Cambridge Assessment International Education

Endorsed for full syllabus coverage

Cambridge International AS & A Level

Computer Science

David Watson
Helen Williams

DYNAMIC LEARNING

HODDER EDUCATION

Cambridge International AS & A Level

Computer Science

David Watson
Helen Williams

HODDER EDUCATION
AN HACHETTE UK COMPANY

Unless otherwise acknowledged, the questions, example answers and comments that appear in this book were written by the authors. In examinations, the way marks are awarded may be different. Questions from the Cambridge International AS & A Level Computer Science papers are reproduced by permission of Cambridge Assessment International Education. Cambridge Assessment International Education bears no responsibility for the example answers to questions taken from its past question papers which are contained in this publication.

The publishers would like to thank the following who have given permission to reproduce the following material in this book:

Page 181 Extract from IEEE Code of Ethics. Reprinted with permission of IEEE with the copyright notice © Copyright 2018 IEEE; **Pages 181–3** Copyright © 1999 by the Institute for Electrical and Electronics Engineers, Inc. and the Association for Computing Machinery, Inc.; **Page 187** eBay software pirates stump up $100,000 – https://www.theregister.co.uk/2006/11/24/ebay_pirates_payup/. Reprinted with permission of Out-Law.com, the news service of international law firm Pinsent Masons; **Page 218** Map data © 2018 Google, Imagery © 2018 Landsat/Copernicus.

Photo credits

Figures 1.1 and 1.2 © David Watson; **Figure 1.3** © Sébastien Delaunay/stock.adobe.com; **Figure 2.18** © Forgem/Shutterstock.com; **Figure 3.1** *tl* © studio306fotolia/stock.adobe.com; *tr* © Chavim/stock.adobe.com; *bl* © pozdeevvs/stock.adobe.com; *br* © Sergey Yarochkin/stock.adobe.com; **Figure 3.4** © Mau Horng/stock.adobe.com; **Figure 3.5** © science photo/stock.adobe.com; **Figure 3.9** © Maksym Dykha/Shutterstock.com; **Figure 3.10** © Hurst Photo/Shutterstock.com; **Figure 3.11** © philipus/stock.adobe.com; **Figure 3.12** © belekekin/ Shutterstock.com; **Figure 4.4** *l* © cybertrone/stock.adobe.com, *c* © Tungphoto/Shutterstock.com, *r* © Luminis/Shutterstock.com; **Figure 5.1** *l* © Stuart Brady (Public Domain) via Wikipedia Commons; *r* © Jiri Hera/stock.adobe.com; **Figure 6.3** © Andrey Burmakin/stock.adobe.com; **Figure 6.4** © bkilzer/stock.adobe.com; **Figure 7.2** *l* © Pres Panayotov/ Shutterstock.com; *c* © James Balog/Getty Images; *r* © caluian/stock.adobe.com; **Figure 18.19** © seewhatmitchsee/ 123rf.com; **Figure 18.21** *b* © Garmon/stock.adobe.com; *t* © Christian Musat/stock.adobe.com; *ct* © Ammit/stock.adobe.com; *cb* © Martina Berg/stock.adobe.com; **Figure 18.24** Harshal/stock.adobe.com; **Figures 18.27 and 18.28** *all* © David Watson.

l = left, *c* = centre, *b* = bottom, *t* = top, *r* = right

Every effort has been made to trace and acknowledge ownership of copyright. The publishers will be glad to make suitable arrangements with any copyright holders whom it has not been possible to contact. Computer hardware and software brand names mentioned in this book are protected by their respective trademarks and are acknowledged.

Although every effort has been made to ensure that website addresses are correct at time of going to press, Hodder Education cannot be held responsible for the content of any website mentioned in this book.

Hachette UK's policy is to use papers that are natural, renewable and recyclable products and made from wood grown in well-managed forests and other controlled sources. The logging and manufacturing processes are expected to conform to the environmental regulations of the country of origin.

Orders: please contact Bookpoint Ltd, 130 Park Drive, Milton Park, Abingdon, Oxon OX14 4SE. Telephone: (44) 01235 827827. Fax: (44) 01235 400401. Email education@bookpoint.co.uk Lines are open from 9 a.m. to 5 p.m., Monday to Saturday, with a 24-hour message answering service. You can also order through our website: www.hoddereducation.com

© David Watson and Helen Williams 2019

First published 2019 by
Hodder Education,
An Hachette UK Company
Carmelite House
50 Victoria Embankment
London EC4Y 0DZ

www.hoddereducation.com

Impression number 10 9 8 7 6 5 4 3 2 1

Year 2023 2022 2021 2020 2019

All rights reserved. Apart from any use permitted under UK copyright law, no part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or held within any information storage and retrieval system, without permission in writing from the publisher or under licence from the Copyright Licensing Agency Limited. Further details of such licences (for reprographic reproduction) may be obtained from the Copyright Licensing Agency Limited, www.cla.co.uk

Cover photo © Terrance Emerson – stock.adobe.com

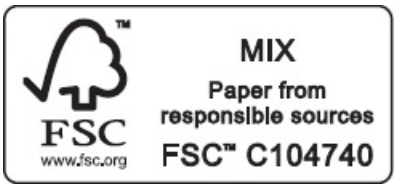
Illustrations by Aptara Inc. and Hodder Education

Typeset by Aptara Inc.

Printed by Bell & Bain Ltd, Glasgow

A catalogue record for this title is available from the British Library.

ISBN: 9781510457591
eISBN: 9781510457607



Contents

Introduction

AS LEVEL

1 Information representation and multimedia
- 1.1 Data representation
- 1.2 Multimedia
- 1.3 File compression

2 Communication
- 2.1 Networking
- 2.2 The internet

3 Hardware
- 3.1 Computers and their components
- 3.2 Logic gates and logic circuits

4 Processor fundamentals
- 4.1 Central processing unit (CPU) architecture
- 4.2 Assembly language
- 4.3 Bit manipulation

5 System software
- 5.1 Operating systems
- 5.2 Language translators

6 Security, privacy and data integrity
- 6.1 Data security
- 6.2 Data integrity

7 Ethics and ownership
- 7.1 Legal, moral, ethical and cultural implications
- 7.2 Copyright issues
- 7.3 Artificial intelligence (AI)

8 Databases
- 8.1 Database concepts
- 8.2 Database management systems (DBMSs)
- 8.3 Data definition language (DDL) and data manipulation language (DML)

9 Algorithm design and problem solving
9.1 Computational thinking skills
9.2 Algorithms

10 Data types and structures
10.1 Data types and records
10.2 Arrays
10.3 Files
10.4 Abstract data types (ADTs)

11 Programming
11.1 Programming basics
11.2 Programming constructs
11.3 Structured programming

12 Software development
12.1 Program development lifecycle
12.2 Program design
12.3 Program testing and maintenance

A LEVEL

13 Data representation
13.1 User-defined data types
13.2 File organisation and access
13.3 Floating-point numbers, representation and manipulation

14 Communication and internet technologies
14.1 Protocols
14.2 Circuit switching and packet switching

15 Hardware
15.1 Processors and parallel processing
15.2 Boolean algebra and logic circuits

16 System software and virtual machines
16.1 Purposes of an operating system (OS)
16.2 Virtual machines (VMs)
16.3 Translation software

17 Security
17.1 Encryption
17.2 Quantum cryptography
17.3 Protocols

17.4 Digital signatures and digital certificates

18 Artificial intelligence (AI)
18.1 Shortest path algorithms
18.2 Artificial intelligence, machine learning and deep learning

19 Computational thinking and problem solving
19.1 Algorithms
19.2 Recursion

20 Further programming
20.1 Programming paradigms
20.2 File processing and exception handling

Glossary

Index

Introduction

This textbook provides the knowledge, understanding and practical skills to support those studying Cambridge International AS & A Level Computer Science. This textbook is part of a suite of resources which include a Programming Skills Workbook and an Online Teacher's Guide.

The syllabus content has been covered comprehensively and is presented in two sections: Chapters 1 to 12 cover the AS Level, Chapters 13 to 20 cover the extra content required for the full A Level.

How to use this book

To make your study of Computer Science as rewarding and successful as possible, this textbook, endorsed by Cambridge Assessment International Education, offers the following important features.

Organisation

The content is presented in the same order as in the syllabus, and the chapter titles match those in the syllabus.

Features to help you learn

Each chapter is broken down into several sections, so that the content is accessible.

At the start of each chapter, there is a blue box that gives a summary of the syllabus points to be covered in that chapter, to show you what you are going to learn.

In this chapter, you will learn about

- binary magnitudes, binary prefixes and decimal prefixes
- binary, denary and hexadecimal number systems
- how to carry out binary addition and subtraction
- the use of hexadecimal and binary coded decimal (BCD) number systems
- the representation of character sets (such as ASCII and Unicode)
- how data for a bit-mapped image is encoded
- how to estimate the file size for a bit-map image
- image resolution and colour depth
- encoding of vector graphics
- the representation of sound in a computer
- the effects of changing sampling rate and resolution on sound quality
- the need for file compression methods (such as lossy and lossless formats)
- how to compress common file formats (such as text files, bit-map images, vector graphics, sound files and video files).

The grey-blue *What you should already know* boxes at the beginning of each chapter or section help you to check you have the right level of knowledge before you begin. You may have already studied Computer Science at IGCSE, O Level or equivalent, or you may not have. These boxes contain questions to find out how much you remember, or to gauge your previous learning. If you are unable to answer the questions, you will need to refresh your memory, or make sure you are familiar with the the relevant ideas, before continuing.

WHAT YOU SHOULD ALREADY KNOW

Try these four questions before you read this chapter.

1 What are the column weightings for the binary number system?

2 Carry out these binary additions. Convert your answers to denary.

a) 0 0 1 1 0 1 0 1 + 0 1 0 0 1 0 0 0

b) 0 1 0 0 1 1 0 1 + 0 1 1 0 1 1 1 0

c) 0 1 0 1 1 1 1 1 + 0 0 0 1 1 1 1 0

d) 0 1 0 0 0 1 1 1 + 0 1 1 0 1 1 1 1

e) 1 0 0 0 0 0 0 1 + 0 1 1 1 0 1 1 1

f) 1 0 1 0 1 0 1 0 + 1 0 1 0 1 0 1 0

3 What are the column weightings for the hexadecimal (base 16) number system?

4 Carry out these hexadecimal additions. Convert your answers to denary.

a) 1 0 7 + 2 5 7

b) 2 0 8 + A 1 7

c) A A A + 7 7 7

d) 1 F F + 7 F 7

e) 1 4 9 + F 0 F

f) 1 2 5 1 + 2 5 6 7

g) 3 4 A B + C 0 0 A

h) A 0 0 1 + D 7 7 F

i) 1 0 0 9 + 9 F F 1

j) 2 7 7 7 + A C F 1

Key terms for each chapter or section are listed, with definitions. When you are reading through the chapter and you come across a term you don't understand, go back and see if it has been explained here.

Key terms

Logic gates – electronic circuits which rely on 'on/off' logic; the most common ones are NOT, AND, OR, NAND, NOR and XOR.

Logic circuit – formed from a combination of logic gates and designed to carry out a particular task; the output from a logic circuit will be 0 or 1.

Truth table – a method of checking the output from a logic circuit; they use all the possible binary input combinations depending on the number of inputs; for example, two inputs have 2^2 (4) possible binary combinations, three inputs will have 2^3 (8) possible binary combinations, and so on.

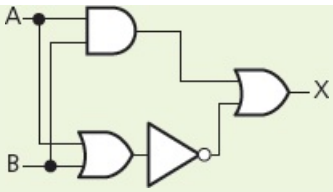
Boolean algebra – a form of algebra linked to logic circuits and based on TRUE and FALSE.

There are *Activities* throughout, so that you can apply what you have learned. Some of these take the form of questions, to allow you to test your knowledge; others aim to give you experience of practical work. Some of these will also give you opportunities to work collaboratively with other students.

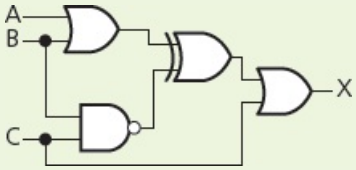
ACTIVITY 3B

Produce truth tables for each of the following logic circuits. You are advised to split them up into intermediate parts to help eliminate errors.

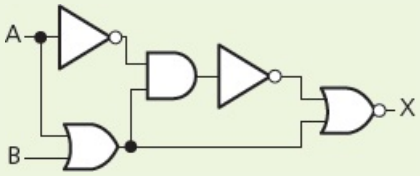
a)



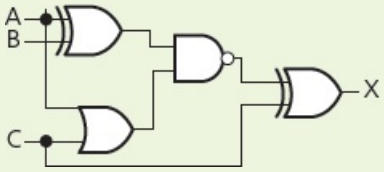
b)



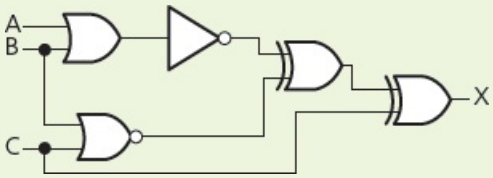
c)



d)



e)



There are also some *Extension* activities. These go beyond the requirements of the syllabus, but it is good to see if you know the answers. We hope they will be of interest to you.

EXTENSION ACTIVITY 3E

1 Look at this simplified diagram of a keyboard; the letter H has been pressed. Explain:

a) how pressing the letter H has been recognised by the computer

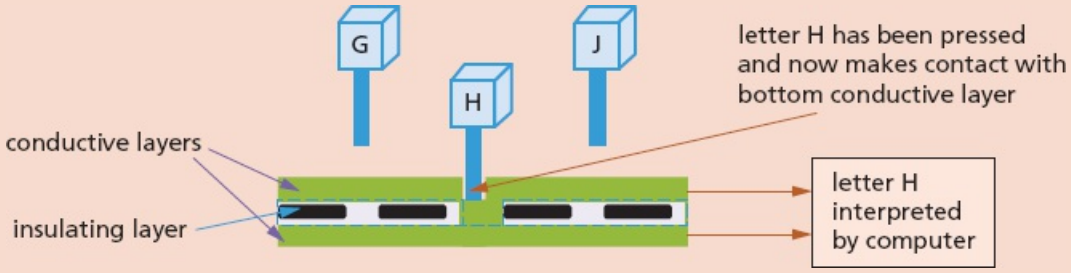
b) how the computer manages the very slow process of inputting data from a keyboard.

2 a) Describe how these types of pointing devices work.

i) Mechanical mouse

ii) Optical mouse

b) Connectivity between mouse and computer can be through USB cable or wireless. Explain these two types of connectivity.



The *End of chapter questions* are practice exam-style questions; these provide a more formal way to check your progress. Some questions from Cambridge International AS & A Level Computer Science past papers are included.

End of chapter questions

1 a) The following bytes represent binary integers using the two's complement form. State the equivalent denary values.

i) 0 1 0 0 1 1 1 1

[1]

ii) 1 0 0 1 1 0 1 0

[1]

iii) Write the integer −53 in two's complement form.

[1]

iv) Write the maximum possible range of numbers using the two's complement form of an 8-bit binary number.
Give your answers in denary.

[2]

b) i) Write the denary integer 798 in binary-coded decimal (BCD) format.

[1]

ii) Write the denary number that is represented by the following BCD number.

1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	0

[2]

c) Give **one** use of binary-coded decimal system.

[1]

Assessment

If you are following the AS Level course, you will take two examination papers:

- Paper 1 Theory Fundamentals (1 hour 30 minutes)
- Paper 2 Fundamental Problem-solving and Programming Skills (2 hours)

If you are studying the A Level course, you will take four examination papers, Papers 1 and 2 and also:

- Paper 3 Advanced Theory (1 hour 30 minutes)
- Paper 4 Practical (2 hours 30 minutes)

Note that calculators must not be used in any paper.

Command words

The table below includes command words used in the assessment for this syllabus. The use of the command word will relate to the subject context. Make sure you are familiar with these.

Command word	What it means
Analyse	examine in detail to show meaning, identify elements and the relationship between them
Assess	make an informed judgement
Calculate	work out from given facts, figures or information
Comment	give an informed opinion
Compare	identify/comment on similarities and/or differences
Complete	add information to an incomplete diagram or table
Consider	review and respond to given information
Contrast	identify/comment on differences
Define	give precise meaning
Demonstrate	show how or give an example
Describe	state the points of a topic/give characteristics and main features
Develop	take forward to a more advanced stage or build upon given information
Discuss	write about issue(s) or topic(s) in depth in a structured way
Draw	draw a line to match a term with a description
Evaluate	judge or calculate the quality, importance, amount, or value of something
Examine	investigate closely, in detail
Explain	set out purposes or reasons/make the relationships between things evident/provide why and/or how and support with relevant evidence
Give	produce an answer from a given source or recall/memory
Identify	name/select/recognise
Justify	support a case with evidence/argument
Outline	set out main points

Predict	suggest what may happen based on available information
Sketch	make a simple freehand drawing showing the key features, taking care over proportions
State	express in clear terms
Suggest	apply knowledge and understanding to situations where there are a range of valid responses in order to make proposals
Summarise	select and present the main points, without detail
Write	write an answer in a specific way

From the authors

We hope you enjoy this book. It encourages you to develop your computational thinking while broadening your understanding of computer science. This should prove helpful when you go on to further study, where topics such as artificial intelligence, quantum cryptography and imperative and declarative programming will be studied; all of these are covered in the later chapters of the book. In order to handle such topics confidently, you will need to be a competent programmer who uses computational thinking to solve problems and has a good understanding of computer architecture. All chapters are designed to build on your previous experience in a way that develops essential skills and at the same time expands the techniques you are able to use.

David Watson

Helen Williams

Notes for teachers

Key concepts

These are the essential ideas that help learners to develop a deep understanding of the subject and to make links between the different topics. Although teachers are likely to have these in mind at all times when they are teaching the syllabus, the following icons are included in the textbook at points where the key concepts relate to the text:

Computational thinking

Computational thinking is a set of fundamental skills that help produce a solution to a problem. Skills such as abstraction, decomposition and algorithmic thinking are used to study a problem and design a solution that can be implemented. This may involve using a range of technologies and programming languages.

Programming paradigms

A programming paradigm is a way of thinking about or approaching problems. There are many different programming styles that can be used, which are suited to unique functions, tools and specific situations. An understanding of programming paradigms is essential to ensure they are used appropriately, when designing and building programs.

Communication

Communication is a core requirement of computer systems. It includes the ability to transfer data from one device or component to another and an understanding of the rules and methods that are used in this data transfer. Communication could range from the internal transfer of data within a computer system, to the transfer of a video across the internet.

Computer architecture and hardware

Computer architecture is the design of the internal operation of a computer system. It includes the rules that dictate how components and data are organised, how data are communicated between components, to allow hardware to function. There is a range of architectures – with different components and rules – that are appropriate for different scenarios. All computers comprise a combination of hardware components, ranging from internal components, such as the central processing unit (CPU) and main memory, to peripherals. To produce effective and efficient programs to run on hardware, it is important to understand how the components work independently and together to produce a system that can be used. Hardware needs software to be able to perform a task. Software allows hardware to become functional. This enables the user to communicate with the hardware to perform tasks.

Data representation and structures

Computers use binary and understanding how a binary number can be interpreted in many

different ways is important. Programming requires an understanding of how data can be organised for efficient access and/or transfer.

Additional support

The Programming Skills Workbook provides practice for the programming papers and includes exercises designed to give students the necessary experience of working in one of the three prescribed high-level programming languages: Java (Console mode), Visual Basic and Python (Console mode). It is a write-in workbook designed to be used throughout the course.

Answers to questions are available in the Online Teacher’s Guide.

1 Information representation and multimedia

In this chapter, you will learn about

- binary magnitudes, binary prefixes and decimal prefixes
- binary, denary and hexadecimal number systems
- how to carry out binary addition and subtraction
- the use of hexadecimal and binary coded decimal (BCD) number systems
- the representation of character sets (such as ASCII and Unicode)
- how data for a bit-mapped image is encoded
- how to estimate the file size for a bit-map image
- image resolution and colour depth
- encoding of vector graphics
- the representation of sound in a computer
- the effects of changing sampling rate and resolution on sound quality
- the need for file compression methods (such as lossy and lossless formats)
- how to compress common file formats (such as text files, bit-map images, vector graphics, sound files and video files).

WHAT YOU SHOULD ALREADY KNOW

Try these four questions before you read this chapter.

1 What are the column weightings for the binary number system?

2 Carry out these binary additions. Convert your answers to denary.

a) 0 0 1 1 0 1 0 1 + 0 1 0 0 1 0 0 0

b) 0 1 0 0 1 1 0 1 + 0 1 1 0 1 1 1 0

c) 0 1 0 1 1 1 1 1 + 0 0 0 1 1 1 1 0

d) 0 1 0 0 0 1 1 1 + 0 1 1 0 1 1 1 1

e) 1 0 0 0 0 0 0 1 + 0 1 1 1 0 1 1 1

f) 1 0 1 0 1 0 1 0 + 1 0 1 0 1 0 1 0

3 What are the column weightings for the hexadecimal (base 16) number system?

4 Carry out these hexadecimal additions. Convert your answers to denary.

a) 1 0 7 + 2 5 7

b) 2 0 8 + A 1 7

c) A A A + 7 7 7

d) 1 F F + 7 F 7

e) 1 4 9 + F 0 F

f) 1 2 5 1 + 2 5 6 7

g) 34AB+C00A
h) A001+D77F
i) 1009+9FF1
j) 2777+ACF1

1.1 Data representation

Key terms

Binary – base two number system based on the values 0 and 1 only.

Bit – abbreviation for binary digit.

One's complement – each binary digit in a number is reversed to allow both negative and positive numbers to be represented.

Two's complement – each binary digit is reversed and 1 is added in right-most position to produce another method of representing positive and negative numbers.

Sign and magnitude – binary number system where left-most bit is used to represent the sign (0 = + and 1 = –); the remaining bits represent the binary value.

Hexadecimal – a number system based on the value 16 (uses the denary digits 0 to 9 and the letters A to F).

Memory dump – contents of a computer memory output to screen or printer.

Binary-coded decimal (BCD) – number system that uses 4 bits to represent each denary digit.

ASCII code – coding system for all the characters on a keyboard and control codes.

Character set – a list of characters that have been defined by computer hardware and software. It is necessary to have a method of coding, so that the computer can understand human characters.

Unicode – coding system which represents all the languages of the world (first 128 characters are the same as ASCII code).

1.1.1 Number systems

Every one of us is used to the decimal or denary (base 10) number system. This uses the digits 0 to 9 which are placed in ‘weighted’ columns.

10000	1000	100	10	units
3	1	4	2	1

The denary number represented above is thirty-one thousand, four hundred and twenty-one.

(Note that dealing with decimal fractions is covered in Chapter 13 since this is slightly more complex.)

Designers of computer systems adopted the **binary** (base 2) number system since this allows only two values, 0 and 1. No matter how complex the system, the basic building block in all computers is the binary number system. Since computers contain millions and millions of tiny ‘switches’, which must be in the ON or OFF position, this lends itself logically to the binary system. A switch in the ON position can be represented by 1; a switch in the OFF position can be represented by 0. Each of the **bi**nary dig**its** are known as **bits**.

1.1.2 Binary number system

The binary system uses 1s and 0s only which gives these corresponding weightings:

128	64	32	16	8	4	2	1
(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

A typical binary number would be:

1	1	1	0	1	1	1	0

Converting from binary to denary and from denary to binary

It is fairly straightforward to change a binary number into a denary number. Each time a 1 appears in a column, the column value is added to the total. For example, the binary number above is:

128 + 64 + 32 + 8 + 4 + 2 = 238 (denary)

The 0 values are simply ignored when calculating the total.

The reverse operation – converting from denary to binary – is slightly more complex. There are two basic ways of doing this.

Consider the conversion of the denary number, 107, into binary …

Method 1

This method involves placing 1s in the appropriate position so that the total equates to 107.

128	64	32	16	8	4	2	1
0	1	1	0	1	0	1	1

ACTIVITY 1A

Convert these binary numbers into denary.

a) 0 0 1 1 0 0 1 1
b) 0 1 1 1 1 1 1 1
c) 1 0 0 1 1 0 0 1
d) 0 1 1 1 0 1 0 0
e) 1 1 1 1 1 1 1 1
f) 0 0 0 0 1 1 1 1
g) 1 0 0 0 1 1 1 1
h) 0 0 1 1 0 0 1 1
i) 0 1 1 1 0 0 0 0
j) 1 1 1 0 1 1 1 0

Method 2

This method involves successive division by 2; the remainders are then written from bottom to

top to give the binary value.

2	107	
2	53	remainder: 1
2	26	remainder: 1
2	13	remainder: 0
2	6	remainder: 1
2	3	remainder: 0
2	1	remainder: 1
2	0	remainder: 1
	0	remainder: 0

Write the remainder from bottom to top to get the binary number:

0 1 1 0 1 0 1 1

ACTIVITY 1B

Convert these denary numbers into binary (using either method).

a) 4 1
b) 6 7
c) 8 6
d) 1 0 0
e) 1 1 1
f) 1 2 7
g) 1 4 4
h) 1 8 9
i) 2 0 0
j) 2 5 5

Binary addition and subtraction

Up until now we have assumed all binary numbers have positive values. There are a number of methods to represent both positive and negative numbers. We will consider:

- one's complement
- two's complement.

In **one's complement**, each digit in the binary number is inverted (in other words, 0 becomes 1 and 1 becomes 0). For example, 0 1 0 1 1 0 1 0 (denary value 90) becomes 1 0 1 0 0 1 0 1 (denary value −90).

In **two's complement**, each digit in the binary number is inverted and a '1' is added to the right-most bit. For example, 0 1 0 1 1 0 1 0 (denary value 90) becomes:

```
   1  0  1  0  0  1  0  1
+                     1
=  1  0  1  0  0  1  1  0  (since 1 + 1 = 0, a carry of 1) = denary value –90
```

Throughout the remainder of this chapter, we will use the two's complement method to avoid confusion. Also, two's complement makes binary addition and subtraction more straightforward. The reader is left to investigate one's complement and the **sign and magnitude** method in binary arithmetic.

Now that we are introducing negative numbers, we need a way to represent these in binary. The two's complement uses these weightings for an 8-bit number representation:

−128	64	32	16	8	4	2	1

This means:

−128	64	32	16	8	4	2	1
1	1	0	1	1	0	1	0
0	0	1	0	0	1	1	0

The first example is: −128 + 64 + 16 + 8 + 2 = −38

EXTENSION ACTIVITY 1A

Show the column headings for a system that uses 16 bits to represent a binary number.

The second example is: 32 + 4 + 2 = 38

The easiest way to convert a number into its negative equivalent is to use two's complement. For example, 104 in binary is 0 1 1 0 1 0 0 0.

To find the binary value for −104 using two's complement:

invert the digits:	1	0	0	1	0	1	1	1 (+104 in denary)
add 1:								1
which gives:	1	0	0	1	1	0	0	0 = −104)

ACTIVITY 1C

Convert these denary numbers into 8-bit binary numbers using two's complement where necessary. Use these binary column weightings:

−128	64	32	16	8	4	2	1

a) +114
b) +61
c) +96
d) −14
e) −116

Binary addition

Consider Examples 1.1 and 1.2.

Example 1.1

Add 0 0 1 0 0 1 0 1 (37 in denary) and 0 0 1 1 1 0 1 0 (58 in denary).

Solution

−128	64	32	16		8	4	2	1
0	0	1	0		0	1	0	1
				+				
0	0	1	1		1	0	1	0
				=				
0	1	0	1		1	1	1	1

This gives us 0 1 0 1 1 1 1 1, which is 95 in denary; the correct answer.

Example 1.2

Add 0 1 0 1 0 0 1 0 (82 in denary) and 0 1 0 0 0 1 0 1 (69 in denary).

Solution

−128	64	32	16		8	4	2	1
0	1	0	1		0	0	1	0
				+				
0	1	0	0		0	1	0	1
				=				
1	0	0	1		0	1	1	1

This gives us 1 0 0 1 0 1 1 1, which is –105 in denary (which is clearly nonsense). When adding two positive numbers, the result should always be positive (likewise, when adding two negative numbers, the result should always be negative). Here, the addition of two positive numbers has resulted in a negative answer. This is due to the result of the addition producing a number which is outside the range of values which can be represented by the 8 bits being used (in this case +127 is the largest value which can be represented, and the calculation produces the value 151, which is larger than 127 and, therefore, out of range). This causes overflow; it is considered in more detail in Chapter 13.

Binary subtraction

To carry out subtraction in binary, we convert the number being subtracted into its negative equivalent using two's complement, and then *add* the two numbers.

Example 1.3

Carry out the subtraction 95 – 68 in binary.

Solution

1 Convert the two numbers into binary:

95 = 0 1 0 1 1 1 1 1

68 = 0 1 0 0 0 1 0 0

2 Find the two's complement of 68:

invert the digits:	1	0	1	1	1	0	1	1	
add 1:								1	
which gives:	1	0	1	1	1	1	0	0	= −68

3 Add 95 and −68:

	−128	64	32	16	8	4	2	1
	0	1	0	1	1	1	1	1
+								
	1	0	1	1	1	1	0	0
=								
1	0	0	0	1	1	0	1	1

The additional ninth bit is simply ignored leaving the binary number 0 0 0 1 1 0 1 1 (denary equivalent of 27, which is the correct result of the subtraction).

Example 1.4

Carry out the subtraction 49 – 80 in binary.

Solution

1 Convert the two numbers into binary:

49 = 0 0 1 1 0 0 0 1

80 = 0 1 0 1 0 0 0 0

2 Find the two's complement of 80:

invert the digits:	1	0	1	0	1	1	1	1	
add 1:								1	
which gives:	1	0	1	1	0	0	0	0	= −80

3 Add 49 and −80:

−128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	1
			+				
1	0	1	1	0	0	0	0
			=				
1	1	1	0	0	0	0	1

This gives us 1 1 1 0 0 0 0 1, which is −31 in denary; the correct answer.

ACTIVITY 1D

Carry out these binary additions and subtractions using these 8-bit column weightings:

−128 64 32 16 8 4 2 1

a) 0 0 1 1 1 0 0 1 + 0 0 1 0 1 0 0 1
b) 0 1 0 0 1 0 1 1 + 0 0 1 0 0 0 1 1
c) 0 1 0 1 1 0 0 0 + 0 0 1 0 1 0 0 0
d) 0 1 1 1 0 0 1 1 + 0 0 1 1 1 1 1 0
e) 0 0 0 0 1 1 1 1 + 0 0 0 1 1 1 0 0
f) 0 1 1 0 0 0 1 1 − 0 0 1 1 0 0 0 0
g) 0 1 1 1 1 1 1 1 − 0 1 0 1 1 0 1 0
h) 0 0 1 1 0 1 0 0 − 0 1 0 0 0 1 0 0
i) 0 0 0 0 0 0 1 1 − 0 1 1 0 0 1 0 0
j) 1 1 0 1 1 1 1 1 − 1 1 0 0 0 0 1 1

Measurement of the size of computer memories

The byte is the smallest unit of memory in a computer. Some computers use larger bytes, such as 16-bit systems and 32-bit systems, but they are always multiples of 8. 1 byte of memory wouldn't allow you to store very much information; so memory size is measured in these multiples. See Table 1.1.

Name of memory size	Equivalent denary value (bytes)
1 kilobyte (1 KB)	1 000
1 megabyte (1 MB)	1 000 000
1 gigabyte (1 GB)	1 000 000 000
1 terabyte (1 TB)	1 000 000 000 000
1 petabyte (1 PB)	1 000 000 000 000 000

Table 1.1 Memory size and denary values

The system of numbering shown in Table 1.1 only refers to some storage devices, but is technically inaccurate. It is based on the SI (base 10) system of units where 1 kilo is equal to 1000. A 1 TB hard disk drive would allow the storage of 1×10^{12} bytes according to this system. However, since memory size is actually measured in terms of powers of 2, another system has been proposed by the International Electrotechnical Commission (IEC); it is based on the binary system. See Table 1.2.

Name of memory size	Number of bytes	Equivalent denary value (bytes)
1 kibibyte (1 KiB)	2^{10}	1 024
1 mebibyte (1 MiB)	2^{20}	1 048 576
1 gibibyte (1 GiB)	2^{30}	1 073 741 824
1 tebibyte (1 TiB)	2^{40}	1 099 511 627 776
1 pebibyte (1 PiB)	2^{50}	1 125 899 906 842 624

Table 1.2 IEC memory size system

This system is more accurate. Internal memories (such as RAM) should be measured using the IEC system. A 64 GiB RAM could, therefore, store 64×2^{30} bytes of data (68 719 476 736 bytes).

See Section 1.2 for examples of how to calculate the size of a file.

1.1.3 Hexadecimal number system

The **hexadecimal** system is very closely related to the binary system. Hexadecimal (sometimes referred to as simply hex) is a base 16 system with the weightings:

1 048 576	65 536	4096	256	16	1
(16^5)	(16^4)	(16^3)	(16^2)	(16^1)	(16^0)

Because it is a system based on 16 different digits, the numbers 0 to 9 and the letters A to F are used to represent hexadecimal digits.

A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15.

Since $16 = 2^4$, **four** binary digits are equivalent to each hexadecimal digit. Table 1.3 summarises the link between binary, hexadecimal and denary.

Binary value	Hexadecimal value	Denary value
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	A	10
1 0 1 1	B	11
1 1 0 0	C	12
1 1 0 1	D	13
1 1 1 0	E	14
1 1 1 1	F	15

Table 1.3 The link between binary, hexadecimal and denary

Converting from binary to hexadecimal and from hexadecimal to

binary

Converting from binary to hexadecimal is a fairly easy process. Starting from the right and moving left, split the binary number into groups of 4 bits. If the last group has less than 4 bits, then simply fill in with 0s from the left. Take each group of 4 bits and convert it into the equivalent hexadecimal digit using Table 1.3.

Examples 1.5 and 1.6 show you how this works.

Example 1.5

Convert 1 0 1 1 1 1 1 0 0 0 0 1 from binary to hexadecimal.

Solution

First split it into groups of 4 bits:

1 0 1 1 1 1 1 0 0 0 0 1

Then find the equivalent hexadecimal digits:

B E 1

Example 1.6

Convert 1 0 0 0 0 1 1 1 1 1 1 1 0 1 from binary to hexadecimal.

Solution

First split it into groups of 4 bits:

1 0 0 0 0 1 1 1 1 1 1 0 1

The left group only contains 2 bits, so add in two 0s to the left:

0 0 1 0 0 0 0 1 1 1 1 1 1 0 1

Now find the equivalent hexadecimal digits:

2 1 F D

ACTIVITY 1E

Convert these binary numbers into hexadecimal.

a) 1 1 0 0 0 0 1 1

b) 1 1 1 1 0 1 1 1

c) 1 0 0 1 1 1 1 1 1 1

d) 1 0 0 1 1 1 0 1 1 1 0

e) 0 0 0 1 1 1 1 0 0 0 0 1

f) 1 0 0 0 1 0 0 1 1 1 1 0

g) 0 0 1 0 0 1 1 1 1 1 1 1 0

h) 0 1 1 1 0 1 0 0 1 1 1 0 0

i) 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1

j) 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0

Converting from hexadecimal to binary is also straightforward. Using the data from Table 1.3, simply take each hexadecimal digit and write down the 4 bit code which corresponds to the digit.

Example 1.7

Convert this hexadecimal number to its binary equivalent.

4 5 A

Solution

Using Table 1.3, find the 4-bit code for each digit:

0 1 0 0 0 1 0 1 1 0 1 0

Put the groups together to form the binary number:

0 1 0 0 0 1 0 1 1 0 1 0

Example 1.8

Convert this hexadecimal number to its binary equivalent.

B F 0 8

Solution

Using Table 1.3:

1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0

Then put all the digits together:

1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0

Use of the hexadecimal system

This section reviews two uses of the hexadecimal system.

Memory dumps

It is much easier to work with:

B 5 A 4 1 A F C

than it is to work with:

1 0 1 1 0 1 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 0

So, hexadecimal is often used when developing new software or when trying to trace errors in programs. When the memory contents are output to a printer or monitor, this is known as a

memory dump.

ACTIVITY 1F

Convert these hexadecimal numbers into binary.

a) 6 C
b) 5 9
c) A A
d) A 0 0
e) 4 0 E
f) B A 6
g) 9 C C
h) 4 0 A A
i) D A 4 7
j) 1 A B 0

A program developer can look at each of the hexadecimal codes (as shown in Table 1.4) and determine where the error lies. The value on the far left shows the memory location, so it is possible to find out exactly where in memory the fault occurs. Using hexadecimal is more manageable than binary. It is a powerful fault-tracing tool, but requires considerable knowledge of computer architecture to be able to interpret the results.

00990F60	54	68	69	73	20	69	73	20	61	6E	20	65	78	61	6D	70	6C	65	20	6F	66
00990F77	61	20	6D	65	6D	6F	72	79	20	64	75	6D	70	20	66	72	6F	6D	20	20	61
00990E8E	74	79	70	69	63	61	6C	20	20	63	6F	6D	70	75	74	65	72	20	20	6D	85
00990EA5	6D	6F	72	79	20	73	68	6F	77	69	6E	67	20	74	68	65	20	20	63	6F	6E
00990EBC	74	65	6E	74	73	20	6F	66	20	61	20	6E	75	6D	62	65	72	20	20	6F	66
00990ED3	6C	6F	63	61	74	69	6F	6E	73	20	20	69	6E	20	20	68	65	78	20	20	20
00990EEA	6E	6F	74	61	74	69	6F	6E	20	20	00	00	00	00	00	00	00	00	00	00	00

Table 1.4 Memory dump

1.1.4 Binary-coded decimal (BCD) system

The **binary-coded decimal (BCD)** system uses a 4-bit code to represent each denary digit:

0 0 0 0 = 0	0 1 0 1 = 5
0 0 0 1 = 1	0 1 1 0 = 6
0 0 1 0 = 2	0 1 1 1 = 7
0 0 1 1 = 3	1 0 0 0 = 8
0 1 0 0 = 4	1 0 0 1 = 9

Therefore, the denary number 3 1 6 5 would be 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 in BCD format.

The 4-bit code can be stored in the computer either as half a byte or two 4-bit codes stored together to form one byte. For example, using 3 1 6 5 again …

Method 1: four single bytes

0	0	0	0	0	0	1	1	3
0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	0	6
0	0	0	0	0	1	0	1	5

Method 2: two bytes

0	0	1	1	0	0	0	1	3	1
0	1	1	0	0	1	0	1	6	5

ACTIVITY 1G

1 Convert these denary numbers into BCD format.
   a) 2 7 1
   b) 5 0 0 6
   c) 7 9 9 0
2 Convert these BCD numbers into denary numbers.
   a) 1 0 0 1 0 0 1 1 0 1 1 1
   b) 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 0

Uses of BCD

The most obvious use of BCD is in the representation of digits on a calculator or clock display.

Each denary digit will have a BCD equivalent value which makes it easy to convert from computer output to denary display.

As you will learn in Chapter 13, it is nearly impossible to represent decimal values exactly in computer memories which use the binary number system. Normally this doesn't cause a major issue since the differences can be dealt with. However, when it comes to accounting and representing monetary values in computers, *exact* values need to be stored to prevent significant errors from accumulating. Monetary values use a fixed-point notation, for example $1.31, so one solution is to represent each denary digit as a BCD value.

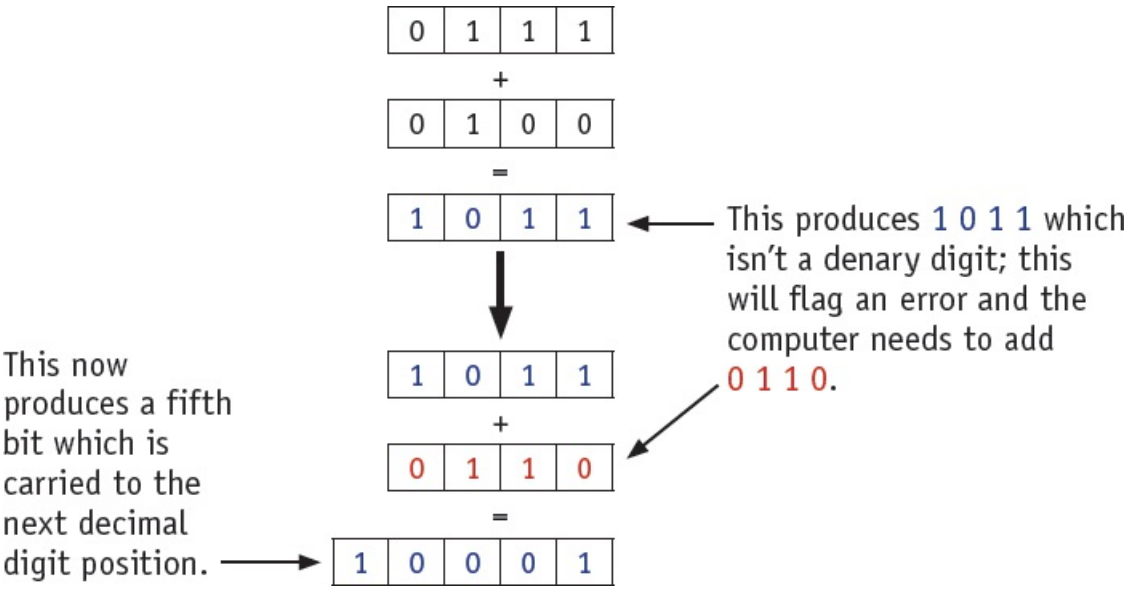
Consider adding $0.37 and $0.94 together using fixed-point decimals.

$0.37	0 0 0 0 0 0 0 0 . 0 0 1 1 0 1 1 1	
+	+	
$0.94	0 0 0 0 0 0 0 0 . 1 0 0 1 0 1 0 0	Expected result = $1.31

Using binary addition, this sum will produce:

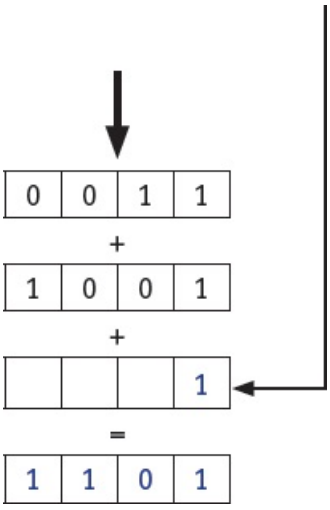
0 0 0 0 0 0 0 0 . 1 1 0 0 1 0 1 1 which produces 1 1 0 0 (denary 12) and 1 0 1 1 (denary 11), which is clearly incorrect. The problem was caused by 3 + 9 = 12 and 7 + 4 = 11, as neither 12 nor 11 are single denary digits. The solution to this problem, enabling the computer to store monetary values accurately, is to add 0 1 1 0 (denary 6) whenever such a problem arises. The computer can be programmed to recognise this issue and add 0 1 1 0 at each appropriate point.

If we look at the example again, we can add .07 and .04 (the two digits in the second decimal place) first.

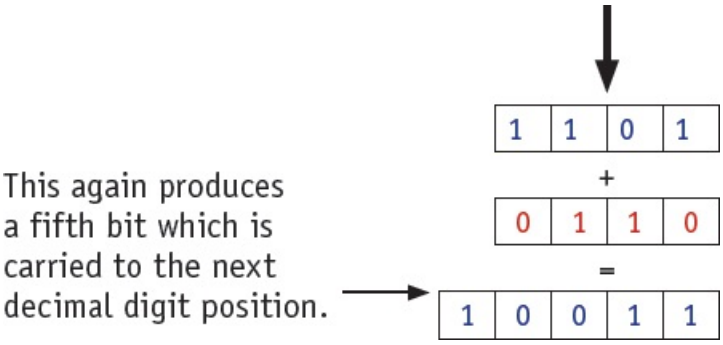


Now we will add .3 and .9 together (the two digits in the first decimal place) remembering the

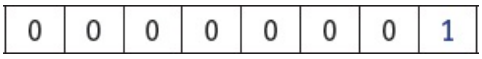
carry bit from the addition above:



This produces 1 1 0 1 which isn't a denary digit; this will flag an error and the computer again needs to add 0 1 1 0.



Adding 1 to 0 0 0 0 0 0 0 0 produces:



Final answer:

which is 1.31 in denary – the correct answer.

ACTIVITY 1H

Carry out these BCD additions.

a) 0.45 + 0.21

b) 0.66 + 0.51

c) 0.88 + 0.75

1.1.5 ASCII codes and Unicodes

The **ASCII code** system (American Standard Code for Information Interchange) was set up in 1963 for use in communication systems and computer systems. The newer version of the code was published in 1986. The standard ASCII code **character set** consists of 7-bit codes (0 to 127 denary or 0 to 7F in hexadecimal); this represents the letters, numbers and characters found on a standard keyboard together with 32 control codes (which use up codes 0 to 31 (denary) or 0 to 19 (hexadecimal)).

Table 1.5 shows part of the standard ASCII code table (only the control codes have been removed from the table).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20	<SPACE>	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	<DELETE>

▲ **Table 1.5** Part of the ASCII code table

Notice the storage of characters with uppercase and lowercase. For example:

a	1	1	0	0	0	0	1	hex 61 (lower case)
A	1	0	0	0	0	0	1	hex 41 (upper case)
y	1	1	1	1	0	0	1	hex 79 (lower case)
Y	1	0	1	1	0	0	1	hex 59 (uppercase)

Notice the sixth bit changes from 1 to 0 when comparing lower and uppercase characters. This makes the conversion between the two an easy operation. It is also noticeable that the character sets (such as a to z, 0 to 9, and so on) are grouped together in sequence, which speeds up usability.

Extended ASCII uses 8-bit codes (128 to 255 in denary or 80 to FF in hex). This allows for non-English characters and for drawing characters to be included.

Since ASCII code has a number of disadvantages and is unsuitable for some purposes, different methods of coding have been developed over the years. One coding system is called **Unicode**. Unicode allows characters in a code form to represent all languages of the world, thus supporting many operating systems, search engines and internet browsers used globally. There is overlap with standard ASCII code, since the first 128 (English) characters are the same, but Unicode can support several thousand different characters in total. As can be seen in Tables 1.5 and 1.6, ASCII uses one byte to represent a character, whereas Unicode will support up to four bytes per character.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	154	9A	Ü	180	B4	┤	206	CE	╬	232	E8	Φ
129	81	ü	155	9B	¢	181	B5	╡	207	CF	╧	233	E9	Θ
130	82	é	156	9C	£	182	B6	╢	208	D0	╨	234	EA	Ω
131	83	â	157	9D	¥	183	B7	╖	209	D1	╤	235	EB	δ
132	84	ä	158	9E	₧	184	B8	╕	210	D2	╥	236	EC	∞
133	85	à	159	9F	ƒ	185	B9	╣	211	D3	╙	237	ED	ø
134	86	å	160	A0	á	186	BA	║	212	D4	╘	238	EE	ε
135	87	ç	161	A1	í	187	BB	╗	213	D5	╒	239	EF	∩
136	88	ê	162	A2	ó	188	BC	╝	214	D6	╓	240	F0	≡
137	89	ë	163	A3	ú	189	BD	╜	215	D7	╫	241	F1	±
138	8A	è	164	A4	ñ	190	BE	╛	216	D8	╪	242	F2	≥
139	8B	ï	165	A5	Ñ	191	BF	┐	217	D9	┘	243	F3	≤
140	8C	î	166	A6	ª	192	C0	└	218	DA	┌	244	F4	⌠
141	8D	ì	167	A7	º	193	C1	┴	219	DB	▉	245	F5	⌡
142	8E	Ä	168	A8	¿	194	C2	┬	220	DC	▄	246	F6	÷
143	8F	Å	169	A9	⌐	195	C3	├	221	DD	▌	247	F7	≈
144	90	É	170	AA	¬	196	C4	─	222	DE	▐	248	F8	°
145	91	æ	171	AB	½	197	C5	┼	223	DF	▀	249	F9	∙
146	92	Æ	172	AC	¼	198	C6	╞	224	E0	α	250	FA	·
147	93	ô	173	AD	¡	199	C7	╟	225	E1	ß	251	FB	√
148	94	ö	174	AE	«	200	C8	╚	226	E2	Γ	252	FC	³
149	95	ò	175	AF	»	201	C9	╔	227	E3	π	253	FD	²
150	96	û	176	B0	░	202	CA	╩	228	E4	∑	254	FE	■
151	97	ù	177	B1	▒	203	CB	╦	229	E5	σ	255	FF	
152	98	ÿ	178	B2	▓	204	CC	╠	230	E6	µ			
153	99	Ö	179	B3	│	205	CD	═	231	E7	τ			

▲ **Table 1.6** Extended ASCII code table

The Unicode consortium was set up in 1991. Version 1.0 was published with five goals, these were to

- create a universal standard that covered all languages and all writing systems
- produce a more efficient coding system than ASCII
- adopt uniform encoding where each character is encoded as 16-bit or 32-bit code
- create unambiguous encoding where each 16-bit or 32-bit value always represents the same character (it is worth pointing out here that the ASCII code tables are not standardised and versions other than the ones shown in tables 1.5 and 1.6 exist)
- reserve part of the code for private use to enable a user to assign codes for their own characters and symbols (useful for Chinese and Japanese character sets).

A sample of Unicode characters are shown in Table 1.7. As can be seen from the table, characters used in languages such as Russian, Greek, Romanian and Croatian can now be represented in a computer).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01A0	Ơ	ơ	Ƣ	ƣ	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Σ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư
01B0	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ
01C0	ǀ	ǁ	ǂ	!	DŽ	Dž	dž	LJ	Lj	lj	NJ	Nj	nj	Ǎ	ǎ	Ǐ
01D0	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ
01E0	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ
01F0	ǰ	DZ	Dz	dz	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ
0200	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
0210	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
0220	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
0230	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ƚ	Ⱦ	ȿ
0240	ɀ	ɁɁ	ɂ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ
0250	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ
0260	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ
0270	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ
0280	ʀ	ʁ	ʂ	ʃ	ʄ	ʅ	ʆ	ʇ	ʈ	ʉ	ʊ	ʋ	ʌ	ʍ	ʎ	ʏ
0290	ʐ	ʑ	ʒ	ʓ	ʔ	ʕ	ʖ	ʗ	ʘ	ʙ	ʚ	ʛ	ʜ	ʝ	ʞ	ʟ
02A0	ʠ	ʡ	ʢ	ʣ	ʤ	ʥ	ʦ	ʧ	ʨ	ʩ	ʪ	ʫ	ʬ	ʭ	ʮ	ʯ
02B0	ʰ	ʱ	ʲ	ʳ	ʴ	ʵ	ʶ	ʷ	ʸ	ʹ	ʺ	ʻ	ʼ	ʽ	ʾ	ʿ

▲ **Table 1.7** Sample of Unicode characters

1.2 Multimedia

Key terms

Bit-map image – system that uses pixels to make up an image.

Pixel – smallest picture element that makes up an image.

Colour depth – number of bits used to represent the colours in a pixel, e.g. 8 bit colour depth can represent $2^8 = 256$ colours.

Bit depth – number of bits used to represent the smallest unit in, for example, a sound or image file – the larger the bit depth, the better the quality of the sound or colour image.

Image resolution – number of pixels that make up an image, for example, an image could contain 4096 × 3192 pixels (12 738 656 pixels in total).

Screen resolution – number of horizontal and vertical pixels that make up a screen display. If the screen resolution is smaller than the image resolution, the whole image cannot be shown on the screen, or the original image will become lower quality.

Resolution – number of pixels per column and per row on a monitor or television screen.

Pixel density – number of pixels per square centimetre.

Vector graphics – images that use 2D points to describe lines and curves and their properties that are grouped to form geometric shapes.

Sampling resolution – number of bits used to represent sound amplitude (also known as bit depth).

Sampling rate – number of sound samples taken per second.

Frame rate – number of video frames that make up a video per second.

Images can be stored in a computer in two common formats: bit-map image and vector graphic.

1.2.1 Bit-map images

Bit-map images are made up of **pixels** (picture elements); the image is stored in a two-dimensional matrix of pixels.

Pixels can take different shapes, such as or

When storing images as pixels, we have to consider

- at least 8 bits (1 byte) per pixel are needed to code a coloured image (this gives 256 possible colours by varying the intensity of the blue, green and red elements)
- true colour requires 3 bytes per pixel (24 bits), which gives more than one million colours
- the number of bits used to represent a pixel is called the **colour depth**.

EXTENSION ACTIVITY 1B

Find out how HTML is used to control the colour of each pixel on a screen. How is HTML used in the design stage of a web page screen layout?

In terms of images, we need to distinguish between **bit depth** and colour depth; for example, the number of bits that are used to represent a single pixel (bit depth) will determine the colour depth of that pixel. As the bit depth increases, the number of possible colours which can be represented also increases. For example, a bit depth of 8 bits per pixel allows 256 (2^8) different colours (the colour depth) to be represented, whereas using a bit depth of 32 bits per pixel results in 4 294 967 296 (2^{32}) different colours. The impact of bit depth and colour depth is considered later.

We will now consider the actual image itself and how it can be displayed on a screen. There are two important definitions here:

- **Image resolution** refers to the number of pixels that make up an image; for example, an image could contain 4096 × 3192 pixels (12 738 656 pixels in total).
- **Screen resolution** refers to the number of horizontal pixels and the number of vertical pixels that make up a screen display (for example, if the screen resolution is smaller than the image resolution then the whole image cannot be shown on the screen or the original image will now be a lower quality).

We will try to clarify the difference by using an example.

Figure 1.1 has been taken by a digital camera using an image resolution of 4096 × 3192 pixels:

Figure 1.1 Image taken by a digital camera

Suppose we wish to display Figure 1.1 on a screen with screen resolution of 1920 × 1080. To display this image the web browser (or other software) would need to re-size Figure 1.1 so that it now fits the screen. This could be done by removing pixels so that it could now be displayed, or part of the image could be cropped (and, in this case, rotated through 90°) as shown in Figure 1.2.

Figure 1.2 Image cropped and rotated through 90°

However, a lower **resolution** copy of Figure 1.1 (for example, 1024 × 798) would now fit on the screen without any modification to the image. We could simply zoom in to enlarge it to full

screen size; however, the image could now become pixelated (in other words, the number of pixels per square inch (known as the **pixel density**) is smaller, causing deterioration in the image quality).

We will now consider a calculation which shows how pixel density can be calculated for a given screen. Imagine we are using an Apple iPhone 8 which has 5.5-inch screen size and screen resolution of 1920 pixels × 1080 pixels:

1. add together the squares of the resolution size ($(1920^2 + 1080^2)$ = (3 686 400 + 16 640) = 4 852 800)
2. find the square root $\left(\sqrt{4852800} = 2202.907\right)$
3. divide by screen size (2202.907 ÷ 5.5 = 401)

This gives us the pixel density of 401 pixels per square inch (ppi) (which is the same as the published figure from the manufacturer).

A pixel-generated image can be scaled up or scaled down; it is important to understand that this can be done when deciding on the resolution. The resolution can be varied on many cameras before taking, for example, a digital photograph. When magnifying an image, the number of pixels that makes up the image remains the same but the area they cover is now increased. This means some of the sharpness could be lost. This is known as the pixel density and is key when scaling up photographs. For example, look at Figure 1.3.

Figure 1.3 Five images of the same car wheel

Image A is the original. By the time it has been scaled up to make image E it has become pixelated ('fuzzy'). This is because images A and E have different pixel densities.

The main drawback of using high resolution images is the increase in file size. As the number of pixels used to represent the image is increased, the size of the file will also increase. This impacts on how many images can be stored on, for example, a hard drive. It also impacts on the time to download an image from the internet or the time to transfer images from device to device. Bit-map images rely on certain properties of the human eye and, up to a point, the amount of file compression used (see Section 1.3 File compression). The eye can tolerate a certain amount of resolution reduction before the loss of quality becomes significant.

Calculating bit-map image file sizes

It is possible to estimate the file size needed to store a bit-map image. The file size will need to take into account the image resolution and bit depth.

For example, a full screen with a resolution of 1920 × 1080 pixels and a bit depth of 24 requires 1920 × 1080 × 24 bits = 49 766 400 bits for the full screen image.

EXTENSION ACTIVITY 1C

Calculate the file size needed to store the screen image on a UHD television.

Dividing by 8 gives us 6 220 800 bytes (equivalent to 6.222 MB using the SI units or 5.933 MiB using IEE units). An image which does not occupy the full screen will obviously result in a smaller file size.

Note: when saving a bit-map image, it is important to include a file header; this will contain items such as file type (.bmp or .jpeg), file size, image resolution, bit depth (usually 1, 8, 16, 24 or 32), any type of data compression employed and so on.

1.2.2 Vector graphics

Vector graphics are images that use 2D points to describe lines and curves and their properties that are grouped to form geometric shapes. Vector graphics can be designed using computer aided design (CAD) software or using an application which uses a drawing canvas on the screen. See Figure 1.4.

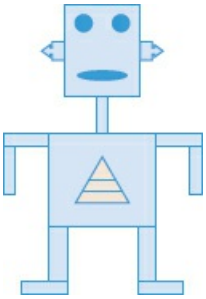


Figure 1.4 Drawing of a robot made up of a number of geometric shapes

A vector graphic will contain a drawing list (included in a file header) that is made up of

- the command used for each object that makes up the graphic image
- the attributes that define the properties that make up each object (for example consider the ellipse of the robot's mouth – this will need the position of the two centres, the radius from centres, the thickness and style of each line, the line colour and any fill colour used)
- the relative position of each object will also need to be included
- the dimensions of each object are not defined, but the relative positions of objects to each other in the final graphic need to be defined; this means that scaling up the vector graphic image will result in no loss of quality.

When printing out vector graphics it is usually necessary to first convert it into a bit-map image to match the format of most printers.

Comparison between vector graphics and bit-map images

Vector graphic images	Bit-map images
made up of geometric shapes which require definition/attributes	made up of tiny pixels of different colours
to alter/edit the design, it is necessary to change each of the geometric shapes	possible to alter/edit each of the pixels to change the design of the image
they do not require large file size since it is made up of simple geometric shapes	because of the use of pixels (which give very accurate designs), the file size is very large
because the number of geometric shapes is limited, vector graphics are not usually very realistic	since images are built up pixel by pixel, the final image is usually very realistic

file formats are usually .svg, .cgm, .odg	file formats are usually .jpeg, .bmp, .png

Table 1.8 Comparison between vector graphics and bit-map images

It is now worth considering whether a vector graphic or a bit-map image would be the best choice for a given application. When deciding which is the better method, we should consider the following:

- Does the image need to be resized? If so, a vector graphic could be the best option.
- Does the image need to be drawn to scale? Again, a vector graphic is probably the best option.
- Does the image need to look real? Usually bit-map images look more realistic than vector graphics.
- Are there file restrictions? If so, it is important to consider whether vector graphic images can be used; if not, it would be necessary to consider the image resolution of a bit-map image to ensure the file size is not too large.

For example, when designing a logo for a company or composing an ‘exploded diagram’ of a car engine, vector graphics are the best choice.

However, when modifying photographs using photo software, the best method is to use bit-map images.

1.2.3 Sound files

Sound requires a medium in which to travel through (it cannot travel in a vacuum). This is because it is transmitted by causing oscillations of particles within the medium. The human ear picks up these oscillations (changes in air pressure) and interprets them as sound. Each sound wave has a frequency and wavelength; the amplitude specifies the loudness of the sound.

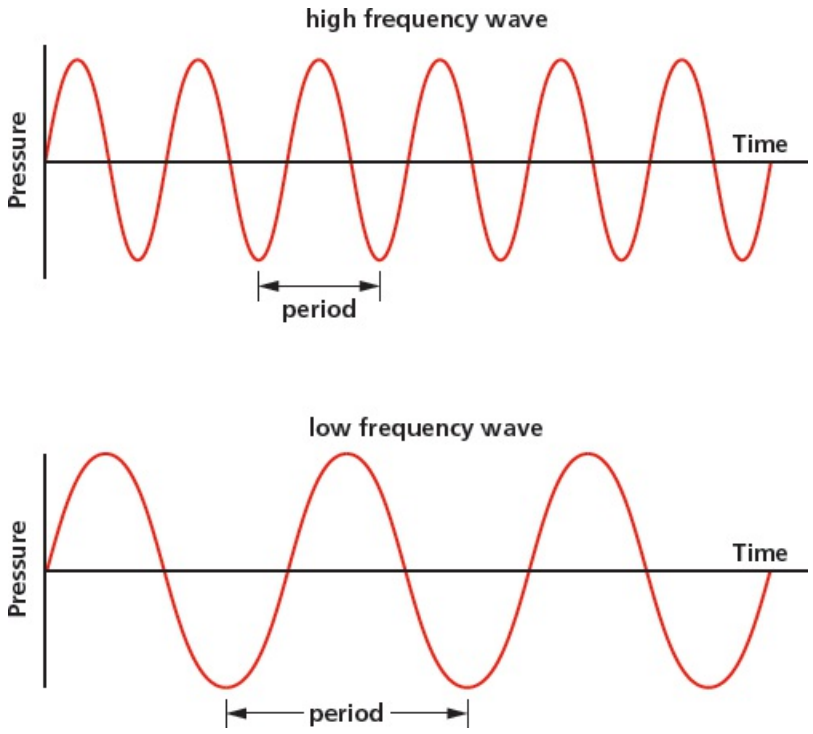


Figure 1.5 High and low frequency wave signals

Sound is an analogue value; this needs to be digitised in order to store sound in a computer. This is done using an analogue to digital converter (ADC). If the sound is to be used as a music file, it is often filtered first to remove higher frequencies and lower frequencies which are outside the range of human hearing. To convert the analogue data to digital, the sound waves are sampled at a given time rate. The amplitude of the sound cannot be measured precisely, so approximate values are stored.

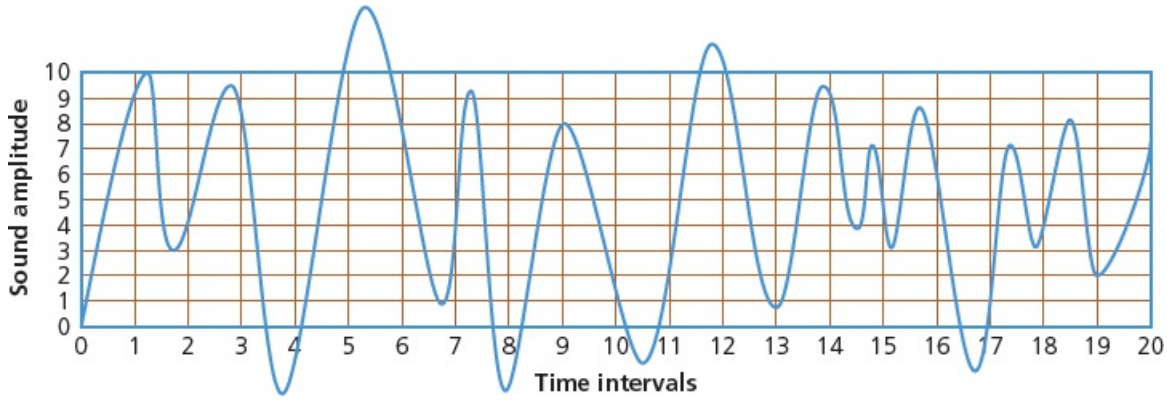


Figure 1.6 A sound wave

Figure 1.6 shows a sound wave. The *x*-axis shows the time intervals when the sound was sampled (0 to 20), and the *y*-axis shows the amplitude of the sampled sound (the amplitudes above 10 and below 0 are filtered out in this example).

At time interval 1, the approximate amplitude is 9; at time interval 2, the approximate amplitude is 4, and so on for all 20 time intervals. Because the amplitude range in Figure 1.6 is 0 to 10, then 4 binary bits can be used to represent each amplitude value (for example, 9 would be represented by the binary value 1001). Increasing the number of possible values used to represent sound amplitude also increases the accuracy of the sampled sound (for example, using a range of 0 to 127 gives a much more accurate representation of the sound sample than using a range of, for example, 0 to 10). This is known as the **sampling resolution** (also known as the bit depth).

Sampling rate is the number of sound samples taken per second. The higher the sampling rate and/or sampling resolution, the greater the file size. For example, a 16-bit sampling resolution is used when recording CDs to give better sound quality.

So, how is sampling used to record a sound clip?

- The amplitude of the sound wave is first determined at set time intervals (the sampling rate).
- This gives an approximate representation of the sound wave.
- The sound wave is then encoded as a series of binary digits.

Using a higher sampling rate or larger resolution will result in a more faithful representation of the original sound source.

Pros	Cons
larger dynamic range	produces larger file size
better sound quality	takes longer to transmit/download sound files
less sound distortion	requires greater processing power

Table 1.9 The pros and cons of using a larger sampling resolution when recording sound

Recorded sound is often edited using software. Common features of such software include the ability to

- edit the start/stop times and duration of a sample
- extract and save (or delete) part of a sample
- alter the frequency and amplitude of a sample
- fade in and fade out
- mix and/or merge multiple sound tracks or sources
- combine various sound sources together and alter their properties
- remove ‘noise’ to enhance one sound wave in a multiple of waves (for example, to identify and extract one person’s voice out of a group of people)
- convert between different audio formats.

1.2.4 Video

This section considers the use of video and extends beyond the syllabus. While this is not specifically mentioned in the syllabus, it has been included here for completeness. Many specialist video cameras exist. However, most digital cameras, smart phones and tablets are also capable of taking moving images by ‘stitching’ a number of still photos (frames) together. They are often referred to as DV (digital video) cameras; they store compressed photo frames at a speed of 25 MB per second – this is known as motion JPEG.

In both single frame and video versions, the camera picks up the light from the image and turns it into an electronic signal using light-sensitive sensors. In the case of the DV cameras, these signals are automatically converted into a compressed digital file format.

When recording video, the **frame rate** refers to the number of frames recorded per second.

1.3 File compression

Key terms

Lossless file compression – file compression method where the original file can be restored following decompression.

Lossy file compression – file compression method where parts of the original file cannot be recovered during decompression, so some of the original detail is lost.

JPEG – Joint Photographic Expert Group – a form of lossy file compression based on the inability of the eye to spot certain colour changes and hues.

MP3/MP4 files – file compression method used for music and multimedia files.

Audio compression – method used to reduce the size of a sound file using perceptual music shaping.

Perceptual music shaping – method where sounds outside the normal range of hearing of humans, for example, are eliminated from the music file during compression.

Bit rate – number of bits per second that can be transmitted over a network. It is a measure of the data transfer rate over a digital telecoms network.

Run length encoding (RLE) – a lossless file compression technique used to reduce text and photo files in particular.

It is often necessary to reduce the file size of a file to either save storage space or to reduce the time taken to stream or transmit data from one device to another (see Chapter 2). The two most common forms of file compression are **lossless file compression** and **lossy file compression**.

Lossless file compression

With this technique, all the data from the original file can be reconstructed when the file is uncompressed again. This is particularly important for files where loss of any data would be disastrous (such as a spreadsheet file of important results).

Lossy file compression

With this technique, the file compression algorithm eliminates unnecessary data (as with MP3 and **JPEG** formats, for example).

Lossless file compression is designed to lose none of the original detail from the file (such as Run-Length Encoding (RLE) which is covered later in this chapter). Lossy file compression usually results in some loss of detail when compared to the original; it is usually impossible to reconstruct the original file. The algorithms used in the lossy technique have to decide which parts of the file are important (and need to be kept) and which parts can be discarded.

We will now consider file compression techniques applied to multimedia files.

1.3.1 File compression applications

MPEG-3 (MP3) and MPEG-4 (MP4)

MPEG-3 (MP3) uses technology known as **audio compression** to convert music and other sounds into an MP3 file format. Essentially, this compression technology will reduce the size of a normal music file by about 90%. For example, an 80 MB music file on a CD can be reduced to 8 MB using MP3 technology.

MP3 files are used in MP3 players, computers or mobile phones. Music files can be downloaded or streamed from the internet in a compressed format, or CD files can be converted to MP3 format. While streamed or MP3 music quality can never match the 'full' version found on a CD, the quality is satisfactory for most purposes.

But how can the original music file be reduced by 90% while still retaining most of the music quality? This is done using file compression algorithms that use **perceptual music shaping.**

Perceptual music shaping removes certain sounds. For example

- frequencies that are outside the human hearing range
- if two sounds are played at the same time, only the louder one can be heard by the ear, so the softer sound is eliminated.

This means that certain parts of the music can be removed without affecting the quality too much. MP3 files use what is known as a lossy format, since part of the original file is lost following the compression algorithm. This means that the original file cannot be put back together again. However, even the quality of MP3 files can be different, since it depends on the **bit rate** – this refers to the number of bits per second used when creating the file. Bit rates are between 80 and 320 kilobits per second; usually 200 kilobits or higher gives a sound quality close to a normal CD.

MPEG-4 (MP4) files are slightly different to MP3 files. This format allows the storage of multimedia files rather than just sound. Music, videos, photos and animation can all be stored in the MP4 format. Videos, for example, could be streamed over the internet using the MP4 format without losing any real discernible quality (see Chapter 2 for notes on video streaming).

EXTENSION ACTIVITY 1D

Find out how file compression can be applied to a photograph without noticeably reducing its quality. Compare this to run-length encoding (RLE), described below.

Photographic (bit-map) images

When a photographic file is compressed, both the file size and quality of image are reduced. A common file format for images is JPEG, which uses lossy file compression. Once the image is subjected to the JPEG compression algorithm, a new file is formed and the original file can no longer be constructed. A JPEG will reduce the raw bit-map image by a factor of between 5 and 15, depending on the quality of the original.

Vector graphics can also undergo some form of file compression. Scalable vector graphics (.svg)

are defined in XML text files which, therefore, allows them to be compressed.

Run-length encoding (RLE)

Run-length encoding (RLE) can be used to compress a number of different file formats.

It is a form of lossless/reversible file compression that reduces the size of a string of adjacent, identical data (such as repeated colours in an image).

A repeating string is encoded into two values.

The first value represents the number of identical data items (such as characters) in the run. The second value represents the code of the data item (such as ASCII code if it is a keyboard character).

RLE is only effective where there is a long run of repeated units/bits.

Using RLE on text data

Consider the text string ‘aaaaabbbbccddddd’.

Assuming each character requires 1 byte, then this string needs 16 bytes. If we assume ASCII code is being used, then the string can be coded as follows:

a	a	a	a	a	b	b	b	b	c	c	d	d	d	d	d
05 97					04 98				02 99		05 100				

This means we have five characters with ASCII code 97, four characters with ASCII code 98, two characters with ASCII code 99, and five characters with ASCII code 100. Assuming each number in the second row requires 1 byte of memory, the RLE code will need 8 bytes. This is half the original file size.

One issue occurs with a string such as ‘cdcdcdcdcd’, where compression is not very effective. To cope with this we use a flag. A flag preceding data indicates that what follows are the number of repeating units (for example, 255 05 97 where 255 is the flag and the other two numbers indicate that there are five items with ASCII code 97). When a flag is not used, the next byte(s) are taken with their face value and a run of 1 (for example, 01 99 means one character with ASCII code 99 follows).

Consider this example:

String	aaaaaaaa	bbbbbbbbbb	c	d	c	d	c	d	eeeeeeee
Code	08 97	10 98	01 99	01 100	01 99	01 100	01 99	01 100	08 101

The original string contains 32 characters and would occupy 32 bytes of storage.

The coded version contains 18 values and would require 18 bytes of storage.

Introducing a flag (255 in this case) produces:

255 08 97 255 10 98 99 100 99 100 99 100 255 08 101

This has 15 values and would, therefore, require 15 bytes of storage. This is a reduction in file

size of about 53%.

Using RLE with images

Black and white images

Figure 1.7 shows the letter F in a grid where each square requires 1 byte of storage. A white square has a value 1 and a black square a value of 0.

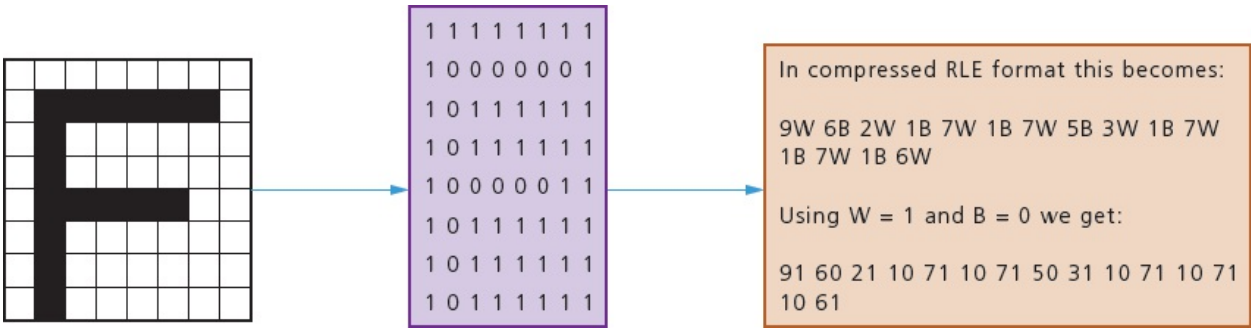
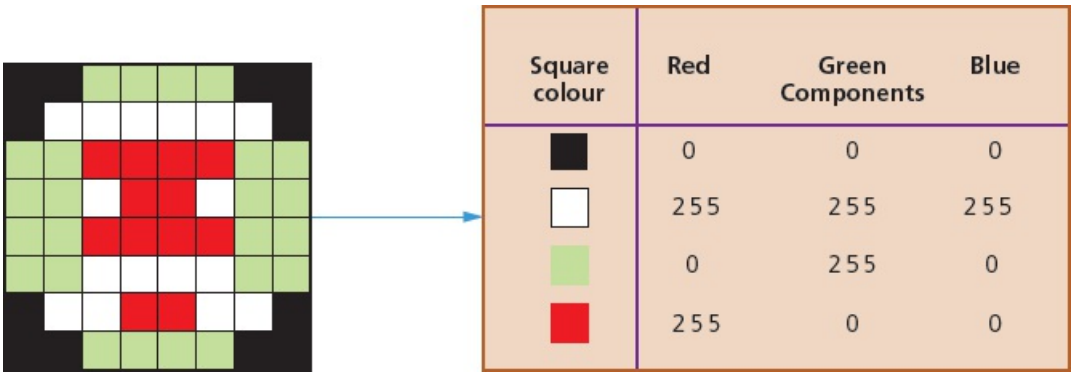


Figure 1.7 Using RLE with a black and white image

The 8 × 8 grid would need 64 bytes; the compressed RLE format has 30 values, and therefore needs only 30 bytes to store the image.

Coloured images

Figure 1.8 shows an object in four colours. Each colour is made up of red, green and blue (RGB) according to the code on the right.



Square colour	Red	Green Components	Blue
	0	0	0
	255	255	255
	0	255	0
	255	0	0

Figure 1.8 Using RLE with a coloured image

This produces the following data:

2 0 0 0 4 0 255 0 3 0 0 0 6 255 255 255 1 0 0 0 2 0 255 0 4 255 0 0 4 0 255 0 1 255 255 255 2 255 0 0 1 255 255 255 4 0 255 0 4 255 0 0 4 0 255 0 4 255 255 255 2 0 255 0 1 0 0 0 2 255 255 255 2 255 0 0 2 255 255 255 3 0 0 0 4 0 255 0 2 0 0 0

The original image (8 × 8 square) would need 3 bytes per square (to include all three RGB values). Therefore, the uncompressed file for this image is 8 × 8 × 3 = 192 bytes.

The RLE code has 92 values, which means the compressed file will be 92 bytes in size. This gives a file reduction of about 52%. It should be noted that the file reductions in reality will not be as large as this due to other data which needs to be stored with the compressed file (such as a

file header).

1.3.2 General methods of compressing files

All the above file compression techniques are excellent for very specific types of file. However, it is also worth considering some general methods to reduce the size of a file without the need to use lossy or lossless file compression:

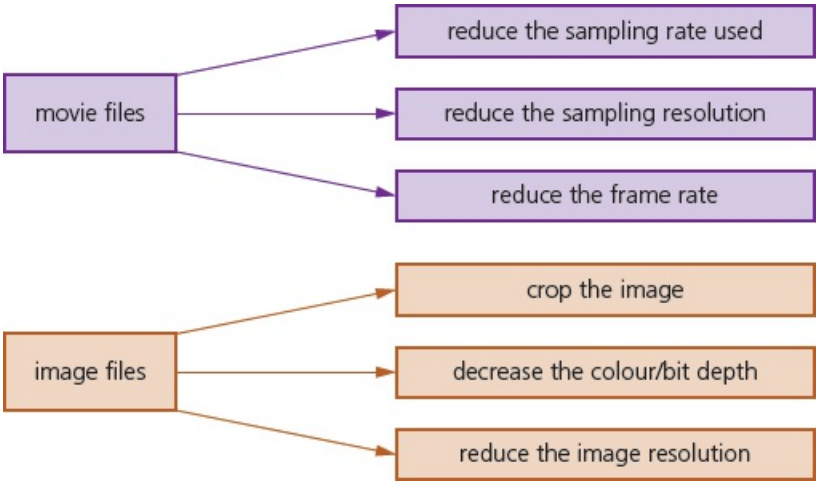


Figure 1.9 General methods of compressing files

ACTIVITY 1I

1 **a)** What is meant by *lossless* and *lossy* file compression?

b) Give an example of a lossless file format and an example of a lossy file format.

2 **a)** Describe how music picked up by a microphone is turned into a digitised music file in a computer.

b) Explain why it is often necessary to compress stored music files. Describe how the music quality is essentially retained.

3 **a)** What is meant by *run length encoding*?

b) Describe how RLE compresses a file. Give an example in your description.

4 **a)** Describe the differences between bit-map images and vector graphics.

b) A software designer needs to incorporate images into her software to add realism. Explain what she needs to consider when deciding between using bit-map images and vector graphics in her software.

End of chapter questions

1 **a)** The following bytes represent binary integers using the two's complement form. State the equivalent denary values.

i) 0 1 0 0 1 1 1 1

[1]

ii) 1 0 0 1 1 0 1 0

[1]

iii) Write the integer −53 in two's complement form.

[1]

iv) Write the maximum possible range of numbers using the two's complement form of an 8-bit binary number.
Give your answers in denary.

[2]

b) i) Write the denary integer 798 in binary-coded decimal (BCD) format.

[1]

ii) Write the denary number that is represented by the following BCD number.

1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	0

[2]

c) Give **one** use of binary-coded decimal system.

[1]

2 A software developer is using a microphone and a sound editing app to collect and edit sounds for his new game.
When collecting sounds, the software developer can decide on the sampling resolution he wishes to use.

a) i) State what is meant by *sampling resolution*.

[1]

ii) Describe how sampling resolution will affect how accurate the stored digitised sound will be.

[2]

b) The software developer will include images in his new game.

i) Explain the term *image resolution*.

[1]

ii) The software developer is using 16-colour bit-map images.
State the number of bits required to encode data for one pixel of his image.

[1]

iii) One of the images is 16 384 pixels wide and 512 pixels high.
The developer decides to save it as a 256-colour bit-map image.
Calculate the size of the image file in gibibytes.

[3]

iv) The bit-map image will contain a *header*.
State **two** items you would expect to see in the header.

[2]

v) Give **three** features you would expect to see in the sound editing app.

[3]

3 The editor of a movie is finalising the music score. They will send the final version of the score to the movie producer by email attachment.

a) Describe how *sampling* is used to record the music sound clips.

[3]

b) The music sound clips need to undergo some form of data compression before the music editor can send them via email.
Identify the type of compression, lossy or lossless, they should use.
Give a justification for your answer.

[3]

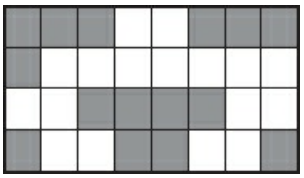
c) One method of data compression is known as *run length encoding (RLE).*

i) Explain what is meant by RLE.

[3]

ii) Show how RLE would be used to produce a compressed file for the image below.
Write down the data you would expect to see in the RLE compressed format (you may assume that the grey squares have a code value of 85 and the white squares have a code value of 255).

[4]



4 a) Write the denary numbers 60, 27 and −27 in 8-bit binary two's complement form.

[3]

b) Show the result of the addition 60 + 27 using 8-bit binary two's complement form. Show all of your working.

[2]

c) Show the result of the subtraction 60 − 27 using 8-bit binary two's complement form.

[2]

d) Give the result of the following addition.

0 1 0 1 1 0 0 1

+

0 1 1 0 0 0 0 1

Explain why the expected result is not obtained.

[2]

5 a) Carry out 0.52 + 0.83 using binary-coded decimal (BCD). Show all of your working.

[4]

b) i) Define the term *hexadecimal.*

[1]

ii) Give **two** uses of the hexadecimal system.

[2]

iii) Convert the following binary number into hexadecimal.

0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0

[2]

6 a) Convert the denary number 95 into binary coded decimal (BCD).

[1]

b) Using two's complement, carry out the binary subtraction:
0 0 1 0 0 0 1 1 – 0 1 0 0 0 1 0 0
and convert your answer into denary.

[3]

c) Convert the denary number 506 into hexadecimal.

[1]

2 Communication

In this chapter, you will learn about

- the benefits of networking devices
- the characteristics of a local area network (LAN) and a wide area network (WAN)
- client-server and peer-to-peer models in networking
- the differences between thin client and thick client
- bus, star, mesh and hybrid networking topologies
- public and private cloud computing
- the differences between wired and wireless networks (including types of cable and wireless technologies)
- the hardware required to support a LAN
- the function of routers
- Ethernet and how data collisions are detected and avoided
- bit streaming (including differences between real-time and on-demand streaming of data)
- the differences between the internet and the World Wide Web (WWW)
- the hardware needed to support the internet
- IP addresses (including IPv4, IPv6, public IP addresses and private IP addresses)
- the use of the uniform resource locator (URL) to locate a resource on the world wide web
- the role of the domain name service (DNS).

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you read this chapter.

1 a) Explain the following terms associated with devices connected to a network/internet.

i) MAC address

ii) IP address

b) Explain the main differences between a MAC address and an IP address and why it is necessary to have both associated with a device connected to the internet.

c) What is the purpose of an internet service provider (ISP)?

d) Explain the function of an internet browser. In what ways is this different to an ISP?

2 A college is about to form a network from 20 stand-alone computers. Describe the hardware and software that might be needed to produce this simple computer network.

3 a) Mobile phones and tablets can be configured to access the internet from any location. Describe the software required to allow this to happen.

b) Describe some of the benefits and drawbacks (when compared to a desktop PC) of accessing website pages from a mobile phone.

2.1 Networking

Key terms

ARPAnet – Advanced Research Projects Agency Network.

WAN – wide area network (network covering a very large geographical area).

LAN – local area network (network covering a small area such as a single building).

MAN – metropolitan area network (network which is larger than a LAN but smaller than a WAN, which can cover several buildings in a single city, such as a university campus).

File server – a server on a network where central files and other data are stored. They can be accessed by a user logged onto the network.

Hub – hardware used to connect together a number of devices to form a LAN that directs incoming data packets to all devices on the network (LAN).

Switch – hardware used to connect together a number of devices to form a LAN that directs incoming data packets to a specific destination address only.

Router – device which enables data packets to be routed between different networks (for example, can join LANs to form a WAN).

Modem – modulator demodulator. A device that converts digital data to analogue data (to be sent down a telephone wire); conversely it also converts analogue data to digital data (which a computer can process).

WLAN – wireless LAN.

(W)AP – (wireless) access point which allows a device to access a LAN without a wired connection.

PAN – network that is centred around a person or their workspace.

Client-server – network that uses separate dedicated servers and specific client workstations. All client computers are connected to the dedicated servers.

Spread spectrum technology – wideband radio frequency with a range of 30 to 50 metres.

Node – device connected to a network (it can be a computer, storage device or peripheral device).

Peer-to-peer – network in which each node can share its files with all the other nodes. Each node has its own data and there is no central server.

Thin client – device that needs access to the internet for it to work and depends on a more powerful computer for processing.

Thick client – device which can work both off line and on line and is able to do some processing even if not connected to a network/internet.

Bus network topology – network using single central cable in which all devices are connected

to this cable so data can only travel in one direction and only one device is allowed to transmit at a time.

Packet – message/data sent over a network from node to node (packets include the address of the node sending the packet, the address of the packet recipient and the actual data – this is covered in greater depth in Chapter 14).

Star network topology – a network that uses a central hub/switch with all devices connected to this central hub/switch so all data packets are directed through this central hub/switch.

Mesh network topology – interlinked computers/devices, which use routing logic so data packets are sent from sending stations to receiving stations only by the shortest route.

Hybrid network – network made up of a combination of other network topologies.

Cloud storage – method of data storage where data is stored on off-site servers.

Data redundancy – situation in which the same data is stored on several servers in case of maintenance or repair.

Wi-Fi – wireless connectivity that uses radio waves, microwaves. Implements IEEE 802.11 protocols.

Bluetooth – wireless connectivity that uses radio waves in the 2.45 GHz frequency band.

Spread spectrum frequency hopping – a method of transmitting radio signals in which a device picks one of 79 channels at random. If the chosen channel is already in use, it randomly chooses another channel. It has a range up to 100 metres.

WPAN – wireless personal area network. A local wireless network which connects together devices in very close proximity (such as in a user's house); typical devices would be a laptop, smartphone, tablet and printer.

Twisted pair cable – type of cable in which two wires of a single circuit are twisted together. Several twisted pairs make up a single cable.

Coaxial cable – cable made up of central copper core, insulation, copper mesh and outer insulation.

Fibre optic cable – cable made up of glass fibre wires which use pulses of light (rather than electricity) to transmit data.

Gateway – device that connects LANs which use different protocols.

Repeater – device used to boost a signal on both wired and wireless networks.

Repeating hubs – network devices which are a hybrid of hub and repeater unit.

Bridge – device that connects LANs which use the same protocols.

Softmodem – abbreviation for software modem; a software-based modem that uses minimal hardware.

NIC – network interface card. These cards allow devices to connect to a network/internet (usually associated with a MAC address set at the factory).

WNIC – wireless network interface cards/controllers.

Ethernet – protocol IEEE 802.3 used by many wired LANs.

Conflict – situation in which two devices have the same IP address.

Broadcast – communication where pieces of data are sent from sender to receiver.

Collision – situation in which two messages/data from different sources are trying to transmit along the same data channel.

CSMA/CD – carrier sense multiple access with collision detection – a method used to detect collisions and resolve the issue.

Bit streaming – contiguous sequence of digital bits sent over a network/internet.

Buffering – store which holds data temporarily.

Bit rate – number of bits per second that can be transmitted over a network. It is a measure of the data transfer rate over a digital telecoms network.

On demand (bit streaming) – system that allows users to stream video or music files from a central server as and when required without having to save the files on their own computer/tablet/phone.

Real-time (bit streaming) – system in which an event is captured by camera (and microphone) connected to a computer and sent to a server where the data is encoded. The user can access the data ‘as it happens’ live.

2.1.1 Networking devices

One of the earliest forms of networking, circa 1970 in the USA, was the **Advanced Research Projects Agency Network (ARPAnet)**. This was an early form of packet switching **wide area network (WAN)** connecting a number of large computers in the Department of Defense. It later expanded to include university computers. It is generally agreed that ARPAnet developed the technical platform for what we now call the internet. Figure 2.1 shows the vast area this network covered.

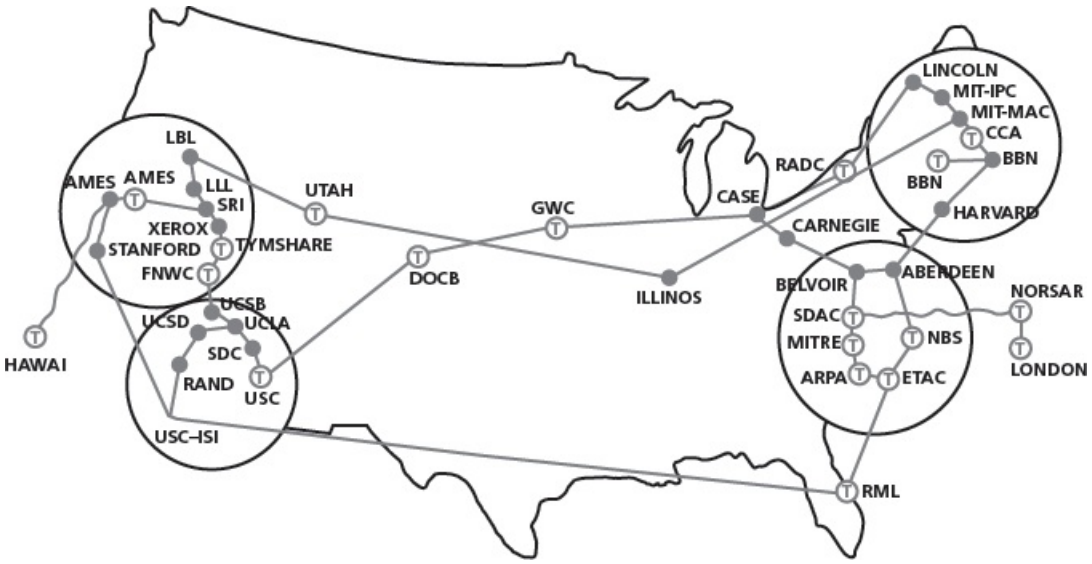


Figure 2.1 ARPAnet coverage, 1973

As personal computers developed through the 1980s, a local network began to appear. This became known as a **LAN local area network (LAN)**. LANs tended to be much smaller networks (usually inside one building) connecting a number of computers and shared devices, such as printers. WANs typically consist of a number of LANs connected via public communications networks (such as telephone lines or satellites). Because a WAN consists of LANs joined together, it may be a private network, and passwords and user IDs are required to access it. This is in contrast to the internet which is a vast number of decentralised networks and computers which have a common point of access, so that anyone with access to the internet can connect to the computers on these networks. This makes it intrinsically different to a WAN.

In recent years, another type of network – a **metropolitan area network (MAN)** – has emerged. MANs are larger than LANs as they can connect together many small computer networks (e.g LANs) housed in different buildings within a city (for example, a university campus). MANs are restricted in their size geographically to, for example, a single city.

In contrast, WANs can cover a much larger geographical area, such as a country or a continent. For example, a multi-national company may connect a number of smaller networks together (e.g. LANs or MANs) to form a world-wide WAN. This is covered in more detail later.

Here are some of the main benefits of networking computers and devices (rather than using a number of stand-alone computers):

- Devices, such as printers, can be shared (thus reducing costs).
- Licences to run software on networks are often far cheaper than buying licences for an equivalent number of stand-alone computers.
- Users can share files and data.
- Access to reliable data that comes from a central source, such as a **file server**.
- Data and files can be backed up centrally at the end of each day.
- Users can communicate using email and instant messaging.
- A network manager can oversee the network and, for example, apply access rights to certain files, or restrict access to external networks, such as the internet.

There are also a number of drawbacks:

- Cabling and servers can be an expensive initial outlay.
- Managing a large network can be a complex and difficult task.
- A breakdown of devices, such as the file servers, can affect the whole network.
- Malware and hacking can affect entire networks (particularly if a LAN is part of a much larger WAN), although firewalls do afford some protection in this respect.

Networked computers

Networked computers form an infrastructure which enables internal and external communications to take place. The infrastructure includes the following:

Hardware

- LAN cards
- routers
- switches
- wireless routers
- cabling

Software

- operation and management of the network
- operation of firewalls
- security applications/utilities

Services

- DSL
- satellite communication channels
- wireless protocols
- IP addressing.

Networks can be categorised as private or public.

Private networks are owned by a single company or organisation (they are often LANs or intranets with restricted user access, for example, passwords and user ids are required to join the network); the companies are responsible for the purchase of their own equipment and software, maintenance of the network and the hiring and training of staff.

Public networks are owned by a communications carrier company (such as a telecoms company); many organisations will use the network and there are usually no specific password requirements to enter the network – but sub-networks may be under security management.

WANs and LANs

Local area networks (LANs)

LANs are usually contained within one building, or within a small geographical area. A typical LAN consists of a number of computers and devices (such as printers) connected to **hubs** or **switches**. One of the hubs or switches is usually connected to a **router** and/or **modem** to allow the LAN to connect to the internet or become part of a wide area network (WAN).

Wireless LANs (WLANs)

Wireless LANs (WLANs) are similar to LANs but there are no wires or cables. In other words, they provide wireless network communications over fairly short distances (up to 100 metres) using radio or infrared signals instead of using cables.

Devices, known as **wireless access points (WAPs)**, are connected into the wired network at fixed locations. Because of the limited range, most commercial LANs (such as those on a college campus or at an airport) need several WAPs to permit uninterrupted wireless communications. The WAPs use either **spread spectrum** technology (which is a wideband radio frequency with a range from a few metres to 100 metres) or infrared (which has a very short range of about 1 to 2 metres and is easily blocked, and therefore has limited use; see Section 2.1.5 Wired and wireless networking).

The WAP receives and transmits data between the WLAN and the wired network structure. End users access the WLAN through wireless LAN adapters which are built into the devices or as a plug in module.

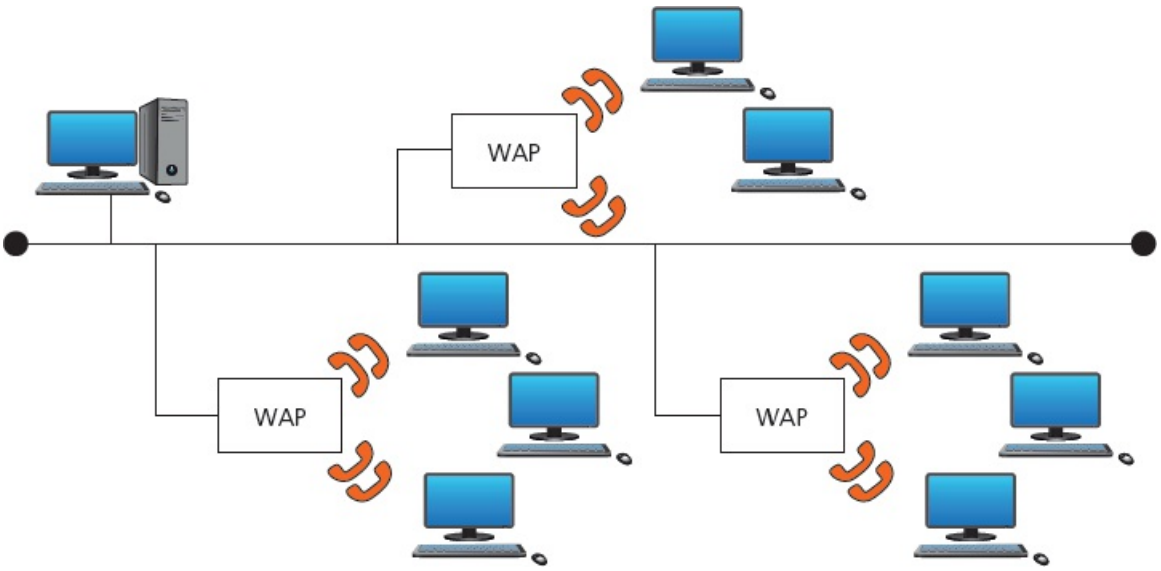


Figure 2.2 Wireless local area networks (WLAN)

Wide area networks (WANs)

Wide area networks (WANs) are used when computers or networks are situated a long distance from each other (for example, they may be in different cities or on different continents). If a number of LANs are joined together using a router or modem, they can form a WAN. The network of automated teller machines (ATMs) used by banks is one of the most common examples of the use of a WAN.

Because of the long distances between devices, WANs usually make use of a public communications network (such as telephone lines or satellites), but they can use dedicated or leased communication lines which can be less expensive and more secure (less risk of hacking, for example).

A typical WAN will consist of end systems and intermediate systems, as shown in Figure 2.3. 1, 3, 7 and 10 are known as end systems, and the remainder are known as intermediate systems. The distance between each system can be considerable, especially if the WAN is run by a multi-national company.

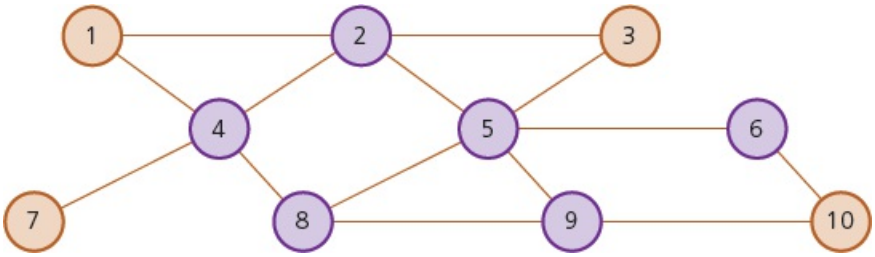


Figure 2.3 A typical WAN

The following is used as a guide for deciding the 'size' of a network:

WAN: 100 km to over 1000 km

MAN: 1 km to 100 km

LAN: 10 m to 1000 m

PAN: 1 m to 10 m (this is not a commonly used term – it means **personal area network**; in other words, a home system)

2.1.2 Client-server and peer-to-peer networking models

We will consider two types of networking models, client-server and peer-to-peer.

Client-server model

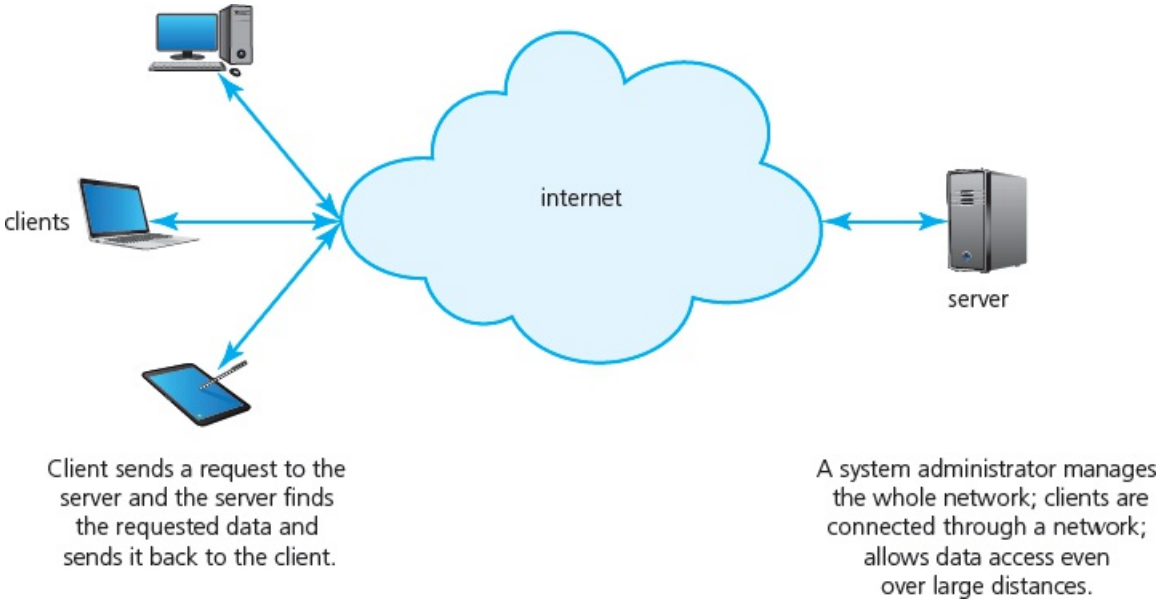


Figure 2.4 Client-server model

- The **client-server** model uses separate dedicated servers and specific client workstations; client computers will be connected to the server computer(s).
- Users are able to access most of the files, which are stored on dedicated servers.
- The server dictates which users are able to access which files. (Note: sharing of data is the most important part of the client-server model; with peer-to-peer, connectivity is the most important aspect.)
- The client-server model allows the installation of software onto a client's computer.
- The model uses central security databases which control access to the shared resources. (Note: passwords and user IDs are required to log into the network.)
- Once a user is logged into the system, they will have access to only those resources (such as a printer) and files assigned to them by the network administrator, so offers greater security than peer-to-peer networks.
- Client-server networks can be as large as you want them to be and they are much easier to scale up than peer-to-peer networks.
- A central server looks after the storing, delivery and sending of emails.
- This model offers the most stable system, for example, if someone deletes a shared resource from the server, the nightly back-up would restore the deleted resource (this is different in peer-to-peer – see later).
- Client-server networks can become bottlenecked if there are several client requests at the same time.

- In the client-server model, a file server is used and is responsible for
  - central storage and management of data files, thus enabling other network users to access files
  - allowing users to share information without the need for offline devices (such as a memory stick)
  - allowing any computer to be configured as the host machine and act as the file server (note that the server could be a storage device (such as SSD or HDD) that could also serve as a remote storage device for other computers, thus allowing them to access this device as if it were a local storage device attached to their computer).

Examples of use of client-server network model

A company/user would choose a client-server network model for the following reasons.

- The company/user has a large user-base (however, it should be pointed out that this type of network model may still be used by a small group of people who are doing independent projects but need to have sharing of data and access to data outside the group).
- Access to network resources needs to be properly controlled.
- There is a need for good network security.
- The company requires its data to be free from accidental loss (in other words, data needs to be backed up at a central location).

An example is the company Amazon; it uses the client-server network model. The user front-end is updated every time a user logs on to the Amazon website and a large server architecture handles items such as order processing, billing customers and data security; none of the Amazon users are aware that other customers are using the website at the same time – there is no interaction between users and server since they are kept entirely separate at all times.

Peer-to-peer model

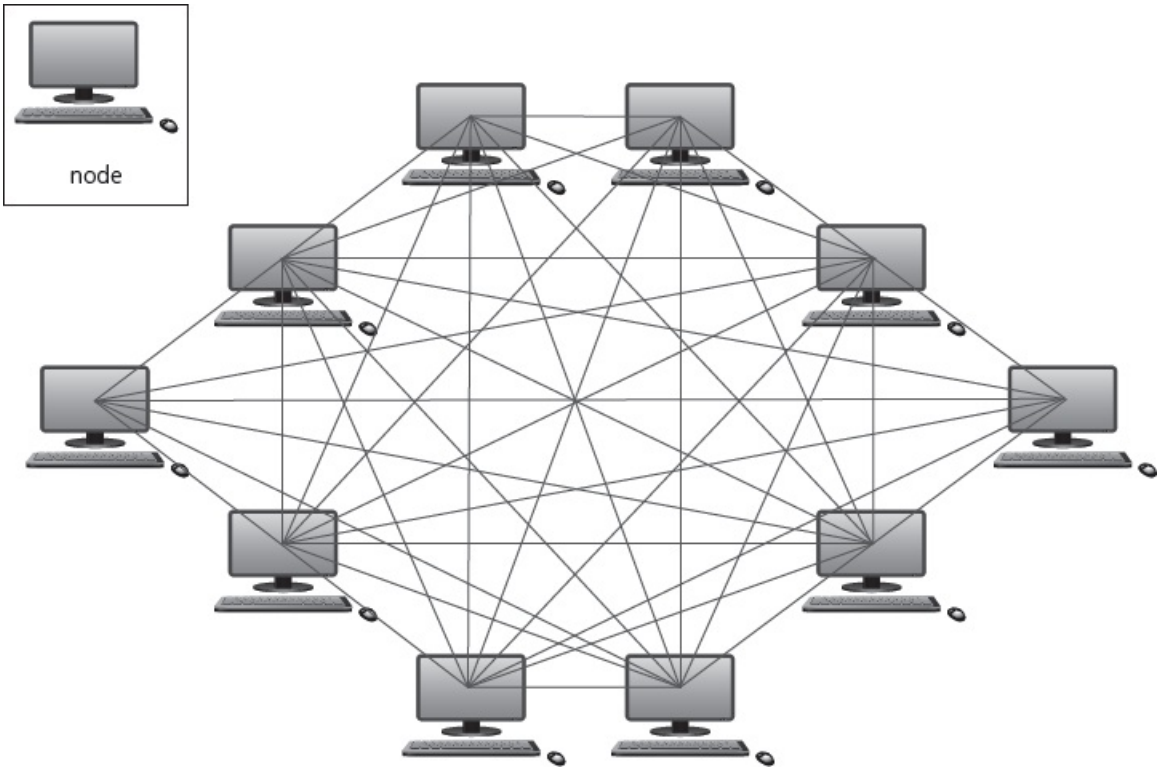


Figure 2.5 Peer-to-peer model

On a peer-to-peer network, each **node** joins the network to allow

- the provision of services to all other network users; the services available are listed on a nominated ‘look up’ computer – when a node requests a service, the ‘look up’ computer is contacted to find out which of the other network nodes can provide the required service
- other users on the network to simply access data from another node
- communication with other peers connected to the network
- peers to be both suppliers and consumers (unlike the client-server model where consumers and resources are kept entirely separate from each other)
- peers to participate as equals on the network (again this is different to the client-server model where a webserver and client have different responsibilities).

The **peer-to-peer** model does not have a central server. Each of the nodes (workstations) on the network can share its files with all the other nodes, and each of the nodes will have its own data.

Because there is no central storage, there is no requirement to authenticate users.

This model is used in scenarios where no more than 10 nodes are required (such as a small business) where it is relatively easy for users to be in contact with each other on a regular basis. More than 10 nodes leads to performance and management issues.

Peer-to-peer offers little data security since there is no central security system. This means it is impossible to know who is authorised to share certain data. Users can create their own network node share point which is the only real security aspect since this gives them some kind of control. However, there are no real authentication procedures.

Examples of peer-to-peer network model

A user would choose the peer-to-peer network model for one or more of following reasons:

- The network of users is fairly small.
- There is no need for robust security.
- They require workstation-based applications rather than being server-based.

An example would be a small business where there is frequent user interaction and there is no need to have the features of a client-server network (for example, a builder with five associated workers located in their own homes who only need access to each other's diaries, previous jobs, skills-base and so on – when the builder is commissioned to do a job they need to access each other's computer to check on who is available and who has the appropriate skills).

Thin clients and thick clients

The client-server model offers **thin clients** and **thick clients**. These can often refer to both hardware and software.

Thin client

A thin client is heavily dependent on having access to a server to allow constant access to files and to allow applications to run uninterrupted. A thin client can either be a device or software which needs to be connected to a powerful computer or server to allow processing to take place (the computer or server could be on the internet or could be part of a LAN/MAN/WAN network). The thin client will not work unless it is connected at all times to the computer or server. A software example would be a web browser which has very limited functions unless it is connected to a server. Other examples include mobile phone apps which need constant access to a server to work. A hardware example is a POS terminal at a supermarket that needs constant access to a server to find prices, charge customers and to do any significant processing.

Thick client

A thick client can either be a device or software that can work offline or online; it is still able to do some processing whether it is connected to a server or not. A thick client can either be connected to a LAN/MAN/WAN, virtual network, the internet or a cloud computing server. A hardware example is a normal PC/laptop/tablet since it would have its own storage (HDD or SSD), RAM and operating system which means it is capable of operating effectively online or offline. An example of software is a computer game which can run independently on a user's computer, but can also connect to an online server to allow gamers to play and communicate with each other.

Table 2.1 highlights some of the pros and cons of using thick client or thin client hardware.

	Pros	**Cons**
Thick clients	• more robust (device can carry out processing even when not connected to server) • clients have more control (they can store their own programs	• less secure (relies on clients to keep their own data secure) • each client needs to update data and software individually

	and data/files)	• data integrity issues, since many clients access the same data which can lead to inconsistencies
Thin clients	• less expensive to expand (low-powered and cheap devices can be used) • all devices are linked to a server (data updates and new software installation done centrally) • server can offer protection against hacking and malware	• high reliance on the server; if the server goes down or there is a break in the communication link then the devices cannot work • despite cheaper hardware, the start-up costs are generally higher than for thick clients

Table 2.1 Summary of pros and cons of thick and thin client hardware

Table 2.2 highlights the differences between thick and thin client software.

Thin client software	Thick client software
• always relies on a connection to a remote server or computer for it to work	• can run some of the features of the software even when not connected to a server
• requires very few local resources (such as SSD, RAM memory or computer processing time)	• relies heavily on local resources
• relies on a good, stable and fast network connection for it to work	• more tolerant of a slow network connection
• data is stored on a remote server or computer	• can store data on local resources such as HDD or SSD

Table 2.2 Differences between thin and thick client software

ACTIVITY 2A

1 A company has 20 employees working on the development of a new type of battery for use in mobile phones. Decide which type of network model (client-server or peer-to-peer) would be most suitable. Give reasons for your choice.

2 Another company is made up of a group of financial consultants who advise other companies on financial matters, such as taxation and exporting overseas. Decide which type of network model (client-server or peer-to-peer) would be most suitable. Give reasons for your choice.

2.1.3 Network topologies

There are many ways to connect computers to make complex networks. Here we will consider

- bus networks
- star networks
- mesh networks
- hybrid networks.

Bus networks

A **bus network topology** uses a single central cable to which all computers and devices are connected. It is easy to expand and requires little cabling. Data can only travel in one direction; if data is being sent between devices then other devices cannot transmit. Terminators are needed at each end to prevent signal reflection (bounce). Bus networks are typically peer-to-peer. The disadvantages of a bus network include:

- If the main cable fails, the whole network goes down.
- The performance of the network deteriorates under heavy loading.
- The network is not secure since each **packet** passes through every node.

The advantages of a bus network include:

- Even if one node fails, the remainder of the network continues to function.
- It is easy to increase the size of the network by adding additional nodes.

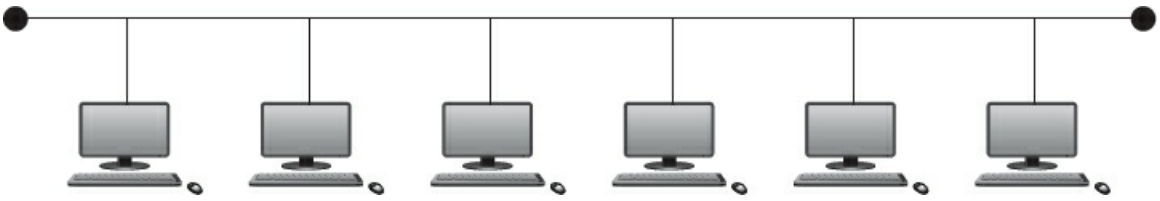


Figure 2.6 Bus network topology

In bus network topology, each node looks at each packet and determines whether or not the address of the recipient in the package matches the node address. If so, the node accepts the packet; if not, the packet is ignored.

These are most suitable for situations with a small number of devices with light traffic occurring. For example, a small company or an office environment.

Star networks

A **star network topology** uses a central hub/switch and each computer/device is connected to the hub/switch. Data going from host to host is directed through the central hub/switch. Each computer/device has its own dedicated connection to the central node (hub/switch) – any type of network cable can be used for the connections (see Section 2.1.5 Wired and wireless networking). This type of network is typically a client-server. The disadvantages of a star network include:

- The initial installation costs are high.

- If the central hub/switch fails, then the whole network goes down.

The advantages of a star network include:

- Data collisions are greatly reduced due to the topology.
- It is a more secure network since security methods can be applied to the central node and packets only travel to nodes with the correct address.
- It is easy to improve by simply installing an upgraded hub.
- If one of the connections is broken it only affects one of the nodes.

How packets are handled depends on whether the central node is a switch or a hub. If it is a hub, all the packets will be sent to every device/node on the star network – if the address in the packet matches that of the node, it will be accepted; otherwise, it is ignored (this is similar to the way packets are handled on a bus network). If the central node is a switch, packets will only be sent to nodes where the address matches the recipient address in the packet. The latter is clearly more secure, since only nodes intended to see the packet will receive it.

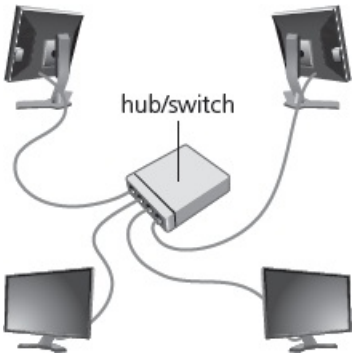


Figure 2.7 Star network topology

Star networks are useful for evolving networks where devices are frequently added or removed. They are well suited to applications where there is heavy data traffic.

Mesh networks

There are two types of **mesh network topologies**: **routing** and **flooding**. Routing works by giving the nodes routing logic (in other words, they act like a router) so that data is directed to its destination by the shortest route and can be re-routed if one of the nodes in the route has failed. Flooding simply sends the data via all the nodes and uses no routing logic, which can lead to unnecessary loading on the network. It is a type of peer-to-peer network, but is fundamentally different. The disadvantages of a mesh network include:

- A large amount of cabling is needed, which is expensive and time consuming.
- Set-up and maintenance is difficult and complex.

The advantages of a mesh network include:

- It is easy to identify where faults on the network have occurred.
- Any broken links in the network do not affect the other nodes.
- Good privacy and security, since packets travel along dedicated routes.
- The network is relatively easy to expand.

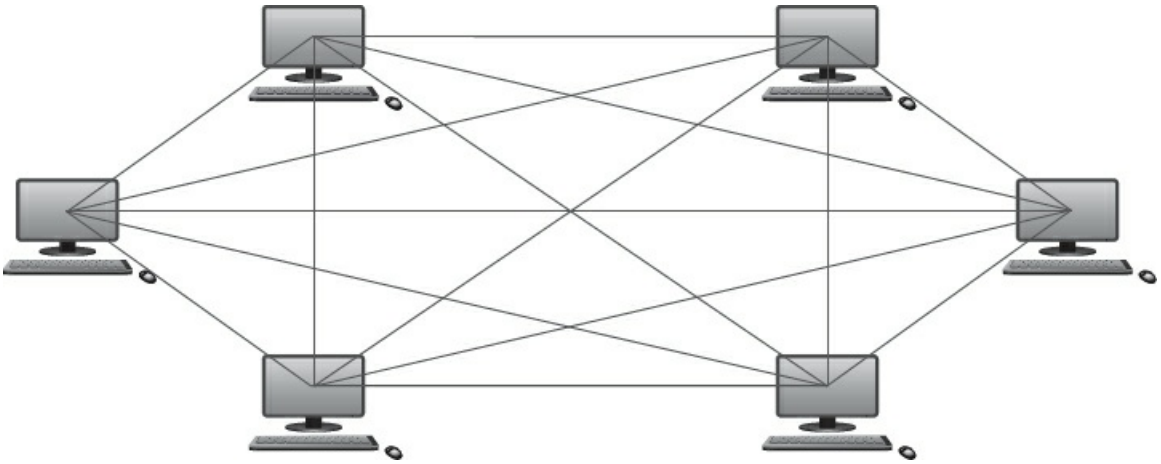


Figure 2.8 Mesh network topology

There are a number of applications worth considering here:

- The internet and WANs/MANs are typical uses of mesh networks.
- Many examples include industrial monitoring and control where sensors are set up in mesh design and feedback to a control system which is part of the mesh, for example
  - medical monitoring of patients in a hospital
  - electronics interconnectivity (for example, systems that link large screen televisions, DVDs, set top boxes, and so on); each device will be in a location forming the mesh
  - modern vehicles use wireless mesh network technology to enable the monitoring and control of many of the components in the vehicle.

EXTENSION ACTIVITY 2A

There appear to be similarities between the peer-to-peer network model and mesh network model.

Describe the differences between the two models.

Hybrid networks

A **hybrid network** is a mixture of two or more different topologies (bus and star, bus and mesh, and so on). The main advantages and disadvantages depend on which types of network are used to make up the hybrid network, but an additional disadvantage is that they can be very complex to install, configure and maintain.

Additional advantages include:

- They can handle large volumes of traffic.
- It is easy to identify where a network fault has occurred.
- They are very well suited to the creation of larger networks.

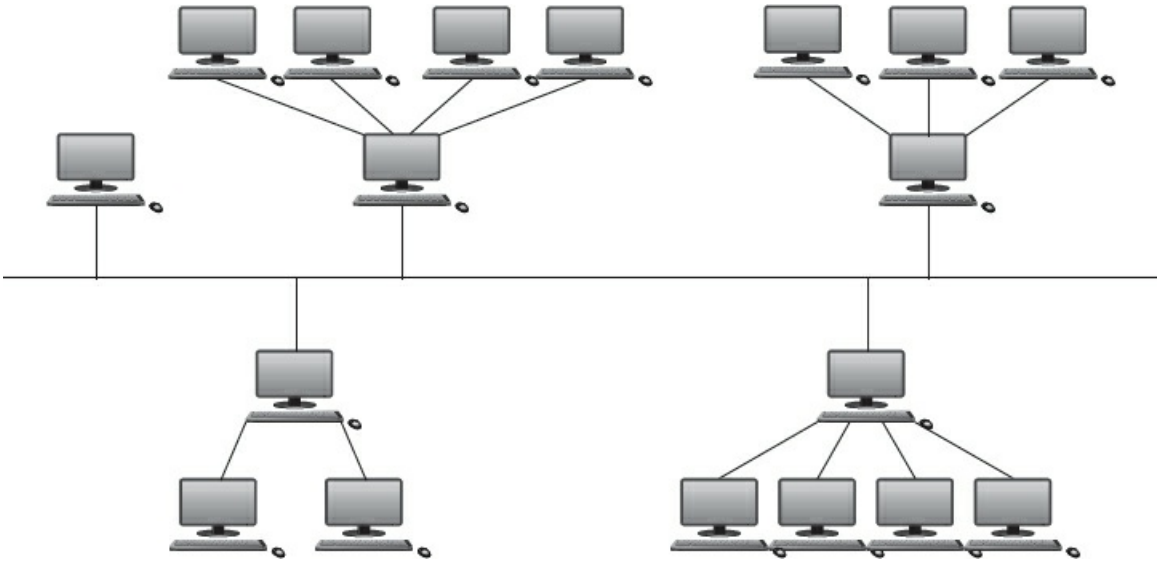


Figure 2.9 Hybrid bus and star network

Note that the handling of packets in hybrid networks will depend on which of the above topologies are used to make up the hybrid structure.

One of the typical applications of hybrid networks is illustrated by the following example, involving three hotel chains, A, B and C.

Suppose hotel chain A uses a bus network, hotel chain B uses a star network and hotel chain C uses a mesh network.

At some point, all three hotel chains are taken over by another company. By using hybrid network technology, all three hotel chains can be connected together even though they are each using a different type of network. The system can also be expanded easily without affecting any of the existing hotels using the network.

There are many other examples; you might want to explore the various applications for each type of network topology.

2.1.4 Public and private cloud computing

Cloud storage is a method of data storage where data is stored on offsite servers – the physical storage covers hundreds of servers in many locations. The same data is stored on more than one server in case of maintenance or repair, allowing clients to access data at any time. This is known as **data redundancy**. The physical environment is owned and managed by a hosting company.

There are three common systems, public cloud, private cloud and hybrid cloud.

Public cloud is a storage environment where the customer/client and cloud storage provider are different companies.

Private cloud is storage provided by a dedicated environment behind a company firewall. Customer/client and cloud storage provider are integrated and operate as a single entity.

Hybrid cloud is a combination of private and public clouds. Some data resides in the private cloud and less sensitive/less commercial data can be accessed from a public cloud storage provider.

Instead of saving data on a local hard disk or other storage device, a user can save their data 'in the cloud'. The pros and cons of using cloud storage are shown in Table 2.3.

Pros of using cloud storage	Cons of using cloud storage
• customer/client files stored on the cloud can be accessed at any time from any device anywhere in the world provided internet access is available • no need for a customer/client to carry an external storage device with them, or use the same computer to store and retrieve information • provides the user with remote back-up of data to aid data loss and disaster recovery • recovers data if a customer/client has a hard disk or back-up device failure • offers almost unlimited storage capacity	• if the customer/client has a slow or unstable internet connection, they would have problems accessing or downloading their data/files • costs can be high if large storage capacity is required • expensive to pay for high download/upload data transfer limits with the customer/client internet service provider (ISP) • potential failure of the cloud storage company is possible – this poses a risk of loss of all back-up data

Table 2.3 Summary of pros and cons of using cloud storage

Data security when using cloud storage

Companies that transfer vast amounts of confidential data from their own systems to a cloud service provider are effectively relinquishing control of their own data security. This raises a number of questions:

- What physical security exists regarding the building where the data is housed?
- How good is the cloud service provider's resistance to natural disasters or power cuts?
- What safeguards exist regarding personnel who work for the cloud service company? Can they

use their authorisation codes to access confidential data for monetary purposes?

Potential data loss when using cloud storage

There is a risk that important and irreplaceable data could be lost from the cloud storage facilities. Actions from hackers (gaining access to accounts or pharming attacks, for example) could lead to loss or corruption of data. Users need to be certain sufficient safeguards exist to overcome these risks.

The following breaches of security involving some of the largest cloud service providers suggest why some people are nervous of using cloud storage for important files:

- The XEN security threat, which forced several cloud operators to reboot all their cloud servers, was caused by a problem in the XEN hypervisor (a hypervisor is a piece of computer software, firmware or hardware that creates and runs virtual machines).
- A large cloud service provider permanently lost data during a routine back-up procedure.
- The celebrity photos cloud hacking scandal, in which more than 100 private photos of celebrities were leaked. Hackers had gained access to a number of cloud accounts, which then enabled them to publish the photos on social networks and sell them to publishing companies.
- In 2016, the National Electoral Institute of Mexico suffered a cloud security breach in which 93 million voter registrations, stored on a central database, were compromised and became publicly available to everyone. To make matters worse, much of the information on this database was also linked to an Amazon cloud server outside Mexico.

Cloud software

Cloud storage is, of course, only one aspect of cloud computing. Other areas covered by cloud computing include databases, networking, software and analytical services using the internet.

Here we will consider cloud software – you can research for yourself how databases and analytical services are provided by cloud computing services.

Software applications can be delivered to a user's computer on demand using cloud computing services. The cloud provider will both host and manage software applications – this will include maintenance, software upgrades and security for a monthly fee. A user will simply connect to the internet (using their web browser on a computer or tablet or mobile phone) and contact their cloud services supplier. The cloud services supplier will connect them to the software application they require.

The main advantages are that the software will be fully tested and it does not need to reside on the user's device. However, the user can still use the software even if the internet connection is lost. Data will simply be stored on the local device and then data will be uploaded or downloaded once the internet connection is restored.

Cloud-based applications can, therefore, perform tasks on a local device. This makes them fundamentally different to web-based apps which need an internet connection at all times.

2.1.5 Wired and wireless networking

Wireless

Wi-Fi and Bluetooth

Both **Wi-Fi** and **Bluetooth** offer wireless communication between devices. They both use electromagnetic radiation as the carrier of data transmission.

Bluetooth sends and receives radio waves in a band of 79 different frequencies (known as channels). These are all centred on a 2.45 GHz frequency. Devices using Bluetooth automatically detect and connect to each other, but they do not interfere with other devices since each communicating pair uses a different channel (from the 79 options).

When a device wants to communicate, it picks one of the 79 channels at random. If the channel is already being used, it randomly picks another channel. This is known as **spread spectrum frequency hopping**. To further minimise the risks of interference with other devices, the communication pairs constantly change the frequencies (channels) they are using (several times a second). Bluetooth creates a secure **wireless personal area network (WPAN)** based on key encryption.

Bluetooth is useful when

- transferring data between two or more devices which are less than 30 metres apart
- the speed of data transmission is not critical
- using low bandwidth applications (for example, sending music files from a mobile phone to a headset).

As mentioned earlier in the chapter, Wi-Fi also uses spread spectrum technology. However, Wi-Fi is best suited to operating full-scale networks, since it offers much faster data transfer rates, better range and better security than Bluetooth. A Wi-Fi-enabled device (such as a computer or smart phone) can access, for example, the internet wirelessly at any wireless access point (WAP) or 'hot spot' up to 100 metres away.

As mentioned, wireless connectivity uses electromagnetic radiation: radio waves, microwaves or infrared. The scale of frequency and wavelength of magnetic radiation is shown in Table 2.4.

	radio waves	microwaves	infrared	visible light	ultra violet	X-rays	gamma rays
Wave length (m)	10^{2}	10^{-1}	10^{-3}	10^{-5}	10^{-7}	10^{-9}	10^{-11}
Frequency (Hz)	3 MHz	3 GHz	300 GHz	30 THz	3 PHz	300 PHz	30 EHz

Table 2.4 Frequency and wavelength of magnetic radiation

EXTENSION ACTIVITY 2B

Frequency and wavelength are linked by the equation:

$$f = \frac{c}{\lambda}$$

where f = frequency (m), λ = wavelength (Hz), and c = velocity of light (3×10^8 m/s).

Confirm the frequency values in Table 2.3 using the wavelengths given.

Table 2.5 compares radio waves, microwaves and infrared. (Please note: the '>' symbol in the table means 'better than').

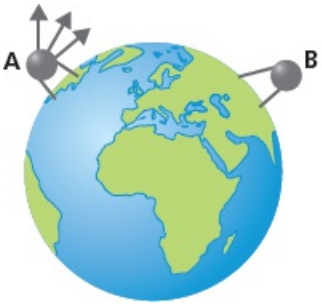
Bandwidth	infrared > microwaves > radio waves (infrared has the largest bandwidth)
Penetration	radio waves > microwaves > infrared (radio waves have the best penetration)
Attenuation	radio waves > microwaves > infrared (radio waves have the best attenuation)

Table 2.5 Comparison of radio waves, microwaves and infrared

Penetration measures the ability of the electromagnetic radiation to pass through different media. Attenuation is the reduction in amplitude of a signal (infrared has low attenuation because it can be affected by, for example, rain or internal walls). Thus, we would expect infrared to be suitable for indoor use only; the fact that it can be stopped by walls is seen as an advantage since this stops the signal causing interference elsewhere. Microwaves seem to offer the best compromise, since they support reasonable bandwidth, and have reasonable penetration and attenuation.

Additional notes on the use of satellites

The use of microwaves and radio waves was previously mentioned as a method for allowing Wi-Fi connectivity in networks. These methods are perfectly satisfactory for short distances – the electromagnetic waves carry the signals – but the curvature of the Earth prevents such methods transmitting data globally.



The electromagnetic radiation from antenna A is transmitted but is unable to reach antenna B due to the Earth's curvature.

Figure 2.10

To overcome this problem, we need to adopt satellite technology:

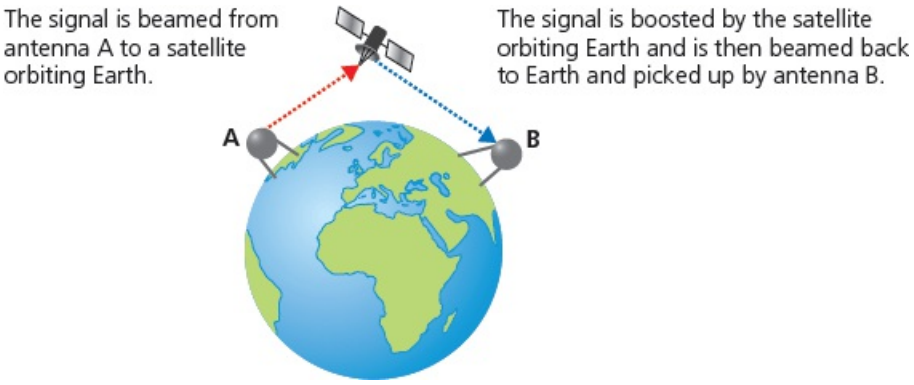


Figure 2.11

The communication between antennae and satellite is carried out by radio waves or microwave frequencies. Different frequency bands are used to prevent signal interference and to allow networks spread across the Earth to communicate through use of satellites (many satellites orbit the Earth – refer to Section 2.2.2 for more information on use of satellite technology with networks).

Wired

There are three main types of cable used in wired networks (see Figure 2.12).

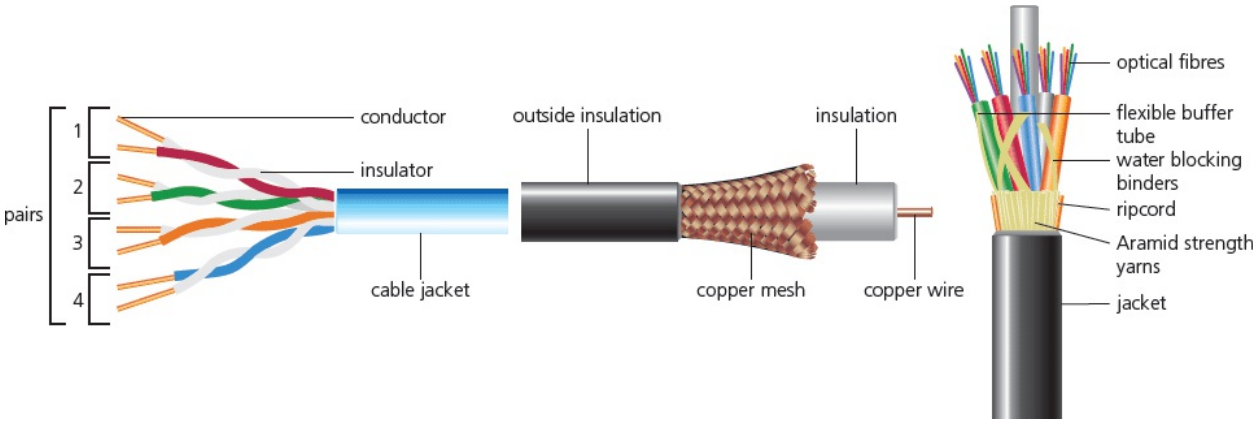


Figure 2.12 (left to right) Twisted pair cable, coaxial cable, fibre optic cable

Twisted pair cables

Twisted pair cables are the most common cable type used in LANs. However, of the three types of cable, it has the lowest data transfer rate and suffers the most from external interference (such as electromagnetic radiation). However, it is the cheapest option. There are two types of twisted pair cable: unshielded and shielded. Unshielded is used by residential users. Shielded is used commercially (the cable contains a thin metal foil jacket which cancels out some of the external interference).

Coaxial cables

Coaxial cables are the most commonly used cables in MANs and by cable television companies. The cost of coaxial cables is higher than twisted pair cables but they offer a better data transfer rate and are affected less by external interference. Coaxial cables also have about 80 times the

transmission capacity of twisted pair. Coaxial suffers from the greatest signal attenuation, but offers the best anti-jamming capabilities.

Fibre optic cables

Fibre optic cables are most commonly used to send data over long distances, because they offer the best data transfer rate, the smallest signal attenuation and have a very high resistance to external interference. The main drawback is the high cost. Unlike the other two types of cable, fibre optics use pulses of light rather than pulses of electricity to transmit data. They have about 26 000 times the transmission capacity of twisted pair cables.

Fibre optic cables can be single- or multi-mode.

Single-mode uses a single mode light source and has a smaller central core, which results in less light reflection along the cable. This allows the data to travel faster and further, making them a good choice for CATV and telecommunications.

Multi core allows for a multi-mode light source; the construction causes higher light reflections in the core, so they work best over shorter distances (in a LAN, for example).

Wired versus wireless

Numerous factors should be considered when deciding if a network should use wired or wireless connectivity, as listed below.

Wireless networking

- It is easier to expand networks and is not necessary to connect devices using cables.
- Devices have increased mobility, provided they are within range of the WAPs.
- Increased chance of interference from external sources.
- Data is less secure than with wired systems; it is easier to intercept radio waves and microwaves than cables so it is essential to protect data transmissions using encryption (such as WEP, WPA2).
- Data transmission rate is slower than wired networks (although it is improving).
- Signals can be stopped by thick walls (in old houses, for example) and signal strength can vary, or ‘drop out’.

Wired networking

- More reliable and stable network (wireless connectivity is often subjected to interference).
- Data transfer rates tend to be faster with no ‘dead spots’.
- Tends to be cheaper overall, in spite of the need to buy and install cable.
- Devices are not mobile; they must be close enough to allow for cable connections.
- Lots of wires can lead to tripping hazards, overheating of connections (potential fire risk) and disconnection of cables during routine office cleaning.

Other considerations

- If mobile phones and tablets are connected to the network, it will need to offer Wi-Fi or Bluetooth capability.
- There may be regulations in some countries regarding which wireless transmission frequencies

can be used legally.

- Permission from authorities and land owners may be required before laying cables underground.
- There are numerous competing signals in the air around us; it is important to consider this when deciding whether to go for wired or wireless connectivity.

2.1.6 Hardware requirements of networks

In this section we will consider a number of hardware items needed to form a LAN network and the hardware needed to form a WAN. Please note

- the concept of the WLAN and the hardware needed to support it have been covered in earlier sections
- the hardware items hub and **gateway** have been included in this section to complete the picture; however, knowledge of these two items is not required by the syllabus.

Hub

Hubs are hardware devices that can have a number of devices or computers connected to them.

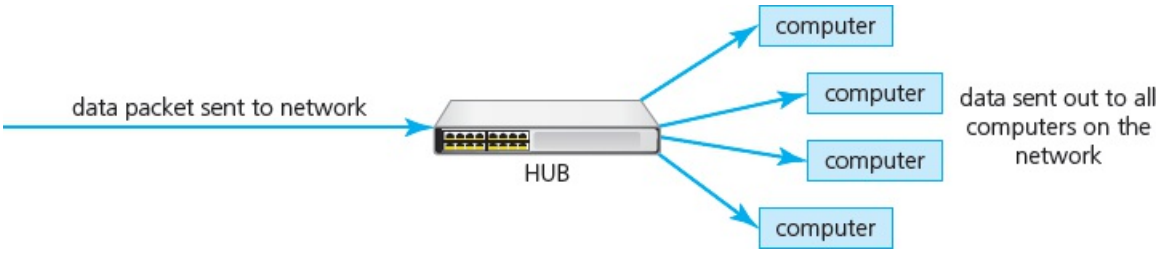


Figure 2.13 Hub flow diagram

They are often used to connect a number of devices to form a local area network (LAN), for example a star network (see Section 2.1.3). A hub's main task is to take any data packet (a group of data being transmitted) received at one of its ports and then send the data to every computer in the network. Using hubs is not a very secure method of data distribution and is also wasteful of bandwidth. Note that hubs can be wired or wireless devices.

Switch

Switches are similar to hubs, but are more efficient in the way they distribute the data packet. As with hubs, they connect a number of devices or computers together to form a LAN (for example, a star network).

However, unlike a hub, the switch checks the data packet received and works out its destination address (or addresses) and sends the data to the appropriate computer(s) only. This makes using a switch a more secure and efficient way of distributing data.

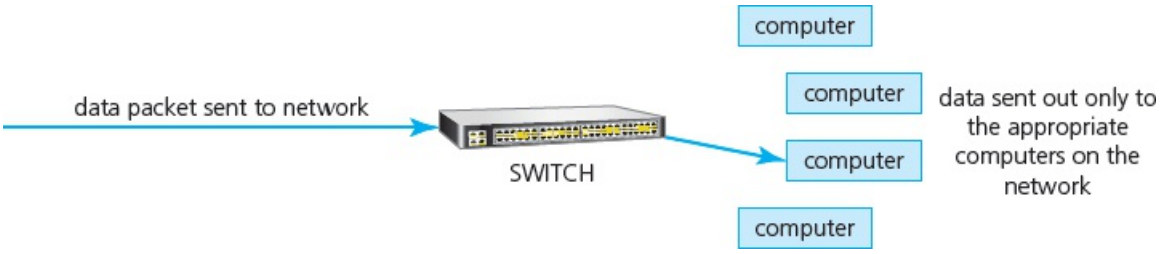


Figure 2.14 Switch flow diagram

Each device or computer on a network has a media access control (MAC) address which

identifies it uniquely. Data packets sent to switches will have a MAC address identifying the source of the data and additional addresses identifying each device which should receive the data. Note that switches can be wired or wireless devices.

Repeater

When signals are sent over long distances, they suffer attenuation or signal loss. **Repeaters** are devices which are added to transmission systems to boost the signal so it can travel greater distances. They amplify signals on both analogue (copper cable) and digital (fibre optic cable) communication links.

Repeaters can also be used on wireless systems. These are used to boost signals to prevent any 'dead spots' in the Wi-Fi zone. These devices plug into electric wall sockets and send out booster signals. They are termed non-logical devices because they will boost all signals which have been detected; they are not selective.

Sometimes, hubs contain repeaters and are known as **repeating hubs**. All signals fed to the hub are boosted before being sent to all devices in the network, thus increasing the operational range.

There are two main drawbacks of repeating hubs:

1. They have only one collision domain. When the signals are boosted and then broadcast to devices, any collisions which might occur are not resolved there and then. One way to deal with this problem is to make use of jamming signals – while this manages the collisions, it also reduces network performance since it involves repeated broadcasts as the collisions are resolved.
2. The devices are referred to as unmanaged since they are unable to manage delivery paths and also security in the network.

Bridge

Bridges are devices that connect one LAN to another LAN that uses the same protocol (communication rules). They are often used to connect together different parts of a LAN so that they can function as a single LAN.

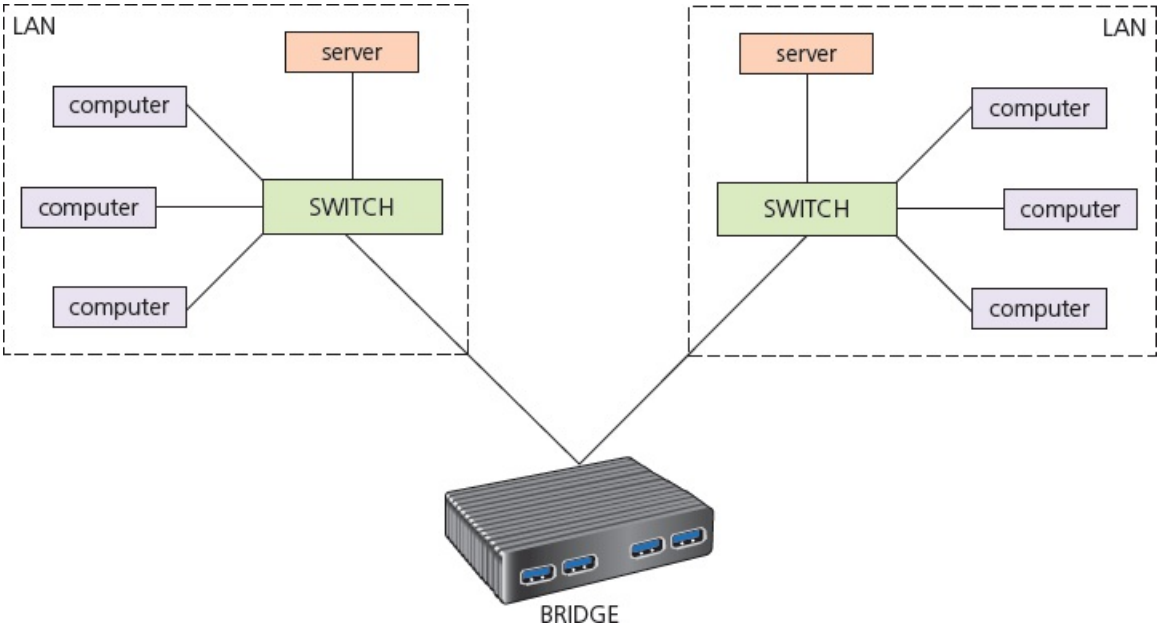


Figure 2.15 Bridge flow diagram

Bridges are used to interconnect LANs (or parts of LANs), since sending out every data packet to all possible destinations would quickly flood larger networks with unnecessary traffic. For this reason, a router is used to communicate with other networks, such as the internet. Note that bridges can be wired or wireless devices.

Router

Routers enable data packets to be routed between the different networks for example, to join a LAN to a WAN. The router takes data transmitted in one format from a network (which is using a particular protocol) and converts the data to a protocol and format understood by another network, thereby allowing them to communicate via the router. We can, therefore, summarise the role of routers as follows. Routers

- restrict broadcasts to a LAN
- act as a default gateway
- can perform protocol translation; for example, allowing a wired network to communicate with a wireless (Wi-Fi) network – the router can take an Ethernet data packet, remove the Ethernet part and put the IP address into a frame recognised by the wireless protocol (in other words, it is performing a protocol conversion)
- can move data between networks
- can calculate the best route to a network destination address.

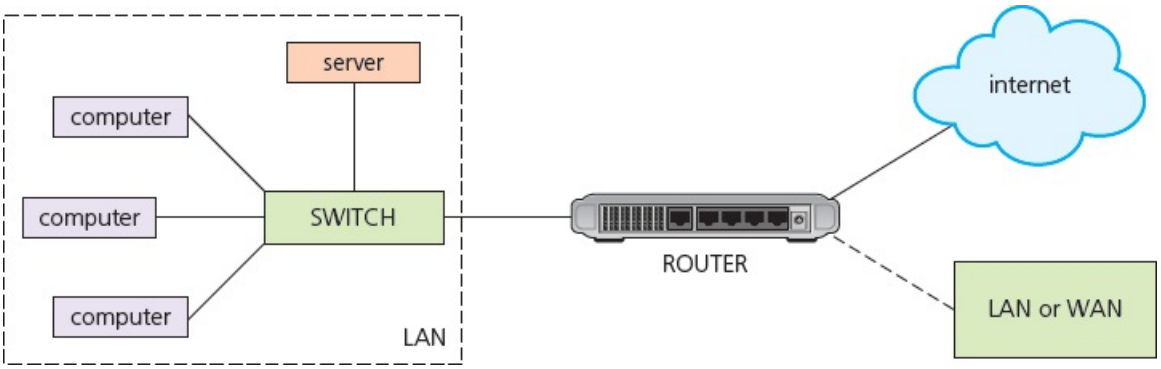


Figure 2.16 Router flow diagram

Broadband routers sit behind a firewall. The firewall protects the computers on a network. The router's main function is to transmit internet and transmission protocols between two networks and allow private networks to be connected.

The router inspects the data package sent to it from any computer on any of the networks connected to it. Since every computer on the same network has the same part of an internet protocol (IP) address, the router is able to send the data packet to the appropriate switch and it will then be delivered using the MAC destination address (see next section). If the MAC address doesn't match any device on the network, it passes on to another switch on the same network until the appropriate device is found. Routers can be wired or wireless devices.

Gateway

A gateway is a network point (or node) that acts as an entrance to another network. It is a key point for data on its way to or from other networks. It can be used to connect two or more dissimilar LANs (LANs using different protocols). The gateway converts data packets from one protocol to another. Gateways can also act as routers, firewalls or servers – in other words, any device that allows traffic to flow in and out of the networks. Gateways can be wired or wireless devices.

All networks have boundaries so that all communication within the network is conducted using devices such as switches or routers. If a network node needs to communicate outside its network, it needs to use a gateway.

Modems

Modern computers work with digital data, whereas many of the public communication channels still only allow analogue data transmission. To allow the transmission of digital data over analogue communication channels we need to use a modem (**mo**dulator **dem**odulator). This device converts digital data to analogue data. It also does the reverse and converts data received over the analogue network into digital data which can be understood by the computer.

Wireless modems transmit data in a modulated form to allow several simultaneous wireless communications to take place without interfering with each other. A modem will connect to the public infrastructure (cable, telephone, fibre-optics or satellite) and will supply the user with a standard Ethernet output which allows connection to a router, thus enabling an internet connection to occur.

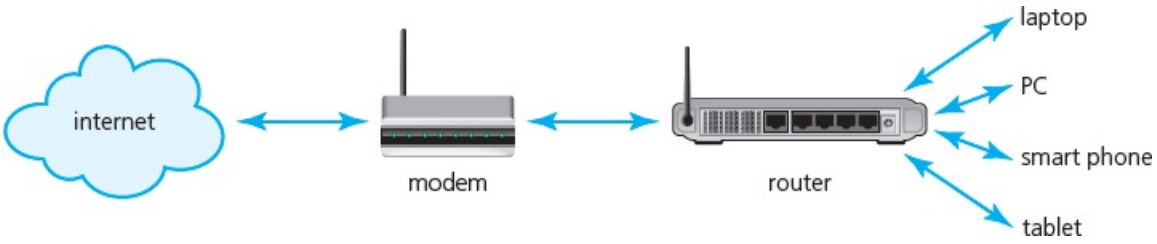


Figure 2.17 Wireless modem flow diagram

While the router will allow the creation of a network in a home, for example, the modem allows for the connection to the external networks (for example, the internet). Routers and modems can be combined into one unit; these devices have the electronics and software to provide both router and modem functions.

Another example of a modem is a **softmodem** (**soft**ware **modem**), which uses minimal hardware and uses software that runs on the host computer. The computer's resources (mainly the processor and RAM) replace the hardware of a conventional modem.

Table 2.6 shows the differences between routers and gateways.

Routers	Gateways
• forward packets of data from one network to another; routers read each incoming packet of data and decide where to forward the packet	• convert one protocol (or data format) to another protocol (format) used in a different network
• can route traffic from one network to another network	• convert data packets from one protocol to another; they act as an entry and exit point to networks
• can be used to join LANs together to form a WAN (sometimes called brouters) and also to connect a number of LANs to the internet	• translate from one protocol to another
• offer additional features such as dynamic routing (ability to forward data by different routes)	• do not support dynamic routing

Table 2.6 Differences between routers and gateways

EXTENSION ACTIVITY 2C

Draw a diagram to show how a gateway could be used to connect together three LANs which are using different protocols. Include all the hardware devices and cables needed.

Network interface card (NIC)

A **network interface card (NIC)** is needed to allow a device to connect to a network (such as the internet). It is usually part of the device hardware and frequently contains the MAC address generated at the manufacturing stage.

Wireless network interface card/controller (WNIC)

Wireless network interface cards/controllers (WNICs) are the same as the more ordinary NICs, in that they are used to connect devices to the internet or other networks. They use an antenna to communicate with networks via microwaves and normally simply plug into a USB port or can be internal integrated circuit plug in.

Figure 2.18 Wireless network interface card/controller (WNIC)

As with usual NICs, they work on layers 1 and 2 of the OSI model (refer to Chapter 14 for more details). WNICs work in two modes.

Infrastructure mode requires WAPs (wireless access points) and all the data is transferred using the WAP and hub/switch; all the wireless devices connect to the WAP and must use the same security and authentication techniques.

Ad hoc mode does not need to have access to WAPs; it is possible for devices to interface with each other directly.

2.1.7 Ethernet

Ethernet is a protocol used by many wired LANs. It was adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) and Ethernet is also known as IEEE 802.3. A network using Ethernet is made up of:

- a node (any device on the LAN)
- medium (path used by the LAN devices, such as an Ethernet cable)
- frame (data is transmitted in frames which are made up of source address and destination address – the addresses are often the MAC address).

Conflicts

When using Ethernet, it is possible for IP addresses to **conflict**; this could show up as a warning such as that in Figure 2.19.

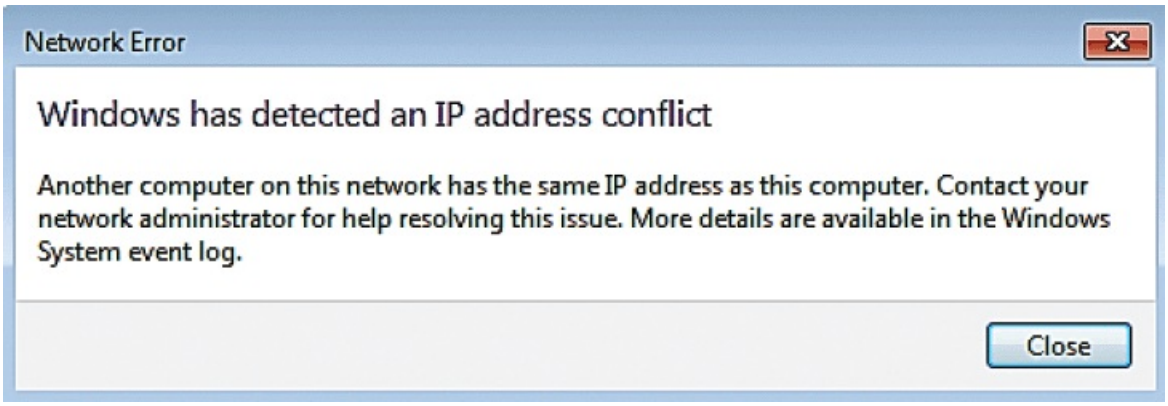


Figure 2.19 IP address conflict error

This may occur if devices on the same network have been given the same IP address; without a unique IP address it is not possible to connect to a network. This is most likely to occur on a LAN where dynamic IP addresses may have been used. Dynamic IP addresses are temporary and may have been assigned to a device on the network, unfortunately, another device using static IP addresses may already have the same IP address. This can be resolved by re-starting the router. Any dynamic IP addresses will be re-assigned, which could resolve the issue.

Collisions

Ethernet supports **broadcast** transmission (communications where pieces of data are sent from sender to receiver) and are used to send messages to all devices connected to a LAN. The risk is that two messages using the same data channel could be sent at the same time, leading to a **collision**. **Carrier sense multiple access with collision detection (CSMA/CD)** was developed to try and resolve this issue. Collison detection depends on simple physics: when a frame is sent it causes a voltage change on the Ethernet cable. When a collision is detected, a node stops transmitting a frame and transmits a ‘jam’ signal and then waits for a random time interval before trying to resend the frame. CSMA/CD protocol will define the random time period for a device to wait before trying again.

Figure 2.20 shows how data collisions can be dealt with using transmission counters (which keep track of how many times the collision detection routine has been entered – there will a defined limit as part of the CSMA/CD protocol) and random time periods.

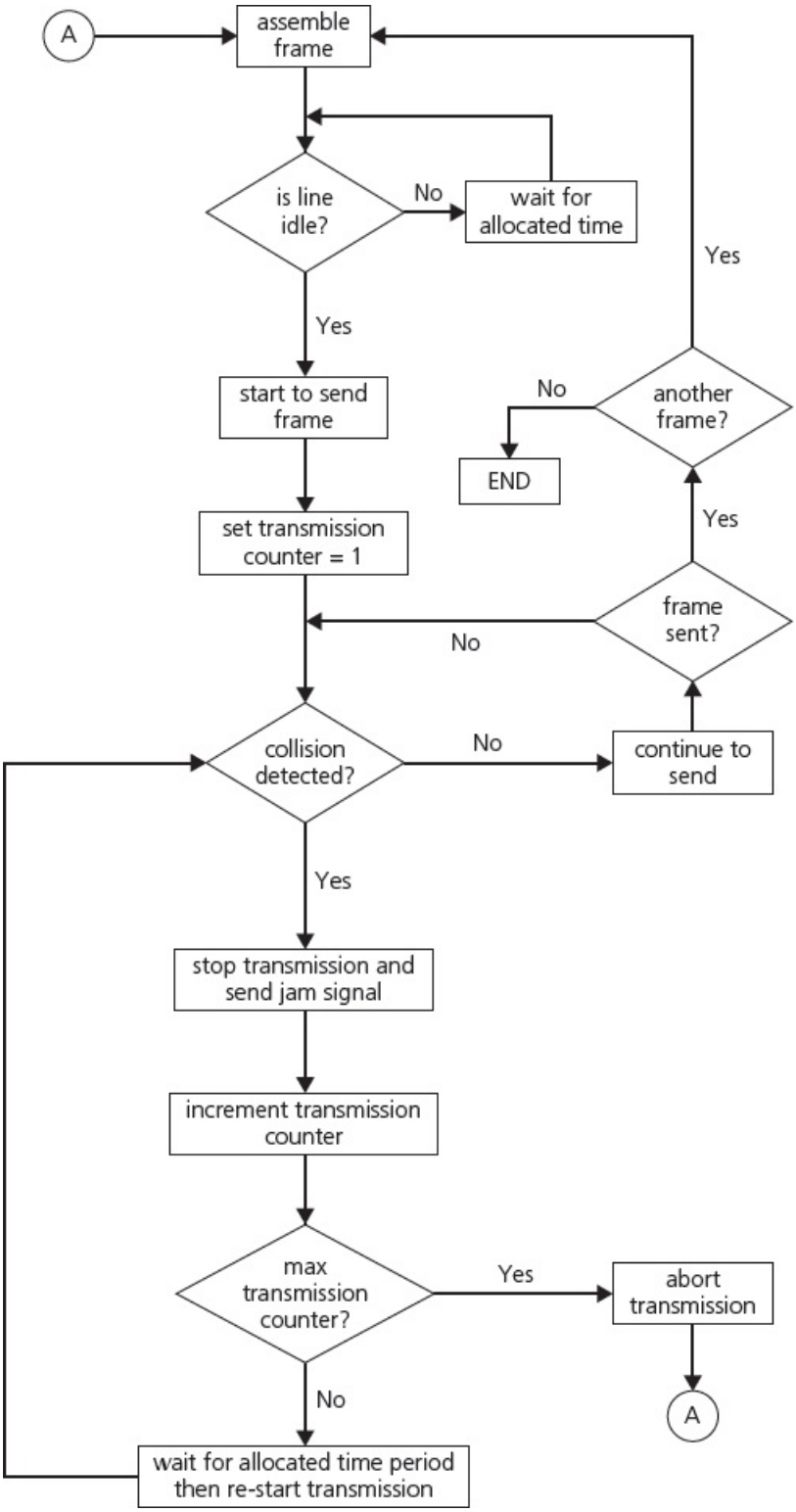


Figure 2.20 How data collisions can be dealt with using transmission counters

EXTENSION ACTIVITY 2D

Review Figure 2.20. As it stands, it is possible for an endless loop to be established.

Suggest a modification to the flow diagram to ensure it terminates if there is a problem with the data channel, or to prevent the data transmission holding up the computer for an unacceptable time period.

2.1.8 Bit streaming

Bit streaming is a contiguous sequence of digital bits sent over the internet or a network that requires a high speed data communication link (such as fast broadband). Since bit streaming often involves very large files (such as video) it is necessary for the files to undergo some data compression before transmission. It is also necessary to have some form of **buffering** to ensure smooth playback of the media files.

The data transmission rate from the file server (containing the video, for example) to the buffer must be greater than the rate at which data is transmitted from buffer to media player. The larger the buffer, the better the control over the **bit rate** being sent to the media player. The media player will always check to ensure data lies between a minimum value (often referred to as low water mark) and a maximum value (often referred to as a high water mark). The difference between the two values is usually about 80% of the total buffer capacity. The buffer is a temporary storage area of the computer.

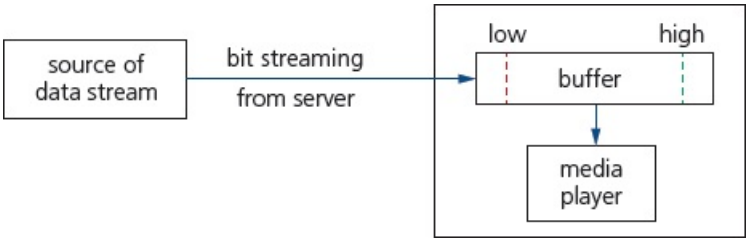


Figure 2.21 Bit streaming

Table 2.7 shows the pros and cons of bit streaming.

Pros of bit streaming	Cons of bit streaming
• no need to wait for a whole video or music file to be downloaded before the user can watch or listen • no need to store large files on your device • allows video files and music files to be played on demand (as required) • no need for any specialist hardware • affords piracy protection (more difficult to copy streamed files than files stored on a hard drive)	• cannot stream video or music files if broadband connection is lost • video or music files will pause to allow the data being streamed to 'catch up' if there is insufficient buffer capacity or slow broadband connection • streaming uses up a lot of bandwidth • security risks associated with downloading files from the internet • copyright issues

Table 2.7 Pros and cons of bit streaming

Bit streaming can be either **on demand** or **real time**.

On demand

- Digital files stored on a server are converted to a bit streaming format (encoding takes place and the encoded files are uploaded to a server).
- A link to the encoded video/music file is placed on the web server to be downloaded.
- The user clicks on the link and the video/music file is downloaded in a contiguous bit stream.
- Because it is on demand, the streamed video/music is broadcast to the user as and when required.
- It is possible to pause, rewind and fast forward the video/music if required.

Real time

- An event is captured by camera and microphone and is sent to a computer.
- The video signal is converted (encoded) to a streaming media file.
- The encoded file is uploaded from the computer to the dedicated video streaming server.
- The server sends the encoded live video to the user's device.
- Since the video footage is live it is not possible to pause, rewind or fast forward.

ACTIVITY 2B

1 a) Explain the differences between LAN, MAN and WAN.

b) Give **three** of the benefits of networking computers.

c) Explain the following terms.

i) Thick client

ii) Thin client

2 a) Draw diagrams to show the following network topologies.

i) Bus

ii) Star

iii) Mesh

b) Give **one** benefit and **one** drawback of using each type of network topology.

3 a) Explain the differences between public and private cloud computing.

b) Give **two** benefits of using cloud computing.

c) Give **two** drawbacks of using cloud computing.

4 You have been asked by a manager to write a report on whether a LAN being set up in their new building should use wired or wireless connectivity. The building has 20 floors. Explain your arguments for and against using both types of connectivity and draw a conclusion to help the manager make their decision.

5 a) What is meant by *bit streaming*?

b) Why is it necessary to use buffers whilst streaming a video from the internet?

c) Explain the differences between on demand and real time bit streaming.

2.2 The internet

Key terms

Internet – massive network of networks, made up of computers and other electronic devices; uses TCP/IP communication protocols.

World Wide Web (WWW) – collection of multimedia web pages stored on a website, which uses the internet to access information from servers and other computers.

HyperText Mark-up Language (HTML) – used to design web pages and to write http(s) protocols, for example.

Uniform resource locator (URL) – specifies location of a web page (for example, www.hoddereducation.co.uk).

Web browser – software that connects to DNS to locate IP addresses; interprets web pages sent to a user's computer so that documents and multimedia can be read or watched/listened to.

Internet service provider (ISP) –
company which allows a user to connect to the internet. They will usually charge a monthly fee for the service they provide.

Public switched telephone network (PSTN) – network used by traditional telephones when making calls or when sending faxes.

Voice over Internet Protocol (VoIP) – converts voice and webcam images into digital packages to be sent over the internet.

Internet protocol (IP) – uses IPv4 or IPv6 to give addresses to devices connected to the internet.

IPv4 – IP address format which uses 32 bits, such as 200.21.100.6.

Classless inter-domain routing (CIDR) – increases IPv4 flexibility by adding a suffix to the IP address, such as 200.21.100.6/18.

IPv6 – newer IP address format which uses 128 bits, such as A8F0:7FFF:F0F1:F000:3DD0:256A:22FF:AA00.

Zero compression – way of reducing the length of an IPv6 address by replacing groups of zeroes by a double colon (::); this can only be applied once to an address to avoid ambiguity.

Sub-netting – practice of dividing networks into two or more sub-networks.

Private IP address – an IP address reserved for internal network use behind a router.

Public IP address – an IP address allocated by the user's ISP to identify the location of their device on the internet.

Domain name service (DNS) – (also known as domain name system) gives domain names for internet hosts and is a system for finding IP addresses of a domain name.

JavaScript® – object-orientated (or scripting) programming language used mainly on the web

to enhance HTML pages.

PHP – hypertext processor; an HTML-embedded scripting language used to write web pages.

2.2.1 The differences between the internet and the World Wide Web

There are fundamental differences between the **internet** and the **World Wide Web (WWW)**.

Internet

- The internet is a massive network of networks (although, as explained in Section 2.1.1, the internet is not a WAN) which are made up of various computers and other electronic devices.
- It stands for interconnected network.
- The internet makes use of transmission control protocol (TCP)/internet protocol (IP).

World Wide Web (WWW)

- This is a collection of multimedia web pages and other documents which are stored on websites.
- http(s) protocols are written using **HyperText Mark-up Language (HTML)**.
- **Uniform resource locators (URLs)** specify the location of all web pages.
- Web resources are accessed by **web browsers**.
- The world wide web uses the internet to access information from servers and other computers.

2.2.2 Hardware and software needed to support the internet

The fundamental requirements for connecting to the internet are

- a device (such as a computer, tablet or mobile phone)
- a telephone line connection or a mobile phone network connection (however, it is possible that a tablet or mobile phone may connect to the internet using a wireless router)
- a router (which can be wired or wireless) or router and modem
- an **internet service provider (ISP)** (combination of hardware and software)
- a web browser.

The telephone network system, **public switched telephone network (PSTN)**, is used to connect computers/devices and LANs between towns and cities. Satellite technology is used to connect to other countries (see later).

In recent years, telephone lines have changed from copper cables to fibre optic cables, which permits greater bandwidth and faster data transfer rates (and less risk of data corruption from interference). Fibre optic telephone networks are usually identified as 'fast broadband'. As discussed earlier, high speed broadband has allowed WLANs to be developed by using WAPs.

High speed communication links allow telephone and video calls to be made using a computer and the internet. Telephone calls require either an internet-enabled telephone connected to a computer (using a USB port) or external/internal microphone and speakers. Video calls also require a webcam. When using the internet to make a phone call, the user's voice is converted to digital packages using **Voice over Internet Protocol (VoIP)**. Data is split into packages (packet switching) and sent over the network via the fastest route. Packet switching and circuit switching are covered in more detail in Chapter 14.

Comparison between PSTN and internet when making a phone call

Public switched telephone network (PSTN)

PSTN uses a standard telephone connected to a telephone line.

The telephone line connection is always open whether or not anybody is talking – the link is not terminated until the receivers are replaced by both parties.

Telephone lines remain active even during a power cut; they have their own power source.

Modern phones are digitised systems and use fibre optic cables (although because of the way it works this is a big waste of capacity – a 10 minute phone call will transmit about 10 MB of data).

Existing phone lines use circuit switching (when a phone call is made the connection (circuit) is maintained throughout the duration of the call – this is the basis of PSTN).

Phone calls using the internet

Phone calls using the internet use either an internet phone or microphone and speakers (video calls also require a webcam).

The internet connection is only ‘live’ while data (sound/video image) is being transmitted.

Voice over Internet Protocol (VoIP) converts sound to digital packages (encoding) which can be sent over the internet.

VoIP uses packet switching; the networks simply send and retrieve data as it is needed so there is no dedicated line, unlike PSTN. Data is routed through thousands of possible pathways, allowing the fastest route to be determined.

The conversation (data) is split into data packages. Each packet contains at least the sender’s address, receiver’s address and order number of packet – the sending computer sends the data to its router which sends the packets to another router, and so on. At the receiving end, the packets are reassembled into the original state (see Chapter 14 for more details).

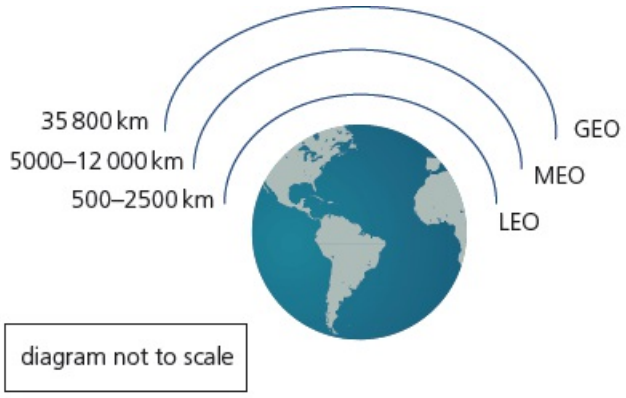
VoIP also carries out file compression to reduce the amount of data being transmitted.

Because the link only exists while data is being transmitted, a typical 10 minute phone call may only contain about 3 minutes where people are talking; thus only 3 MB of data is transmitted making it much more efficient than PSTN.

Cellular networks and satellites

Other devices, such as mobile phones, use the cellular network. Here, the mobile phone providers act as the ISPs and the phones contain communication software which allows them to access the telephone network and also permits them to make an internet connection.

Satellites are an important part of all network communications that cover vast distances. Due to the curvature of the Earth, the height of the satellite’s orbit determines how much coverage it can give. Figure 2.22 shows how satellites are classified according to how high they orbit in relation to the Earth’s surface.



Geostationary Earth Orbit (GEO) provide long distance telephone and computer network communications; orbital period = 24 hours

Medium Earth Orbit (MEO) used for GPS systems (about 10 MEO satellites are currently orbiting the Earth); orbital period = 2 to 12 hours

Low Earth Orbit (LEO) used by the mobile phone networks (there are currently more than 100 LEO satellites orbiting the Earth); orbital period = 80 mins to 2 hours

Figure 2.22 Satellite classification

Satellites have the advantage that they will always give complete coverage and don’t suffer from signal attenuation to the same extent as underground/undersea cables. It is also difficult to isolate and resolve faults in cables on the sea bed.

2.2.3 IP addresses

The internet is based on TCP/IP protocols. Protocols define the rules that must be agreed by senders and receivers on the internet. Protocols can be divided into TCP layers (see Chapter 14). We will first consider **internet protocols (IP)**.

Internet protocols (IP)

IPv4 addressing

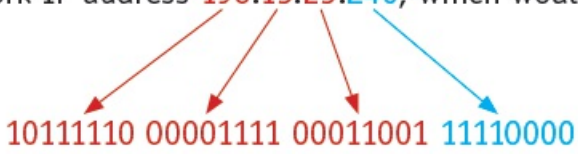
The most common type of addressing on the internet is **IPv4 IP version 4 (IPv4)**. This is based on 32 bits giving 2^{32} (4 294 967 296) possible addresses. The 32 bits are split into four groups of 8 bits (thus giving a range of 0 to 255). For example, 254.0.128.77.

The system uses the group of bits to define network (netID) and network host (hostID). The netID allows for initial transmission to be routed according to the netID and then the hostID is looked at by the receiving network. Networks are split into five different classes, as shown in Table 2.8 below.

Network class	IPv4 range	Number of netID bits	Number of hostID bits	Types of network
A	0.0.0.0 to 127.255.255.255	8	24	very large
B	128.0.0.0 to 191.255.255.255	16	16	medium size
C	192.0.0.0 to 223.255.255.255	24	8	small networks
D	224.0.0.0 to 239.255.255.255	–	–	multi-cast
E	240.0.0.0 to 255.255.255.255	–	–	experimental

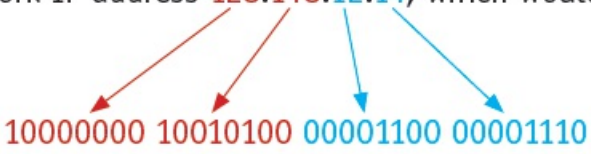
Table 2.8 The five network classes

Consider the class C network IP address 190.15.25.240, which would be written in binary as:



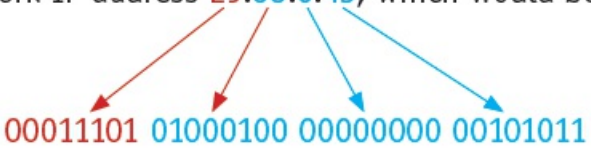
Here the network id is 190.15.25 and the host ID is 240.

Consider the class B network IP address 128.148.12.14, which would be written in binary as:



Here the network ID is 128.148 and the host ID is 12.14 (made up of sub-net ID 12 and host ID of 14).

Consider the class A network IP address 29.68.0.43, which would be written in binary as:



Here the network ID is 29 and the host ID is 68.0.43 (made up of sub-net ID 68.0 and host ID of 43).

Here the network ID is 29 and the host ID is 68.0.43 (made up of sub-net ID 68.0 and host ID of 43).

However, it soon became clear that this IPv4 system provides insufficient address range. For example, a user with a medium sized network (class B) might have 284 host machines and their class B licence allows them 2^{16} (65534; note the value is not 65536 since two values are not assigned). This means several of the allocated host IDs will not be used, which is wasteful.

Classless inter-domain routing (CIDR) reduces this problem by increasing the flexibility of the IPv4 system. A suffix is used, such as 192.30.250.00/18, which means 18 bits will be used for the net ID and the last 14 bits will be used for the host ID (rather than the normal 24 bits and 8 bits for a class C network). The suffix clearly increases the flexibility regarding which bits represent the net ID and which represent the host ID.

EXTENSION ACTIVITY 2E

Network address translation (NAT) removes the need for each IP address to be unique. Find out how it works.

IPv6 addressing

IPv6 addressing has been developed to overcome some of the problems associated with IPv4. This system uses 128-bit addressing, which allows for much more complex addressing structures. An IPv6 address is broken into 16-bit chunks and because of this, it adopts the hexadecimal notation. For example:

A8FB:7A88:FFF0:0FFF:3D21:2085:66FB:F0FA

Note how a colon (:) rather than a decimal point (.) is used here.

It has been designed to allow the internet to grow in terms of number of hosts and the potential amount of data traffic. IPv6 has benefits over IPv4, it

- has no need for NATs (network address translation)
- removes risk of private IP address collisions
- has built in authentication
- allows for more efficient routing.

Zero compression

IPv6 addresses can be quite long; but there is a way to shorten them using **zero compression**. For example, 900B:3E4A:AE41:0000:0000:AFF7:DD44:F1FF can be written as:

900B:3E4A:AE41::AFF7:DD44:F1FF

With the section 0000:0000 replaced by ::

The zero compression can only be applied ONCE to an IPv6 address, otherwise it would be impossible to tell how many zeros were replaced on each occasion where it was applied. For example, 8055:F2F2:0000:0000:FFF1:0000:0000:DD04 can be rewritten either as:

8055:F2F2::FFF1:0000:0000:DD04

or as:

8055:F2F2:0000:0000:FFF1::DD04

8055:F2F2::FFF1::DD04 is not a legal way of compressing the original address – we have no way of knowing whether the original address was

8055:F2F2:0000:FFF1:0000:0000:0000:DD04

or

8055:F2F2:0000:0000:0000:FFF1:0000:DD04

or

8055:F2F2:0000:0000:FFF1:0000:0000:DD04

It would, therefore, be regarded as ambiguous.

Sub-netting

CIDR is actually based on **sub-netting** and the two are similar in many ways. Sub-netting divides a LAN into two or more smaller networks. This helps reduce network traffic and can also hide the complexity of the overall network. Recall that the IP address (using IPv4) is made up of the netID and hostID. Suppose a university network has eight departments and has a netID of 192.200.20 (11000000.11001000.00010100). All of the devices on the university network will be associated with this netID and can have hostID values from 00000001 to 1111110 (hostIDs

containing all 0s or all 1s are forbidden). The university network will look something like this:

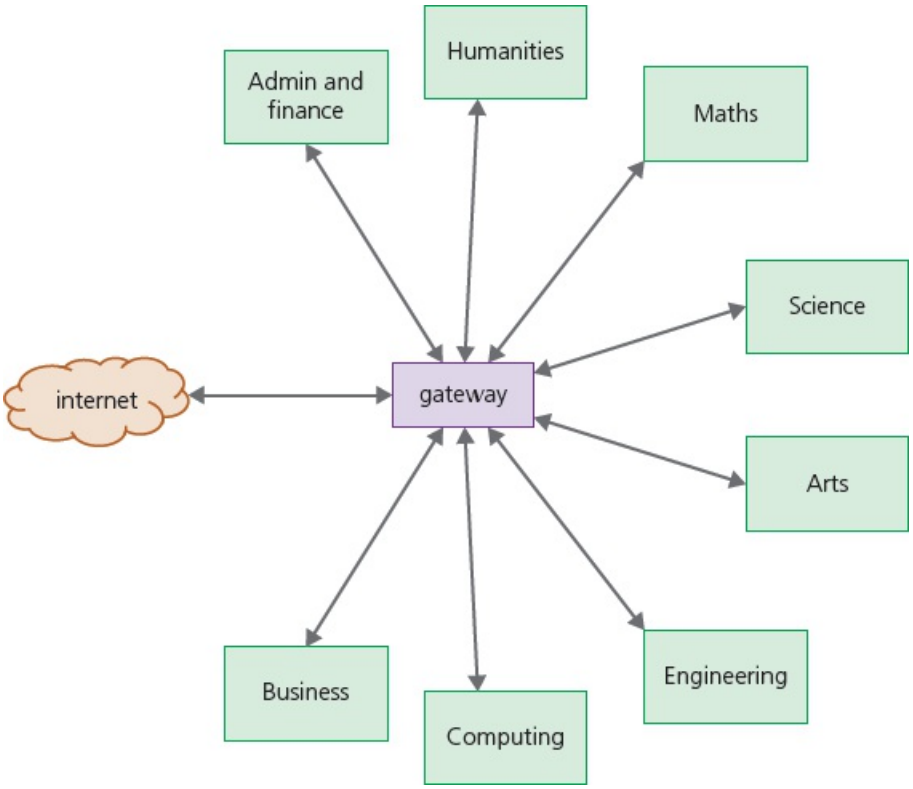


Figure 2.23 An example of a university network

So, for example, the devices in the Admin and finance department might have hostIDs of 1, 8, 240, 35, 67, 88, 134, and so on, with similar spreads for the other seven departments.

It would be beneficial to organise the netIDs and hostIDs so that the network was a lot less complex in nature. With sub-netting, the hostID is split as follows:

000 00000, where the first 3 bits are netID expansion and the last 5 bits are the hostIDs.

Thus, we have eight sub-nets with the same range of hostIDs.

Department	**netID**	**hostID range**
Admin and finance	192.200.20.0	00001 to 11110
Humanities	192.200.20.1	00001 to 11110
Maths	192.200.20.2	00001 to 11110
Science	192.200.20.3	00001 to 11110
Arts	192.200.20.4	00001 to 11110
Engineering	192.200.20.5	00001 to 11110
Computing	192.200.20.6	00001 to 11110

Business	192.200.20.7	00001 to 11110

Table 2.9

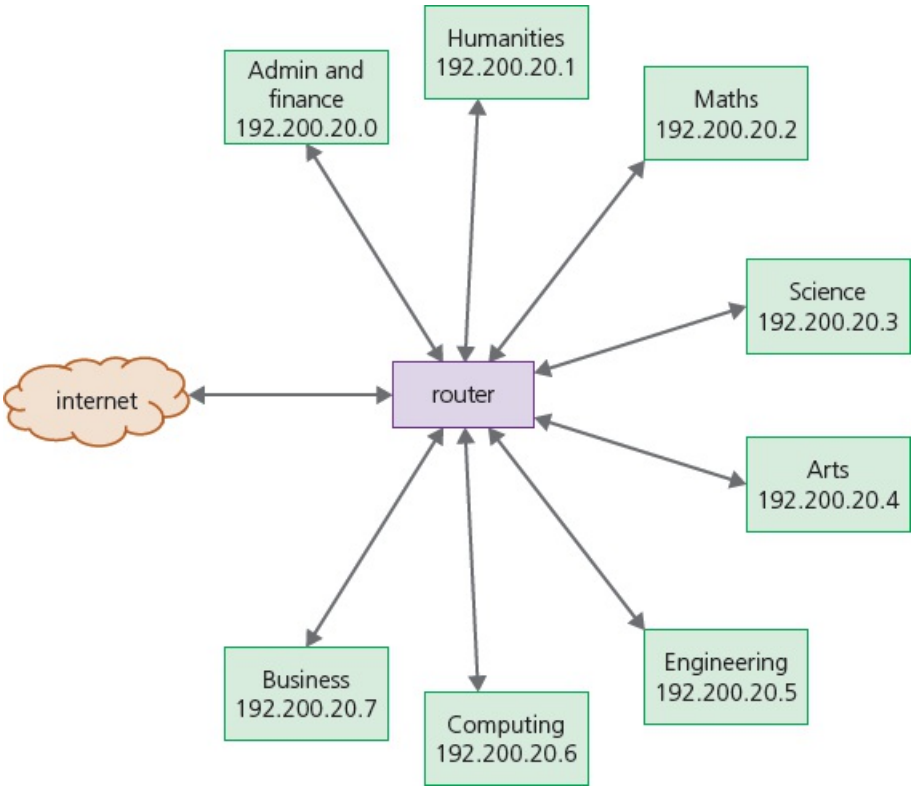


Figure 2.24 An example of a university network with netIDs

The devices in the Admin and finance department will have IP addresses

192.200.20.000 00001 to 192.200.20.000 11110

The Humanities department will have IP addresses

192.200.20.001 00001 to 192.200.20.001 11110

And so on for the other departments.

To obtain the netID from the IP address we can apply the AND mask (recall that 1 AND 1 = 1, 0 AND 0 = 0 or 1 AND 0 = 0). Thus, if a device has an IP address of

11000000.11001000.00010100.011 00011

we can apply the AND mask

11111111.11111111.11111111.111 00000

which results in the netID value

11000000.11001000.00010100.011 00000 (or 192.200.20.03)

This is the Science department. Consequently, the whole network is more efficient (for the reasons stated above) and less complex. Compare this to CIDR 192/200/20/0/27, which extends the size of the netID to 27 bits and has a hostID of only 5 bits, but would not reduce the complexity of the network.

Private IP addresses and public IP addresses

Private IP addresses are reserved for internal use behind a router or other NAT device. The following blocks are reserved for private IP addresses.

Class A	10.0.0.0 to 10.255.255.255	16 million possible addresses
Class B	172.16.0.0 to 172.31.255.255	1 million possible addresses
Class C	192.168.0.0 to 192.168.255.255	65 600 possible addresses

Table 2.10

Private IP addresses (which are internal value only) allow for an entirely separate set of addresses within a network. They allow access to the network without taking up a public IP address space. However, devices using these private IP addresses cannot be reached by internet users.

Public IP addresses are the ones allocated by a user's ISP to identify the location of their device. Devices using these IP addresses are accessible from anybody using the internet. Public IP addresses are used by

- DNS servers
- network routers
- directly-controlled computers.

2.2.4 Uniform resource service (URLs)

Web browsers are software that allow users to access and display web pages on their screens. They interpret HTML sent from websites and display the results. Web browsers use uniform resource locators (URL) to access websites; these are represented by a set of four numbers, such as 109.108.158.1.

But it is much easier to type this into a browser using the following format:

protocol://website address/path/filename

Protocol is usually http or https

Website address is

- domain host (www)
- domain name (name of website)
- domain type (.com, .org, .net, .gov, and so on)
- (sometimes) a country code (.uk, .de, .cy, .br, and so on).

Path is the web page (if this is omitted then it is the root directory of the website)

Filename is the item from the web page

http://www.hoddereducation.co.uk/computerscience

2.2.5 Domain name service (DNS)

The **domain name service (DNS)** (also known as domain name system) gives domain names for internet hosts and is a system for finding IP addresses of a domain name. Domain names eliminate the need for a user to memorise IP addresses. The DNS process involves converting a host name (such as www.hoddereducation.co.uk) into an IP address the computer can understand (such as 107.162.140.19).

Often, DNS servers contain a database of URLs with the matching IP addresses.

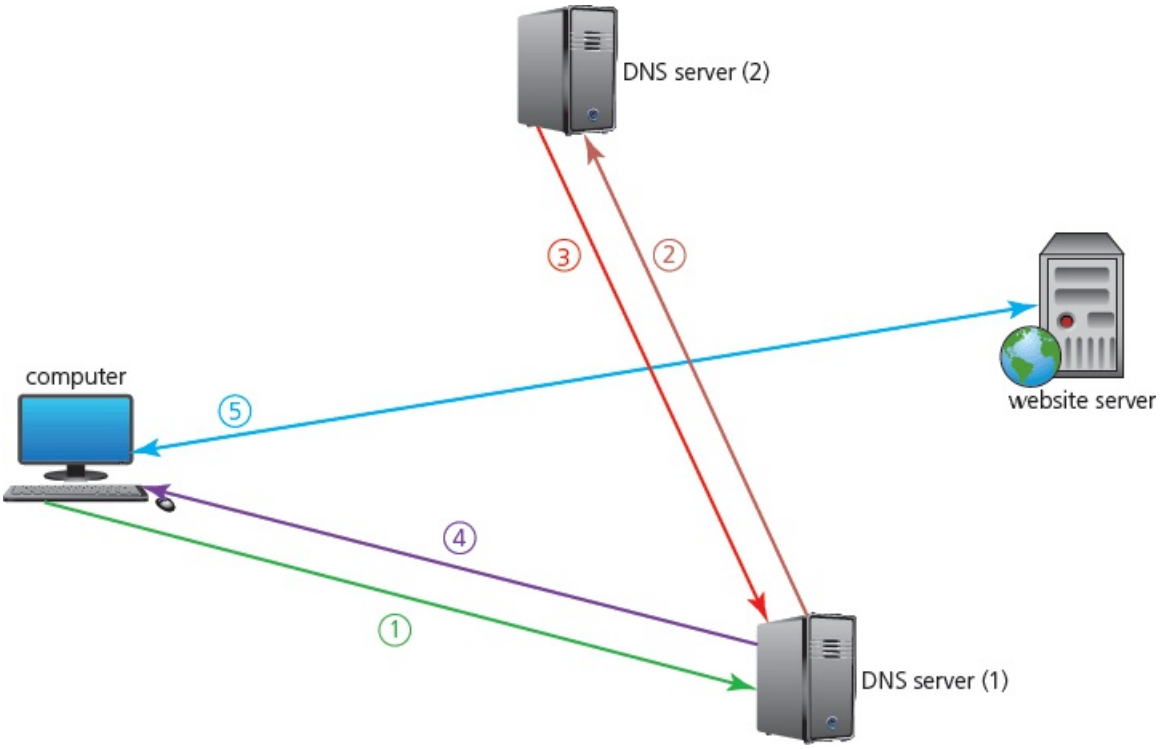


Figure 2.25 An example of the DNS process

① The user opens their web browser and types in the URL (www.hoddereducation.co.uk) and the web browser asks the DNS server (1) for the IP address of the website.

② The DNS server can't find www.hoddereducation.co.uk in its database or its cache and sends out a request to DNS server (2).

③ DNS server (2) finds the URL and can map it to 107.162.140.19; the IP address is sent back to DNS server (1) which now puts the IP address and associated URL into its cache/database.

④ This IP address is then sent back to the user's computer.

⑤ The computer now sets up a communication with the website server and the required pages are downloaded. The web browser interprets the HTML and displays the information on the user's screen.

2.2.6 Scripting in HTML

This section considers HTML scripting using JavaScript and PHP. While this extends beyond the syllabus, it is included here to help you understand how HTML is used to create websites and how web browsers communicate with servers. It is included here for information and to aid understanding.

A user may wish to develop a web application, which is client-server based, on their own computer. To do this they would need to:

- download the necessary server software
- install the application on the chosen/allocated server
- use the web browser on their computer to access and interpret the application web pages.

Each web page would need to be created using HTML. A domain name would have to be purchased from a web-hosting company. The HTML files would need to be uploaded to the server which was allocated to the user by the web-hosting company.

HTML would be used to create a file using tags. For example:

```
<html>
<body>
<p> Example <p/>
[program code]
</html>
```

Between the HTML tags the inclusion of JavaScript or PHP can be used.

JavaScript

JavaScript (unlike HTML) is a programming language which will run on the client-side. What is the difference between running on the client-side and running on the server-side?

- Client-side – the script runs on the computer, which is making the request, processing the web page data that is being sent to the computer from the server.
- Server-side – the script is run on the web server and the results of processing are then sent to the computer that made the request.

The following short program inputs a temperature and outputs ‘HIGH’ if it is 200 °C or over, ‘OK’ if it is 100 °C or over and ‘LOW’ if it is below 100 °C.

```
01  <html>
02  <body>
03  <p>Enter the temperature</p>
04  <input id="Temp" value="0"
05  <button onclick="checkReading()>"Enter</button>
06  <script>
07    function checkReading() {
08      var temp, result;
09      temp = document.getElementById("Temp").value;
10      if (temp >= 200) {
11      result = "HIGH"
12      } else if (temp >= 100) {
13        result = "OK"
14      } else {
15        result = "LOW"
16      }
17      alert("The result is " + result)
18    }
19  </script>
20  </body>
21  </html>
```

PHP

PHP is another language which can be embedded within HTML. However, when PHP is used it is processed on the server-side. Again, the code will be sandwiched inside HTML and will be stored as a .php file.

The following example is similar to the JavaScript example; again temperatures are input but this time ‘H’, ‘O’ and ‘L’ are output depending on the result. Note that variables begin with $ and are case-sensitive.

```
01  <?php
02    if(isset($ _ GET['temp'])) {
03    echo "Result: " . checkReading($ _ GET['temp']);
04    } else {
05  ?>
06  <form action="#" method="get">
07    Enter Temp: <input type="text" name="temp" /><br />
08    <input type="submit" value="Calculate" />
09    </form>
10
11  <?php
12    }
13      function checkReading($inputTemp) {
14        $resultChar = "L";
15      if($inputTemp >= 200) $resultChar = "H";
16      else if($inputTemp >= 100) $resultChar = "O";
17      return $resultChar;
18    }
19  ?>
```

EXTENSION ACTIVITY 2F

Look at the two pieces of code in the previous JavaScript and PHP sections, then answer these questions.

a) Write down the names of two variables which are used in each piece of code.

b) In each case, identify which statement(s) correspond(s) to an output.

c) What is the purpose of the statement shown in line:

- **i)** 09 of the JavaScript code
- **ii)** 03 of the PHP code?

e) What is the purpose of line 05 in the JavaScript?

ACTIVITY 2C

1 a) Describe what happens when a telephone call is made using PSTN.

b) Describe what happens when a computer, equipped with microphone and speakers, is used to make a 'telephone' call over the internet.

c) Communication links between continents frequently involve the use of satellite

technology. Explain the differences between GEO, MEO and LEO satellites.

2 **a)** Class A computer networks are identified by IP addresses starting with 0.0.0.0, class B computer networks are identified by IP addresses starting with 128.0.0.0 and class C computer networks are identified by IP addresses starting with 192.0.0.0. (Class D networks begin with 224.0.0.0.)
Write these starting IP addresses in binary format.

b) Using the data above, write down the upper IP addresses of the three network classes A, B and C.

c) A device on a network has the IP address:
10111110 00001111 00011001 11110000

i) Which class of network is the device part of?

ii) Which bits are used for the net ID and which bits are used for the host ID?

iii) A network uses IP addresses of the form 200.35.254.25/18.

Explain the significance of the appended value 18.

d) Give **two** differences between IPv4 and IPv6.

3 **a)** Describe the differences between private IP addresses and public IP addresses.

b) Identify the protocol, domain name and file name used in the following URL:
https://www.exampleofaurl.co.de/computer_logic.html

c) Describe how DNS is used to retrieve a web page from the website used in part b).

4 **a)** Explain the differences between the internet and the world wide web (www).

b) Hasina wrote,
'The internet is not necessarily a type of WAN.'
Is Hasina's statement correct? Give reasons for your answer.

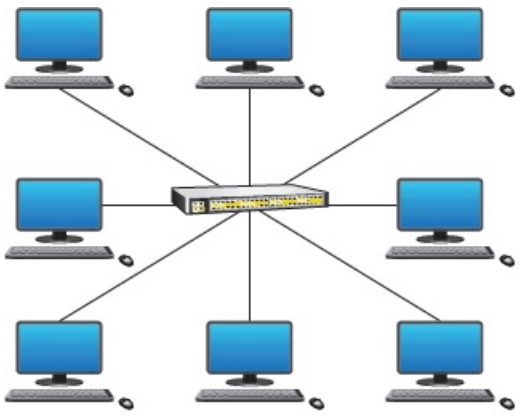
c) Explain these two terms.

i) Web browser

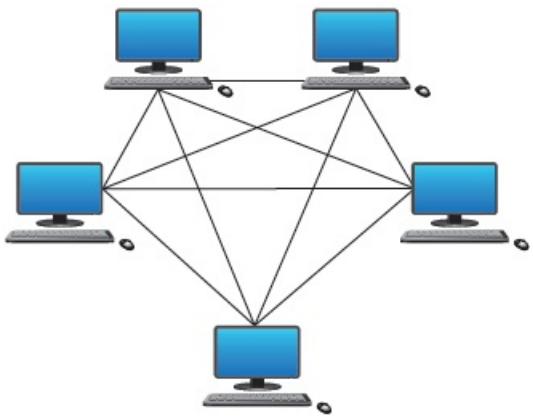
ii) Internet service provider (ISP)

End of chapter questions

1 Star and mesh are two types of network topology that can be used to make a LAN.



Star network



Mesh network

a) i) State **one** benefit and **one** drawback of the star network topology.

[2]

ii) State **one** benefit and **one** drawback of the mesh network topology.

[2]

b) Copy the diagram below and connect each description to either a client-server or peer-to-peer network.

[4]

Type of network	Description
	Connectivity is the most important aspect of this type of network
	Uses separate dedicated servers and specific workstations
Client-server	Has no central storage and doesn't require authentication of users
	Sharing of data is the most important aspect of this type of network
	Has no central server; each workstation shares its files/data with the others
Peer-to-peer	Performance and management issues can occur if the number of workstations exceeds ten
	Once logged in, a user can only access resources that the network manager allows them to use
	More stable system since there is centralised backing up of files

2 a) Conventional telephone calls are made using the public service telephone network (PSTN). The national network uses both copper cables and fibre optic cables.

i) Explain the difference between copper cabling and fibre optic cabling.

[2]

ii) Describe **two** benefits and **two** drawbacks of both types of cabling.

[4]

b) Satellite technology is often used in long distance communications.
Compare the differences between GEO, MEO and LEO satellites.

[3]

c) Some telephones use Bluetooth to connect to the telephone network. Explain what is meant by:

i) the attenuation of a signal

[2]

ii) spread spectrum frequency hopping.

[2]

3 a) Explain the term bit streaming.

[2]

b) A person watches a film streamed from a website on a tablet computer.

i) Give **two** benefits of using bit streaming for this purpose.

[2]

ii) State **two** potential problems of using bit streaming for this purpose.

[2]

c) Explain the terms on-demand bit streaming and real-time bit streaming.

[4]

Cambridge International AS & A Level Computer Science 9608 Paper 11 Q1 November 2015

4 A buffer is 2 MiB in size. The lower limit of the buffer is set at 200 KiB and the higher limit is set at 1.8 MiB.
Data is being streamed at 1.5 Mbps and the media player is taking data at the rate 600 kbps.
You may assume a megabit is 1 048 576 bits and a kilobit is 1024 bits.

a) Explain why the buffer is needed.

[2]

b) i) Calculate the amount of data stored in the buffer after 2 seconds of streaming and playback.
You may assume that the buffer already contains 200 KiB of data.

[4]

ii) By using different time values (such as 4 secs, 6 secs, 8 secs, and so on) determine how long it will take before the buffer reaches its higher limit (1.8 MiB).

[5]

c) Describe how the problem calculated in part b) ii) can be overcome so that a 30-minute video can be watched without frequent pausing of playback.

[2]

5 a) When data is transmitted over a LAN network there is the possible risk of data collision.

i) Explain the term *data collision*.

[2]

ii) Describe how CSMA/CD is able to detect collisions.

[1]

iii) Explain how CSMA/CD can be used to resolve the problem of data collision.

[2]

b) Copy the diagram below and connect each network device to its description.

[5]

Network device	Description
gateway	device that analyses packets of data transmitted from one network to another or analyses data within a single network
switch	network point (node) that connects two networks that use different protocols
hub	device that connects LANs that use the same protocol to allow them to work as a single network
router	device on a network that redirects data received to only those destinations on the LAN network that match the address in the data packet
bridge	device that sends all the received data packets to every device in the network irrespective of any data packet addresses

3 Hardware

In this chapter, you will learn about

- primary storage/memory devices
- secondary storage (including removable devices)
- the benefits and drawbacks of embedded systems
- hardware devices used as input, output and storage
- the differences between RAM, ROM, SRAM, DRAM, PROM and EPROM
- the use of RAM, ROM, SRAM and DRAM in a range of devices
- monitoring and control systems
- the use of logic gates: NOT, AND, OR, NAND, NOR and XOR
- the construction and use of truth tables
- the construction of logic circuits, truth tables and logic expressions from a variety of logic information.

WHAT YOU SHOULD ALREADY KNOW

Try these five questions before you read this chapter.

1 What is the difference between *memory* and *storage*?
2 Why is it necessary to have both internal and external memory/storage devices?
3 Can you recognise the memory/storage devices on the right?
4 What is the difference between *online* and *offline* storage?
5 What is the difference between *data access time* and *data transfer rate* when using memory and storage devices?

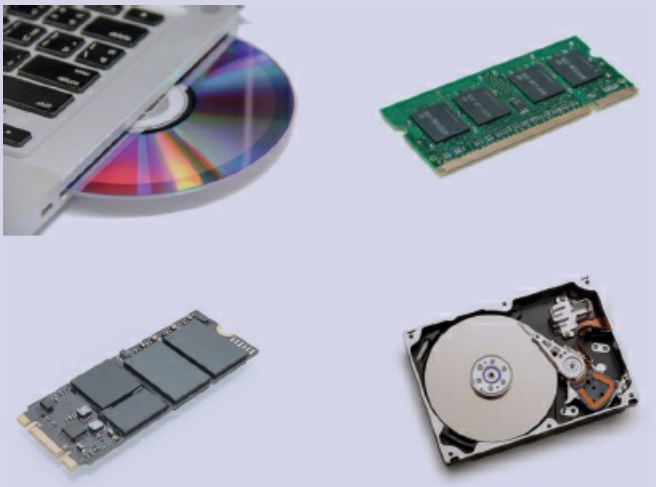


Figure 3.1 Memory/storage devices

3.1 Computers and their components

Key terms

Memory cache – high speed memory external to processor which stores data which the processor will need again.

Random access memory (RAM) – primary memory unit that can be written to and read from.

Read-only memory (ROM) – primary memory unit that can only be read from.

Dynamic RAM (DRAM) – type of RAM chip that needs to be constantly refreshed.

Static RAM (SRAM) – type of RAM chip that uses flip-flops and does not need refreshing.

Refreshed – requirement to charge a component to retain its electronic state.

Programmable ROM (PROM) – type of ROM chip that can be programmed once.

Erasable PROM (EPROM) – type of ROM that can be programmed more than once using ultraviolet (UV) light.

Hard disk drive (HDD) – type of magnetic storage device that uses spinning disks.

Latency – the lag in a system; for example, the time to find a track on a hard disk, which depends on the time taken for the disk to rotate around to its read-write head.

Fragmented – storage of data in non-consecutive sectors; for example, due to editing and deletion of old data.

Removable hard disk drive – portable hard disk drive that is external to the computer; it can be connected via a USB part when required; often used as a device to back up files and data.

Solid state drive (SSD) – storage media with no moving parts that relies on movement of electrons.

Electronically erasable programmable read-only memory (EEPROM) – read-only (ROM) chip that can be modified by the user, which can then be erased and written to repeatedly using pulsed voltages.

Flash memory – a type of EEPROM, particularly suited to use in drives such as SSDs, memory cards and memory sticks.

Optical storage – CDs, DVDs and Blu-ray™ discs that use laser light to read and write data.

Dual layering – used in DVDs; uses two recording layers.

Birefringence – a reading problem with DVDs caused by refraction of laser light into two beams.

Binder 3D printing – 3D printing method that uses a two-stage pass; the first stage uses dry powder and the second stage uses a binding agent.

Direct 3D printing – 3D printing technique where print head moves in the *x*, *y* and *z* directions. Layers of melted material are built up using nozzles like an inkjet printer.

Digital to analogue converter (DAC) – needed to convert digital data into electric currents that can drive motors, actuators and relays, for example.

Analogue to digital converter (ADC) – needed to convert analogue data (read from sensors, for example) into a form understood by a computer.

Organic LED (OLED) – uses movement of electrons between cathode and anode to produce an on-screen image. It generates its own light so no back lighting required.

Screen resolution – number of pixels in the horizontal and vertical directions on a television/computer screen.

Touch screen – screen on which the touch of a finger or stylus allows selection or manipulation of a screen image; they usually use capacitive or resistive technology.

Capacitive – type of touch screen technology based on glass layers forming a capacitor, where fingers touching the screen cause a change in the electric field.

Resistive – type of touch screen technology. When a finger touches the screen, the glass layer touches the plastic layer, completing the circuit and causing a current to flow at that point.

Virtual reality headset – apparatus worn on the head that covers the eyes like a pair of goggles. It gives the user the 'feeling of being there' by immersing them totally in the virtual reality experience.

Sensor – input device that reads physical data from its surroundings.

3.1.1 Types of memory and storage

Computers require some form of memory and storage.

Memory is usually referred to as the internal devices which the computer can access directly. This memory can be the user's workspace, temporary data or data that is key to running the computer.

Storage devices allow users to store applications, data and files. The user's data is stored permanently and they can change it or read it as they wish. Storage needs to be larger than internal memory since the user may wish to store large files (such as music files or photographic images).

Storage devices can also be removable to allow data, for example, to be transferred between computers. Removable devices allow a user to store important data in a different building in case of data loss.

However, all of this has become a lot less important with the advent of technology such as 'data drop' (which uses Bluetooth) and cloud storage.

Internal memory includes components such as registers (which are part of the processor). There is also **memory cache** (which is external to the processor); this is used to store data which the processor will probably need to use again.

Figure 3.2 summarises the types of memory and storage devices covered in this chapter.

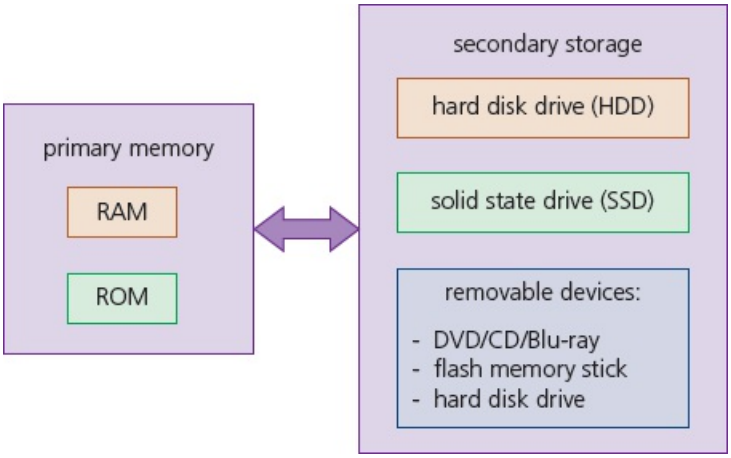


Figure 3.2 Memory and storage devices

Primary memory

Primary memory is the part of computer memory which can be accessed directly from the CPU and, as Figure 3.2 shows, contains the **random access memory (RAM)** and **read-only memory (ROM)** memory chips. Primary memory allows the processor to access applications and services temporarily stored in memory locations. The structure of primary memory is shown in Figure 3.3.

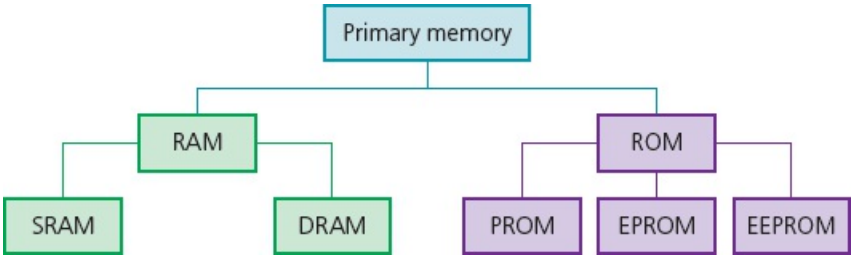


Figure 3.3 Structure of primary memory

All computer systems come with some form of RAM. These memory devices are not really random, it refers to the fact that any memory location can be accessed independent of which memory location was last used. Access time to locate data is much faster in RAM than in secondary devices. RAM can also be

- written to or read from, and the data stored can be changed by the user or by the computer
- used to store data, files, part of an application or part of the operating system ***currently in use***
- volatile (memory contents are lost on powering off the computer).

In general, the larger the RAM, the faster the computer will operate. In reality, RAM never runs out of memory, it continues to operate but just becomes slower and slower as more data is stored. As RAM becomes 'full', the processor has to continually access the secondary data storage devices to overwrite old data on RAM with new data. By increasing the RAM size, the number of times this has to be done is considerably reduced, thus making the computer operate more quickly.

There are currently two types of RAM technology, **dynamic RAM (DRAM)** and **static RAM (SRAM)**.

Dynamic RAM (DRAM)

Each DRAM chip consists of a number of transistors and capacitors. Each of these parts is tiny since a single RAM chip will contain millions of capacitors and transistors.

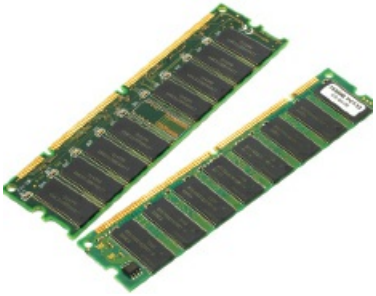


Figure 3.4 Two pieces of dynamic random access memory (DRAM)

- Capacitors hold the bits of information (0 or 1).
- Transistors act like switches; they allow the chip control circuitry to read the capacitor or change the capacitor's value.

This type of RAM needs to be constantly **refreshed** (that is, the capacitor needs to be re-charged every 15 microseconds otherwise it would lose its value). If it is not refreshed, the capacitor's charge will leak away very quickly, leaving every capacitor with the value 0.

DRAMs have a number of advantages over SRAMs. They:

- are much less expensive to manufacture than SRAMs
- consume less power than SRAMs
- have a higher memory capacity than SRAMs.

Static RAM (SRAM)

A major difference between SRAM and DRAM is that SRAM does not need to be constantly refreshed.

It makes use of flip flops (see Chapter 15) which hold each bit of memory.

SRAM is much faster than DRAM when it comes to data access (typically, access time for SRAM is 25 nanoseconds and for DRAM is 60 nanoseconds).

DRAM is the most common type of RAM used in computers, but where absolute speed is essential, for example in the processor's memory cache, SRAM is the preferred technology. Memory cache is a high speed portion of the memory. It is effective because most programs access the same data or instructions many times. By keeping as much of this information as possible in SRAM, the computer avoids having to access the slower DRAM.

Figure 3.5 Static RAM

Table 3.1 summarises the differences between DRAM and SRAM.

DRAM	SRAM
• consists of a number of transistors and capacitors • needs to be constantly refreshed • less expensive to manufacture than SRAM • has a higher memory capacity than SRAM • main memory is constructed from DRAM • consumes more power than SRAM under reasonable levels of access, as it needs to be constantly refreshed	• uses flip-flops to hold each bit of memory • does not need to be constantly refreshed • has a faster data access time than DRAM • processor memory cache makes use of SRAM • if accessed at a high frequency, power usage can exceed that of DRAM

Table 3.1 Differences between DRAM and SRAM

Another form of primary memory is the read-only memory (ROM). This is similar to RAM in that it shares the same random access properties, but it cannot be written to or changed. As the name suggests, ROM is a read-only memory device.

ROMs are

- non-volatile (the contents are not lost after powering off the computer)
- permanent memory devices (the contents cannot be changed)
- often used to store data which the computer needs to access when powering up for the first time for example, the basic input/output system (BIOS).

Table 3.2 summarises the main differences between RAM and ROM.

RAM	ROM
• temporary memory device • volatile memory • can be written to and read from • used to store data, files, programs, part of OS currently in use • can be increased in size to improve operational speed of a computer	• permanent memory device • non-volatile memory device • data stored cannot be altered • sometimes used to store BIOS and other data needed at start up

Table 3.2 Differences between RAM and ROM

PROM and EPROM

A **programmable read-only memory (PROM)** is a type of ROM chip that can be altered once. A PROM is made up of a matrix of fuses. Programming a PROM requires the use of a PROM writer which uses an electric current to alter specific cells by 'burning' fuses in the matrix. Due to the method of programming (writing), a PROM can only be written to once. They are often used in mobile phones and in RFID tags.

An **erasable programmable read-only memory (EPROM)** is different to a PROM because they use floating gate transistors and capacitors rather than fuses. Ultra violet (UV) light is used to program an EPROM through a quartz window. They are used in applications which are under development, such as the programming of new games consoles.

Embedded systems

Embedded systems involve installing microprocessors into devices to enable operations to be controlled in a more efficient way. Devices such as cookers, refrigerators and central heating systems can now all be activated by a web-enabled device (such as a mobile phone or tablet). The time a central heating system switches on or off and the temperature can all be set from an app on a mobile phone from anywhere in the world.

There are pros and cons of devices being controlled in this manner, as shown in Table 3.3.

Pros of embedded systems	Cons of embedded systems
• small in size and therefore easy to fit into devices • relatively low cost to make • usually dedicated to one task,	• difficult to upgrade devices to take advantage of new technology • troubleshooting faults in the device becomes a specialist task

making for simple interfaces and often no requirement of an operating system • consume very little power • very fast reaction to changing input (operate in real time) • with mass production comes reliability	• although the interface can appear to be simple, in reality it can be more confusing (changing the time on a cooker clock can require several steps, for example) • any device that can be accessed over the internet is also open to hackers, viruses, and so on • due to the difficulty in upgrading and fault finding, devices are often just thrown away rather than being repaired (wasteful)

Table 3.3 Pros and cons of controlling devices with embedded systems

EXTENSION ACTIVITY 3A

Describe how ROM and RAM chips could be used in:

a) a microwave oven

b) a refrigerator

c) a remote-controlled model aeroplane (the movement of the aeroplane is controlled by a hand-held device).

Secondary storage devices

Secondary storage includes storage devices that are not directly accessible by the CPU. They are non-volatile devices which allow data to be stored as long as required by the user. This type of storage is much larger than primary memory, but data access time is considerably slower than RAM and ROM. All applications, the operating system, device drivers and general files (for example, documents, photos and music) are stored on secondary storage. The following section discusses the various types of secondary storage that can be found on the majority of computers. Secondary storage devices fall into three categories: magnetic, solid state and optical.

Hard disk drives (HDD)

Hard disk drives (HDD) are still one of the most common methods used to store data on a computer.

Data is stored in a digital format on the magnetic surfaces of the disks (or platters, as they are frequently called). The hard disk drive will have a number of platters which can spin at about 7000 times a second. A number of read-write heads can access all of the surfaces in the disk drive. Normally each platter will have two surfaces which can be used to store the data. These read-write heads can move very quickly – typically they can move from the centre of the disk to the edge of the disk (and back again) 50 times a second.

Data is stored on the surface in sectors and tracks.

A sector on a given track will contain a fixed number of bytes.

Unfortunately, hard disk drives have very slow data access when compared to, for example, RAM. Many applications require the read-write heads to constantly seek for the correct blocks of data; this means a large number of head movements. The effects of **latency** then become very

significant. Latency is defined as the time it takes for a specific block of data on a data track to rotate around to the read-write head.

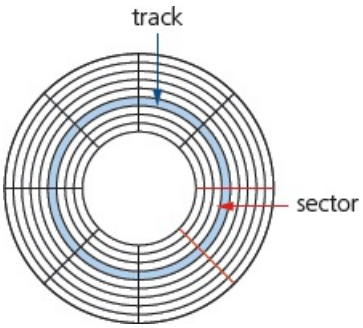


Figure 3.6 Tracks and sectors on a hard disk drive

Users will sometimes notice the effect of latency when they see messages such as, 'Please wait' or, at its worst, 'not responding'.

When a file or data is stored on an HDD, the required number of sectors needed to store the data will be allocated. However, the sectors allocated may not be adjacent to each other. Through time, the HDD will undergo numerous deletions and editing, which leads to sectors becoming increasingly **fragmented**, resulting in a gradual deterioration of the HDD performance (in other words, it takes longer and longer to access data). Defragmentation software can improve on this situation by 'tidying up' the disk sectors.

An HDD is a direct access device; however, data in a given sector will be read sequentially.

Removable hard disk drives are essentially HDDs that are external to the computer and can be connected to the computer using one of the USB ports. In this way, they can be used as back-up devices or as another way of transferring files between computers.

EXTENSION ACTIVITY 3B

The length of a track on each disk in an HDD disk pack becomes much shorter towards the centre of the disk. Find out how manufacturers have overcome this issue with regards to disk data capacity and data access time.

Solid state drives (SSD)

Latency is an issue in HDDs, as discussed earlier. **Solid state drives (SSD)** reduce this issue considerably. They have no moving parts and all data is retrieved at the same rate. They do not rely on magnetic properties. The most common type of solid state storage devices store data by controlling the movement of electrons within NAND chips. The data is stored as 0s and 1s in millions of tiny transistors (at each junction one transistor is called a floating gate and the other is called a control gate) within the chip. This effectively produces a non-volatile rewritable memory.

However, a number of solid state storage devices sometimes use **electronically erasable PROM (EEPROM)** technology. The main difference is the use of NOR chips rather than NAND. This makes them faster in operation but devices using EEPROM are considerably more expensive than those that use NAND technology. EEPROM also allows data to be read or erased in single

bytes at a time. Use of NAND only allows blocks of data to be read or erased. This makes EEPROM technology more useful in certain applications where data needs to be accessed or erased in byte-size chunks.

Because of the cost implications, the majority of solid state storage devices use NAND technology. The two are usually distinguished by the terms **flash memory** (use NAND) and EEPROM (use NOR).

So, what are the main benefits of using an SSD rather than an HDD?

Solid state drives

- are more reliable (no moving parts to go wrong)
- are considerably lighter (which makes them suitable for laptops)
- do not have to 'get up to speed' before they work properly
- have a lower power consumption
- run much cooler than HDDs (both these points again make them very suitable for laptop computers)
- are very thin (because they have no moving parts)
- access data considerably faster.

The main drawback of SSD is the still unknown longevity of the technology. Most solid state storage devices are conservatively rated at only 20 GB write operations per day over a three year period – this is known as SSD endurance. For this reason, SSD technology is not commonly used in servers, for example, where a huge number of write operations take place every day. However, this issue is being addressed by a number of manufacturers to improve the durability of these solid state systems and they are rapidly becoming more common in applications such as servers and cloud storage devices.

Note that it is also not possible to over-write existing data on a flash memory device; it is necessary to first erase the old data and then write the new data at the same location.

Memory sticks/flash memories (also known as pen drives) use solid state technology. They usually connect to the computer through the USB port. Their main advantage is that they are very small, lightweight devices which make them suitable for transferring files between computers. They can also be used as small back-up devices for music or photo files, for example.

Complex or expensive software, such as an expert system, will often use a memory stick as a dongle. The dongle contains additional files which are needed to run the software. Without this dongle, the software will not work properly. It therefore prevents illegal or unauthorised use of the software, and also prevents copying of the software since, without the dongle, it is useless.

Optical media: CDs, DVDs and Blu-ray discs

CDs and DVDS are described as **optical storage devices**. Laser light is used to read data from, and write data onto, the surface of a disk.

Both CDs and DVDs use a thin layer of metal alloy or light-sensitive organic dye to store the data. As shown in Figure 3.7, both systems use a single, spiral track which runs from the centre of the disk to the edge. When a disk spins, the optical head moves to the point where the laser beam 'contacts' the disk surface and follows the spiral track from the centre outwards. As with an HDD, a CD/DVD is divided into sectors allowing direct access of data. Also, as in the case of

an HDD, the outer part of the disk runs faster than the inner part of the disk.

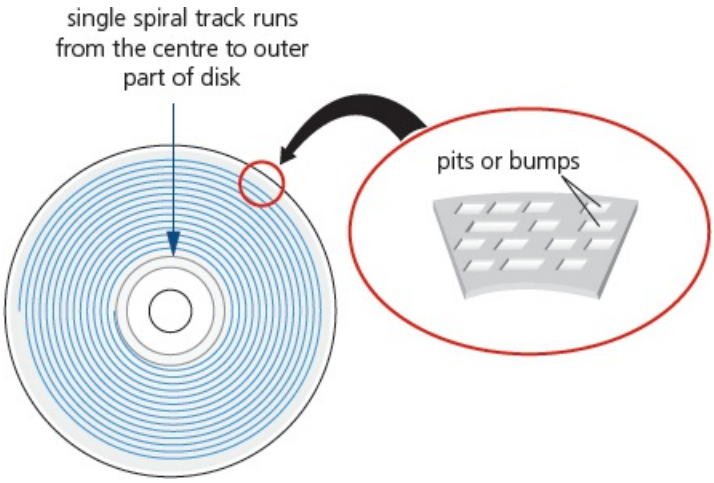


Figure 3.7 CDs and DVDs use a single, spiral track

EXTENSION ACTIVITY 3C

The outer part of an optical disk runs faster than the inner part of the disk. Find out how manufacturers have overcome this issue with regards to disk data capacity and data access time.

The data is stored in ‘pits’ and ‘bumps’ on the spiral track. A red laser is used to read and write the data. CDs and DVDs can be designated R (write once only) or RW (can be written to or read from many times).

DVD technology is slightly different to that used in CDs. One of the main differences is the use of **dual layering** which considerably increases the storage capacity. This means that there are two individual recording layers. Two layers of a standard DVD are joined together with a transparent (polycarbonate) spacer, and a very thin reflector is sandwiched between the two layers. Reading and writing of the second layer is done by a red laser focusing at a fraction of a millimetre difference compared to the first layer.

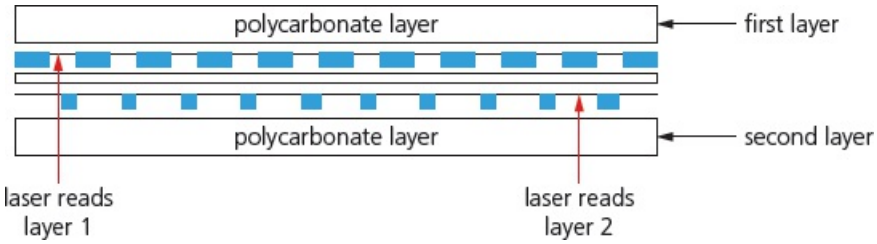


Figure 3.8 Dual layering in a DVD

Standard, single layer DVDs still have a larger storage capacity than CDs because the ‘pit’ size and track width are both smaller. This means that more data can be stored on the DVD surface. DVDs use lasers with a wavelength of 650 nanometres; CDs use lasers with a wavelength of 780 nanometres. The shorter the wavelength of the laser light, the greater the storage capacity of the medium.

- Blu-ray discs are another example of optical storage media. However, they are fundamentally different to DVDs in their construction and in the way they carry out read-write operations.
- Blu-ray uses a blue laser, rather than a red laser, to carry out read and write operations; the wavelength of blue light is only 405 nanometres (compared to 650 nm for red light).
- Using blue laser light means that the 'pits' and 'bumps' can be much smaller; consequently, a Blu-ray can store up to five times more data than a DVD.
- Blu-ray uses a single 1.1 mm thick polycarbonate disk; DVDs use a sandwich of two 0.6 mm thick disks.
- Using two sandwiched layers can cause **birefringence** (light is refracted into two separate beams causing reading errors); because Blu-ray uses only one layer, the discs do not suffer from birefringence.
- Blu-ray discs automatically come with a secure encryption system which helps to prevent piracy and copyright infringement.

Table 3.4 summarises the main differences between CDs, DVDs and Blu-ray.

disk type	laser colour	wavelength of laser light	disk construction	track pitch (distance between tracks)
CD	red	780 nm	single 1.2 mm polycarbonate layer	1.60 µm
DVD	red	650 nm	two 0.6 mm polycarbonate layers	0.74 µm
Blu-ray	blue	405 nm	single 1.1 mm polycarbonate layer	0.30 µm
nm = 10^{-9} metres µm = 10^{-6} metres				

Table 3.4 Main differences between CDs, DVDs and Blu-ray

All these optical storage media are used as back-up systems (for photos, music and multimedia files). This also means that CDs and DVDs can be used to transfer files between computers. Manufacturers sometimes supply their software (such as printer drivers) on CDs and DVDs. When the software is supplied in this way, the disk is usually in a read-only format.

The most common use of DVD and Blu-ray is the supply of movies or games. The memory capacity of CDs is not big enough to store most movies.

EXTENSION ACTIVITY 3D

A recent development is PRAM (parameter RAM) or PCRAM (phase-change RAM) which utilises chalogenide glass. This is glass containing elements such as sulphur, antimony, selenium, germanium or tellurium. Chalogenide compounds used in PRAMs/PCRAMs can be changed between the amorphous (glass-like) state and crystalline state, which changes the optical and electrical properties allowing the storage of data when used as a film on the surface of optical media.

Find out more about this technology and determine whether this could result in the demise of

the current solid state removable devices.

3.1.2 Input and output devices

This section will consider laser printers, inkjet printers, 3D printers, speakers, microphones, screens and sensors.

Laser printers

Laser printers use dry powder ink rather than liquid ink and make use of the properties of static electricity to produce the text and images. Unlike inkjet printers, for example, laser printers print the whole page in one go. Colour laser printers use four toner cartridges – blue, cyan, magenta and black. Although the actual technology is different to monochrome printers, the printing method is similar, but colour dots are used to build up the text and images.

Figure 3.9 A laser printer

When a user wishes to print a document using a laser printer, the following sequence of events takes place.

Stage	Description of what happens
1	data from the document is sent to a printer driver
2	printer driver ensures that the data is in a format that the chosen printer can understand
3	check is made by the printer driver to ensure that the chosen printer is available to print (is it busy? is it off-line? is it out of ink? and so on)
4	data is sent to the printer and stored in a temporary memory known as a printer buffer
5	printing drum given a positive charge. As this drum rotates, a laser beam scans across it removing the positive charge in certain areas, leaving negatively charged areas which exactly match the text/images of the page to be printed
6	drum is coated with positively charged toner (powdered ink). Since the toner is positively charged, it only sticks to the negatively charged parts of the drum
7	negatively charged sheet of paper is rolled over the drum
8	toner on the drum sticks to the paper to produce an exact copy of the page sent to the printer
9	to prevent the paper sticking to the drum, the electric charge on the paper is removed

	after one rotation of the drum
10	the paper goes through a fuser (a set of heated rollers), where the heat melts the ink so that it fixes permanently to the paper
11	a discharge lamp removes all the electric charge from the drum so it is ready to print the next page

Table 3.5 Sequence to print using a laser printer

Inkjet printers

Inkjet printers are made up of

- a print head consisting of nozzles that spray droplets of ink onto the paper to form characters
- an ink cartridge or cartridges; either one cartridge for each colour (blue, yellow and magenta) and a black cartridge, or one single cartridge containing all three colours and black (note: some systems use six colours)
- a stepper motor and belt which moves the print head assembly across the page from side to side
- a paper feed which automatically feeds the printer with pages as they are required.

Figure 3.10 An inkjet printer

The ink droplets are currently produced using one of two technologies: thermal bubble or piezoelectric.

Thermal bubble – tiny resistors create localised heat which makes the ink vaporise. This causes the ink to form a tiny bubble, as the bubble expands some of the ink is ejected from the print head onto the paper. When the bubble collapses, a small vacuum is created which allows fresh ink to be drawn into the print head. This continues until the printing cycle is completed.

Piezoelectric – a crystal is located at the back of the ink reservoir for each nozzle. The crystal is given a tiny electric charge which makes it vibrate. This vibration forces ink to be ejected onto the paper and at the same time more ink is drawn in for further printing.

When a user wishes to print a document using an inkjet printer, the following sequence of events takes place. Whatever technology is used, the basic steps in the printing process are the same.

Stage	Description of what happens
1	data from the document is sent to a printer driver
2	printer driver ensures that the data is in a format that the chosen printer can understand

3	check is made by the printer driver to ensure that the chosen printer is available to print (is it busy? is it off-line? is it out of ink? and so on)
4	data is sent to the printer and stored in a temporary memory known as a printer buffer
5	a sheet of paper is fed into the main body of the printer. A sensor detects whether paper is available in the paper feed tray – if it is out of paper (or the paper is jammed), an error message is sent back to the computer
6	as the sheet of paper is fed through the printer, the print head moves from side to side across the paper printing the text or image. The four ink colours are sprayed in their exact amounts to produce the desired final colour
7	at the end of each full pass of the print head, the paper is advanced very slightly to allow the next line to be printed. This continues until the whole page has been printed
8	if there is more data in the printer buffer, then the whole process from stage 5 is repeated until the buffer is empty
9	once the printer buffer is empty, the printer sends an interrupt to the processor in the computer, which is a request for more data to be sent to the printer. The process continues until the whole of the document has been printed

Table 3.6 Sequence to print using a laser printer

3D printers

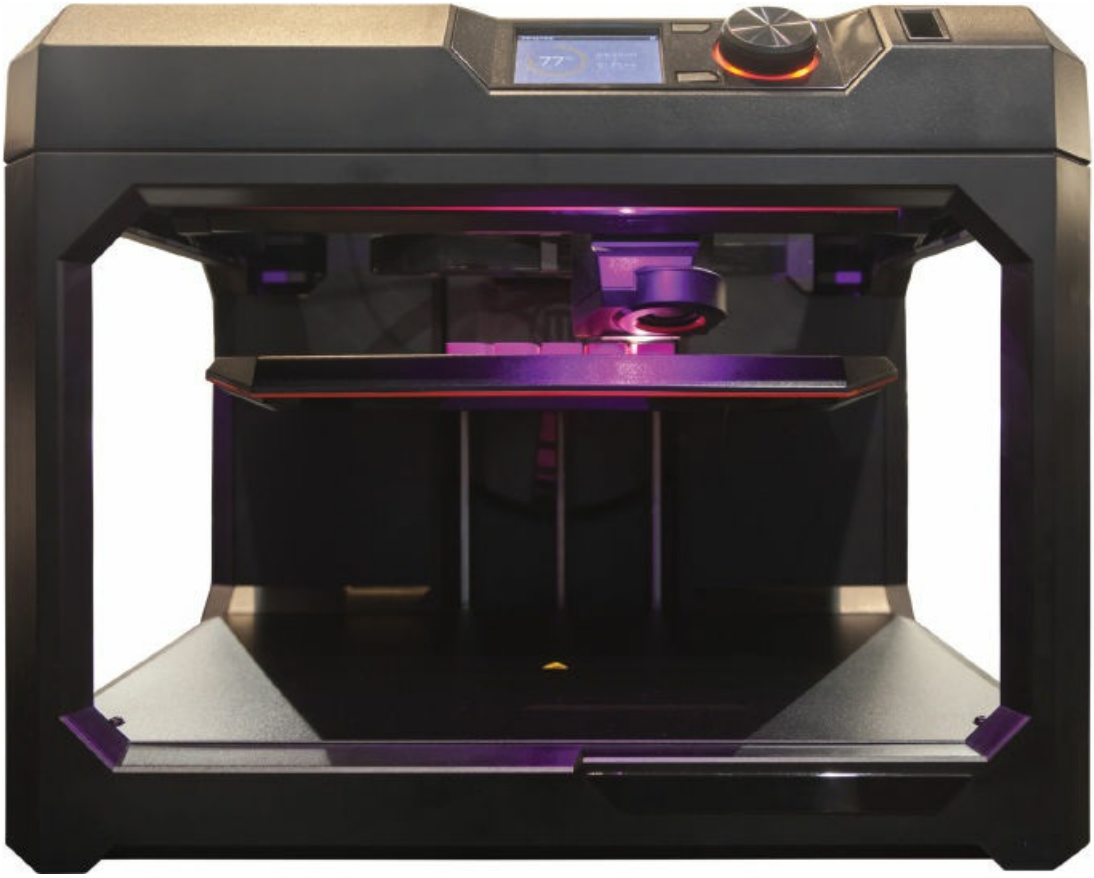


Figure 3.11 A 3D printer

3D printers are used to produce working, solid objects. They are primarily based on inkjet and laser printer technology. The solid object is built up layer by layer using materials such as powdered resin, powdered metal, paper or ceramic.

The artificial bone framework in Figure 3.12 was made from many layers (100 µm thick) of powered metal using a technology known as **binder 3D printing**.

Figure 3.12 Artificial bone framework made using an industrial 3D printer

Various types of 3D printers exist; they range from the size of a microwave oven up to the size of a small car.

3D printers use additive manufacturing (the object is built up layer by layer); this is in contrast to the more traditional method of subtractive manufacturing (removal of material to make the object). For example, making a statue using a 3D printer would involve building it up layer by

layer using powdered stone until the final object was formed. The subtractive method would involve carving the statue out of solid stone (removing the stone not required) until the final item was produced. Similarly, CNC machining removes metal to form an object; 3D printing would produce the same item by building up the object from layers of powdered metal.

Direct 3D printing uses inkjet technology; a print head can move left to right as in a normal printer. However, the print head can also move up and down to build up the layers of an object.

Binder 3D printing is similar to direct 3D printing. However, this method uses two passes for each of the layers; the first pass sprays dry powder and then on the second pass a binder (a type of glue) is sprayed to form a solid layer.

Newer technologies use lasers and UV light to harden liquid polymers; this further increases the diversity of products which can be made.

Speakers and microphones

Speakers

Digitised sound stored in a file on a computer can be converted into sound as follows:

- The digital data is first passed through a digital to analogue converter (DAC) where it is converted into an electric current.
- This is then passed through an amplifier (since the current generated by the DAC will be small) to create a current large enough to drive a loudspeaker.
- This electric current is then fed to a loudspeaker where it is converted into sound.

The following schematic shows how this is done.

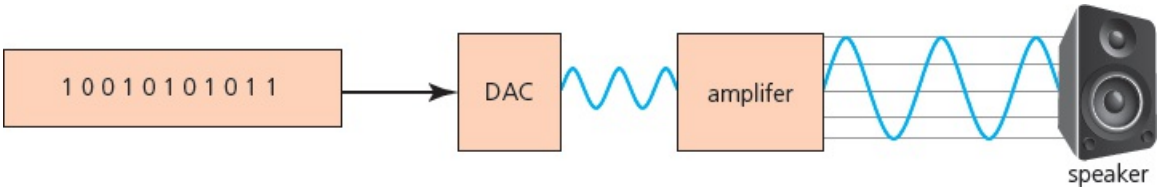


Figure 3.13 Digital to analogue conversion

As Figure 3.13 shows, if the sound is stored in a computer file, it must first pass through a **digital to analogue converter (DAC)** to convert the digital data into an electric current which can be used to drive the loudspeaker. Figure 3.14 shows how a loudspeaker can convert electric signals into sound waves.

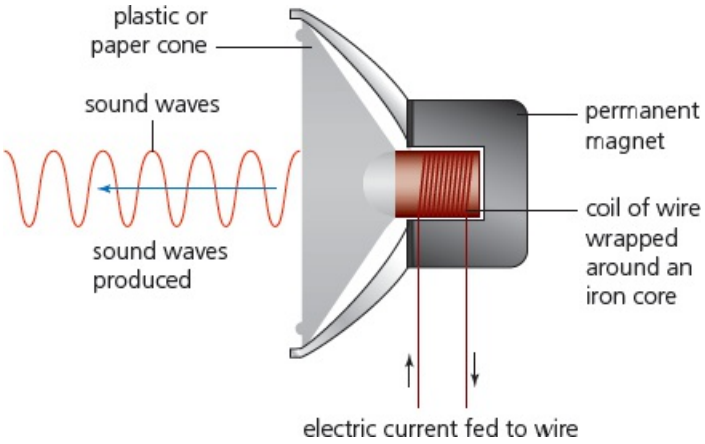


Figure 3.14 Diagram showing how a loudspeaker works

- When an electric current flows through a coil of wire that is wrapped around an iron core, the core becomes a temporary electromagnet; a permanent magnet is also positioned very close to this electromagnet.
- As the electric current through the coil of wire varies, the induced magnetic field in the iron core also varies. This causes the iron core to be attracted towards the permanent magnet and as the current varies this will cause the iron core to vibrate.
- Since the iron core is attached to a cone (made from paper or thin synthetic material), this causes the cone to vibrate, producing sound.

The rate at which the DAC can translate the digital output into analogue voltages is known as the sampling rate. If the DAC is a 16-bit device, then it can accept numbers between +32 767 (2^{16} – 1) and –32 768 (2^{16}); the digital value containing all zeros is ignored.

Microphones

Microphones are either built into the computer or are external devices connected through the USB port or through wireless connectivity.

Figure 3.15 shows how a microphone can convert sound waves into an electric current. The current produced can either be stored as sound (on, for example, a CD), amplified and sent to a loudspeaker, or sent to a computer for storage.

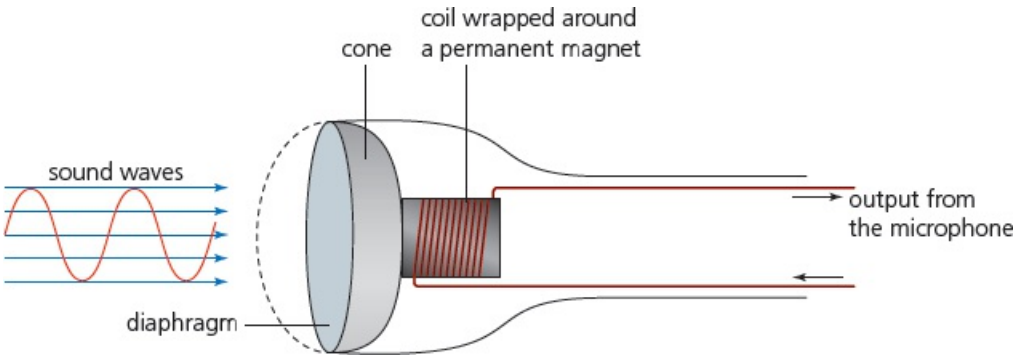


Figure 3.15 Diagram of how a microphone works

- When sound is created, it causes the air to vibrate.

- When a diaphragm in the microphone picks up the air vibrations, the diaphragm also begins to vibrate.
- A copper coil is wrapped around a permanent magnet and the coil is connected to the diaphragm using a cone. As the diaphragm vibrates, the cone moves in and out causing the copper coil to move backwards and forwards.
- This forwards and backwards motion causes the magnetic field around the permanent magnet to be disturbed, inducing an electric current.
- The electric current is then either amplified or sent to a recording device. The electric current is analogue in nature.

The electric current output from the microphone can also be sent to a computer where a sound card converts the current into a digital signal which can then be stored in the computer. The following diagram shows what happens when the word ‘hut’ is picked up by a microphone and is converted into digital values:

Look at Figure 3.16. The word ‘hut’ (in the form of a sound wave) has been picked up by a microphone; this is then converted using an **analogue to digital converter (ADC)** into digital values which can then be stored in a computer or manipulated as required using appropriate software.

Figure 3.16 Analogue to digital conversion

Screens

Screens are used to show the output from a computer. Modern screens use an LCD, backlit with LEDs or the newer **organic light emitting diode (OLED)** technology.

Figure 3.17 shows a simplified form of how OLED technology works.

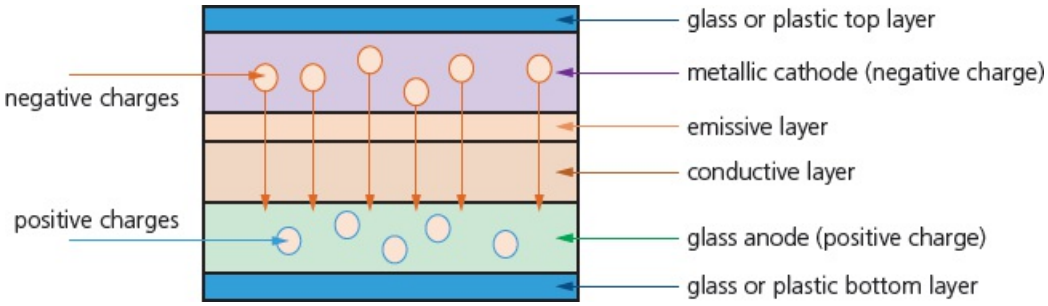


Figure 3.17 Simplified form of how OLED technology works

OLEDs use organic materials (made up of carbon compounds) to create flexible semiconductors. Organic films are sandwiched between two charged electrodes (one is a metallic cathode and the other a glass anode). When an electric field is applied to the electrodes, they give off light. This means that no form of back lighting is required. This allows for very thin screens. It also means

that there is no longer a need to use LCD technology, since OLED is a self-contained system.

Screen displays are based on the pixel (the smallest picture element) concept where each screen pixel is made up of three sub-pixels, which are red, green and blue. By varying the intensity of the three sub-pixels, it is possible to generate millions of colours. The greater the number of pixels on a screen, the greater is the **screen resolution** (the number of pixels which can be viewed horizontally and vertically on screen; for example, 1680 × 1080 pixels). LCD and OLED screens use this type of pixel matrix to make up the picture.

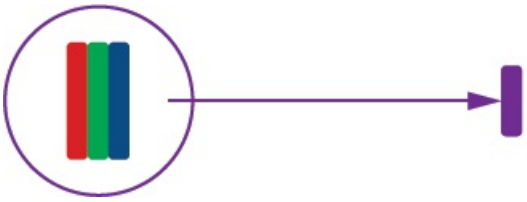


Figure 3.18 The pixel matrix

Touch screens (which act as both input and output devices) also make use of LCD and OLED technology. They are particularly used in mobile phones and tablets.

We shall now consider LCD **capacitive** and **resistive** touch screen technologies.

Capacitive

- Made up of many layers of glass that act like a capacitor creating electric fields between the glass plates in layers.
- When the top glass layer is touched, the electric current changes and the coordinates where the screen was touched are determined by an on board microprocessor.

Benefits

- Medium cost technology.
- Screen visibility is good even in strong sunlight.
- Permits multi-touch capability.
- Screen is very durable; it takes a major impact to break the glass.

Drawbacks

- Only allows use of bare fingers as the form of input; although the latest screens permit the use of a special stylus to be used.

Resistive

- Makes use of an upper layer of polyester (a form of plastic) and a bottom layer of glass.
- When the top polyester layer is touched, the top layer and bottom layer complete a circuit.
- Signals are then sent out, which are interpreted by a microprocessor and the calculations determine the coordinates of where the screen was touched.

Benefits

- Relatively inexpensive technology.
- Possible to use bare fingers, gloved fingers or stylus to carry out an input operation.

Drawbacks

- Screen visibility is poor in strong sunlight.
- Does not permit multi-touch capability.
- Screen durability is only fair; it is vulnerable to scratches and the screen wears out through time.

Virtual headsets

Virtual reality has now been around for many years and has many applications. For example, it is possible to ‘walk around’ inside dangerous areas – such as a nuclear power plant – without actually being there.

It allows engineers to plan modifications or repairs to a plant in complete safety and to try out different scenarios first before implementing them. One of the devices used is a **virtual reality headset** which gives the engineer the feeling of being there. We will now describe how these devices work.

- Video is sent from a computer to the headset (either using an HDMI cable or a smartphone fitted into the headset).
- Two feeds are sent to an LCD/OLED display (sometimes two screens are used, one for the left side of the image and one for the right side of the image); lenses placed between the eyes and the screen allow for focusing and reshaping of the image/video for each eye, thus giving a 3D effect and adding to the realism.
- Most headsets use 110° field of view which is enough to give a pseudo 360° surround image/video.
- A frame rate of 60 to 120 images per second is used to give a true/realistic image.
- As the user moves their head (up and down or left to right), a series of sensors and/or LEDs measure this movement, which allows the image/video on the screen to react to the user’s head movements (sensors are usually gyroscopic or accelerometers; LEDs are used in conjunction with mini cameras to further monitor head movements).
- Headsets also use binaural sound (surround sound) so that the speaker output appears to come from behind, from the side or from a distance, giving very realistic 3D sound.
- Some headsets also use infrared sensors to monitor eye movement (in addition to head movement), which allows the depth of field on the screen to be more realistic; an example of this is to make objects in the foreground appear fuzzy when the user’s eyes indicate they are looking into the distance (and vice versa).

Sensors

Sensors are input devices which read or measure physical properties, such as temperature, pressure, acidity, and so on.

Real data is analogue in nature – this means it is constantly changing and does not have a discrete value. Analogue data usually requires some form of interpretation, for example, the temperature shown on a mercury thermometer requires the user to look at the height of the mercury to work out the temperature. The temperature, therefore, can have an infinite number of values depending on the precision of how the height of the mercury is measured. Equally, an analogue clock requires the user to look at the hands on the clock face. The area swept out by the hands allows the number of hours and minutes to be interpreted. There are many other examples.

Computers cannot make any sense of these physical quantities and the data needs to be converted into a digital format. This is usually achieved by an analogue to digital converter (ADC). This device converts physical values into discrete digital values.

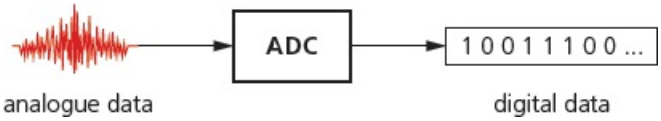


Figure 3.19 Converting analogue data into digital data

When a computer is used to control devices, such as a motor or a valve, it is often necessary to use a digital to analogue converter (DAC), since these devices need analogue data to operate in many cases. Frequently, an actuator is used in these control applications. Although these are technically output devices, they are mentioned here since they are an integral part of the control system. An actuator is an electromechanical device such as a relay, solenoid or motor. Note that a solenoid is an example of a digital actuator as part of the device is connected to a computer which opens and closes a circuit as required. When energized, the solenoid may operate a plunger or armature to control, for example, a fuel injection system. Other actuators, such as motors and valves, may require a DAC so that they receive an electric current rather than a simple digital signal direct from the computer.

Notice the importance of (positive) feedback, which is where the output from the system can affect the next input. This is due to the fact that sensor readings may cause the microprocessor to alter a valve or a motor, for example, which will then change the next reading taken by the sensor. So the output from the microprocessor will impact on the next input received as it attempts to bring the system within the desired parameters.

Table 3.7 shows a number of common sensors and examples of their applications.

Sensor	Example applications
temperature	• control a central heating system • control/monitor a chemical process • control/monitor temperature in a greenhouse
moisture/humidity	• control/monitor moisture/humidity levels in soil/air in a greenhouse • monitor dampness levels in an industrial application (for example, monitor moisture in a paint spray booth in a car factory)
light	• switch street lighting on at night and off during the day • monitor/control light levels in a greenhouse • switch on car headlights when it gets dark
infrared/motion	• turn on windscreen wipers on a car when it rains • detect an intruder in a burglar alarm system • count people entering or leaving a building
pressure	• detect intruders in a burglar alarm system

	• check weight (such as the weight of a vehicle) • monitor/control a process where gas pressure is important
acoustic/sound	• pick up noise levels (such as footsteps or breaking glass) in a burglar alarm system • detect noise of liquids dripping from a pipe
gas (such as O_2 or CO_2)	• monitor pollution levels in a river or air • measure O_2 and CO_2 levels in a greenhouse • check for CO_2 or NO_2 leaks in a power station
pH	• monitor/control acidity/alkalinity levels in soil • monitor pollution in rivers
magnetic field	• detect changes in in cell phones, CD players, and so on • used in anti-lock braking systems in motor vehicles

Table 3.7 Common sensors and examples of applications

Sensors are used in both monitoring and control applications. There is a subtle difference between how these two methods work. The flowchart (Figure 3.21 overleaf) shows a simplification of the process.

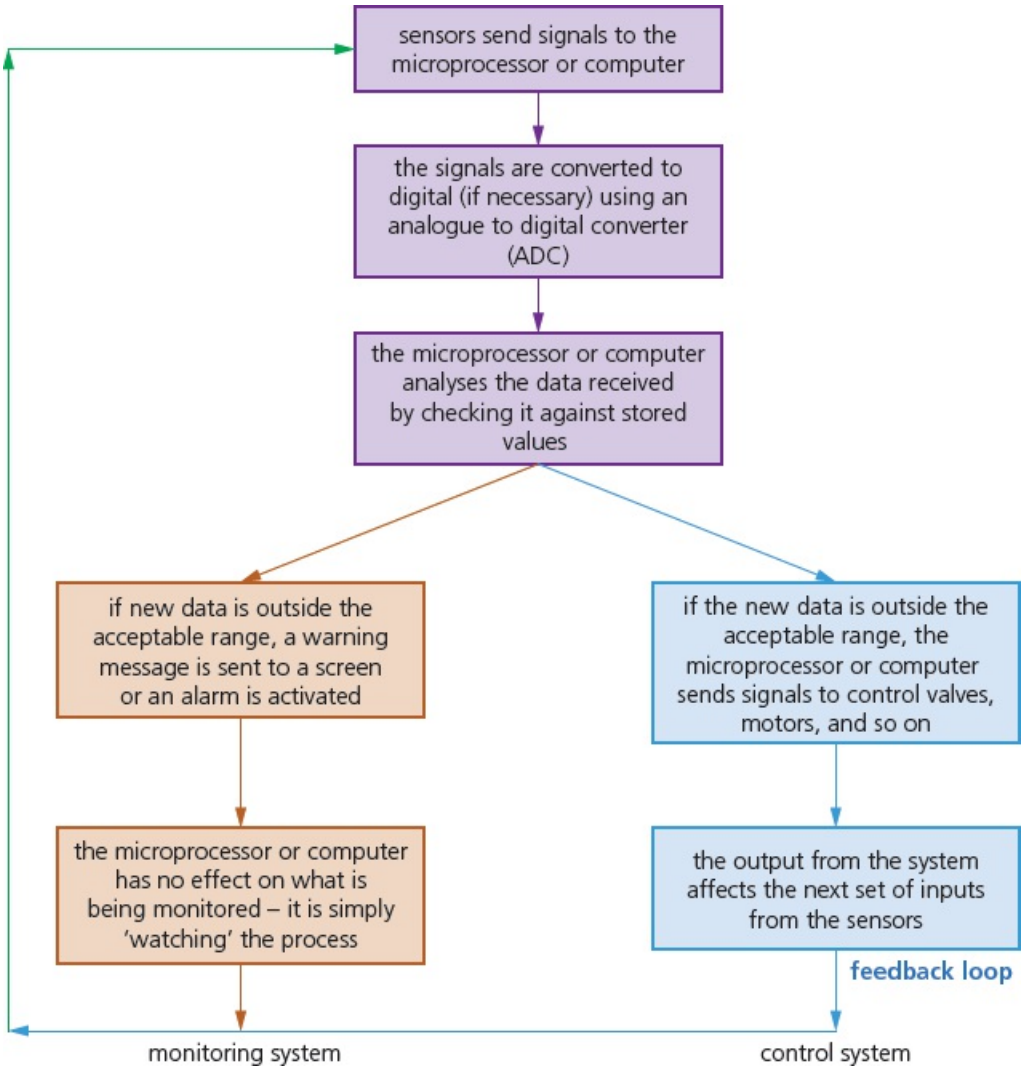


Figure 3.20 Sensors for monitoring and controlling systems

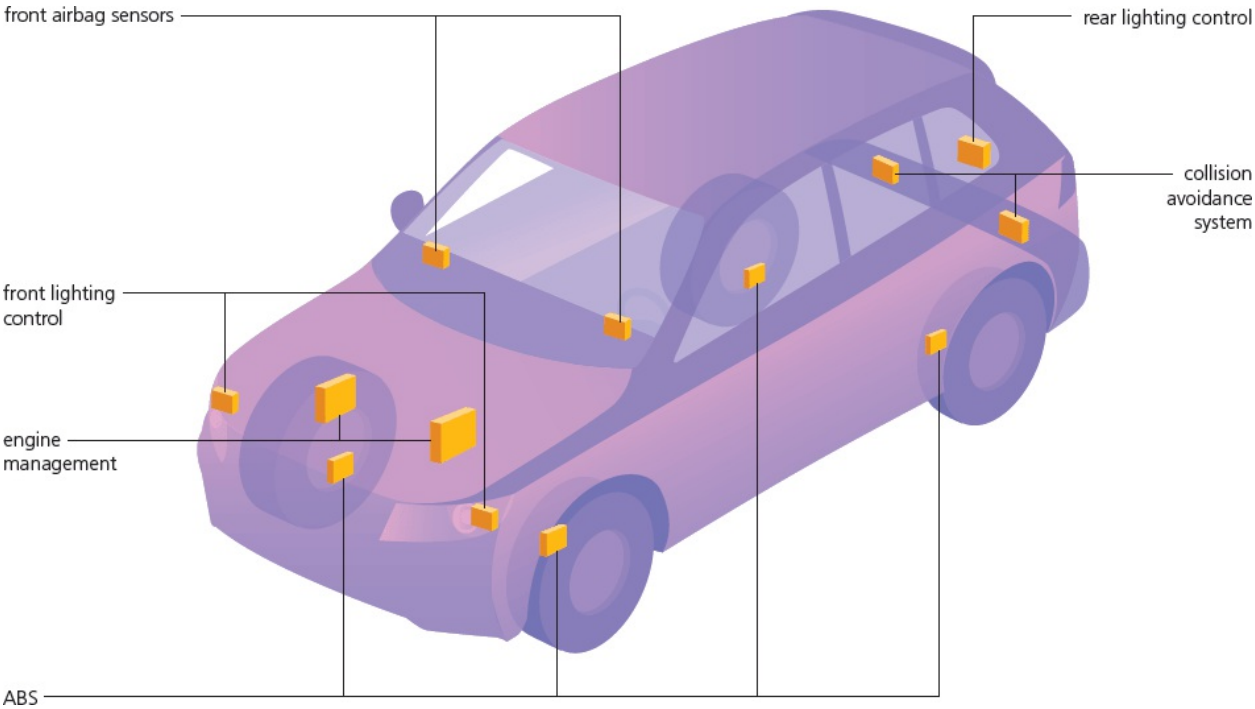


Figure 3.21 Sensors on a typical modern car

Table 3.8 shows some examples of monitoring and control applications of sensors.

Examples of monitoring	Examples of control
• monitoring a patient in a hospital for vital signs such as heart rate, temperature, and so on • checking for intruders in a burglar alarm system • checking the temperature levels in a car engine • monitoring pollution levels in a river	• turning street lights on at night and turning them off again during daylight • controlling the temperature in a central heating/air conditioning system • controlling the traffic lights at a road junction • operating anti-lock brakes on a car when necessary • controlling the environment in a greenhouse

Table 3.8 Examples of monitoring and control applications of sensors

One of the most common uses of sensors in modern times is in the monitoring and control of a number of functions in motor vehicles and aeroplanes. Look at Figure 3.21 showing a typical modern car and its many sensors used to control or monitor several functions.

Below is an in-depth look at just one of the sensor systems labelled on Figure 3.21.

Anti-lock braking systems (on cars)

Anti-lock braking systems (ABS) on cars use magnetic field sensors to stop the wheels locking up on the car if the brakes have been applied too sharply.

- When one of the car wheels rotates too slowly (it is locking up), a magnetic field sensor sends data to a microprocessor.
- The microprocessor checks the rotation speed of the other three wheels.
- If they are different (rotating faster), the microprocessor sends a signal to the braking system and the braking pressure to the affected wheel is reduced.
- The wheel's rotational speed is then increased to match the other wheels.
- The checking of the rotational speed using these magnetic field sensors is done several times a second and the braking pressure to all the wheels can be constantly changing to prevent any of the wheels locking up under heavy braking.
- This is felt as a 'judder' on the brake pedal as the braking system is constantly switched off and on to equalise the rotational speed of all four wheels.
- If one of the wheels is rotating too quickly, braking pressure is increased to that wheel until it matches the other three.

ACTIVITY 3A

1 a) i) Describe **three** differences between RAM and ROM.

ii) Compare the relative advantages and disadvantages of SRAM and DRAM. Include examples of where each type of memory would be used in a computer.

b) Secondary storage can be magnetic, optical or solid state. Describe **two** features of each type of storage which differentiates it from the other two types.

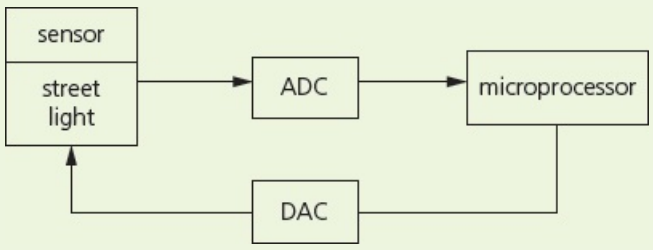
2 a) Explain the main differences in operation of a laser printer compared with an inkjet printer.

b) i) Name **one** application of a laser printer and **one** application of an inkjet printer.

ii) For each of your named applications in part b) i), give a reason why the chosen printer is the most suitable.

3 An art gallery took several photographs of a valuable, fragile painting. The images were sent to a computer where they were processed by a 3D printing application. A 3D printout of the painting was produced showing the texture of the oil paint, canvas and any flaws in the painting.
Give reasons why the art gallery would wish to make this 3D replica.

4 The following diagram shows a schematic of a microprocessor-controlled street lighting system.



The microprocessor is used to control the operation of the street lamp. The lamp is fitted with a light sensor which constantly sends data to the microprocessor. The data value from

the sensor changes according to whether it is sunny, cloudy, raining, night time, and so on.

Describe how the microprocessor would be used to automatically switch on the light at night and switch it off again when it becomes light. Include a feature to stop the light constantly flickering on and off when it becomes overcast or cars go past with full headlights at night.

EXTENSION ACTIVITY 3E

1 Look at this simplified diagram of a keyboard; the letter H has been pressed. Explain:

a) how pressing the letter H has been recognised by the computer

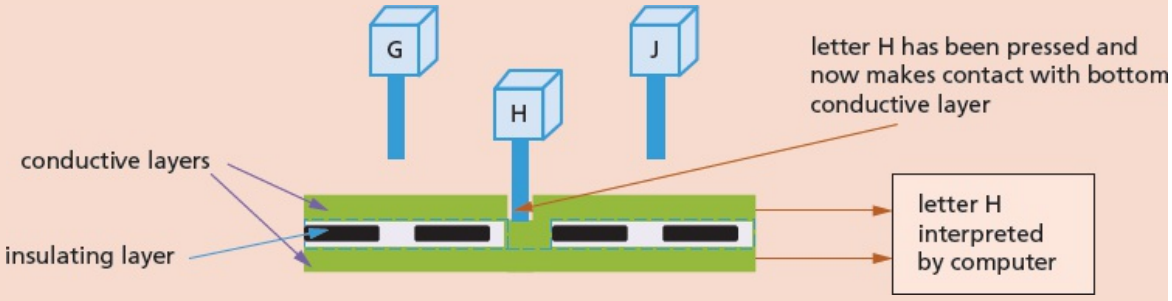
b) how the computer manages the very slow process of inputting data from a keyboard.

2 a) Describe how these types of pointing devices work.

i) Mechanical mouse

ii) Optical mouse

b) Connectivity between mouse and computer can be through USB cable or wireless. Explain these two types of connectivity.



EXTENSION ACTIVITY 3F

Another new screen technology is known as **quantum LED (QLED)**, which is in direct competition with **organic (LED)**. Look at this statement:

'QLED televisions are simply LED televisions that use quantum dots to enhance their overall performance in key picture quality areas.'

Find out the main differences between QLED and OLED technologies.

3.2 Logic gates and logic circuits

Key terms

Logic gates – electronic circuits which rely on ‘on/off’ logic. The most common ones are NOT, AND, OR, NAND, NOR and XOR.

Logic circuit – formed from a combination of logic gates and designed to carry out a particular task. The output from a logic circuit will be 0 or 1.

Truth table – a method of checking the output from a logic circuit. They use all the possible binary input combinations depending on the number of inputs; for example, two inputs have 2^2 (4) possible binary combinations, three inputs will have 2^3 (8) possible binary combinations, and so on.

Boolean algebra – a form of algebra linked to logic circuits and based on TRUE and FALSE.

3.2.1 Logic gates

Electronic circuits in computers, many memories and controlling devices are made up of thousands of **logic gates**. Logic gates take binary inputs and produce a binary output. Several logic gates combined together form a **logic circuit** and these circuits are designed to carry out a specific function. The checking of the output from a logic gate or logic circuit can be done using a **truth table**.

This section will consider the function and role of logic gates, logic circuits and truth tables. A number of possible applications of logic circuits will also be considered. A reference to **Boolean algebra** will be made throughout this section, although this is covered in more depth in Chapter 15.

Six different logic gates will be considered in this section.

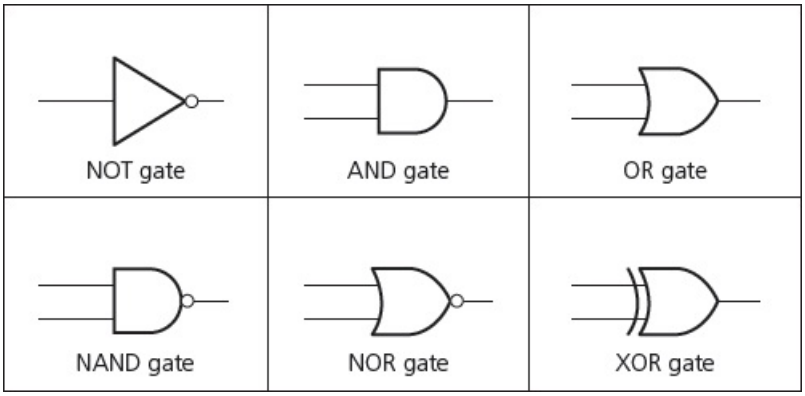


Figure 3.22 Six types of logic gate

3.2.2 Truth tables

Truth tables are used to trace the output from a logic gate or logic circuit. The NOT gate is the only logic gate with one input; the other five gates have two inputs. When constructing truth tables, all possible combinations of 1s and 0s which can be input are considered. For the NOT gate (one input) there are only 2^1 (2) possible binary combinations. For all other gates (two inputs), there are 2^2 (4) possible binary combinations.

For logic circuits, the number of inputs can be more than 2; for example, three inputs give a possible 2^3 (8) binary combinations. And for four inputs, the number of possible binary combinations is 2^4 (16). It is clear that the number of possible binary combinations is a multiple of the number 2 in every case. Table 3.9 summarises this.

Inputs	
A	B
0	0
0	1
1	0
1	1

Inputs		
A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Inputs			
A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Table 3.9

3.2.3 The function of the six logic gates

NOT gate

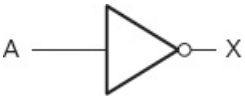


Figure 3.23 NOT gate

Description

The output, X, is 1 if the input A is NOT 1

How to write this

X = NOT A (logic notation)

X = Ā (Boolean algebra)

Truth table

Input	Output
A	**X**
0	1
1	0

Table 3.10

AND gate

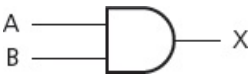


Figure 3.24 AND gate

Description

The output, X, is 1 if input A is 1 and input B is 1

How to write this

X = A AND B (logic notation)

X = A.B (Boolean algebra)

Truth table

Inputs		Output
A	**B**	**X**

0	0	0
0	1	0
1	0	0
1	1	1

Table 3.11

OR gate

Figure 3.25 OR gate

Description

The output, X, is 1 if input A is 1 or input B is 1

How to write this

X = A OR B (logic notation)

X = A + B (Boolean algebra)

Truth table

Inputs		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.12

NAND gate (NOT AND)

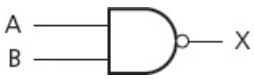


Figure 3.26 NAND gate

Description

The output, X, is 1 if input A is NOT 1 or input B is NOT 1

How to write this

X = A NAND B (logic notation)

$X = \overline{A.B}$ (Boolean algebra)

Truth table

Inputs		Output
A	**B**	**X**
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.13

NOR gate (NOT OR)

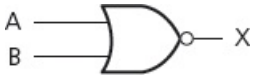


Figure 3.27 NOR gate

Description

The output, X, is 1 if:

input A is NOT 1 and input B is NOT 1

How to write this

X = A NOR B (logic notation)

$X = \overline{A + B}$ (Boolean algebra)

Truth table

Inputs		Output
A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	0

Table 3.14

XOR gate

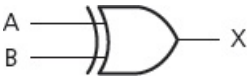


Figure 3.28 XOR gate

Description

The output, X, is 1 if (input A is 1 AND input B is NOT 1) OR (input A is NOT 1 AND input B is 1)

How to write this

X = A XOR B (logic notation)

X = (A.$\overline{B}$) + ($\bar{A}$.B) (Boolean algebra)

(Note: this is sometimes written as: (A + B) . $\overline{A.B}$)

Truth table

Inputs		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.15

EXTENSION ACTIVITY 3G

Using truth tables show that X = (A.$\overline{B}$) + ($\bar{A}$.B) and X = (A + B) . $\overline{A.B}$ both represent the XOR logic gate.

You will notice, in the Boolean algebra, three new symbols.

- A dot (.) represents the AND operation (it can be written as $\wedge$).
- A plus sign (+) represents the OR operation (it can be written as $\vee$).
- A dash above a letter (for example, $\bar{A}$) represents the NOT operation.

3.2.4 Logic circuits

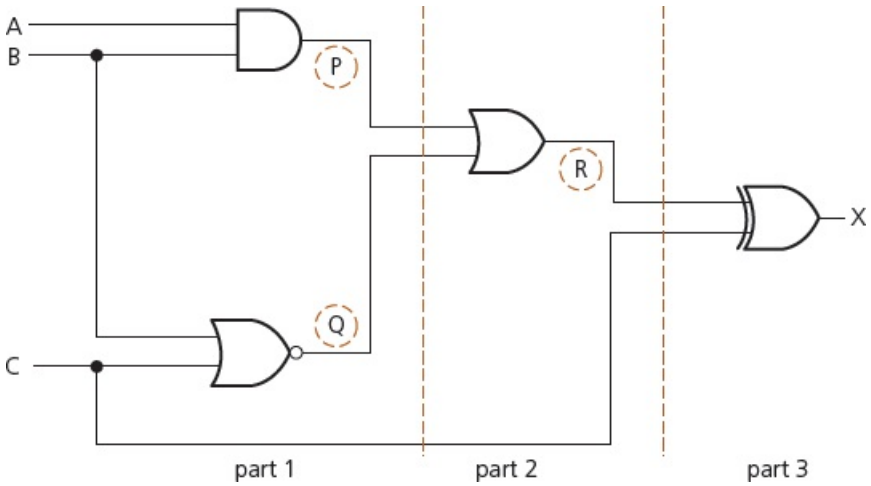
When logic gates are combined to carry out a particular function, such as controlling a robot, they form a **logic circuit**.

The output from the logic circuit is checked using a truth table. The following three examples show how to:

- produce a truth table
- design a logic circuit from a given logic statement/Boolean algebra
- design a logic circuit to carry out an actual safety function.

Example 3.1

Produce a truth table for the following logic circuit (note the use of • at junctions):



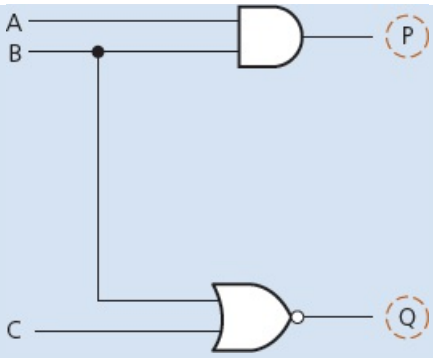
Solution

There are three inputs to this logic circuit; therefore, there will be eight possible binary values which can be input.

To show step-wise how the truth table is produced, the logic circuit has been split up into three parts and intermediate values are shown as P, Q and R.

Part 1

This is the first part of the logic circuit; the first task is to find the intermediate values P and Q.

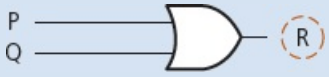


The value of P is found from the AND gate where the inputs are A and B. The value of Q is found from the NOR gate where the inputs are B and C. An intermediate truth table is produced:

Inputs			Outputs	
A	B	C	P	Q
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	0

Part 2

The second part of the logic circuit has P and Q as inputs and the intermediate output, R.



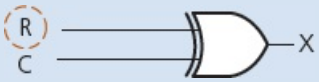
This produces the following intermediate truth table (Note: even though there are only two inputs to the logic gate, we have generated eight binary values in Part 1 and these must all be used in this second truth table).

Inputs		**Output**
P	**Q**	**R**
0	1	1
0	0	0
0	0	0
0	0	0

0	1	1
0	0	0
1	0	1
1	0	1

Part 3

The final part of the logic circuit has R and C as inputs and the final output, X.



This gives the third intermediate truth table.

Putting all three intermediate truth tables together produces the final truth table which represents the original logic circuit.

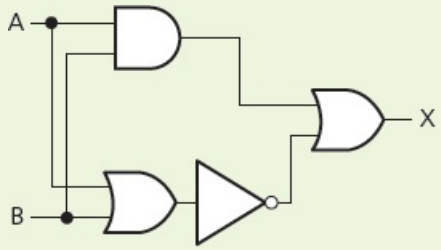
Inputs		**Output**
R	**C**	**X**
1	0	1
0	1	1
0	0	0
0	1	1
1	0	1
0	1	1
1	0	1
1	1	0

Inputs			Intermediate values			Output
A	B	C	P	Q	R	X
0	0	0	*0*	*1*	*1*	1
0	0	1	*0*	*0*	*0*	1
0	1	0	*0*	*0*	*0*	0
0	1	1	*0*	*0*	*0*	1
1	0	0	*0*	*1*	*1*	1
1	0	1	*0*	*0*	*0*	1
1	1	0	*1*	*0*	*1*	1
1	1	1	*1*	*0*	*1*	0

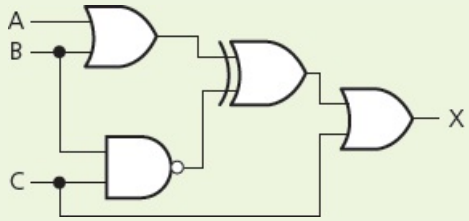
ACTIVITY 3B

Produce truth tables for each of the following logic circuits. You are advised to split them up into intermediate parts to help eliminate errors.

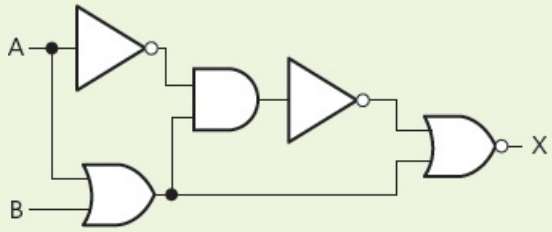
a)



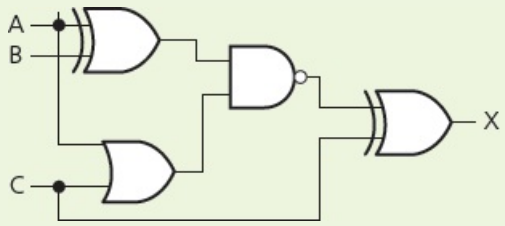
b)



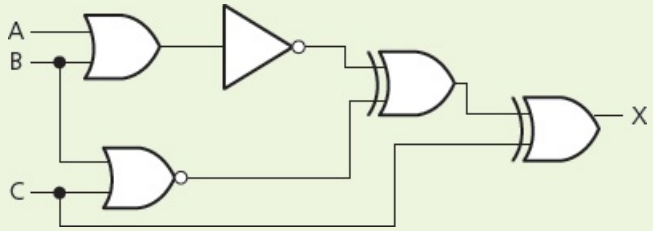
c)



d)



e)



Example 3.2

A safety system uses three inputs to a logic circuit. An alarm, X, sounds if input A represents ON and input B represents OFF, or if input B represents ON and input C represents OFF.

Produce a logic circuit and truth table to show the conditions which cause the output X to be 1.

Solution

The first thing to do is to write down the logic statement representing the scenario in this example. To do this, it is necessary to recall that ON = 1 and OFF = 0 and also that 0 is considered to be NOT 1.

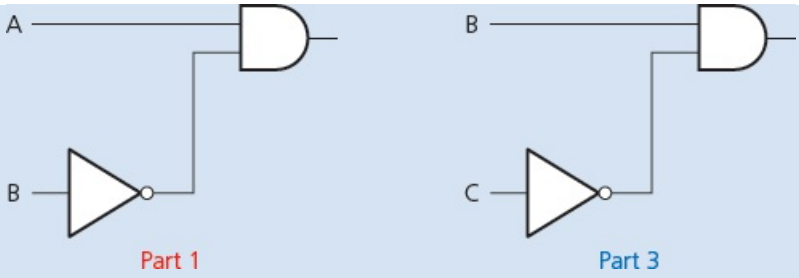
So, we get the following logic statement:

X = 1 if	(A = 1 AND B = NOT 1)	OR	(B = 1 AND C = NOT 1)
	this equates to A is ON and B is OFF	the two parts are connected by the OR gate	this equates to B is ON AND C is OFF
	Part 1	Part 2	Part 3

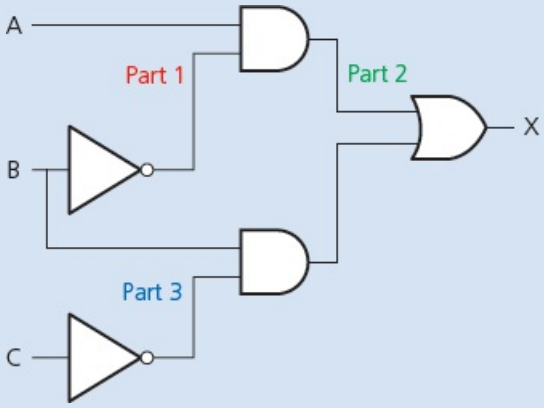
This statement can also be written in Boolean algebra as:

$$(A.\overline{B}) + (B.\overline{C})$$

The logic circuit is made up of three parts as shown in the logic statement. We will produce the logic gate for the Part 1 and Part 3, then join both parts together with the OR gate.



Now, combining both parts with Part 2 (the OR gate) gives us:



There are two ways to produce the truth table.

- Trace through the logic circuit using the method described in Example 3.1.
- Use the original logic statement; this allows you to check that your logic circuit is correct.

We will use the second method in this example.

Inputs			Intermediate values		Output
A	B	C	(A=1 AND B=NOT 1)	(B=1 AND C=NOT 1)	X
0	0	0	*0*	*0*	0
0	0	1	*0*	*0*	0
0	1	0	*0*	*1*	1
0	1	1	*0*	*0*	0
1	0	0	*1*	*0*	1
1	0	1	*1*	*0*	1
1	1	0	*0*	*1*	1
1	1	1	*0*	*0*	0

ACTIVITY 3C

Draw the logic circuits and complete the truth tables for these logic statements and Boolean algebra statements.

a) X = 1 if (A = 1 OR B = 1) OR (A = 0 AND B = 1)

b) Y = 1 if (A = 0 AND B = 0) AND (B = 0 OR C = 1)

c) T = 1 if (switch K is ON or switch L is ON) OR (switch K is ON and switch M is OFF) OR (switch M is ON)

d) X = (A.B) + (B.C)

e) R = 1 if (switch A is ON and switch B is ON) AND (switch B is ON or switch C is OFF)

Example 3.3

A wind turbine has a safety system which uses three inputs to a logic circuit. A certain combination of conditions results in an output, X, from the logic circuit being equal to 1. When the value of X = 1, the wind turbine is shut down.

The following table shows which parameters are being monitored and form the three inputs to the logic circuit.

Parameter description	Parameter	Binary value	Description of condition
turbine speed	S	0	turbine speed ≤ 1000 rpm
		1	turbine speed > 1000 rpm
bearing temperature	T	0	bearing temperature ≤ 80 °C
		1	bearing temperature > 80 °C
wind velocity	W	0	wind velocity ≤ 120 kph
		1	wind velocity > 120 kph

The output, X, will have a value of 1 if any of the following combination of conditions occur:

- either turbine speed ≤ 1000 rpm and bearing temperature > 80 °C
- or turbine speed > 1000 rpm and wind velocity > 120 kph
- or bearing temperature ≤ 80 °C and wind velocity > 120 kph

Design the logic circuit and complete the truth table to produce a value of X = 1 when either of the three conditions occur.

Solution

This is a different type of problem to those covered in Examples 3.1 and 3.2. This time, a real situation is given and it is necessary to convert the information into a logic statement and then produce the logic circuit and truth table. It is advisable in problems as complex as this to produce the logic circuit and truth table separately (based on the conditions given) and then check them against each other to see if there are any errors.

Stage 1

The first thing to do is to convert each of the three statements into logic statements. Use the information given in the table and the three condition statements to find how the three

parameters S, T and W, are linked. We usually look for the key words AND, OR and NOT when converting actual statements into logic.

We end up with these three logic statements:

① turbine speed ⮹ 1000 rpm and bearing temperature > 80 °C
logic statement: (S = NOT 1 AND T = 1)

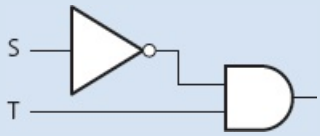
② turbine speed > 1000 rpm and wind velocity > 120 kph
logic statement: (S = 1 AND W = 1)

③ bearing temperature ⮹ 80 °C and wind velocity > 120 kph
logic statement: (T = NOT 1 AND W = 1)

Stage 2

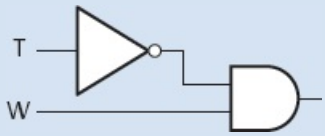
This produces three intermediate logic circuits:

①



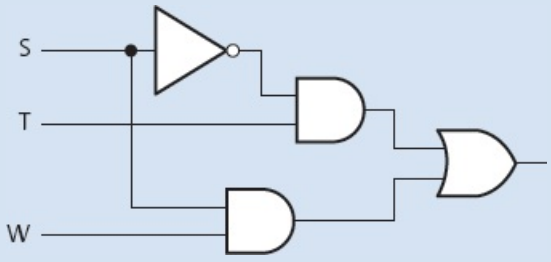
②

③

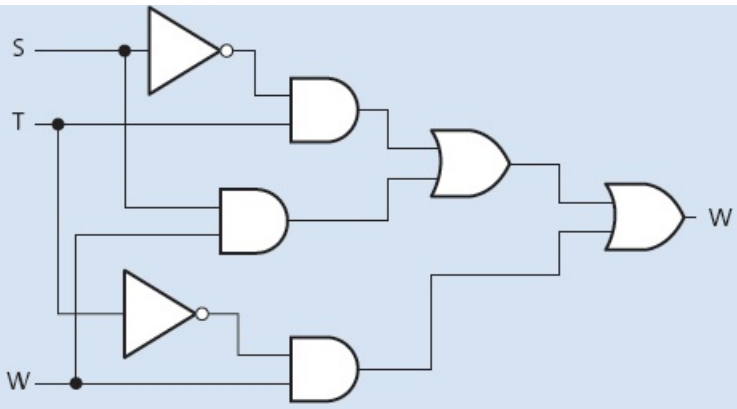


Each of the three original statements were joined together by the word OR. So, we need to join all of the three intermediate logic circuits by two OR gates to get the final logic circuit.

We will start by joining ① and ② together using an OR gate.



Now, we connect this to logic circuit ③ to obtain the final logic circuit.



The final part is to produce the truth table. We will do this using the original logic statement, since this method allows an extra check to be made on the final logic circuit.

There were three parts to the problem, so the truth table will first evaluate each part. Then, by applying OR gates, as shown below, the final value, X, is obtained:

① (S = NOT 1 AND T = 1)

② (S = 1 AND W = 1)

③ (T = NOT 1 AND W = 1)

We find the outputs from ① and ② and then OR these two outputs to obtain a new intermediate, which we will label part ④.

We then OR parts ③ and ④ together to get the value of X.

Inputs			Intermediate values				Output
A	B	C	① (S=NOT 1 AND T=1)	② (S=1 AND W=1)	③ (T=NOT 1 AND W=1)	④	X
0	0	0	*0*	*0*	*0*	*0*	0
0	0	1	*0*	*0*	*1*	*0*	1
0	1	0	*1*	*0*	*0*	*1*	1
0	1	1	*1*	*0*	*0*	*1*	1
1	0	0	*0*	*0*	*0*	*0*	0
1	0	1	*0*	*1*	*1*	*1*	1
1	1	0	*0*	*0*	*0*	*0*	0
1	1	1	*0*	*1*	*0*	*1*	1

ACTIVITY 3D

There are two scenarios described below. In each case, produce the logic circuit and complete a truth table to represent the scenario.

a) A chemical process is protected by a logic circuit. There are three inputs to the logic circuit representing key parameters in the chemical process.
An alarm, X, will give an output value of 1 depending on certain conditions in the chemical process.
This table describes the process conditions being monitored.

Parameter description	Parameter	Binary value	Description of condition
chemical reaction rate	R	0	reaction rate < 40 mol/l/sec
		1	reaction rate $\geqslant 40$ mol/l/sec
process temperature	T	0	temperature > 115 °C
		1	temperature $\leqslant 115$ °C
concentration of chemicals	C	0	concentration = 4 mol
		1	concentration $>$ 4 mol

An alarm, X, will generate the value 1 if:
either reaction rate < 40 mol/l/sec
or concentration > 4 mol AND temperature > 115 °C
or reaction rate ⁝↑ 40 mol/l/sec AND temperature > 115 °C.

b) A power station has a safety system controlled by a logic circuit. Three inputs to the logic circuit determine whether the output, S, is 1.

When S = 1 the power station shuts down.

The following table describes the conditions being monitored.

Parameter description	Parameter	Binary value	Description of condition
gas temperature	G	0	gas temperature $\leqslant 160$ °C
		1	gas temperature > 160 °C
reactor pressure	R	0	reactor pressure $\leqslant 10$ bar
		1	reactor pressure $>$ 10 bar
water temperature	W	0	water temperature $\leqslant 120$ °C
		1	water temperature > 120 °C

Output, S, will generate a value of 1, if:

either gas temperature > 160 °C AND water temperature ⍐ 120 °C

or gas temperature ⍐ 160 °C AND reactor pressure > 10 bar

or water temperature > 120 °C AND reactor pressure > 10 bar.

3.2.5 Logic circuits in the real world

The design of logic circuits is considerably more complex than has, so far, been described. We have discussed some of the fundamental theories, providing sufficient coverage of the Cambridge International A Level syllabus. However, it is worth discussing some of the more advanced aspects of logic circuit design, to strengthen understanding.

Electronics companies need to consider the cost of components, ease of fabrication and time constraints when designing and building logic circuits.

Ways electronics companies review logic circuit design include:

- using 'off-the-shelf' logic units and building up the logic circuit as a number of 'building blocks'
- simplifying the logic circuit as far as possible; this may be necessary where room is at a premium (for example, building circuit boards for use in satellites for space exploration).

Using logic 'building blocks'

One common 'building block' is the NAND gate. It is possible to build up any logic gate, and therefore any logic circuit, by simply linking together a number of NAND gates, such as:

- the AND gate

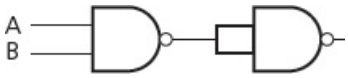


Figure 3.29 AND gate made from NAND gates

- the OR gate

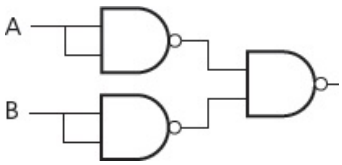


Figure 3.30 OR gate made from NAND gates

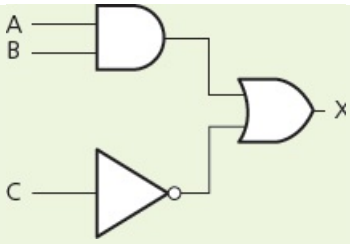
- the NOT gate

Figure 3.31 NOT gate made from NAND gates

ACTIVITY 3E

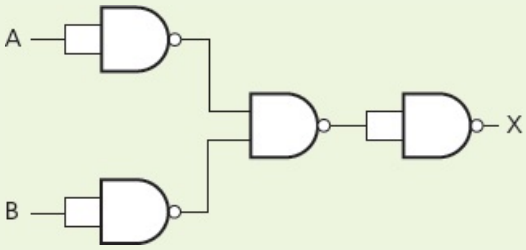
1 By drawing the truth tables, show that the three logic circuits shown above can be used to represent AND, OR and NOT gates.

2 a) Show how the following logic circuit could be built using NAND gates only.
Complete truth tables for both logic circuits to show that they produce identical outputs.



b) Show how the XOR gate could be built from NAND gates only.
Complete a truth table for your final design to show that it produces the same output as a single XOR gate.

3 By drawing a truth table, discover which single logic gate has the same function as the following logic circuit made up of NAND gates only.

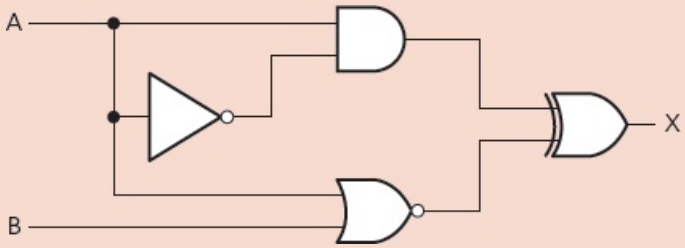


Simplification of logic circuits

The second method involves the simplification of logic circuits. By reducing the number of components, the cost of production can be less. This can also improve reliability and make it easier to trace faults if they occur. This is covered in more depth in Chapter 15.

EXTENSION ACTIVITY 3H

By drawing a truth table, show which single logic gate has the same function as the logic circuit drawn below.



3.2.6 Multi-input logic gates

This section looks at logic gates with more than two inputs (apart from the NOT gate). Students are not expected to answer questions about multi-input logic gates at Cambridge International AS Level, but this information is included here for completeness and for those with an electronics background. This is intended to complete the picture for interested students who may have seen multi-input gates in other textbooks, or online, and it leads neatly into topics covered in Chapter 15.

Logic gates (apart from the NOT gate) can have more than two inputs. While it is still acceptable to use two-input logic gates, it is worth considering the multi-input option when designing logic circuits; they can simplify the overall result.

Multi-input AND gates

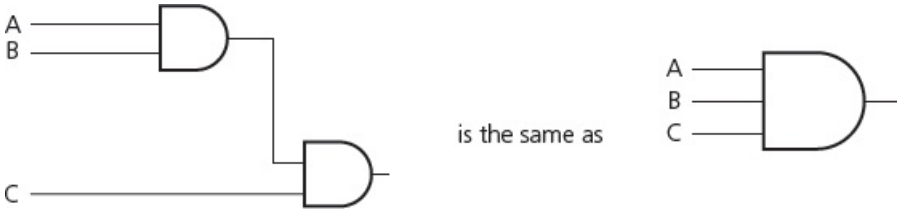


Figure 3.32 Multi-input AND gate

Both sets of AND gates have the output A.B.C and they share identical truth tables.

Inputs			Output
A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 3.16

Now consider the following:

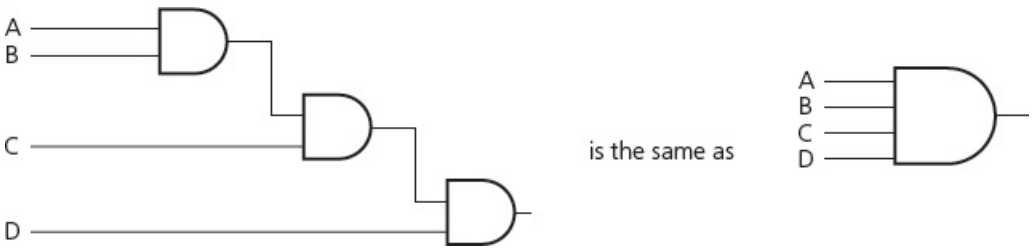


Figure 3.33 4-input AND gate

Both sets of AND gates have the output A.B.C.D and they share identical truth tables.

Inputs				Output
A	B	C	D	A.B.C.D
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Table 3.17

Multi-input OR gates

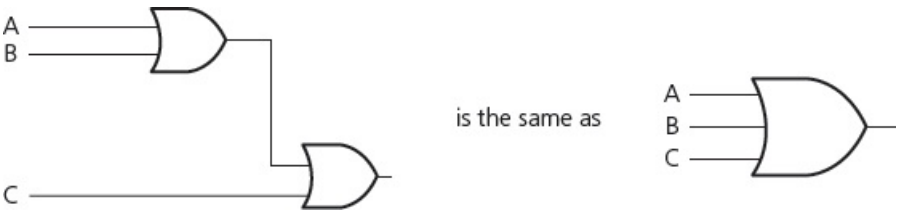


Figure 3.34 Multi-input OR gate

Both sets of OR gates have the output A + B + C and they share identical truth tables.

Inputs			Output
A	B	C	A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.18

Now consider the following:

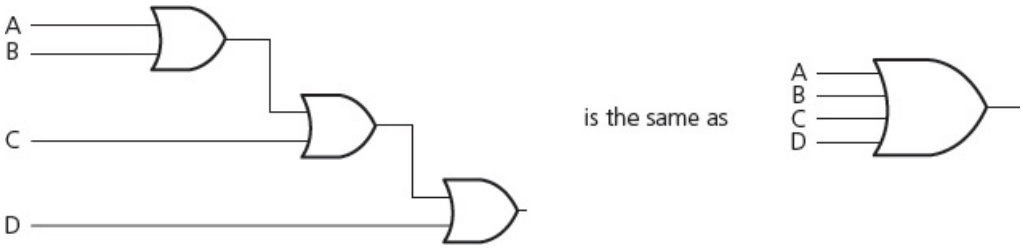


Figure 3.35 4-input OR gate

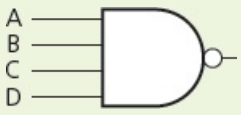
Both sets of OR gates have the output A + B + C + D and they share identical truth tables.

Inputs				Output
A	B	C	D	A + B + C + D
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3.19

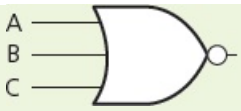
ACTIVITY 3F

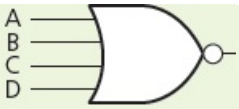
1 a) Draw the following multi-input NAND gate using two-input NAND gates only:



b) Construct the truth tables for the above 4-input NAND gate and for your circuit drawn in part a). Confirm that they are identical.

2 a) Draw the following multi-input NOR gates using two-input NOR gates only.

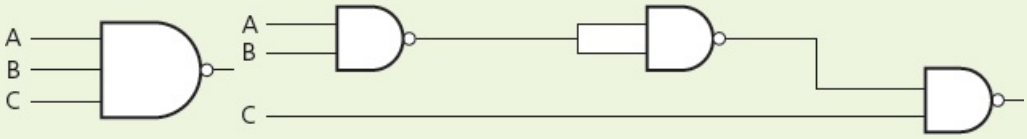




b) Construct the truth tables for the above 3-input NOR gate and for your equivalent circuit drawn in part a).
Confirm they are identical.

c) Construct the truth tables for the above 4-input NOR gate and for your equivalent circuit drawn in part a).
Confirm they are identical.

3 Confirm that the following two logic circuits are identical by constructing the truth tables for each circuit.



End of chapter questions

1 a) Many mobile phone and tablet manufacturers are moving to OLED screen technology.
Give **three** reasons why this is happening.

[3]

b) A television manufacturer makes the following advertising claim:
'Our OLED screens allow the user to enjoy over one million vivid colours in true-to-life vision.'
Comment on the validity of this claim.

[4]

2 a) A company is developing a new games console. The game will be stored on a ROM chip once the program to run the new game has been fully tested and developed.

i) Give **two** advantages of putting the game's program on a ROM chip.

[2]

ii) Explain why the manufacturers would use an EPROM chip during development.

[2]

iii) The manufacturers are also using RAM chips on the internal circuit board.
Explain why they are doing this.

[2]

iv) The games console will have four USB ports.
Apart from the need to attach games controllers, give reasons why USB ports are incorporated.

[2]

b) During development of the games console the plastic parts are being made by a 3D printer.
Give **two** reasons why the manufacturer would use 3D printers.

[2]

3 An air conditioning unit in a car is being controlled by a microprocessor and a number of sensors.

a) Describe the main differences between control and monitoring of a process.

[2]

b) Describe how the sensors and microprocessor would be used to control the air conditioning unit in the car.
Name at least **two** different sensors that might be used and explain the role of positive feedback in your description.
You might find drawing a diagram of your intended process to be helpful.

[6]

4 The nine stages in printing a page using an inkjet printer are shown below. They are **not** in the correct order.
Write the letters A to I so that the stages are in the correct order.

[9]

A	The data is then sent to the printer and it is stored in a temporary memory known as a printer buffer.
B	As the sheet of paper is fed through the printer, the print head moves from side to side across the paper printing the text or image. The four ink colours are sprayed in their exact amounts to produce the desired final colour.
C	The data from the document is sent to a printer driver.
D	Once the printer buffer is empty, the printer sends an interrupt to the processor in the computer, which is a request for more data to be sent to the printer. The whole process continues until the whole of the document has been printed.
E	The printer driver ensures that the data is in a format that the chosen printer can understand.
F	At the end of each full pass of the print head, the paper is advanced very slightly to allow the next line to be printed. This continues until the whole page has been printed.
G	A check is made by the printer driver to ensure that the chosen printer is available to print (is it busy? is it off line? is it out of ink? and so on).
H	If there is more data in the printer buffer, then the whole process from stage 5 is repeated until the buffer is finally empty.
I	A sheet of paper is then fed into the main body of the printer, where a sensor detects whether paper is available in the paper feed tray – if it is out of paper (or the paper is jammed) then an error message is sent back to the computer.

5 **a)** There are two types of RAM: dynamic RAM (DRAM) and static RAM (SRAM). Five statements about DRAM and RAM are shown below. Copy the diagram below and connect each statement to the appropriate type of RAM.

[5]

Statement	Type of RAM
requires the data to be refreshed periodically in order to retain data	
has more complex circuitry	DRAM
does not need to be refreshed as the circuit holds the data as long as the power supply is on	
requires higher power consumption which is significant when used in battery-powered devices	SRAM
used predominantly in cache memory of processors where speed is important	

b) Give **three** differences between RAM and ROM.

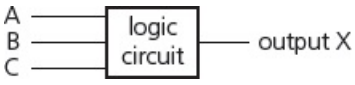
[3]

c) DVD-RAM and flash memory are two examples of storage devices.
Describe **two** differences in how they operate.

[2]

Cambridge International AS & A Level Computer Science 9608 Paper 13 Q4 June 2015

6 a) Three digital sensors, A, B and C, are used to monitor a process. The outputs from the sensors are used as the inputs to a logic circuit. A signal, X, is output from the logic circuit:



Output, X, has a value of 1 if either of the following two conditions occur:
– Sensor A outputs the value 1 OR sensor B outputs the value 0.
– Sensor B outputs the value 1 AND sensor C outputs the value 0.
Draw a logic circuit to represent these conditions.

[5]

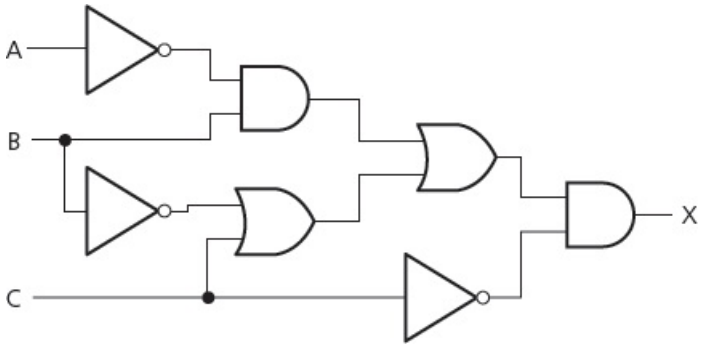
b) Copy and complete the truth table for the logic circuit described in part a).

[4]

A	B	C	working space	X
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

c) Write a logic statement that describes the following logic circuit.

[3]



Cambridge International AS & A Level Computer Science 9608 Paper 13 Q6 June 2015

4 Processor fundamentals

In this chapter, you will learn about

- the basic Von Neumann model of a computer system
- the purpose and role of the registers PC, MDR, MAR, ACC, IX, CIR, and the status registers
- the purpose and role of the arithmetic logic unit (ALU), control unit (CU), system clock and immediate access store (IAS)
- functions of the address bus, data bus and control bus
- factors affecting computer performance (such as processor type, bus width, clock speeds, cache memory and use of core processors)
- the connection of computers to peripheral devices such as Universal Serial Bus (USB), high definition multimedia interface (HDMI) and Video Graphics Array (VGA)
- the fetch-execute cycle and register transfers
- the purpose of interrupts
- the relationship between assembly language and machine code (such as symbolic, absolute and relative addressing)
- different stages for a two-pass assembler
- tracing sample assembly language programming code
- assembly language instruction groups (such as data movement, I/O operations, arithmetic operations, comparisons, and so on)
- addressing modes (immediate, direct, indirect, indexed and relative)
- how to perform binary shifts (including logical, arithmetic, cyclic, left shift and right shift)
- how bit manipulation is used to monitor/control a device.

4.1 Central processing unit (CPU) architecture

WHAT YOU SHOULD ALREADY KNOW

Try these four questions before you read the first part of this chapter.

1 a) Name the main components that make up a typical computer system.

b) Tablets and smart phones carry out many of the functions of a desktop or laptop computer. Describe the main differences between the operations of a desktop or laptop computer and a tablet or phone.

2 When deciding on which computer, tablet or phone to buy, which are the main factors that determine your final choice?

3 Look at a number of computers, laptops and phones and list (and name) the types of input and output ports found on each device.

4 At the centre of all of the above electronic devices is the microprocessor. How has the development of the microprocessor changed over the last ten years?

Key terms

Von Neumann architecture – computer architecture which introduced the concept of the stored program in the 1940s.

Arithmetic logic unit (ALU) – component in the processor which carries out all arithmetic and logical operations.

Control unit – ensures synchronisation of data flow and programs throughout the computer by sending out control signals along the control bus.

System clock – produces timing signals on the control bus to ensure synchronisation takes place.

Immediate access store (IAS) – holds all data and programs needed to be accessed by the control unit.

Accumulator – temporary general purpose register which stores numerical values at any part of a given operation.

Register – temporary component in the processor which can be general or specific in its use that holds data or instructions as part of the fetch-execute cycle.

Status register – used when an instruction requires some form of arithmetic or logical processing.

Flag – indicates the status of a bit in the status register, for example, N = 1 indicates the result of an addition gives a negative value.

Address bus – carries the addresses throughout the computer system.

Data bus – allows data to be carried from processor to memory (and vice versa) or to and from

input/output devices.

Control bus – carries signals from control unit to all other computer components.

Unidirectional – used to describe a bus in which bits can travel in one direction only.

Bidirectional – used to describe a bus in which bits can travel in both directions.

Word – group of bits used by a computer to represent a single unit.

Clock cycle – clock speeds are measured in terms of GHz; this is the vibrational frequency of the clock which sends out pulses along the control bus – a 3.5 GHZ clock cycle means 3.5 billion clock cycles a second.

Overclocking – changing the clock speed of a system clock to a value higher than the factory/recommended setting.

BIOS – basic input/output system.

Cache memory – a high speed auxiliary memory which permits high speed data transfer and retrieval.

Core – a unit made up of ALU, control unit and registers which is part of a CPU. A CPU may contain a number of cores.

Dual core – a CPU containing two cores.

Quad core – a CPU containing four cores.

Port – external connection to a computer which allows it to communicate with various peripheral devices. A number of different port technologies exist.

Universal Serial Bus (USB) – a type of port connecting devices to a computer.

Asynchronous serial data transmission – serial refers to a single wire being used to transmit bits of data one after the other. Asynchronous refers to a sender using its own clock/timer device rather sharing the same clock/timer with the recipient device.

High-definition multimedia interface (HDMI) – type of port connecting devices to a computer.

Video Graphics Array (VGA) – type of port connecting devices to a computer.

High-bandwidth digital copy protection (HDCP) – part of HDMI technology which reduces risk of piracy of software and multimedia.

Fetch-execute cycle – a cycle in which instructions and data are fetched from memory and then decoded and finally executed.

Program counter (PC) – a register used in a computer to store the address of the instruction which is currently being executed.

Current instruction register – a register used to contain the instruction which is currently being executed or decoded.

Register Transfer Notation (RTN) – short hand notation to show movement of data and instructions in a processor, can be used to represent the operation of the fetch-execute cycle.

Interrupt – signal sent from a device or software to a processor requesting its attention; the

processor suspends all operations until the interrupt has been serviced.

Interrupt priority – all interrupts are given a priority so that the processor knows which need to be serviced first and which interrupts are to be dealt with quickly.

Interrupt service routine (ISR) or interrupt handler – software which handles interrupt requests (such as ‘printer out of paper’) and sends the request to the CPU for processing.

4.1.1 Von Neumann model

Early computers were fed data while the machines were running. It was not possible to store programs or data; that meant they could not operate without considerable human intervention.

In the mid-1940s, John Von Neumann developed the concept of the stored program computer. It has been the basis of computer architecture for many years. The main, previously unavailable, features of the **Von Neumann architecture** were

- a central processing unit (CPU or processor)
- a processor able to access the memory directly
- computer memories that could store programs as well as data
- stored programs made up of instructions that could be executed in sequential order.

Figure 4.1 shows a simple representation of Von Neumann architecture.

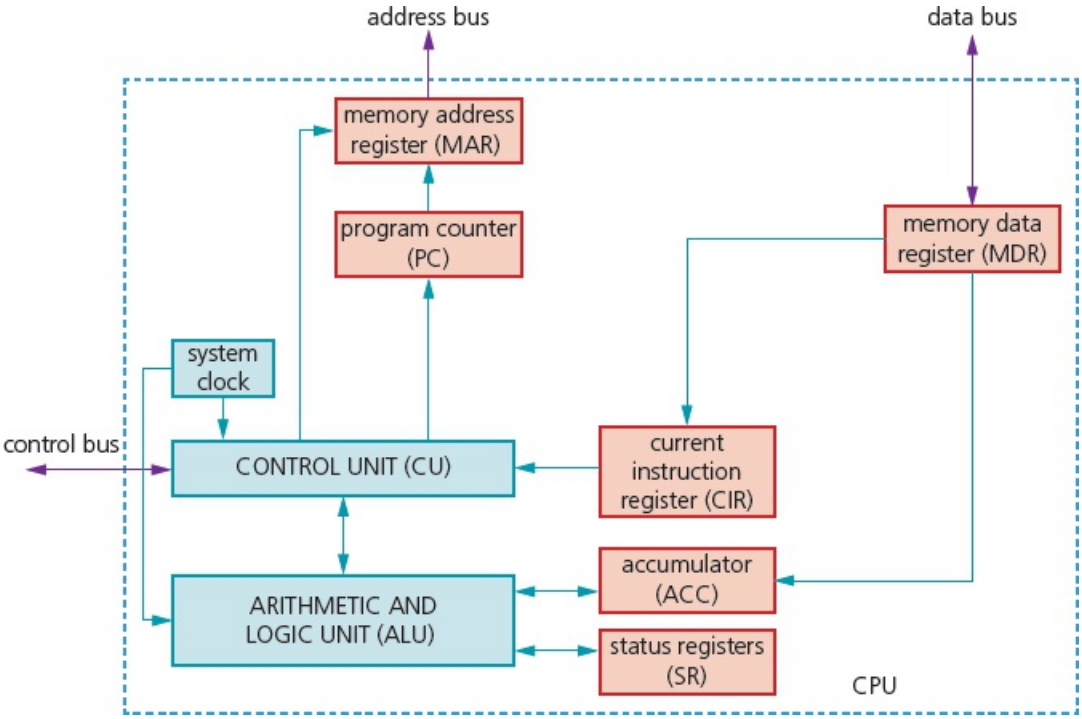


Figure 4.1 Representation of Von Neumann architecture

4.1.2 Components of the processor (CPU)

The main components of the processor are the **arithmetic logic unit (ALU)**, the **control unit (CU)**, the **system clock** and the **immediate access store (IAS)**.

Arithmetic logic unit (ALU)

The ALU allows the required arithmetic or logic operations to be carried out while a program is being run. It is possible for a computer to have more than one ALU – one will perform fixed point operations and the other floating-point operations (see Chapter 13).

Multiplication and division are carried out by a sequence of addition, subtraction and left/right shifting operations (for example, shifting 0 0 1 1 0 1 1 1 two places to the left gives 1 1 0 1 1 1 0 0, which is equivalent to multiplying by a factor of 4).

The **accumulator (ACC)** is a temporary register used when carrying out ALU calculations.

Control unit (CU)

The CU reads an instruction from memory (the address of the location where the instruction can be found is stored in the program counter (PC)). This instruction is then interpreted. During that process, signals are generated along the control bus to tell the other components in the computer what to do. The CU ensures synchronisation of data flow and program instructions throughout the computer.

System clock

A system clock is used to produce timing signals on the control bus to ensure this vital synchronisation takes place – without the clock the computer would simply crash. (See Section 4.1.4 System buses.)

Immediate access store (IAS)

The IAS holds all the data and programs that the processor (CPU) needs to access. The CPU takes data and programs held in backing store and puts them into the IAS temporarily. This is done because read/write operations carried out using the IAS are considerably faster than read/write operations to backing store. Consequently, any key data needed by an application will be stored temporarily in IAS to speed up operations. The IAS is another name for primary (RAM) memory.

4.1.3 Registers

One of the most fundamental components of the Von Neumann system is the **register**. Registers can be general purpose or special purpose. General purpose registers hold data that is frequently used by the CPU or can be used by the programmer when addressing the CPU directly. The accumulator is a good example of a general purpose register and will be used as such throughout this book. Special purpose registers have a specific function within the CPU and hold the program state.

The most common special registers referred to in this book are shown in Table 4.1. The use of many of these registers is explained more fully in Section 4.1.6 (fetch-execute cycle) and in Section 4.2 (tracing of assembly code programs).

Register	Abbreviation	Function/purpose of register
current instruction register	CIR	stores the current instruction being decoded and executed
index register	IX	used when carrying out index addressing operations (assembly code)
memory address register	MAR	stores the address of the memory location currently being read from or written to
memory data/buffer register	MDR/MBR	stores data which has just been read from memory or data which is about to be written to memory (sometimes referred to as MBR)
program counter	PC	stores the address where the next instruction to be read can be found
status register	SR	contain bits which can be set or cleared depending on the operation (for example, to indicate overflow in a calculation)

Table 4.1 Common registers

All of the registers listed in Table 4.1 (apart from status and index registers) are used in the fetch-execute cycle, which is covered later in this chapter.

Index registers are best explained when looking at addressing techniques in assembly code (again, this is covered later in the chapter).

A **status register** is used when an instruction requires some form of arithmetic or logic processing. Each bit is known as a **flag**. Most systems have the following four flags.

- Carry flag (C) is set to 1 if there is a CARRY following an addition operation (refer to Chapter 1).
- Negative flag (N) is set to 1 if the result of a calculation yields a NEGATIVE value.
- Overflow flag (V) is set to 1 if an arithmetic operation results in an OVERFLOW being

produced.

- Zero flag (Z) is set to 1 if the result of an arithmetic or logic operation is ZERO.

Consider this arithmetic operation:

```
  01110111
+ 00111000
  10101111
```

Flags:
N V C Z
1 1 0 0

Since we have two positive numbers being added, the answer should not be negative. The flags indicate two errors: a negative result, and an overflow occurred.

Now consider this operation:

```
  10001000
+ 11000111
  1010011111
```

Flags:
N V C Z
0 1 1 0

Since we have two negative numbers being added, the answer should be negative. The flags indicate that two errors have occurred: a carry has been generated, and a ninth bit overflow has occurred.

Other flags can be generated, such as a parity flag, an interrupt flag or a half-carry flag.

EXTENSION ACTIVITY 4A

Find out what conditions could cause:

a) a parity flag (P) being set to 1

b) an interrupt flag (I) being set to 1

c) a zero flag (Z) being set to 1

d) a half-carry flag (H) being set to 1.

4.1.4 System buses

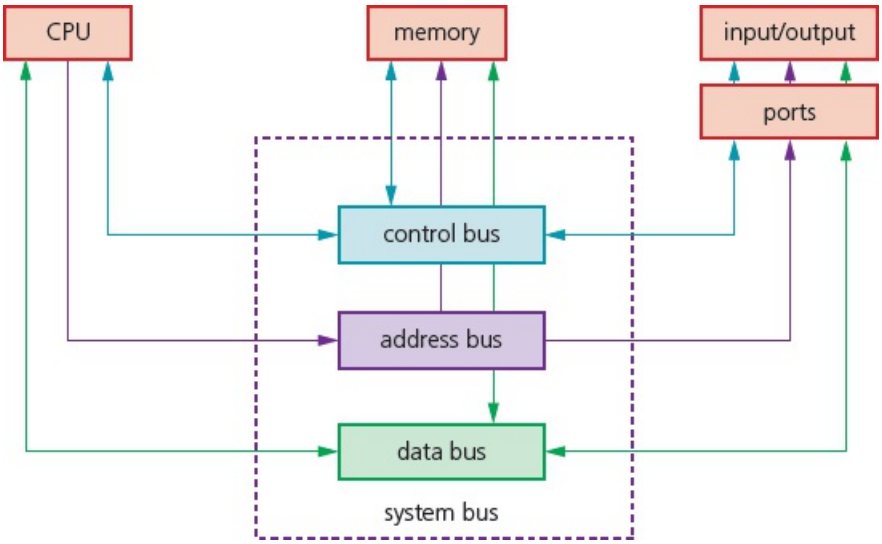


Figure 4.2 System buses

(System) buses are used in computers as a parallel transmission component; each wire in the bus transmits one bit of data. There are three common buses used in the Von Neumann architecture known as **address bus**, **data bus** and **control bus**.

Address bus

As the name suggests, the address bus carries addresses throughout the computer system. Between the CPU and memory the address bus is **unidirectional** (in other words, bits can travel in one direction only). This prevents addresses being carried back to the CPU, which would be undesirable.

The width of a bus is important. The wider the bus, the more memory locations which can be directly addressed at any given time; for example, a bus of width 16 bits can address 2^{16} (65 536) memory locations, whereas a bus width of 32 bits allows 4 294 967 296 memory locations to be simultaneously addressed. Even this is not large enough for modern computers, but the technology behind even wider buses is outside the scope of this book.

Data bus

The data bus is **bidirectional** (in other words, it allows data to be sent in both directions along the bus). This means data can be carried from CPU to memory (and vice versa) as well as to and from input/output devices. It is important to point out that data can be an address, an instruction or a numerical value. As with the address bus, the width of the data bus is important: the wider the bus, the larger the word length that can be transported. (A **word** is a group of bits which can be regarded as a single unit, for example, 16-bit, 32-bit or 64-bit word lengths are the most common). Larger word lengths can improve the computer's overall performance.

Control bus

The control bus is also bidirectional. It carries signals from the CU to all the other computer components. It is usually 8-bits wide since it only carries control signals.

It is worth mentioning here the role of the system clock. The clock defines the **clock cycle** which synchronises all computer operations. As mentioned earlier, the control bus transmits timing signals, ensuring everything is fully synchronised. By increasing clock speed, the processing speed of the computer is also increased (a typical current value is 3.5 GHz – which means 3.5 billion clock cycles a second). Although the speed of the computer may have been increased, it is not possible to say that a computer's overall performance is necessarily increased by using a higher clock speed. Four other factors need to be considered.

1 Width of the address bus and data bus can affect computer performance.
2 **Overclocking**: the clock speed can be changed by accessing the **BIOS basic input/output system (BIOS)** and altering the settings. However, using a clock speed higher than the computer was designed for can lead to problems, such as
   – execution of instructions outside design limits, which can lead to seriously unsynchronised operations (in other words, an instruction is unable to complete in time before the next one is due to be executed) and the computer would frequently crash and become unstable
   – serious overheating of the CPU leading to unreliable performance.
3 The use of **cache memory** can also improve processor performance. It is similar to RAM in that its contents are lost when the power is turned off. Cache uses SRAM (see Chapter 3) whereas most computers use DRAM for main memory. Therefore, cache memories will have faster access times, since there is no need to keep refreshing, which slows down access time. When a processor reads memory, it first checks out cache and then moves on to main memory if the required data is not there. Cache memory stores frequently used instructions and data that need to be accessed faster. This improves processor performance.
4 The use of a different number of **cores** (one core is made up of an ALU, a CU and the registers) can improve computer performance. Many computers are **dual core** (the CPU is made up of two cores) or **quad core** (the CPU is made up of four cores). The idea of using more cores alleviates the need to continually increase clock speeds. However, doubling the number of cores does not necessarily double the computer's performance since we have to take into account the need for the CPU to communicate with each core; this will reduce overall performance. For example
   – dual core has one channel and needs the CPU to communicate with both cores, reducing some of the potential increase in its performance
   – quad core has six channels and needs the CPU to communicate with all four cores, considerably reducing potential performance.

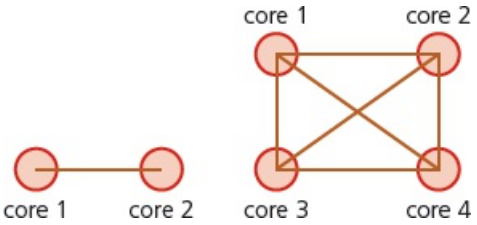


Figure 4.3 Two cores, one channel (left) and four cores, six channels (right)

All of these factors need to be taken into account when considering computer performance.

In summary

- increasing bus width (data and address buses) increases the performance and speed of a computer system
- increasing clock speed usually increases the speed of a computer
- a computer's performance can be changed by altering bus width, clock speed and use of multi-core CPUs
- use of cache memories can also speed up a processor's performance.

4.1.5 Computer ports

Input and output devices are connected to a computer via **ports**. The interaction of the ports with connected input and output is controlled by the control unit. Here we will summarise some of the more common types of ports found on modern computers.

Figure 4.4 (from left to right) USB cable, HDMI cable, VGA cable

USB ports

The **Universal Serial Bus (USB)** is an **asynchronous serial data transmission** method. It has quickly become the standard method for transferring data between a computer and a number of devices.

The USB cable consists of a four-wired shielded cable, with two wires for power and the earth, and two wires used for data transmission. When a device is plugged into a computer using one of the USB ports

- the computer automatically detects that a device is present (this is due to a small change in the voltage level on the data signal wires in the cable)
- the device is automatically recognised, and the appropriate device driver is loaded up so that computer and device can communicate effectively
- if a new device is detected, the computer will look for the device driver which matches the device. If this is not available, the user is prompted to download the appropriate software.

The USB system has become the industry standard, but there are still pros and cons to using this system, as summarised in Table 4.2.

Pros of USB system	Cons of USB system
• devices plugged into the computer are automatically detected and device drivers are automatically loaded up • the connectors can only fit one way, which prevents incorrect connections being made • this has become the industry standard, which means that considerable support is available to users • several different data transmission rates are supported • newer USB standards are backward compatible	• the present transmission rate is limited to less than 500 megabits per second • the maximum cable length is presently about five metres • the older USB standard (such as 1.1) may not be supported in the near future

with older USB standards	

Table 4.2 Pros and cons of the USB system

High-definition multimedia interface (HDMI)

High-definition multimedia interface (HDMI) ports allow output (both audio and visual) from a computer to an HDMI-enabled device. They support high-definition signals (enhanced or standard). HDMI was introduced as a digital replacement for the older **Video Graphics Array (VGA)** analogue system. Modern HD (high definition) televisions have the following features, which are making VGA a redundant technology:

- They use a widescreen format (16:9 aspect ratio).
- The screens use a greater number of pixels (typically 1920 × 1080).
- The screens have a faster refresh rate (such as 120 Hz or 120 frames a second).
- The range of colours is extremely large (some companies claim up to four million different colour variations).

This means that modern HD televisions require more data, which has to be received at a much faster rate than with older televisions (around 10 gigabits per second). HDMI increases the bandwidth, making it possible to supply the necessary data for high quality sound and visual effects.

HDMI can also afford some protection against piracy since it uses **high-bandwidth digital copy protection (HDCP)**. HDCP uses a type of authentication protocol (see Chapters 6 and 17). For example, a Blu-ray player will check the authentication key of the device it is sending data to (such as an HD television). If the key can be authenticated, then handshaking takes place and the Blu-ray can start to transmit data to the connected device.

Video Graphics Array (VGA)

VGA was introduced at the end of the 1980s. VGA supports 640 × 480 pixel resolution on a television or monitor screen. It can also handle a refresh rate of up to 60 Hz (60 frames a second) provided there are only 16 different colours being used. If the pixel density is reduced to 200 × 320, then it can support up to 256 colours.

The technology is analogue and, as mentioned in the previous section, is being phased out.

Table 4.3 summarises the pros and cons of HDMI and VGA.

Pros of HDMI	**Cons of HDMI**
• the current standard for modern televisions and monitors • allows for a very fast data transfer rate • improved security (helps prevent piracy) • supports modern digital systems	• not a very robust connection (easy to break connection when simply moving device) • limited cable length to retain good signal • there are currently five cable/connection standards
Pros of VGA	**Cons of VGA**

• simpler technology • only one standard available • it is easy to split the signal and connect a number of devices from one source • the connection is very secure	• old out-dated analogue technology • it is easy to bend the pins when making connections • the cables must be of a very high grade to ensure good undistorted signal

Table 4.3 Pros and cons of HDMI and VGA

4.1.6 Fetch-execute cycle

We have already considered the role of buses and registers in the processor. This next section shows how an instruction is decoded and executed in the **fetch-execute cycle** using various components in the processor.

To execute a set of instructions, the processor first fetches data and instructions from memory and stores them in suitable registers. Both the address bus and data bus are used in this process. Once this is done, each instruction needs to be decoded before being executed.

Fetch

The next instruction is fetched from the memory address currently stored in the **program counter (PC)** and is then stored in the **current instruction register (CIR)**. The PC is then incremented (increased by 1) so that the next instruction can be processed. This is decoded so that each instruction can be interpreted in the next part of the cycle.

Execute

The processor passes the decoded instruction as a set of control signals to the appropriate components within the computer system. This allows each instruction to be carried out in its logical sequence.

Figure 4.5 shows how the fetch-execute cycle is carried out in the Von Neumann computer model.

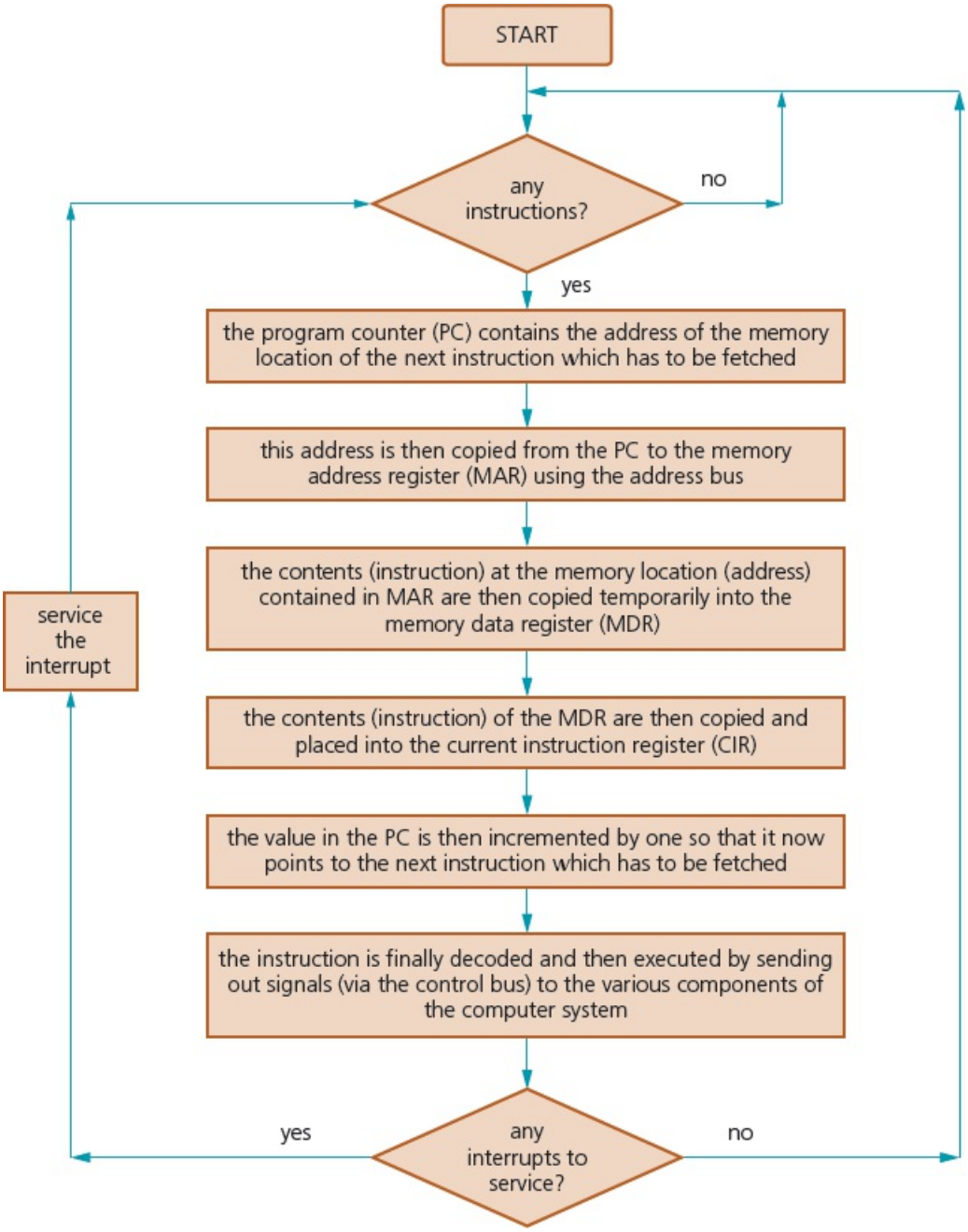


Figure 4.5 How the fetch-execute cycle is carried out in the Von Neumann computer model

When registers are involved, it is possible to describe what is happening by using **Register Transfer Notation (RTN)**. In its simplest form:

```
MAR ← [PC]      contents of PC copied into MAR
PC  ← [PC] + 1  PC is incremented by 1
MDR ← [[MAR]]   data stored at address shown in MAR is
                copied into MDR
CIR ← [MDR]     contents of MDR copied into CIR
```

Double brackets are used in the third line because it is not MAR contents being copied into MDR but it is the data stored at the address shown in MAR that is being copied to MDR.

Compare the above instructions to those shown in Figure 4.5. Inspection should show the register transfer notation is carrying out the same function.

RTN can be abstract (generic notation – as shown on page **117**) or concrete (specific to a particular machine – example shown below). For example, on a RISC computer:

```
instruction _ interpretation := (¬Run/Start →
Run ← 1; instruction _ interpretation):
Run → (CIR ← M[PC]:PC ← PC + 4; instruction _ execution)
```

Use of interrupts in the fetch-execute cycle

Section 4.1.7 gives a general overview of how a computer uses **interrupts** to allow a computer to operate efficiently and to allow it, for example, to carry out multi-tasking functions. Just before we discuss interrupts in this general fashion, the following notes explain how interrupts are specifically used in the fetch-execute cycle.

A special register called the interrupt register is used in the fetch-execute cycle. While the CPU is in the middle of carrying out this cycle, an interrupt could occur, which will cause one of the bits in the interrupt register to change its status. For example, the initial status might be 0000 0000 and a fault might occur while writing data to the hard drive; this would cause the register to change to 0000 1000. The following sequence now takes place.

- At the next fetch-execute cycle, the interrupt register is checked bit by bit.
- The contents 0000 1000 would indicate an interrupt occurred during a previous cycle and it still needs servicing. The CPU would now service this interrupt or ignore it for now, depending on its priority.
- Once the interrupt is serviced by the CPU, it stops its current task and stores the contents of its registers (see Section 4.1.7 for more details about how this is done).
- Control is now transferred to the interrupt handler (or interrupt service routine, ISR).
- Once the interrupt is fully serviced, the register is reset and the contents of registers are restored.

Figure 4.6 summarises the interrupt process during the fetch-execute cycle.

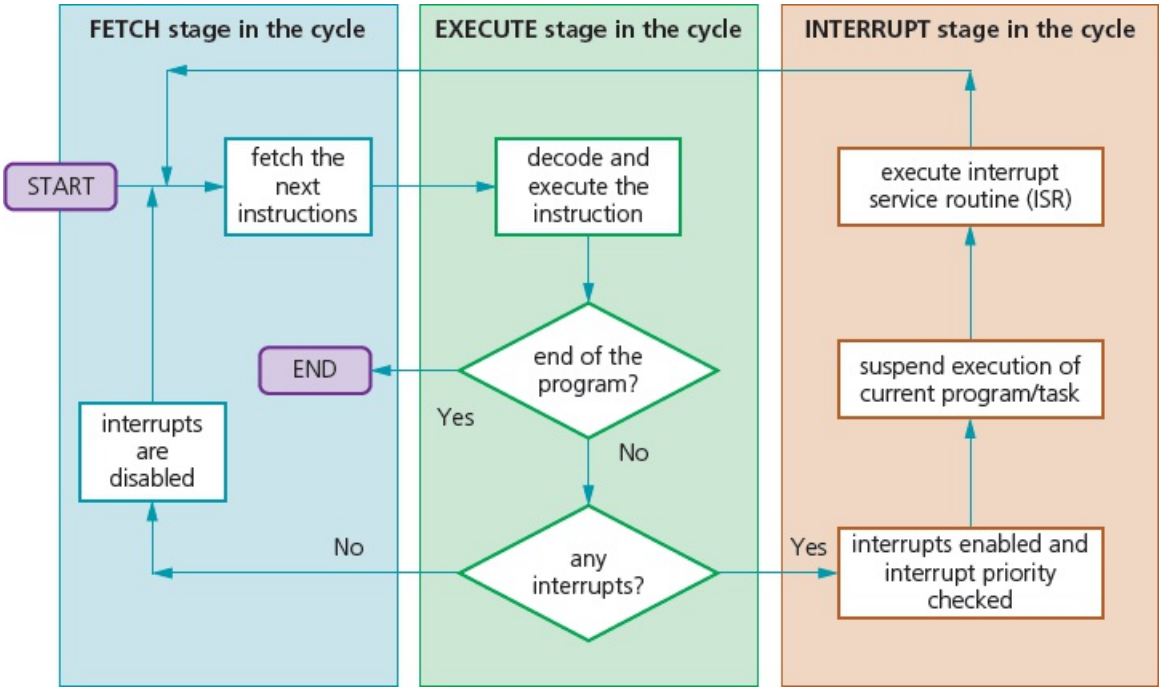


Figure 4.6 The interrupt process during the fetch-execute cycle

4.1.7 Interrupts

An interrupt is a signal sent from a device or from software to the processor. This will cause the processor to temporarily stop what it is doing and service the interrupt. Interrupts can be caused by, for example

- a timing signal
- input/output processes (a disk drive is ready to receive more data, for example)
- a hardware fault (an error has occurred such as a paper jam in a printer, for example)
- user interaction (the user pressed a key to interrupt the current process, such as <CTRL><ALT><BREAK>, for example)
- a software error that cannot be ignored (if an .exe file could not be found to initiate the execution of a program OR an attempt to divide by zero, for example).

Once the interrupt signal is received, the processor either carries on with what it was doing or stops to service the device/program that generated the interrupt. The computer needs to identify the interrupt type and also establish the level of **interrupt priority**.

Interrupts allow computers to carry out many tasks or to have several windows open at the same time. An example would be downloading a file from the internet at the same time as listening to some music from the computer library. Whenever an interrupt is serviced, the status of the current task being run is saved. The contents of the program counter and other registers are saved. Then, the **interrupt service routine (ISR)** is executed by loading the start address into the program counter. Once the interrupt has been fully serviced, the status of the interrupted task is reinstated (contents of saved registers retrieved) and it continues from the point prior to the interrupt being sent.

ACTIVITY 4a

1 a) Describe the functions of the following registers.

i) Current instruction register

ii) Memory address register

iii) Program counter

b) Status registers contain flags. Three such flags are named N, C and V.

i) What does each of the three flags represent?

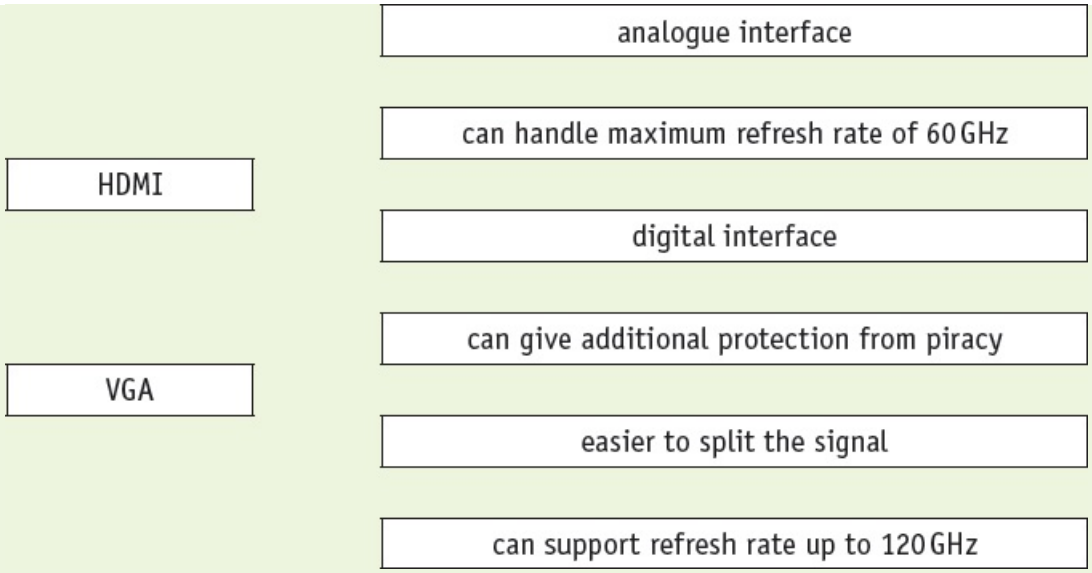
ii) Give an example of the use of each of the three flags.

2 a) Name **three** buses used in the Von Neumann architecture.

b) Describe the function of each named bus.

c) Describe how bus width and clock speed can affect computer performance.

3 Copy the diagram below and connect each feature to the correct port, HDMI or VGA.



4 **a)** What is meant by the *fetch-execute cycle*?

b) Using register transfer notation, show the main stages in a typical fetch-execute cycle.

5 Copy and complete this paragraph by using terms from this chapter.

The processor_____data and instructions required for an application and temporarily stores them in the_____until they can be processed.

The_____is used to hold the address of the next instruction to be executed. This address is copied to the_____using the_____.

The contents at this address are stored in the_____.

Each instruction is then_____and finally_____sending out_____using the_____. Any calculations carried out are done using the_____. During any calculations, data is temporarily held in a special register known as the_____.

4.2 Assembly language

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you start the second part of this chapter.

1 a) Name **two** types of low-level programming language.
   b) Name the only type of programming language that a CPU recognises.
   c) Why do programmers find writing in this type of programming language difficult?
2 Find at least **two** different types of CPU and the language they use.
3 Look at your computer and/or laptop and/or phone and list the programming language(s) they use.

Key terms

Machine code – the programming language that the CPU uses.

Instruction – a single operation performed by a CPU.

Assembly language – a low-level chip/machine specific programming language that uses mnemonics.

Opcode – short for operation code, the part of a machine code instruction that identifies the action the CPU will perform.

Operand – the part of a machine code instruction that identifies the data to be used by the CPU.

Source code – a computer program before translation into machine code.

Assembler – a computer program that translates programming code written in assembly language into machine code. Assemblers can be one pass or two pass.

Instruction set – the complete set of machine code instructions used by a CPU.

Object code – a computer program after translation into machine code.

Addressing modes – different methods of using the operand part of a machine code instruction as a memory address.

Absolute addressing – mode of addressing in which the contents of the memory location in the operand are used.

Direct addressing – mode of addressing in which the contents of the memory location in the operand are used, which is the same as absolute addressing.

Indirect addressing – mode of addressing in which the contents of the contents of the memory location in the operand are used.

Indexed addressing – mode of addressing in which the contents of the memory location found by adding the contents of the index register (IR) to the address of the memory location in the

operand are used.

Immediate addressing – mode of addressing in which the value of the operand only is used.

Relative addressing – mode of addressing in which the memory address used is the current memory address added to the operand.

Symbolic addressing – mode of addressing used in assembly language programming, where a label is used instead of a value.

4.2.1 Assembly language and machine code

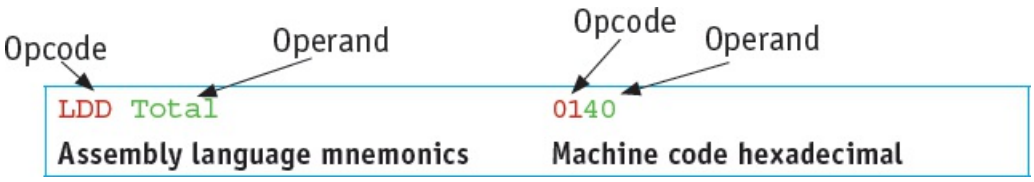
The only programming language that a CPU can use is **machine code**. Every different type of computer/chip has its own set of machine code **instructions**. A computer program stored in main memory is a series of machine code instructions that the CPU can automatically carry out during the fetch-execute cycle. Each machine code instruction performs one simple task, for example, storing a value in a memory location at a specified address. Machine code is binary, it is sometimes displayed on a screen as hexadecimal so that human programmers can understand machine code instructions more easily.

Writing programs in machine code is a specialised task that is very time consuming and often error prone, as the only way to test a program written in machine code is to run it and see what happens. In order to shorten the development time for writing computer programs, other programming languages were developed, where the instructions were easier to learn and understand. Any program not written in machine code needs to be translated before the CPU can carry out the instructions, so language translators were developed.

The first programming language to be developed was **assembly language**, this is closely related to machine code and uses mnemonics instead of binary.

LDD Total	0140	00000000110000000
ADD 20	0214	00000001000011000
STO Total	0340	00000001110000000
Assembly language mnemonics	**Machine code hexadecimal**	**Machine code binary**

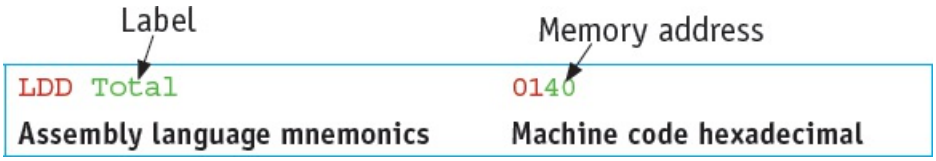
The structure of assembly language and machine code instructions is the same. Each instruction has an **opcode** that identifies the operation to be carried out by the CPU. Most instructions also have an **operand** that identifies the data to be used by the opcode.



4.2.2 Stages of assembly

Before a program written in assembly language (**source code**) can be executed, it needs to be translated into machine code. The translation is performed by a program called an **assembler**. An assembler translates each assembly language instruction into a machine code instruction. An assembler also checks the syntax of the assembly language program to ensure that only opcodes from the appropriate machine code **instruction set** are used. This speeds up the development time, as some errors are identified during translation before the program is executed.

There are two types of assembler: single pass assemblers and two pass assemblers. A single pass assembler puts the machine code instructions straight into the computer memory to be executed. A two pass assembler produces an object program in machine code that can be stored, loaded then executed at a later stage. This requires the use of another program called a loader. Two pass assemblers need to scan the source program twice, so they can replace labels in the assembly program with memory addresses in the machine code program.



Pass 1

- Read the assembly language program one line at a time.
- Ignore anything not required, such as comments.
- Allocate a memory address for the line of code.
- Check the opcode is in the instruction set.
- Add any new labels to the symbol table with the address, if known.
- Place address of labelled instruction in the symbol table.

Pass 2

- Read the assembly language program one line at a time.
- Generate **object code**, including opcode and operand, from the symbol table generated in Pass 1.
- Save or execute the program.

The second pass is required as some labels may be referred to before their address is known. For example, Found is a forward reference for the JPN instruction.

```
Label      Opcode Operand
Notfound: LDD    200
          CMP    #0
          JPN    Found
          JPE    Notfound
Found:    OUT
```

If the program is to be loaded at memory address 100, and each memory location contains 16 bits, the symbol table for this small section of program would look like this:

Label	Address
Notfound	100
Found	104

4.2.3 Assembly language instructions

There are different types of assembly language instructions. Examples of each type are given below.

Data movement instructions

These instructions allow data stored at one location to be copied into the accumulator. This data can then be stored at another location, used in a calculation, used for a comparison or output.

Instruction		Explanation
Opcode	**Operand**	
LDM	#n	Load the number into ACC (immediate addressing is used)
LDD	<address>	Load the contents of the specified address into ACC (direct or absolute addressing is used)
LDI	<address>	The address to be used is the contents of the specified address. Load the contents of the contents of the given address into ACC (indirect addressing is used)
LDX	<address>	The address to be used is the specified address plus the contents of the index register. Load the contents of this calculated address into ACC (indexed addressing is used)
LDR	#n	Load the number n into IX (immediate addressing is used)
LDR	ACC	Load the number in the accumulator into IX
MOV	<register>	Move the contents of the accumulator to the register (IX)
STO	<address>	Store the contents of ACC into the specified address (direct or absolute addressing is used)
END		Return control to the operating system
ACC is the single accumulator IX is the Index Register All numbers are denary unless identified as binary or hexadecimal B is a binary number, for example B01000011 & is a hexadecimal number, for example &7B # is a denary number		

Table 4.4 Data movement instructions

Input and output of data instructions

These instructions allow data to be read from the keyboard or output to the screen.

Instruction		Explanation
Opcode	**Operand**	
IN		Key in a character and store its ASCII value in ACC
OUT		Output to the screen the character whose ASCII value is stored in ACC
No opcode is required as a single character is either input to the accumulator or output from the accumulator		

Table 4.5 Input and output of data instructions

Arithmetic operation instructions

These instructions perform simple calculations on data stored in the accumulator and store the answer in the accumulator, overwriting the original data.

Instruction		Explanation
Opcode	**Operand**	
ADD	<address>	Add the contents of the specified address to the ACC (direct or absolute addressing is used)
ADD	#n	Add the denary number n to the ACC
SUB	<address>	Subtract the contents of the specified address from the ACC
SUB	#n	Subtract the number n from the ACC
INC	<register>	Add 1 to the contents of the register (ACC or IX)
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX)
Answers to calculations are always stored in the accumulator		

Table 4.6 Arithmetic operation instructions

Unconditional and conditional instructions

Instruction		Explanation
Opcode	**Operand**	
JMP	<address>	Jump to the specified address
JPE	<address>	Following a compare instruction, jump to the specified address if the comparison is True
JPN	<address>	Following a compare instruction, jump to the specified address if the comparison is False

END		Returns control to the operating system
Jump means change the PC to the address specified, so the next instruction to be executed is the one stored at the specified address, not the one stored at the next location in memory		

Table 4.7 Unconditional and conditional instructions

Compare instructions

Instruction		**Explanation**
Opcode	**Operand**	
CMP	<address>	Compare the contents of ACC with the contents of the specified address (direct or absolute addressing is used)
CMP	#n	Compare the contents of ACC with the number n
CMI	<address>	The address to be used is the contents of the specified address; compare the contents of the contents of the given address with ACC (indirect addressing is used)
The contents of the accumulator are always compared		

Table 4.8 Compare instructions

4.2.4 Addressing modes

Assembly language and machine code programs use different **addressing modes** depending on the requirements of the program.

Absolute addressing – the contents of the memory location in the operand are used. For example, if the memory location with address 200 contained the value 20, the assembly language instruction LDD 200 would store 20 in the accumulator.

Direct addressing – the contents of the memory location in the operand are used. For example, if the memory location with address 200 contained the value 20, the assembly language instruction LDD 200 would store 20 in the accumulator. Absolute and direct addressing are the same.

Indirect addressing – the contents of the contents of the memory location in the operand are used. For example, if the memory location with address 200 contained the value 20 and the memory location with address 20 contained the value 5, the assembly language instruction LDI 200 would store 5 in the accumulator.

Indexed addressing – the contents of the memory location found by adding the contents of the index register (IR) to the address of the memory location in the operand are used. For example, if IR contained the value 4 and memory location with address 204 contained the value 17, the assembly language instruction LDX 200 would store 17 in the accumulator.

Immediate addressing – the value of the operand only is used. For example, the assembly language instruction LDM #200 would store 200 in the accumulator.

Relative addressing – the memory address used is the current memory address added to the operand. For example, JMR #5 would transfer control to the instruction 5 locations after the current instruction.

Symbolic addressing – only used in assembly language programming. A label is used instead of a value. For example, if the memory location with address labelled MyStore contained the value 20, the assembly language instruction LDD MyStore would store 20 in the accumulator.

Labels make it easier to alter assembly language programs because when absolute addresses are used every reference to that address needs to be edited if an extra instruction is added, for example.

Label	**Instruction**		**Explanation**
	Opcode	**Operand**	
<label>:	<opcode>	<operand>	Labels an instruction
<label>:	n		Gives a symbolic address <label> to the memory location with the contents n

Table 4.9 Labels

4.2.5 Simple assembly language programs

A program written in assembly language will need many more instructions than a program written in a high-level language to perform the same task.

In a high-level language, adding three numbers together and storing the answer would typically be written as a single instruction:

```
total = first + second + third
```

The same task written in assembly language could look like this:

```
Label   Opcode  Operand
start:  LDD     first
        ADD     second
        ADD     third
        STO     total
        END

first:  #20
second: #30
third:  #40
total:  #0
```

If the program is to be loaded at memory address 100 after translation and each memory location contains 16 bits, the symbol table for this small section of program would look like this:

Label	Address
start	100
first	106
second	107
third	108
total	109

When this section of code is executed, the contents of ACC, CIR and the variables used can be traced using a trace table.

CIR	Opcode	Operand	ACC	first 106	second 107	third 108	total 109
100	LDD	first	20	20	30	40	0
101	ADD	second	50	20	30	40	0
102	ADD	third	90	20	30	40	0
103	STO	total	90	20	30	40	90
104	END						

In a high-level language, adding a list of numbers together and storing the answer would typically be written using a loop.

```
FOR counter = 1 TO 3
    total = total + number[counter]
NEXT counter
```

The same task written in assembly language would require the use of the index register (IX). The assembly language program could look like this:

Label	Opcode	Operand	Comment
	LDM	#0	Load 0 into ACC
	STO	total	Store 0 in total
	STO	counter	Store 0 in counter
	LDR	#0	Set IX to 0
loop:	LDX	number	Load the number indexed by IX into ACC
	ADD	total	Add total to ACC
	STO	total	Store result in total
	INC	IX	Add 1 to the contents of IX
	LDD	counter	Load counter into ACC
	INC	ACC	Add 1 to ACC
	STO	counter	Store result in counter
	CMP	#3	Compare with 3
	JPN	loop	If ACC not equal to 3 then return to start of loop
	END		
number:	#5		List of three numbers
	#7		
	#3		
counter:			counter for loop
total:			Storage space for total

If the program is to be loaded at memory address 100 after translation and each memory location contains 16 bits, the symbol table for this small section of program would look like this:

Label	Address
loop	104
number	115
counter	118
total	119

When this section of code is executed the contents of ACC, CIR, IX and the variables used can be traced using a trace table:

CIR	Opcode	Operand	ACC	IX	Counter 118	Total 119
100	LDM	#0	0			
101	STO	total	0			0
102	STO	counter	0		0	0
103	LDR	#0	0	0	0	0
104	LDX	number	5	0	0	0
105	ADD	total	5	0	0	0
106	STO	total	5	0	0	5
107	INC	IX	5	1	0	5
108	LDD	counter	0	1	0	5
109	INC	ACC	1	1	0	5
110	STO	counter	1	1	1	5
111	CMP	#3	1	1	1	5
112	JPN	loop	1	1	1	5
104	LDX	number	7	1	1	5
105	ADD	total	12	1	1	5
106	STO	total	12	1	1	12
107	INC	IX	12	2	1	12
108	LDD	counter	1	2	1	12
109	INC	ACC	2	2	1	12
110	STO	counter	2	2	2	12
111	CMP	#3	2	2	2	12
112	JPN	loop	2	2	2	12
104	LDX	number	3	2	2	12
105	ADD	total	15	2	2	12
106	STO	total	15	2	2	15
107	INC	IX	15	3	2	15
108	LDD	counter	2	3	2	15
109	INC	ACC	3	3	2	15
110	STO	counter	3	3	3	15
111	CMP	#3	3	3	3	15
112	JPN	loop	3	3	3	15
113	END					

ACTIVITY 4B

1 a) State the contents of the accumulator after the following instructions have been executed. The memory location with address 200 contains 300, the memory location with address 300 contains 50.

i) LDM #200

ii) LDD 200

iii) LDI 200

b) Write an assembly language instruction to:

i) compare the accumulator with 5

ii) jump to address 100 if the comparison is true.

2 a) Copy and complete the symbol table for this assembly language program. Assume that the translated program will start at memory address 100.

b) Complete a trace table to show the execution of this assembly language program.

c) State the task that this assembly language program performs.

```
Label    Opcode   Operand
         LDD      number1
         SUB      number2
         ADD      number3
         CMP      #10
         JPE      nomore
         ADD      number4
nomore:  STO      total
         END

number1: #30
number2: #40
number3: #20
number4: #50
total:   #0
```

3 a) Using the assembly language instructions given in this section, write an assembly language program to output the ASCII value of each element of an array of four elements.

b) Complete the symbol table for your assembly language program. Assume that the translated program will start at memory address 100.

c) Complete a trace table to show the execution of your assembly language program.

4.3 Bit manipulation

WHAT YOU SHOULD ALREADY KNOW

Try these two questions before you start the third part of this chapter.

1) Copy and complete the truth table for AND, OR and XOR.

		AND	OR	XOR
0	0			
0	1			
1	0			
1	1			

2) Identify **three** different types of shift used in computer programming.

Key terms

Shift – moving the bits stored in a register a given number of places within the register; there are different types of shift.

Logical shift – bits shifted out of the register are replaced with zeros.

Arithmetic shift – the sign of the number is preserved.

Cyclic shift – no bits are lost, bits shifted out of one end of the register are introduced at the other end of the register.

Left shift – bits are shifted to the left.

Right shift – bits are shifted to the right.

Monitor – to automatically take readings from a device.

Control – to automatically take readings from a device, then use the data from those readings to adjust the device.

Mask – a number that is used with the logical operators AND, OR or XOR to identify, remove or set a single bit or group of bits in an address or register.

4.3.1 Binary shifts

A **shift** involves moving the bits stored in a register a given number of places within the register. Each bit within the register may be used for a different purpose. For example, in the IR each bit identifies a different interrupt.

There are several different types of shift.

Logical shift – bits shifted out of the register are replaced with zeros. For example, an 8-bit register containing the binary value 10101111 shifted left logically three places would become 01111000.

Arithmetic shift – the sign of the number is preserved. For example, an 8-bit register containing the binary value 10101111 shifted right arithmetically three places would become 11110101. Arithmetic shifts can be used for multiplication or division by powers of two.

Cyclic shift – no bits are lost during a shift. Bits shifted out of one end of the register are introduced at the other end of the register. For example, an 8-bit register containing the binary value 10101111 shifted left cyclically three places would become 01111101.

Left shift – bits are shifted to the left; gives the direction of shift for logical, arithmetic and cyclic shifts.

Right shift – bits are shifted to the right; gives the direction of shift for logical, arithmetic and cyclic shifts.

Table 4.10 shows the logical shifts that you are expected to use in assembly language programming.

Instruction		Explanation
Opcode	**Operand**	
LSL	n	Bits in ACC are shifted logically n places to the left. Zeros are introduced on the right-hand end
LSR	n	Bits in ACC are shifted logically n places to the right. Zeros are introduced on the left-hand end
Shifts are always performed on the ACC		

Table 4.10 Logical shifts in assembly language programming

4.3.2 Bit manipulation used in monitoring and control

In **monitoring** and **control**, each bit in a register or memory location can be used as a flag and would need to be tested, set or cleared separately.

For example, a control system with eight different sensors would need to record when the data from each sensor had been processed. This could be shown using 8 different bits in the same memory location.

- AND is used to check if the bit has been set.
- OR is used to set the bit.
- XOR is used to clear a bit that has been set.

Table 4.11 shows the instructions used to check, set and clear a single bit or group of bits.

Instruction		Explanation
Opcode	**Operand**	
AND	n	Bitwise AND operation of the contents of ACC with the operand
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
XOR	n	Bitwise XOR operation of the contents of ACC with the operand
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>
OR	n	Bitwise OR operation of the contents of ACC with the operand
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>
The results of logical bit manipulation are always stored in the ACC. <address> can be an absolute address or a symbolic address. The operand is used as the **mask** to set or clear bits		

Table 4.11 Instructions used to check, set and clear a single bit or group of bits

The assembly language code to test sensor 3 could be:

```
Opcode Operand Comment
LDD    sensors Load content of sensors into ACC
AND    #B100   Mask to select bit 3 only
CMP    #B100   Check if bit 3 is set
JPN    process Jump to process routine if bit not set
LDD    sensors Load sensors into ACC
XOR    #B100   Clear bit 3 as sensor 3 has been processed
```

ACTIVITY 4C

1 a) State the contents of the accumulator after the following instructions have been executed. The accumulator contains B00011001.

i) LSL #4

ii) LSR #5

b) Write an assembly language instruction to:

i) set bit 4 in the accumulator

ii) clear bit 1 in the accumulator.

2 a) Describe the difference between *arithmetic shifts* and *logical shifts*.

b) Explain, with the aid of examples, how a *cyclic shift* works.

c) This register is shown before and after it has been shifted. Identify the type of shift that has taken place.

0	0	1	1	0	1	0	1

1	0	1	0	1	0	0	0

End of chapter questions

1 a) Write these six stages of the Von Neumann fetch-execute cycle in the correct order.

[6]

– instruction is copied from the MDR and is placed in the CIR
– the instruction is executed
– the instruction is decoded
– the address contained in PC is copied to the MAR
– the value in PC is incremented by 1
– instruction is copied from memory location in MAR and placed in MDR

b) Explain how the following affect the performance of a computer system.

i) Width of the data bus and address bus.

[2]

ii) The clock speed.

[2]

iii) Use of dual core or quad core processors.

[2]

c) A student accessed the BIOS on their computer. They increased the clock speed from 2.5 GHz to 3.2 GHz.
Explain the potential dangers in doing this.

[2]

2 a) Explain the main differences between HDMI, VGA and USB ports when sending data to peripherals.

[5]

b) Describe how interrupts can be used to service a printer printing out a large 1000 page document.

[5]

3 a) i) Name **three** special registers used in a typical processor.

[3]

ii) Explain the purpose of the three registers named in part i).

[3]

b) Explain how interrupts are used when a processor sends a document to a printer.

[4]

4 A programmer is writing a program in assembly language. They need to use shift instructions.
Describe, using examples, three types of shift instructions the programmer could use.

[6]

5 An intruder detection system for a large house has four sensors. An 8-bit memory location stores the output from each sensor in its own bit position.
The bit value for each sensor shows:

– 1 – the sensor has been triggered
– 0 – the sensor has not been triggered

The bit positions are used as follows:

Not used				Sensor 4	Sensor 3	Sensor 2	Sensor 1

The output from the intruder detection system is a loud alarm.

a) i) State the name of the type of system to which intruder detection systems belong.

[1]

ii) Justify your answer to part i).

[1]

b) Name **two** sensors that could be used in this intruder detection system.
Give a reason for your choice.

[4]

c) The intruder system is set up so that the alarm will only sound if two or more sensors have been triggered. An assembly language program has been written to process the contents of the memory location.
This table shows part of the instruction set for the processor used.

Instruction		**Explanation**
Opcode	**Operand**	
LDD	<address>	Direct addressing. Load the contents of the given address to ACC
STO	<address>	Store the contents of ACC at the given address

INC	<register>	Add 1 to the contents of the register (ACC or IX)
ADD	<address>	Add the contents of the given address to the contents of ACC
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>
CMP	#n	Compare the contents of ACC with the number n
JMP	<address>	Jump to the given address
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True
JGT	<address>	Following a compare instruction, jump to <address> if the content of ACC is greater than the number used in the compare instruction
END		End the program and return to the operating system

Part of the assembly code is:

	Opcode	**Operand**
SENSORS:		B00001010
COUNT:		0
VALUE:		1
LOOP:	LDD	SENSORS
	AND	VALUE
	CMP	#0
	JPE	ZERO
	LDD	COUNT
	INC	ACC
	STO	COUNT
ZERO:	LDD	VALUE
	CMP	#8
	JPE	EXIT
	ADD	VALUE
	STO	VALUE

	JMP	LOOP
EXIT:	LDD	COUNT
TEST:	CMP	…
	JGT	ALARM

i) Copy the table below and dry run the assembly language code.
Start at LOOP and finish when EXIT is reached.

[4]

BITREG	COUNT	VALUE	ACC
B00001010	0	1	

ii) The operand for the instruction labelled TEST is missing.
State the missing operand.

[1]

iii) The intruder detection system is improved and now has eight sensors. One instruction in the assembly language code will need to be amended.
Identify this instruction. Write the amended instruction.

[2]

5 System software

In this chapter, you will learn about

- why computers need an operating system
- key management tasks, such as memory management, file management, security management, hardware management and process management
- the need for utility software, including disk formatters, virus checkers, defragmentation software, disk content analyse and repair software, file compression and back-up software
- program libraries, software under development using program library software and the benefits to software developers, including the use of dynamic link library (DLL) files
- the need for these language translators: assemblers, compilers and interpreters
- the benefits and drawbacks of using compilers or interpreters
- an awareness that high level language programs may be partially compiled and partially interpreted (such as Java™)
- the features of a typical integrated development environment (IDE) for
  - coding (using context-sensitive prompts)
  - initial error detection (including dynamic syntax checks)
  - presentation (including pretty print, expand and collapse code blocks)
  - debugging (for example, single stepping, use of breakpoints, variables/expressions report windows).

5.1 Operating systems

WHAT YOU SHOULD ALREADY KNOW

Try these five questions before you read the first part of this chapter.

1 Microprocessors are commonly used to control microwave ovens, washing machines and many other household items.
Explain why it is **not** necessary for these devices to have an operating system.

2 **a)** Name **three** of the most common operating systems used in computers and other devices, such as mobile phones and tablets.

b) A manufacturer makes laptop computers, mobile phones and tablets.
Explain why it is necessary for the manufacturer to develop different versions of its operating system for use on its computers, mobile phones and tablets.

3 Most operating systems offer a graphic user interface (GUI) as well as a command line interface (CLI).

a) What are the main differences between the two types of interface?

b) What are the pros and cons of both types of interface?

c) Who would use each type of interface?

4 Before the advent of the operating system, computers relied on considerable human intervention.
Find out the methods used to start up early computers to prepare them for the day's tasks.

5 Describe the role of **buffers** and **interrupts** when a printing job is being sent to an inkjet printer.
Consider the different operational speeds of a processor and a printer, together with size of printing job and interrupt priorities.
Describe potential error scenarios – such as paper jam, out of paper or out of ink – and how these could affect the printing job.

Key terms

CMOS – complementary metal-oxide semiconductor.

Operating system – software that provides an environment in which applications can run and provides an interface between hardware and human operators.

HCI – human–computer interface.

GUI – graphical user interface.

CLI – command line interface.

Icon – small picture or symbol used to represent, for example, an application on a screen.

WIMP – windows, icons, menu and pointing device.

Post-WIMP – interfaces that go beyond WIMP and use touch screen technology rather than a

pointing device.

Pinching and rotating – actions by fingers on a touch screen to carry out tasks such as move, enlarge, reduce, and so on.

Memory management – part of the operating system that controls the main memory.

Memory optimisation – function of memory management that determines how memory is allocated and deallocated.

Memory organisation – function of memory management that determines how much memory is allocated to an application.

Security management – part of the operating system that ensures the integrity, confidentiality and availability of data.

Contiguous – items next to each other.

Virtual memory systems – memory management (part of OS) that makes use of hardware and software to enable a computer to compensate for shortage of actual physical memory.

Memory protection – function of memory management that ensures two competing applications cannot use same memory locations at the same time.

Process management – part of the operating system that involves allocation of resources and permits the sharing and exchange of data.

Hardware management – part of the operating system that controls all input/output devices connected to a computer (made up of sub-management systems such as printer management, secondary storage management, and so on).

Device driver – software that communicates with the operating system and translates data into a format understood by the device.

Utility program – parts of the operating system which carry out certain functions, such as virus checking, defragmentation or hard disk formatting.

Disk formatter – utility that prepares a disk to allow data/files to be stored and retrieved.

Bad sector – a faulty sector on an HDD which can be soft or hard.

Antivirus software – software that quarantines and deletes files or programs infected by a virus (or other malware). It can be run in the background or initiated by the user.

Heuristic checking – checking of software for behaviour that could indicate a possible virus.

Quarantine – file or program identified as being infected by a virus which has been isolated by antivirus software before it is deleted at a later stage.

False positive – a file or program identified by a virus checker as being infected but the user knows this cannot be correct.

Disk defragmenter – utility that reorganises the sectors on a hard disk so that files can be stored in contiguous data blocks.

Disk content analysis software – utility that checks disk drives for empty space and disk usage by reviewing files and folders.

Disk compression – software that compresses data before storage on an HDD.

Back-up utility – software that makes copies of files on another portable storage device.

Program library – a library on a computer where programs and routines are stored which can be freely accessed by other software developers for use in their own programs.

Library program – a program stored in a library for future use by other programmers.

Library routine – a tested and ready-to-use routine available in the development system of a programming language that can be incorporated into a program.

Dynamic link file (DLL) – a library routine that can be linked to another program only at the run time stage.

5.1.1 The need for an operating system

Early computers had no operating system at all. Control software had to be loaded each time the computer was started – this was done using either paper tape or punched cards.

In the 1970s, the home computer was becoming increasingly popular. Early examples, such as the Acorn BBC B, used an internal ROM chip to store part of the operating system. A cassette tape machine was also used to load the remainder of the operational software (see Figure 5.1). This was necessary to 'get the computer started' and used a welcome cassette tape which had to be used each time the computer was turned on.

Figure 5.1 An Acorn BBC B (left) and its cassette tape machine (right)

As the hard disk drive (HDD) was developed, operating systems were stored on the hard disk, and start-up of the motherboard was handled by the basic input/output system (BIOS). Initially, the BIOS was stored on a ROM chip but, in modern computers, the BIOS contents are stored on a flash memory chip. The BIOS configuration is stored in **CMOS memory (complementary metal-oxide semiconductor)** which means it can be altered or deleted as required.

The required part of the operating system is copied into RAM – since operating systems are now so large, it would seriously affect a computer's performance if it was all loaded into RAM at once. An **operating system** provides both the environment in which applications can be run, and a useable interface between humans and computer. An operating system also disguises the complexity of computer hardware. Common examples include Microsoft Windows®, Apple Mac OS, Google Android and IOS (Apple mobile phones and tablets).

The **human–computer interface (HCI)** is usually achieved through a **graphical user interface (GUI)**, although it is possible to use a **command line interface (CLI)** if the user wishes to directly communicate with the computer.

A CLI requires a user to type instructions to choose options from menus, open software, and so on. There are often a number of commands that need to be typed; for example, to save or load a file. The user, therefore, has to learn a number of commands (which must be typed exactly with no errors) just to carry out basic operations. Furthermore, it takes time to key in commands every time an operation has to be carried out.

The advantage of CLI is that the user is in direct communication with the computer and is not restricted to a number of pre-determined options.

For example, the following section of CLI imports data from table A into table B. It shows how

complex it is just to carry out a straightforward operation.

```
1. SQLPrepare(hStmt,
2. ?   (SQLCHAR *) "INSERT INTO tableB SELECT * FROM tableA",
3. ?   SQL_NTS):
4. ?   SQLExecute(hStmt);
```

A GUI allows the user to interact with a computer (or MP3 player, gaming device, mobile phone, and so on) using pictures or symbols (**icons**). For example, the whole of the above CLI code could have been replaced by a single icon, like the one on the left.

Table update

Selecting this icon would execute all of the steps shown in the CLI without the need to type them.

GUIs use various technologies and devices to provide the user interface. One of the first commonly used GUI environments was known as **windows, icons, menu and pointing device (WIMP)**, which was developed for use on personal computers (PCs). Here, a mouse is used to control a cursor and icons are selected to open and run windows. Each window contains an application. Modern computer systems allow several windows to be open at the same time. An example is shown in Figure 5.2.

Figure 5.2 An example of WIMP

A windows manager looks after the interaction between windows, the applications and windowing system (which handles the pointing devices and the cursor's position).

However, smart phones, tablets and many computers now use a **post-WIMP** interaction where fingers are in contact with the screen, allowing actions such as **pinching** and **rotating** which are difficult using a single pointer and device such as a mouse. Also, simply tapping the icon with a finger (or stylus) will launch the application. Developments in touch screen technology mean these flexible HCIs are now readily available.

5.1.2 Operating system tasks

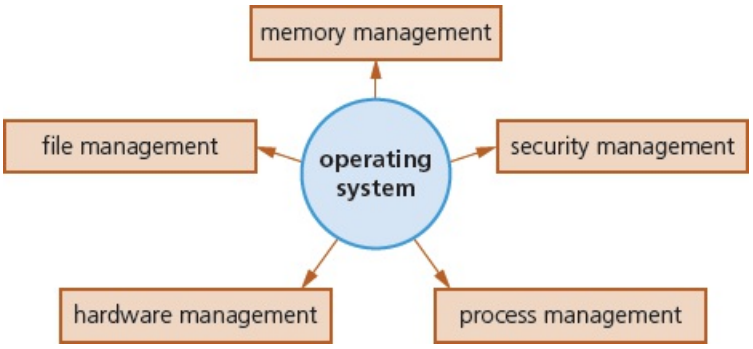


Figure 5.3 Operating system tasks

Memory management

Memory management, as the name suggests, is the management of a computer's main memory. This can be broken down into three parts: **memory optimisation**, **memory organisation** and **memory protection**.

Memory optimisation

Memory optimisation is used to determine how computer memory is allocated and deallocated when a number of applications are running simultaneously. It also determines *where* they are stored in memory. It must, therefore, keep track of all allocated memory and free memory available for use by applications. To maintain optimisation of memory, it will also swap data to and from the HDD or SSD.

Memory organisation

Memory organisation determines how much memory is allocated to an application, and how the memory can be split up in the most appropriate or efficient manner.

This can be done with the use of

- a single (**contiguous**) allocation, where all of the memory is made available to a single application. This is used by MS-DOS and by embedded systems
- partitioned allocation, where the memory is split up into contiguous partitions (or blocks) and memory management then allocates a partition (which can vary in size) to an application
- paged memory, which is similar to partitioned allocation, but each partition is of a fixed size. This is used by **virtual memory systems**
- segmented memory, which is different because memory blocks are not contiguous – each segment of memory will be a logical grouping of data (such as the data which may make up an array).

Memory protection

Memory protection ensures that two competing applications cannot use the same memory locations at the same time. If this was not done, data could be lost, applications could produce incorrect results, there could be security issues, or the computer may crash.

Memory protection and memory organisation are different aspects of an operating system. An operating system may use a typical type of memory organisation (for example, it may use paging or segmentation) but it is always important that no two applications can occupy the same part of memory. Therefore, memory protection must *always* be a part of any type of memory organisation used.

Figure 5.4 shows how different applications can be kept separate from each other.

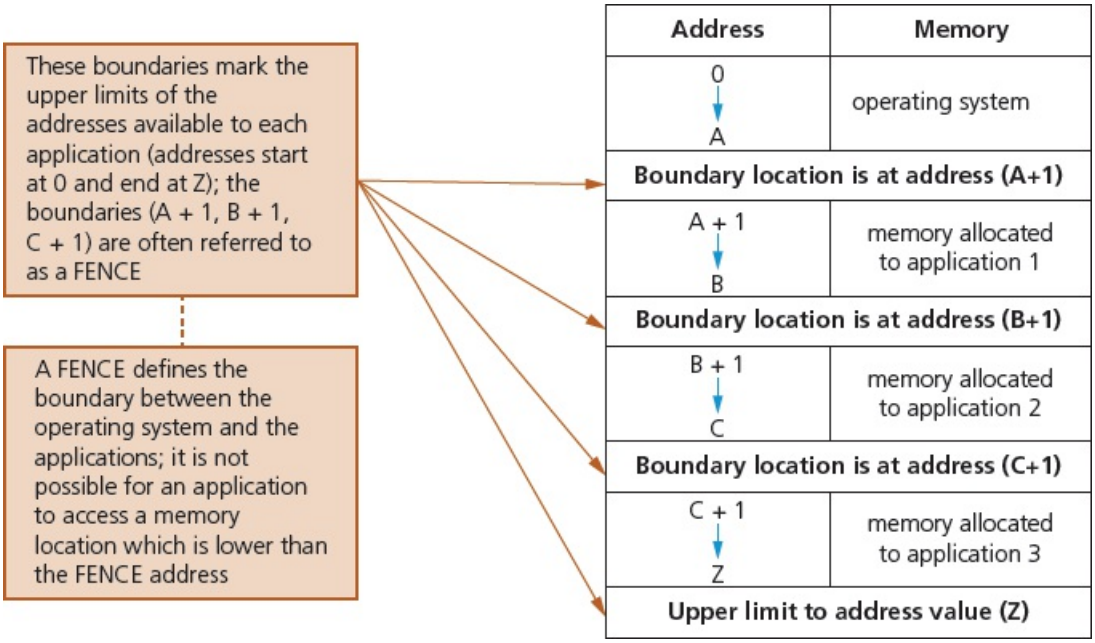


Figure 5.4 Memory protection

Security management

Security management is another part of a typical operating system. The function of security management is to ensure the integrity, confidentiality and availability of data.

This can be achieved by

- carrying out operating system updates as and when they become available
- ensuring that antivirus software (and other security software) is always up-to-date
- communicating with, for example, a firewall to check all traffic to and from the computer
- making use of privileges to prevent users entering 'private areas' on a computer which permits multi-user activity (this is done by setting up user accounts and making use of passwords and user IDs). This helps to ensure the privacy of data
- maintaining access rights for all users
- offering the ability for the recovery of data (and system restore) when it has been lost or corrupted
- helping to prevent illegal intrusion to the computer system (also ensuring the privacy of data).

Note: many of these features are covered in more depth elsewhere in this chapter or in other chapters.

EXTENSION ACTIVITY 5A

While working through the remainder of Chapters 5 and 6, find out all of the methods available to ensure the security, privacy and integrity of data and how these link into the operating system security management. It is important to distinguish between what constitutes security, privacy and integrity of data.

Process management

A process is a program which is being run on a computer. **Process management** involves the allocation of resources and permits the sharing and exchange of data, thus allowing all processes to be fully synchronised (for example, by the scheduling of resources, resolution of software conflicts, use of queues and so on). This is covered in more depth in Chapter 16.

Hardware management

Hardware management involves all input and output peripheral devices.

The functions of hardware management include

- communicating with all input and output devices using **device drivers**
- translating data from a file (defined by the operating system) into a format that the input/output device can understand using device drivers
- ensuring each hardware resource has a priority so that it can be used and released as required.

The management of input/output devices is essentially the control and management of queues and buffers. For example, when printing out a document, the printer management

- locates and loads the printer driver into memory
- sends data to a printer buffer ready for printing
- sends data to a printer queue (if the printer is busy or the print job has a low priority) before sending to the printer buffer
- sends various control commands to the printer throughout the printing process
- receives and handles error messages and interrupts from the printer.

EXTENSION ACTIVITY 5B

Write down the tasks carried out by a keyboard manager when a user types text using a word processor. Consider the use of buffers and queues in your answer.

File management

The main tasks of file management include

- defining the file naming conventions which can be used (filename.docx, where the extension can be .bat, .htm, .dbf, .txt, .xls, and so on)
- performing specific tasks, such as create, open, close, delete, rename, copy, move
- maintaining the directory structures
- ensuring access control mechanisms are maintained, such as access rights to files, password

protection, making files available for editing, locking files, and so on

- specifying the logical file storage format (such as FAT or NTFS if Windows is being used), depending on which type of disk formatter is used (see Section 5.1.3)
- ensuring memory allocation for a file by reading it from the HDD/SSD and loading it into memory.

5.1.3 Utility software

Computer users are provided with a number of **utility programs** that are part of the operating system. However, users can also install their own utility software in addition. This software is usually initiated by the user, but some, such as virus checkers, can be set up to constantly run in the background. Utility software offered by most operating systems includes

- hard disk formatter
- virus checker
- defragmentation software
- disk contents analysis/repair software
- file compression
- back-up software.

Hard disk formatter

A new hard disk drive needs to be initialised ready for formatting. The operating system needs to know how to store files and where the files will be stored on the hard disks. A **disk formatter** will organise storage space by assigning it to data blocks (partitions). A disk surface may have a number of partitions (see Chapter 3 for more details regarding the organisation of data on hard disks). Note that partitions are contiguous blocks of data.

Once the partitions have been created, they must be formatted. This is usually done by writing files which will hold directory data and tables of contents (TOC) at the beginning of each partition. This allows the operating system to recognise a file and know where to find it on the disk surface. Different operating systems will use different filing systems; Windows, for example, uses new technology filing system (NTFS).

When carrying out full formatting using NTFS, all disk sectors are filled with zeros; these zeros are read back, thus testing the sector, but any data already stored there will be lost. So, it is important to remember that reformatting an HDD which has already been used will result in loss of data during the formatting procedure.

Disk formatters also have checking tools, which are non-destructive tests that can be carried out on each sector. If any **bad sector** errors are discovered, the sectors will be flagged as ‘bad’ and the file tracking records will be reorganised – this is done by replacing the bad sectors with new unused sectors, effectively repairing the faulty disk. A damaged file will now contain an ‘empty’ sector, which allows the file to be read but it will be corrupted since the bad sector will have contained important (and now lost) data. It would, therefore, be prudent to delete the damaged file leaving the rest of the HDD effectively repaired.

Bad sectors can be categorised as hard or soft. There are a number of ways that they can be produced, as shown in Table 5.1.

Hard bad sectors (difficult to repair)	Soft bad sectors
• caused by manufacturing errors • damage to disk surface caused by allowing the read-	• sudden loss of power leading to data corruption in some of the sectors

write head to touch the disk surface (for example, by moving HDD without first parking the read-write head) • system crash which could lead to damage to the disk surface(s)	• effect of static electricity leading to corruption of data in some of the sectors on the hard disk surfaces

▲ **Table 5.1** Hard and soft bad sectors

Virus checkers

Any computer (including mobile phones and tablets) can be subject to a virus attack (see Chapter 6).

There are many ways to help prevent viruses, such as being careful when downloading material from the internet, not opening files or links in emails from unknown senders, and by only using certified software. However, virus checkers – which are offered by operating systems – still provide the best defence against malware, as long as they are kept up to date and constantly run in the background.

Running **antivirus software** in the background on a computer will constantly check for virus attacks. Although various types of antivirus software work in different ways, they have some common features. They

- check software or files before they are run or loaded on a computer
- compare possible viruses against a database of known viruses
- carry out **heuristic checking** – this is the checking of software for types of behaviour that could indicate a possible virus, which is useful if software is infected by a virus not yet on the database
- put files or programs which may be infected into **quarantine**, to
  - automatically delete the virus, or
  - allow the user to decide whether to delete the file (it is possible that the user knows that the file or program is not infected by a virus – this is known as a **false positive** and is one of the drawbacks of antivirus software).

Antivirus software needs to be kept up to date since new viruses are constantly being discovered. Full system checks need to be carried out once a week, for example, since some viruses lie dormant and would only be picked up by this full system scan.

Defragmentation software

As an HDD becomes full, blocks used for files will become scattered all over the disk surface (in potentially different sectors and tracks as well as different surfaces). This happens as files are deleted, partially-deleted, extended and so on. The consequence is slower data access time: the HDD read-write head requires several movements just to find and retrieve the data making up the required file. It would be advantageous if files could be stored in contiguous sectors, considerably reducing HDD head movements.

Note that, due to their different operation when accessing data, this is less of a problem with SSDs.

Consider the following example using a disk with 12 sectors per surface.

We have three files (1, 2 and 3) stored on track 8 of the disk surface.

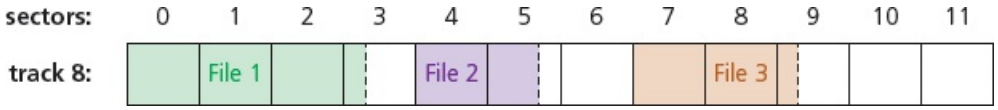


Figure 5.5

File 2 is deleted by the user and file 1 has data added to it. However, the file 2 sectors which become vacant are not filled up straight away by new file 1 data since this would require 'too much effort' for the HDD resources.

We get the following.

Figure 5.6

File 1 has been extended to write data in sectors 10 and 11.

Now, suppose file 3 is extended with the equivalent of 3.25 blocks of data. This requires filling up sector 9 and then moving to some empty sectors to write the remainder of the data – the next free sectors are on track 11.

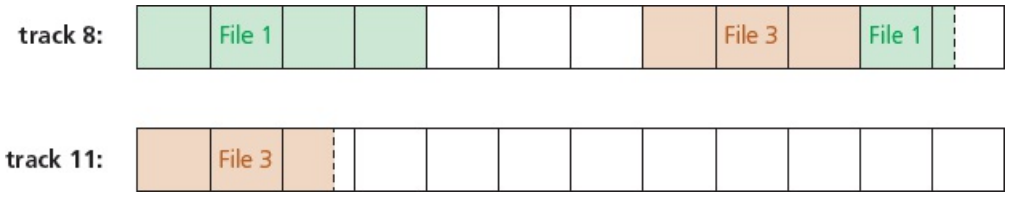


Figure 5.7

If this continues, the files just become more and more scattered throughout the disk surfaces. It is possible for sectors 4, 5 and 6 (on track 8) to eventually become used if the disk starts to fill up and it has to use up whatever space is available. A **disk defragmenter** will rearrange the blocks of data to store files in contiguous sectors wherever possible; however, if the disk drive is almost full, defragmentation may not work. Assuming we can carry out defragmentation, then track 8 now becomes:

Figure 5.8

This allows for much faster data access and retrieval since the HDD now requires fewer read-write head movements to access and read files 1 and 3. Some defragmenters also carry out clean up operations. Data blocks can become damaged after several read/write operations (this is different to bad sectors). If this happens, they are flagged as 'unusable' and any subsequent write operation will avoid writing data to data blocks which have become affected.

Disk content analysis/repair software

The concept of disk repair software was discussed in the above section. **Disk content analysis software** is used to check disk drives for empty space and disk usage by reviewing files and file folders. This can lead to optimal use of disk space by the removal of unwanted files and downloads (such as the deletion of auto saving files, cookies, download files, and so on).

Disk compression and file compression

File compression is essential to save storage space and make it quicker to download/upload files and quicker to send files via email. It was discussed in Chapter 1.

Disk compression is much less common these days due to the vast size of HDDs (often more than 2 TB). The disk compression utility compresses data before writing it to hard disk (and decompresses it again when reading this data). It is a high priority utility and will essentially override all other operating system routines – this is essential because all applications need to have access to the HDD. It is important not to uninstall disk compression software since this would render any previously saved data to be unreadable.

Back-up software

While it is sensible to take manual back-ups using, for example, a memory stick or portable HDD, it is also good practice to use the operating system **back-up utility**. This utility will

- allow a schedule for backing up files to be made
- only carry out a back-up procedure if there have been any changes made to a file.

For total security, there should be three versions of a file:

1. The current (working) version stored on the internal HDD.
2. A locally backed up copy of the file (stored on a portable SSD, for example).
3. A remote back-up version stored well away from the computer (using cloud storage, for example).

Windows environment offers the following facilities using the back-up utility:

- The ability to restore data, files or the computer from the back-up (useful if there has been a problem and files have been lost and need to be recovered).
- The ability to create a restore point (this restores a computer to its state at some point in the past; this can be very useful if a very important file has been deleted and cannot be recovered by any of the other utilities).
- Options of where to save back-up files; this can be set up from the utility to ensure files are automatically backed up to a chosen device.

Windows uses *File History*, which takes snapshots of files and stores them on an external HDD at regular intervals. Over a period of time, *File History* builds up a vast library of past versions of files – this allows a user to choose which version of the file they want to use. *File History* defaults to backing up every hour and retains past versions of files forever unless the user changes the settings.

Mac OS offers the *Time Machine* back-up utility. This erases the contents of a selected drive and replaces them with the contents from the back-up. To use this facility it is necessary to have an external HDD or SSD (connected via USB port) and ensure that the *Time Machine* utility is installed and activated on the selected computer. *Time machine* will automatically

- back up every hour
- keep daily back-ups for the past month, and
- keep weekly back-ups for all the previous months.

Note that once the back-up HDD or SSD is almost full, the oldest back-ups are deleted and replaced with the newest back-up data. Figure 5.9 shows the *Time Machine* message:

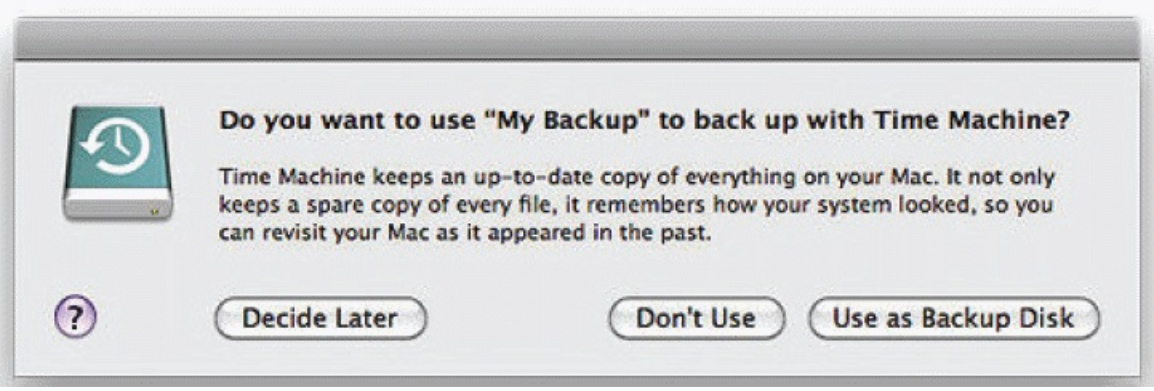


Figure 5.9 Screen shot of *Time Machine* message

5.1.4 Program libraries

Program libraries are used

- when software is under development and the programmer can utilise pre-written subroutines in their own programs, thus saving considerable development time
- to help a software developer who wishes to use dynamic link library (DLL) subroutines in their own program, so these subroutines must be available at run time.

When software routines are written (such as a sort routine), they are frequently saved in a program library for future use by other programmers. A program stored in a program library is known as a **library program**. We also have the term **library routines** to describe subroutines which could be used in another piece of software under development.

Suppose we are writing a game for children with animated graphics (of a friendly panda) using music routines and some scoreboard graphics.

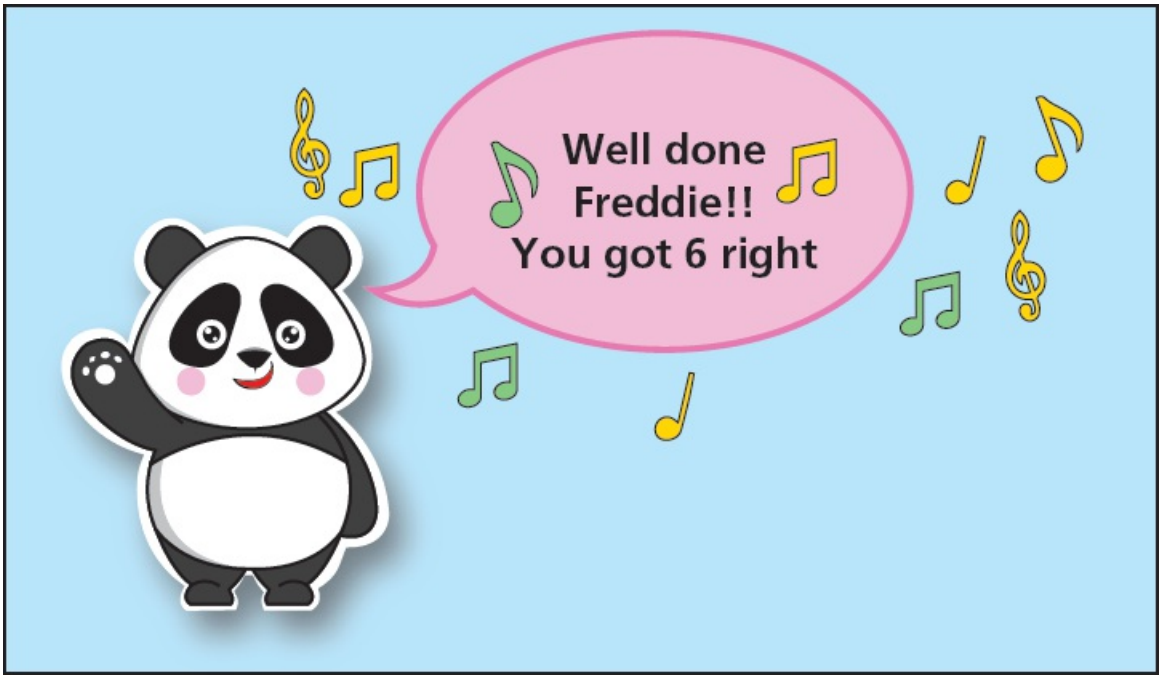


Figure 5.10

This game could be developed using existing routines from a library.

This game could be developed using existing routines from a library.

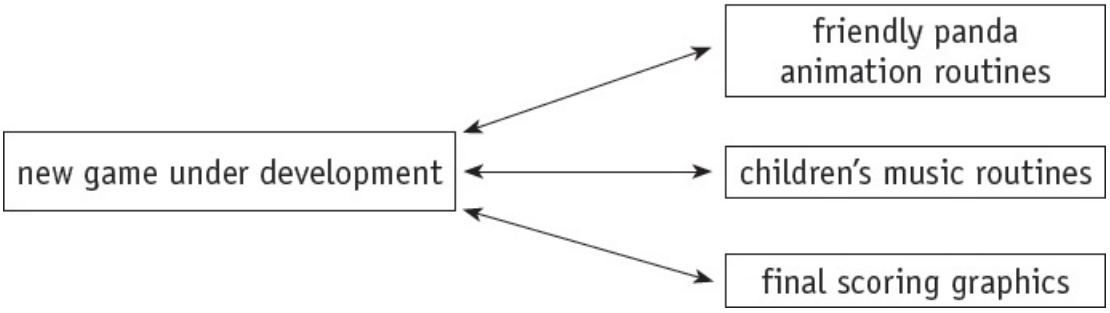


Figure 5.11

Developing software in this way

- removes the need to rewrite the many routines every single time (thus saving considerable time and cost)
- leads to modular programming, which means several programmers can be working on the same piece of software at the same time
- allows continuity with other games that may form part of a whole range (in education, where there may be a whole suite of programs, for example)
- allows the maintenance of a 'corporate image' in all the software being developed by a particular company
- saves considerable development time having to test each routine, since the routines are all fully tested in other software and should be error-free.

All operating systems have two program libraries containing library programs and library routines: static and dynamic.

In static libraries, software being developed is linked to executable code in the library at the time of compilation. So the library routines would be embedded directly into the new program code.

In dynamic libraries, software being developed is not linked to the library routines until actual run time (these are known as **dynamic link library files** or **DLL**). These library routines would be stand-alone files only being accessed as required by the new program – the routines will be available to several applications at the same time.

When using DLL, since the library routines are not loaded into RAM until required, memory is saved, and software runs faster. For example, suppose we are writing new software which allows access to a printer as part of its specification. The main program will be developed and compiled. Once the object code is run, it will only access (and load up) the printer routine from DLL when required by the user of the program. The main program will only contain a link to the printer library routine and will not contain any of the actual printer routine coding in the main body. Table 5.2 summarises the pros and cons of using DLL files.

Pros of using DLL files	Cons of using DLL files
the executable code of the main program is much smaller since DLL files are only loaded into memory at run time	the executable code is not self-contained, therefore all DLL files need to be available at run time otherwise error messages (such as

	missing .dll error) will be generated and the software may even crash
it is possible to make changes to DLL files independently of the main program, consequently if any changes are made to the DLL files it will not be necessary to recompile the main program	any DLL linking software in the main program needs to be available at run time to allow links with DLL files to be made
DLL files can be made available to a number of applications at the same time	if any of the DLL files have been changed (either intentionally or through corruption) this could lead to the main program giving unexpected results or even crashing
all of the above save memory and also save execution time	malicious changes to DLL files could be due to the result of malware, thus presenting a risk to the main program following the linking process

Table 5.2 Pros and cons of using DLL files

ACTIVITY 5A

1 a) i) Explain why a computer needs an operating system.

ii) Name **two** management tasks carried out by the operating system.

b) A new program is to be written in a high level language. The developer has decided to use DLL files in the design of the new program.

i) Explain what is meant by a DLL file. How does this differ from a static library routine?

ii) Describe **two** potential drawbacks of using DLL files in the new program.

2 A company produces glossy geography magazines. Each magazine is produced using a network of computers where thousands of photographs and drawings need to be stored. The computers also have an external link to the internet.
Name, and describe the function of, **three** utility programs the company would use on all its computers.

3 A computer user has a number of important issues, listed below.
For each issue, name a utility which could help solve it. Give a reason for each choice.

a) The user wants to send a number of very large attachments by email, but the recipient cannot accept attachments greater than 20 MB.

b) The user has accidentally deleted files in the past. It is essential that this cannot happen in the future.

c) The user has had their computer for a number a years. The time to access and retrieve data from the hard disk drive is increasing.

d) Last week, the user clicked on a link in an email from a friend, since then the user's computer is running slowly, files are being lost, and they are receiving odd messages.

e) Some of the files on the user's HDD have corrupted and will not open and this is affecting the performance of the HDD.

5.2 Language translators

WHAT YOU SHOULD ALREADY KNOW

Try these two questions before you read the second part of this chapter:

1 **a)** Name **two** types of language translator.

b) Identify a method, other than using a translator, of executing a program written in a high-level language.

2 Most modern language translators offer an Integrated Development Environment (IDE) for program development.

a) Which IDE are you using?

b) Describe **five** features offered by the IDE you use.

c) Which feature do you find most useful? Why is it useful to you?

Key terms

Translator – the systems software used to translate a source program written in any language other than machine code.

Compiler – a computer program that translates a source program written in a high-level language to machine code or p-code, object code.

Interpreter – a computer program that analyses and executes a program written in a high-level language line by line.

Prettyprinting – the practice of displaying or printing well set out and formatted source code, making it easier to read and understand.

Integrated development environment (IDE) – a suite of programs used to write and test a computer program written in a high-level programming language.

Syntax error – an error in the grammar of a source program.

Logic error – an error in the logic of a program.

Debugging – the process of finding logic errors in a computer program by running or tracing the program.

Single stepping – the practice of running a program one line/instruction at a time.

Breakpoint – a deliberate pause in the execution of a program during testing so that the contents of variables, registers, and so on can be inspected to aid debugging.

Report window – a separate window in the run-time environment of the IDE that shows the contents of variables during the execution of a program.

5.2.1 Translation and execution of programs

Instructions in a program can only be executed when written in machine code and loaded into the main memory of a computer. Programming instructions written in any programming language other than machine code must be translated before they can be used. The systems software used to translate a source program written in any language other than machine code are **translators**. There are three types of translator available, each translator performs a different role.

Assemblers

Programs written in assembly language are translated into machine code by an assembler program. Assemblers either store the program directly in main memory, ready for execution, as it is translated, or they store the translated program on a storage medium to be used later. If stored for later use, then a loader program is also needed to load the stored translated program into main memory before it can be executed. The stored translated program can be executed many times without being re-translated.

Every different type of computer/chip has its own machine code and assembly language. For example, MASM is an assembler that is used for the X86 family of chips, while PIC and GENIE are used for microcontrollers. Assembly language programs are machine dependent; they are not portable from one type of computer/chip to another.

Here is a short sample PIC assembly program:

```
        movlw B'00000000'
        tris  PORTB
        movlw B'00000011'
        movwf PORTB
stop:   goto  stop
```

EXTENSION ACTIVITY 5C

Find out what task this very short sample PIC assembly program is performing.

Assembly language programs are often written for tasks that need to be speedily executed, for example, parts of an operating system, central heating system or controlling a robot.

Compilers and interpreters

Programs written in a high-level language can be either translated into machine code by a **compiler** program, or directly executed line-by-line using an **interpreter** program.

Compilers usually store the translated program (object program) on a storage medium ready to be executed later. A loader program is needed to load the stored translated program into main memory before it can be executed. The stored translated program can be executed many times without being retranslated. The program will only need to be retranslated when changes are made to the source code.

With an interpreter, no translated program is generated in main memory or stored for later use. Every line in a program is interpreted then executed each time the program is run.

High-level language programs are machine independent, portable and can be run on any type of computer/chip, provided there is a compiler or interpreter available. For example, Java, Python and Visual Basic® (VB) are high-level languages often used for teaching programming.

EXTENSION ACTIVITY 5D

Find out about three more high-level programming languages that are being used today.

The similarities and differences between assemblers, compilers and interpreters are shown in Table 5.3.

	Assembler	Compiler	Interpreter
Source program written in	assembly language	high-level language	high-level language
Machine dependent	yes	no	no
Object program generated	yes, stored on disk or in main memory	yes, stored on disk or in main memory	no, instructions are executed under the control of the interpreter
Each line of the source program generates	one machine code instruction, one to one translation	many machine code instructions, instruction explosion	many machine code instructions, instruction explosion

Table 5.3 Similarities and differences between assemblers, compilers and interpreters

5.2.2 Pros and cons of compiling or interpreting a program

Both compilers and interpreters are used for programs written in high-level languages. Some **integrated development environments (IDEs)** have both available for programmers, since interpreters are most useful in the early stages of development and compilers produce a stand-alone program that can be executed many times without needing the compiler.

Table 5.4 shows the pros (in the blue cells) and cons (in the white cells) of compilers and interpreters.

Compiler	Interpreter
The end user only needs the executable code, therefore, the end user benefits as there is no need to purchase a compiler to translate the program before it is used.	The end user will need to purchase a compiler or an interpreter to translate the source code before it is used.
The developer keeps hold of the source code, so it cannot be altered or extended by the end user, therefore, the developer benefits as they can charge for upgrades and alterations.	The developer relinquishes control of the source code, making it more difficult to charge for upgrades and alterations. Since end users can view the source code, they could potentially use the developer's intellectual property.
Compiled programs take a shorter time to execute as translation has already been completed and the machine code generated may have been optimised by the compiler.	An interpreted program can take longer to execute than the same program when compiled, since each line of the source code needs to be translated before it is executed every time the program is run.
Compiled programs have no syntax or semantic errors.	Interpreted programs may still contain syntax or semantic errors if any part of the program has not been fully tested, these errors will need to be debugged.
The source program can be translated on one type of computer then executed on another type of computer.	Interpreted programs cannot be interpreted on one type of computer and run on another type of computer.
A compiler finds all errors in a program. One error detected can mean that the compiler finds other dependent errors later on in the program that will not be errors when the first error is corrected. Therefore, the number of errors found may be more than the actual number of errors.	**It is easier to develop and debug a program using an interpreter as errors can be corrected on each line and the program restarted from that place, enabling the programmer to easily learn from any errors.**

Untested programs with errors may cause the computer to crash.	**Untested programs should not be able to cause the computer to crash.**
The developer needs to write special routines in order to view partial results during development, making it more difficult to assess the quality of particular sections of code.	**Partial results can be viewed during development, enabling the developer to make informed decisions about a section of code, for example whether to continue, modify, or scrap and start again.**
End users do not have access to the source code and the run-time libraries, meaning they are unable to make modifications and are reliant on the developer for updates and alterations.	**If an interpreted program is purchased, end users have all the source code and the run-time libraries, enabling the program to be modified as required without further purchase.**

Table 5.4 Pros (blue cells) and cons (white cells) of compilers and interpreters.

5.2.3 Partial compiling and interpreting

In order to achieve shorter execution times, many high-level languages programs use a system that is partially compilation and partially interpretation. The source code is checked and translated by a compiler into object code. The compiled object code is a low-level machine independent code, called intermediate code, p-code or bytecode. To execute the program, the object code can be interpreted by an interpreter or compiled using a compiler.

For example, Java and Python programs can be translated by a compiler into a set of instructions for a virtual machine. These instructions, called bytecode, are then interpreted by an interpreter.

Below are examples of Java and Python intermediate code (bytecode):

Source code:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello World");
  }
}
```

Bytecode:

```
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0: aload_0
   1: invokespecial   #1; //Method java/lang/
Object."<init>":()V
   4: return
public static void main(java.lang.String[]);
  Code:
   0: getstatic       #2; //Field java/lang/System.
out:Ljava/io/PrintStream;
   3: ldc         #3; //String Hello World
   5: invokevirtual   #4; //Method java/io/PrintStream.
println:(Ljava/lang/String;)V
   8: return
```

Source code:

```
print ("Hello World")
```

Bytecode:

```
1   0 LOAD_NAME       0 (print)
    2 LOAD_CONST      0 ('Hello World')
    4 CALL_FUNCTION   1 (1 positional, 0 keyword pair)
    6 RETURN_VALUE
```

EXTENSION ACTIVITY 5E

Visual Basic also has an interpreter for bytecode. Find an example of bytecode for Visual Basic. See if you can find the bytecode for displaying 'Hello World' on the screen as in the Python example above.

5.2.4 Integrated development environment (IDE)

An integrated development environment (IDE) is used by programmers to aid the writing and development of programs. There are many different IDEs available; some just support one programming language, others can be used for several different programming languages. NetBeans®, PyCharm®, Visual Studio® and SharpDevelop are all IDEs currently in use.

EXTENSION ACTIVITY 5F

In small groups investigate different IDEs. See how many different features are available for your group's IDE and identify which programming language(s) are supported. Compare the features of the IDE investigated by your group with the IDEs investigated by other groups in the class.

IDEs usually have

- a source code editor
- a compiler, an interpreter, or both
- a run-time environment with a debugger
- an auto-documenter.

Source code editor

A source code editor allows a program to be written and edited without the need to use a separate text editor. The use of an integrated source code editor speeds up the development process, as editing can be done without changing to a different piece of software each time the program needs correcting or adding to. Most source code editors colour code the words in the program and layout the program in a meaningful way (**prettyprinting**). Some source code editors also offer context sensitive prompts with text completion for variable names and reserved words, and provide dynamic syntax checking. Figures 5.12 and 5.13 show these features in the PyCharm source code editor.

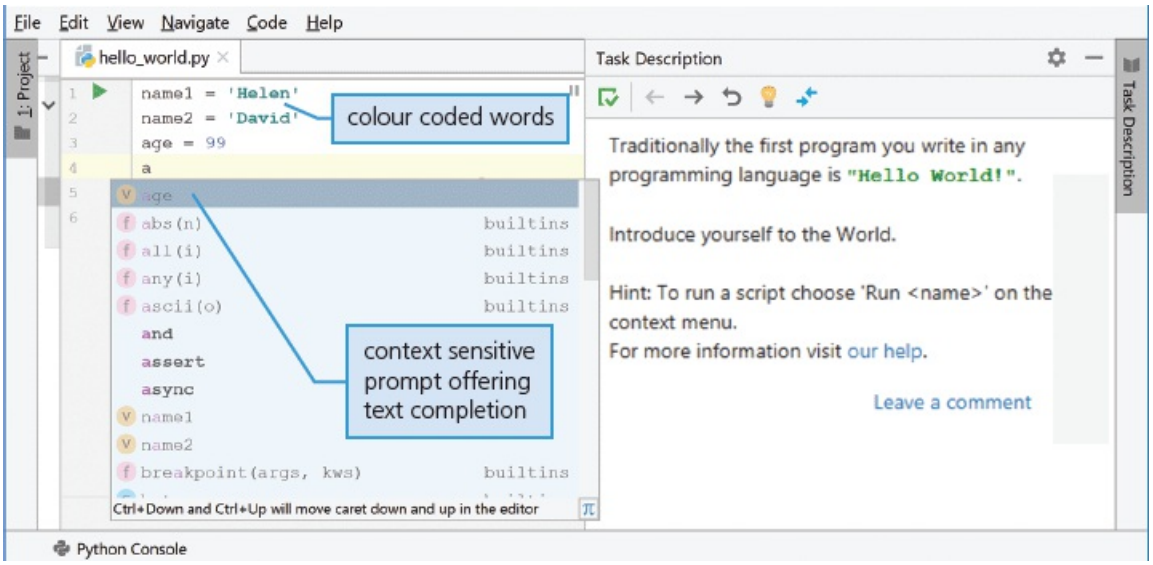


Figure 5.12 PyCharm IDE showing source code editor

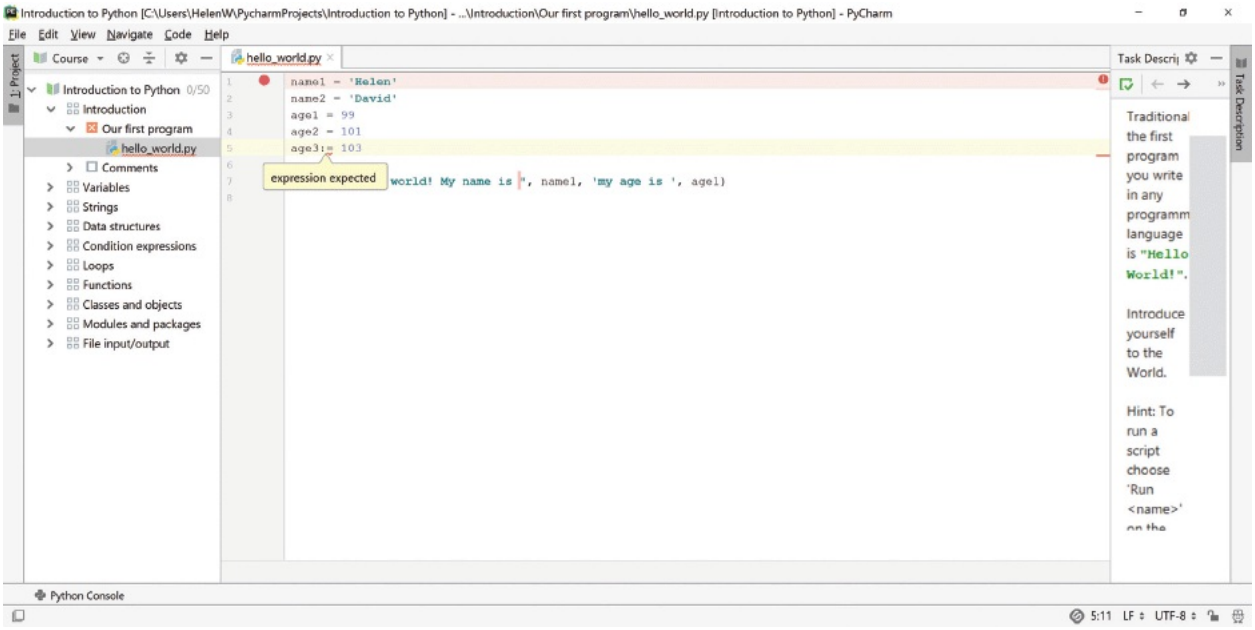


Figure 5.13 PyCharm IDE showing dynamic syntax checking

Here, string values are shown coloured green and integer values are shown coloured blue.

Dynamic syntax checking finds possible **syntax errors** as the program code is being typed in to the source code editor and alerts the programmer at the time, before the source code is interpreted. Many errors can therefore be found and corrected during program writing and editing before the program is run. **Logic errors** can only be found when the program is run.

For larger programs that have more than one code block, some code blocks can be collapsed to a single line in the editor allowing the programmer to just see the code blocks that are currently being developed.

Compilers and interpreters

Most IDEs usually provide a compiler and/or an interpreter to run the program. The interpreter is often used for developing the program and the compiler to produce the final version of the object code.

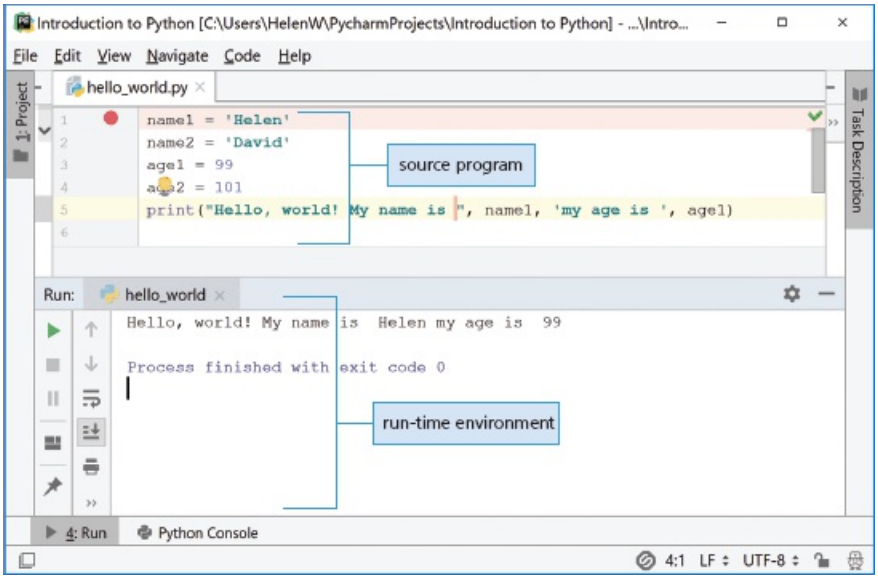
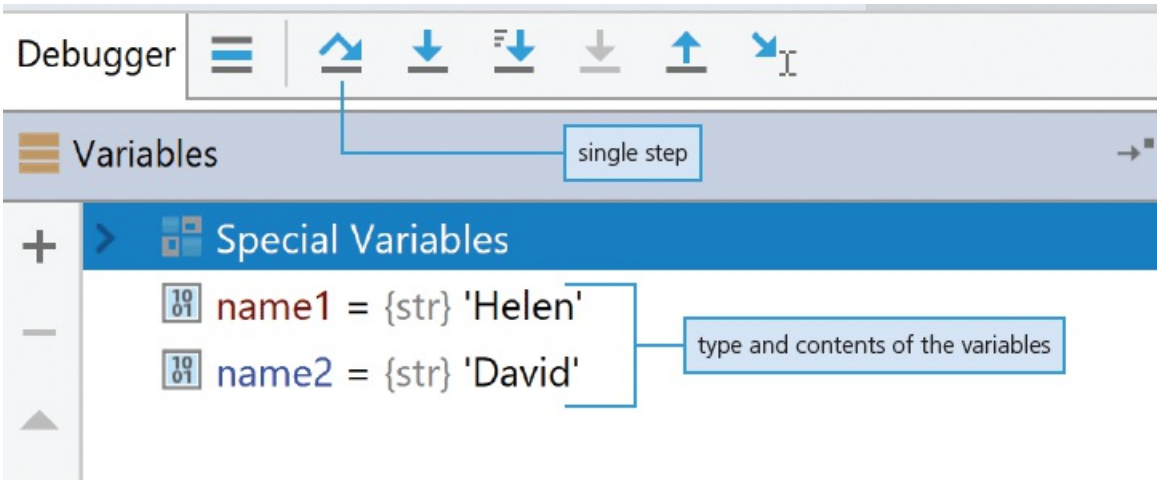


Figure 5.14 PyCharm IDE showing both program code and program run

With PyCharm there can be more than one interpreter available for different versions of the Python language. The program results are shown using the run-time environment provided.

A run-time environment with a debugger

A debugger is a program that runs the program under development and aids the process of **debugging**. It allows the programmer to single step through the program a line at a time (**single stepping**) or to set a **breakpoint** to stop the execution of the program at a certain point in the source code. A **report window** then shows the contents of the variables and expressions evaluated at that point in the program. This allows the programmer to see if there are any logic errors in the program and check that the program works as intended.



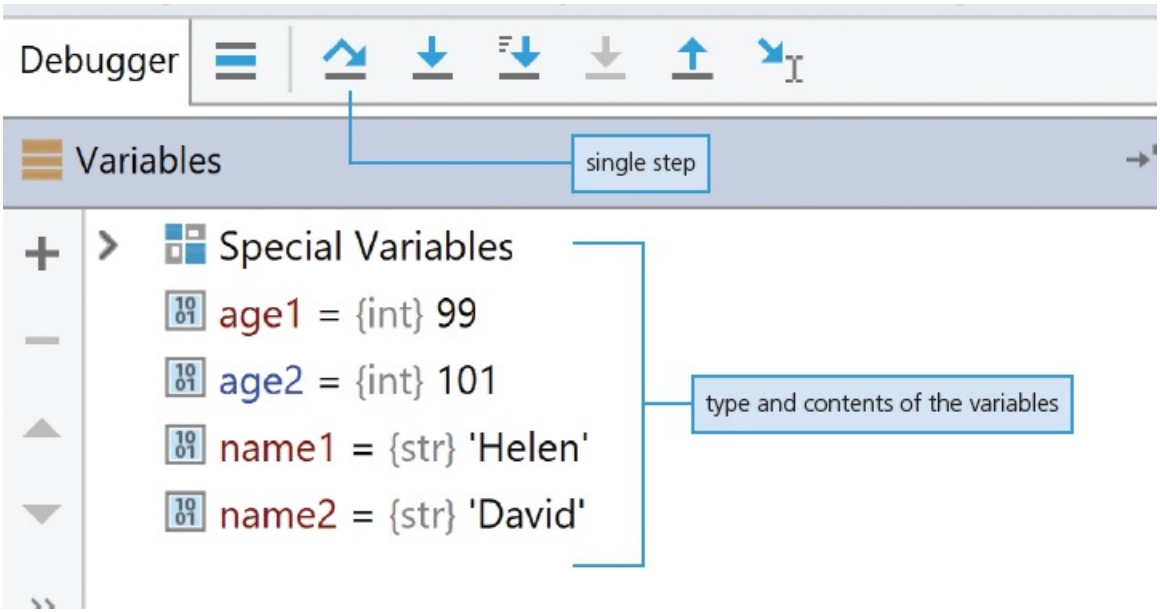


Figure 5.15 PyCharm IDE showing the report window after line 2 (page 155) and after line 4 (above)

Each variable used is shown in the report window together with the type and the contents of the variable at that point in the program. The top variable shown is the last one that was used.

Answers to calculations and other expressions can also be shown.

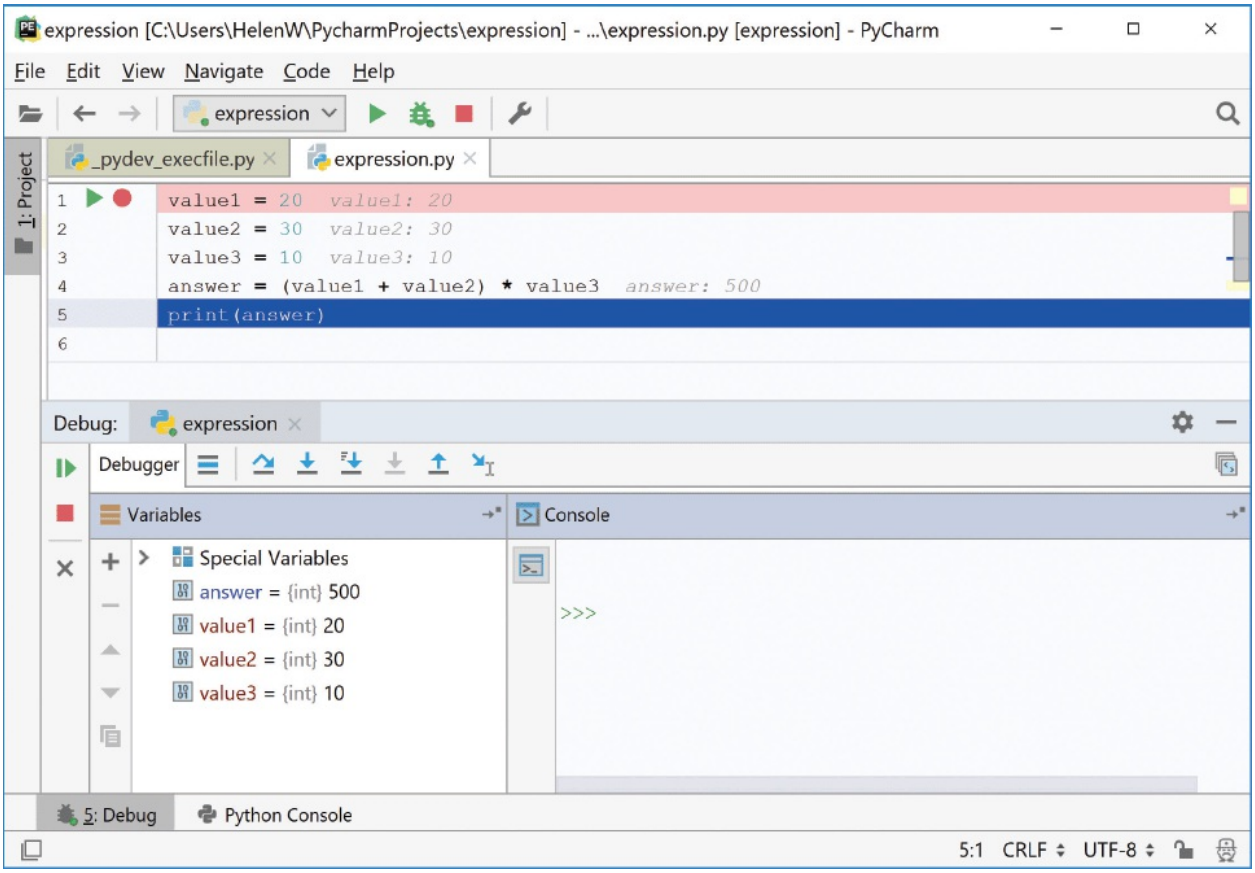


Figure 5.16 PyCharm IDE showing the report window with the answer to an expression

Auto-documenter

Most IDEs usually provide an auto-documenter to explain the function and purpose of programming code.

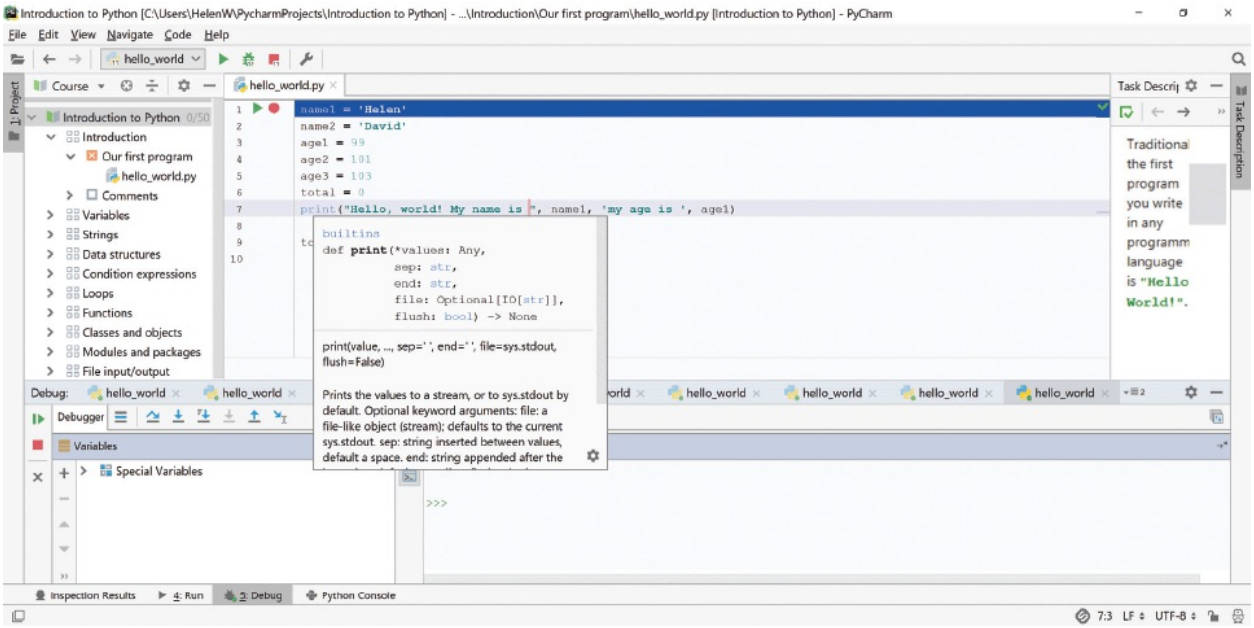


Figure 5.17 PyCharm IDE showing the quick documentation window for print

ACTIVITY 5B

1 a) i) Describe the difference between a compiler and an assembler.

ii) Describe the difference between a compiler and an interpreter.

b) State **two** benefits **and** two drawbacks of using an interpreter.

2 A new program is to be written in a high-level language. The programmer has decided to use an IDE to develop the new program.

a) Explain what is meant by an IDE.

b) Describe **three** features of an IDE.

End of chapter questions

1 A programmer is writing a program that includes code from a program library.

a) Describe **two** benefits to the programmer of using one or more library routines.

[4]

b) The programmer decides to use a Dynamic Link Library (DLL) file.

i) Describe **two** benefits of using DLL files.

[4]

ii) State **one** drawback of using DLL files.

[2]

Cambridge International AS & A Level Computer Science 9608 Paper 12 Q8 November 2016

2 a) The operating system contains code for performing various management tasks. The appropriate code is run when the user performs actions.
Copy the diagram below and connect each OS management task to the appropriate user action.

[3]

OS management task	action
main memory management	The user moves the mouse on the desktop
input/output management	The user closes the spreadsheet program
secondary storage management	The user selects the SAVE command to save their spreadsheet
human–computer interface management	The user selects the PRINT command to output their spreadsheet

b) A user has the following issues with the use of his PC.
State the utility software which should provide a solution.

i) The hard disk stores a large number of video files. The computer frequently runs out of storage space.

[1]

ii) The user is unable to find an important document. He thinks it was deleted in error some weeks ago. This must not happen again.

[1]

iii) The operating system reports 'bad sector' errors.

[1]

iv) There have been some unexplained images and advertisements appearing on the screen. The user suspects it is malware.

[1]

Cambridge International AS & A Level Computer Science 9608 Paper 11 Q6 June 2017

3 *File History* and *Time Machine* are examples of back-up utilities offered as part of two different operating systems.

a) Explain why it is important to back up files on a computer.

[2]

b) One of the features offered by both utilities is the possibility of 'turning back the internal

computer clock'.
Explain why this is an important feature and give **two** occasions when a user may wish to use this feature.

[4]

c) By using diagrams and written explanation, describe how *defragmentation software* works.

[4]

4 Assemblers, compilers and interpreters are all used to translate programs. Discuss the different roles played by each translator.

[6]

5 State **four** features of an IDE that are helpful when coding a program.

[4]

6 Security, privacy and data integrity

In this chapter, you will learn about

- the terms security, privacy and integrity of data
- the need for security of data and security of computer systems
- security measures to protect computer systems such as user accounts, passwords, digital signatures, firewalls, antivirus and anti-spyware software and encryption
- security threats such as viruses and spyware, hacking, phishing and pharming
- methods used to reduce security risks such as encryption and access rights
- the use of validation to protect data integrity
- the use of verification during data entry and data transfer to reduce or eliminate errors.

6.1 Data security

WHAT YOU SHOULD ALREADY KNOW

Try these five questions before you read the first part of this chapter.

1 a) What is meant by *hacking*?

b) Is hacking always an illegal act? Justify your answer.

2 Contactless credit cards and debit cards are regarded by some as a security risk. Discuss the advantages and disadvantages of using contactless cards with particular reference to data security.

3 What are the main differences between *cracking* and *hacking*?

4 a) What are pop-ups when visiting a website? Are they a security risk?

b) What are cookies? Do cookies pose a security threat?

c) Describe:

i) session cookies
ii) permanent cookies
iii) third party cookies.

5 Why must the correct procedures be carried out when removing a memory stick from a computer?

Key terms

Data privacy – the privacy of personal information, or other information stored on a computer, that should not be accessed by unauthorised parties.

Data protection laws – laws which govern how data should be kept private and secure.

Data security – methods taken to prevent unauthorised access to data and to recover data if lost or corrupted.

User account – an agreement that allows an individual to use a computer or network server, often requiring a user name and password.

Authentication – a way of proving somebody or something is who or what they claim to be.

Access rights (data security) – use of access levels to ensure only authorised users can gain access to certain data.

Malware – malicious software that seeks to damage or gain unauthorised access to a computer system.

Firewall – software or hardware that sits between a computer and external network that monitors and filters all incoming and outgoing activities.

Anti-spyware software – software that detects and removes spyware programs installed illegally on a user's computer system.

Encryption – the use of encryption keys to make data meaningless without the correct decryption key.

Biometrics – use of unique human characteristics to identify a user (such as fingerprints or face recognition).

Hacking – illegal access to a computer system without the owner's permission.

Malicious hacking – hacking done with the sole intent of causing harm to a computer system or user (for example, deletion of files or use of private data to the hacker's advantage).

Ethical hacking – hacking used to test the security and vulnerability of a computer system. The hacking is carried out with the permission of the computer system owner, for example, to help a company identify risks associated with malicious hacking of their computer systems.

Phishing – legitimate-looking emails designed to trick a recipient into giving their personal data to the sender of the email.

Pharming – redirecting a user to a fake website in order to illegally obtain personal data about the user.

DNS cache poisoning – altering IP addresses on a DNS server by a 'pharmer' or hacker with the intention of redirecting a user to their fake website.

6.1.1 Data privacy

Data stored about a person or an organisation must remain private and unauthorised access to the data must be prevented – **data privacy** is required.

This is achieved partly by **data protection laws**. These laws vary from country to country, but all follow the same eight guiding principles.

1. Data must be fairly and lawfully processed.
2. Data can only be processed for the stated purpose.
3. Data must be adequate, relevant and not excessive.
4. Data must be accurate.
5. Data must not be kept longer than necessary.
6. Data must be processed in accordance with the data subject's rights.
7. Data must be kept secure.
8. Data must not be transferred to another country unless that country also has adequate protection.

Data protection laws usually cover organisations rather than private individuals. Such laws are no guarantee of privacy, but the legal threat of fines or jail sentences deters most people.

6.1.2 Preventing data loss and restricting data access

Data security refers to the methods used to prevent unauthorised access to data, as well as to the data recovery methods if it is lost.

User accounts

User accounts are used to **authenticate** a user (prove that a user is who they say they are). User accounts are used on both standalone and networked computers in case the computer can be accessed by a number of people. This is often done by a screen prompt asking for a username and password:

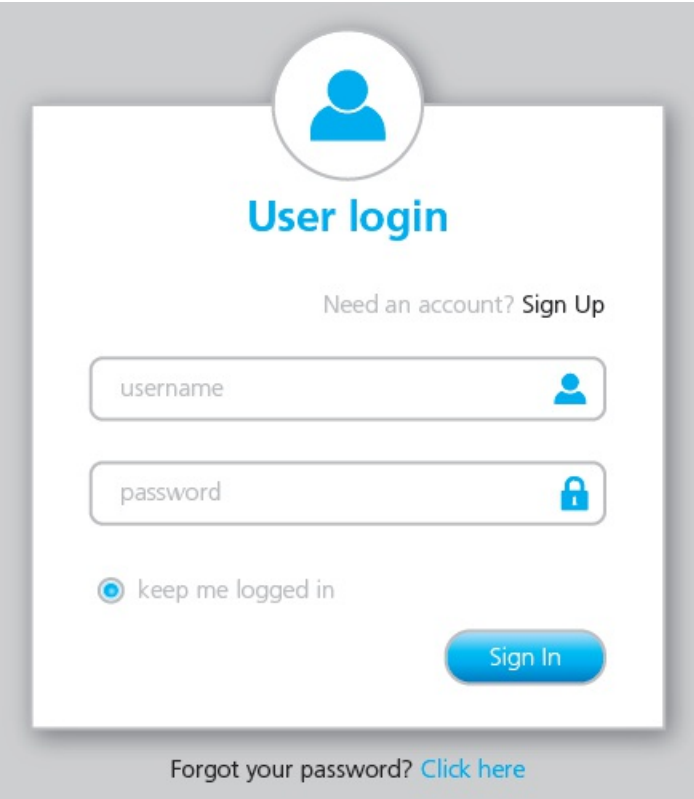


Figure 6.1 A login screen

User accounts control **access rights**. This often involves levels of access. For example, in a hospital it would not be appropriate for a cleaner to have access to data about one of the patients. However, a consultant would need such access. Therefore, most systems have a hierarchy of access levels depending on a person's level of security. This could be achieved by username and password with each username (account) linked to the appropriate level of access.

EXTENSION ACTIVITY 6A

An airport uses a computer system to control security, flight bookings, passenger lists, administration and customer services. Describe how it is possible to ensure the safety of the data on the system so that senior staff can see all data, while customers can only access flight

times (arrivals and departures) and duty free offers.

Use of passwords

Passwords are used to restrict access to data or systems. They should be hard to crack and changed frequently to retain security. Passwords can also take the form of biometrics (such as on a mobile phone, as discussed later). Passwords are also used, for example, when

- accessing email accounts
- carrying out online banking or shopping
- accessing social networking sites.

It is important that passwords are protected. Some ways of doing this are to

- run anti-spyware software to make sure your passwords are not being relayed to whoever put the spyware on your computer
- regularly change passwords in case they have been seen by someone else, illegally or accidentally
- make sure passwords are difficult to crack or guess (for example, do not use your date of birth or pet's name).

Passwords are grouped as either strong (hard to crack or guess) or weak (relatively easy to crack or guess). Strong passwords should contain

- at least one capital letter
- at least one numerical value
- at least one other keyboard character (such as @, *, &)

Example of a strong password: Sy12@#TT90kj=0

Example of a weak password: GREEN

EXTENSION ACTIVITY 6B

Which of the following are weak passwords and which are strong passwords?

Explain your decision in each case.

a) 25-May-2000

b) Pas5word

c) ChapTer@06

d) N55!

e) 12345X

Digital signatures

Digital signatures protect data by providing a way of identifying the sender of, for example, an email. These are covered in more depth in Chapter 17.

Use of firewalls

A **firewall** can be software or hardware. It sits between the user's computer and an external network (such as the internet) and filters information in and out of the computer. This allows the user to decide to allow communication with an external source and warns a user that an external source is trying to access their computer. Firewalls are the primary defence to any computer system to protect from hacking, **malware** (viruses and spyware), phishing and pharming.

Figure 6.2 Firewall

The tasks carried out by a firewall include

- examining the traffic between the user's computer (or internal network) and a public network (such as the internet)
- checking whether incoming or outgoing data meets a given set of criteria
- blocking the traffic if the data fails to meet the criteria, and giving the user (or network manager) a warning that there may be a security issue
- logging all incoming and outgoing traffic to allow later interrogation by the user (or network manager)
- preventing access to certain undesirable sites – the firewall can keep a list of all undesirable IP addresses
- helping to prevent viruses or hackers entering the user's computer (or internal network)
- warning the user if some software on their system is trying to access an external data source (such as an automatic software upgrade). The user is given the option of allowing it to go ahead or request that such access is denied.

The firewall can be a hardware interface which is located somewhere between the computer (or internal network external link) and the internet connection. In these cases, it is often referred to as a gateway. Alternatively, the firewall can be software installed on a computer, sometimes as part of the operating system.

However, sometimes the firewall cannot prevent potential harmful traffic. It cannot

- prevent individuals, on internal networks, using their own modems to by-pass the firewall
- control employee misconduct or carelessness (for example, control of passwords or user accounts)
- prevent users on stand-alone computers from disabling the firewall.

These issues require management and/or personal control to ensure the firewall can work effectively.

Antivirus software

Running antivirus software in the background on a computer will constantly check for virus attacks. Although different types of antivirus software work in different ways, they all

- check software or files before they are run or loaded on a computer
- compare possible viruses against a database of known viruses
- carry out heuristic checking (check software for behaviour that could indicate a virus, which is

useful if software is infected by a virus not yet on the database)

- quarantine files or programs which are possibly infected and
  - allow the virus to be automatically deleted, or
  - allow the user to make the decision about deletion (it is possible that the user knows that the file or program is not infected by a virus – this is known as a false positive and is one of the drawbacks of antivirus software).

Antivirus software needs to be kept up to date since new viruses are constantly being discovered. Full system checks need to be carried out regularly (once a week, for example), since some viruses lie dormant and would only be picked up by this full system scan.

Anti-spyware software

Anti-spyware software detects and removes spyware programs installed illegally on a user's computer system. The software is either based on rules (it looks for typical features associated with spyware) or based on known file structures which can identify common spyware programs.

Encryption

If data on a computer has been accessed illegally (by a hacker, for example) it is possible to encrypt the data, making it virtually impossible to understand without **encryption** keys to decode it. This cannot stop a hacker from deleting the files, but it will stop them using the data for themselves. This is covered in more depth in Chapter 17.

Biometrics

In an attempt to stay one step ahead of hackers and malware writers, many modern computer devices use **biometrics** as part of the password system. Biometrics rely on the unique characteristics of human beings. Examples include fingerprint scans, retina scans (pattern of blood capillary structure), face recognition and voice recognition.

Fingerprint scans

Images of fingerprints are compared against previously scanned fingerprints stored in a database; if they match then access is allowed; the system compares patterns of 'ridges' and 'valleys' which are fairly unique (accuracy is about 1 in 500).

Figure 6.3 Fingerprint

Retina scans

Retina scans use infra-red to scan the unique pattern of blood vessels in the retina (at the back of the eye). It requires a person to stay still for 10 to 15 seconds while the scan takes place; it is very secure since nobody has yet found a way to duplicate the blood vessels patterns' (accuracy is about 1 in 10 million).

Figure 6.4 Retina scan

Mobile phones use biometrics to identify if the phone user is the owner.

6.1.3 Risks to the security of stored data

Hacking

You will see the term **hacking** used throughout this textbook. There are two types of hacking: malicious and ethical.

Malicious hacking is the illegal access to a computer system without the user's permission or knowledge. It is usually employed with the intention of deleting, altering or corrupting files, or to gain personal details such as bank account details. Strong passwords, firewalls and software which can detect illegal activity all guard against hacking.

Ethical hacking is authorised by companies to check their security measures and how robust their computer systems are to resist hacking attacks. It is legal, and is done with a company's permission with a fee paid to the ethical hacker.

Malware

Malware is one of the biggest risks to the integrity and security of data on a computer system. Many software applications sold as antivirus are capable of identifying and removing most of the forms of malware described below.

Viruses

Programs or program code that can replicate and/or copy themselves with the intention of deleting or corrupting files or causing the computer to malfunction. They need an active host program on the target computer or an operating system that has already been infected before they can run.

Worms

A type of stand-alone virus that can replicate themselves with the intention of spreading to other computers; they often use networks to search out computers with weak security.

Logic bombs

Code embedded in a program on a computer. When certain conditions are met (such as a specific date) they are activated to carry out tasks such as deleting files or sending data to a hacker.

Trojan horses

Malicious programs often disguised as legitimate software. They replace all or part of the legitimate software with the intent of carrying out some harm to the user's computer system.

Bots (internet robots)

Not always harmful and can be used, for example, to search automatically for an item on the internet. However, they can cause harm by taking control over a computer system and launching attacks.

Spyware

Software that gathers information by monitoring, for example, key presses on the user's

keyboard. The information is then sent back to the person who sent the software – sometimes referred to as key logging software.

Phishing

Phishing is when someone sends legitimate-looking emails to users. They may contain links or attachments which, when clicked, take the user to a fake website, or they may trick the user into responding with personal data such as bank account details or credit card numbers. The email often appears to come from a trusted source such as a bank or service provider. The key is that the recipient has to carry out a task (click a link, for example) before the phishing scam causes harm.

There are numerous ways to help prevent phishing attacks:

- Users need to be aware of new phishing scams. Those people in industry or commerce should undergo frequent security awareness training to become aware of how to identify phishing (and pharming) scams.
- Do not click on links unless certain that it is safe to do so; fake emails can often be identified by greetings such as 'Dear Customer' or 'Dear emailperson@gmail.com', and so on.
- It is important to run anti-phishing toolbars on web browsers (this includes tablets and mobile phones) since these will alert the user to malicious websites contained in an email.
- Look out for https and/or the green padlock symbol in the address bar (both suggest that traffic to and from the website is encrypted).
- Regularly check online accounts and frequently change passwords.
- Ensure an up-to-date browser, with all of the latest security upgrades, is running, and run a good firewall in the background at all times. A combination of a desktop firewall (usually software) and a network firewall (usually hardware) considerably reduces risk.
- Be wary of pop-ups – use the web browser to block them; if pop-ups get through your defences, do not click on 'cancel' since this often leads to phishing or pharming sites – the best option is to select the small X in the top right hand corner of the pop-up window, which closes it down.

Pharming

Pharming is malicious code installed on a user's computer or on a web server. The code re-directs the user to a fake website without their knowledge (the user does not have to take any action, unlike phishing). The creator of the malicious code can gain personal data such as bank details from users. Often, the website appears to belong to a trusted company and can lead to fraud or identity theft.

Why does pharming pose a threat to data security?

Pharming redirects users to a fake or malicious website set up by, for example, a hacker. Redirection from a legitimate website can be done using **DNS cache poisoning**.

Every time a user types in a URL, their web browser contacts the DNS server. The IP address of the website is then sent back to their web browser. However, DNS cache poisoning changes the real IP address values to those of the fake website consequently, the user's computer connects to the fake website.

Pharmers can also send malicious programming code to a user's computer. The code is stored on the HDD without their knowledge. Whenever the user types in the website address of the targeted website, the malicious programming code alters the IP address sent back to their browser which redirects it to the fake website.

Protection against pharming

It is possible to mitigate the risk of pharming by

- using antivirus software, which can detect unauthorised alterations to a website address and warn the user
- using modern web browsers that alert users to pharming and phishing attacks
- checking the spelling of websites
- checking for https and/or the green padlock symbol in the address bar.

It is more difficult to mitigate risk if the DNS server itself has been infected (rather than the user's computer).

EXTENSION ACTIVITY 6C

Pharmers alter IP addresses in order to send users to fake websites. However, the internet does not only have one DNS server. Find out how a user's internet service provider (ISP) uses its own DNS servers which cache information from other internet DNS servers.

6.1.4 Data recovery

This section covers the potential impact on data caused by accidental mal-operation, hardware malfunction and software malfunction.

In each case, the method of data recovery and safeguards to minimise the risk are considered.

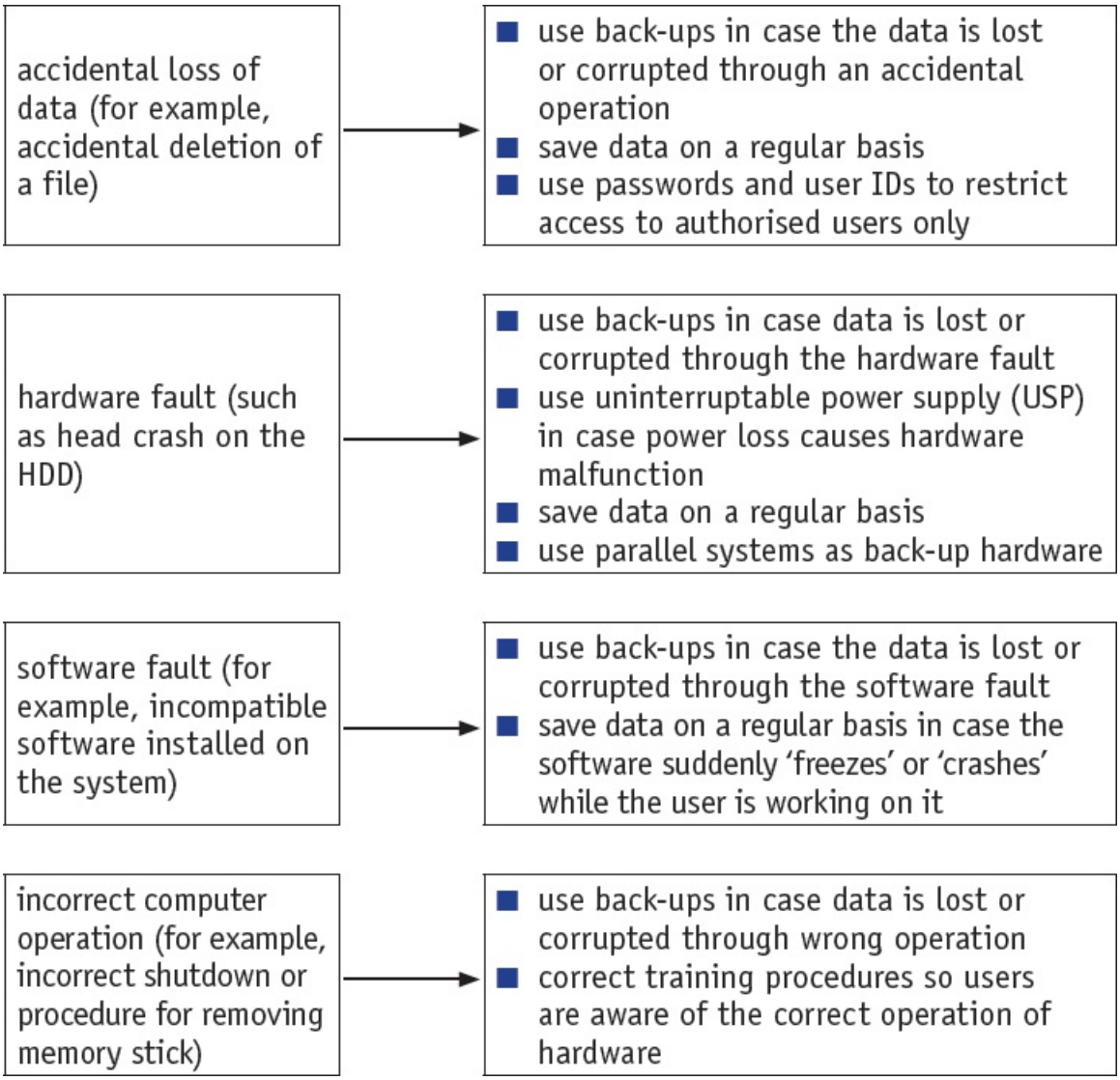


Figure 6.5 Safeguards

In all cases, the backing up of data regularly (automatically and/or manually at the end of the day) onto another medium (such as cloud storage, or removable HDD) is key to data recovery. The back-up should be stored in a separate location in case of, for example, a fire or an office break-in. Somebody should be given the role of carrying out back-ups, to ensure it it always done.

Backing up data may not be a suitable method of recovery in the case of a virus infection, as the backed up data may contain strands of the virus which could re-infect the 'cleaned' computer.

ACTIVITY 6A

1 A company has offices in four different countries. Communication and data sharing between the offices is done via computers connecting over the internet.
Describe **three** data security issues the company might encounter during their day to day communications and data sharing.
For each issue described, explain why it could be a threat to the security of the company.
For each issue described, describe a way to mitigate the threat which has been posed.

2 Define these three terms.

a) Worm

b) Logic bomb

c) Trojan horse

3 John works for a car company. He maintains the database which contains all the personal data of the people working for the car company. John was born on 28 February 1990 and has two pet cats called Felix and Max.

a) John needs to use a password and a username to log onto the database. Why would the following three passwords **not** be a good choice?

i) 280290

ii) FeLix1234

iii) John04

b) Describe how John could improve his passwords.
How should he maintain his passwords to maximise database security?

c) When John enters a password on his computer, he is presented with the following question on screen.

> Would you like to save the password on this device?

Why is it important that John always says ‘no’ to this question?

d) John frequently orders goods from an online company called NILE.com.
He opens an email which purports to be from NILE.com.

> Dear NILE.com user
>
> This is to confirm your recent order for:
>
> 01230123 A level Computer Science Workbook, $15.90
>
> If this is not your order, please click on the following link and update your details:
>
> myorders@NILE.com
>
> Thank you. Customer services.

Explain why John should be suspicious of the email.

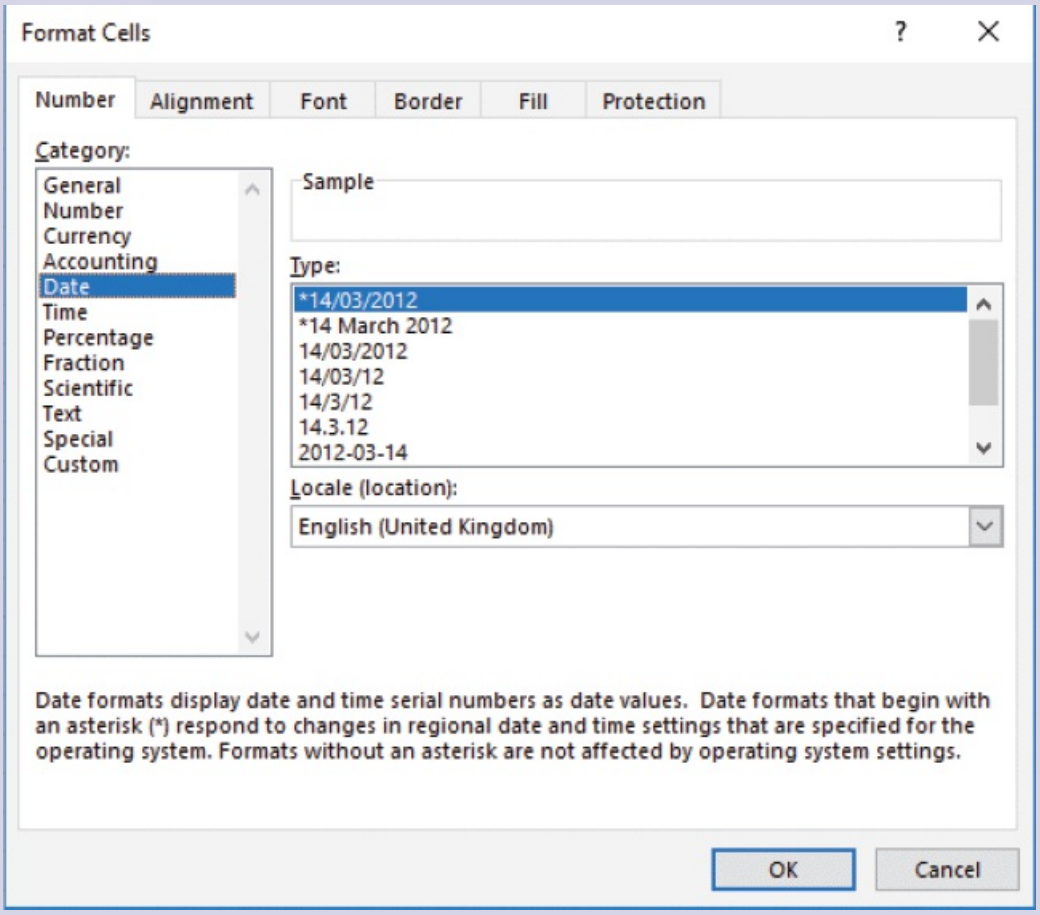
Include, in your explanation, the type of security threat identified by this email.

6.2 Data integrity

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you read the second part of this chapter.

1 Look at the following validation screen from a spreadsheet.
Why is it important to have validation in applications such as spreadsheets?



2 Why is proofreading not the same as verification?

3 Discuss **one** way online form designers can ensure that only certain data can be input by a user. Use the date: 12 March 2019 as the example.

Key terms

Data integrity – the accuracy, completeness and consistency of data.

Validation – method used to ensure entered data is reasonable and meets certain input criteria.

Verification – method used to ensure data is correct by using double entry or visual checks.

Check digit – additional digit appended to a number to check if entered data is error free.

Modulo-11 – method used to calculate a check digit based on modulus division by 11.

Checksum – verification method used to check if data transferred has been altered or corrupted, calculated from the block of data to be sent.

Parity check – method used to check if data has been transferred correctly that uses even or odd parity.

Parity bit – an extra bit found at the end of a byte that is set to 1 if the parity of the byte needs to change to agree with sender/receiver parity protocol.

Odd parity – binary number with an odd number of 1-bits.

Even parity – binary number with an even number of 1-bits.

Parity block – horizontal and vertical parity check on a block of data being transferred.

Data stored on a computer should always be accurate, consistent and up to date. Two of the methods used to ensure data integrity are validation and verification.

The accuracy (**integrity**) of data can be compromised

- during the data entry and data transmission stages
- by malicious attacks on the data, for example caused by malware and hacking
- by accidental data loss caused through hardware issues.

These risks – together with ways of mitigating them – are discussed in the rest of this chapter.

6.2.1 Validation

Validation is a method of checking if entered data is reasonable (and within a given criteria), but it cannot check if data is correct or accurate. For example, if somebody accidentally enters their age as 62 instead of 26, it is reasonable but not accurate or correct. Validation is carried out by computer software; the most common types are shown in Table 6.1.

Validation test	Description	Example of data failing validation test	Example of data passing validation test
type	checks whether non-numeric data has been input into a numeric-only field	typing sk.34 in a field which should contain the price of an item	typing 34.50 in a field which should contain the price of an item
range	checks whether data entered is between a lower and an upper limit	typing in somebody's age as –120	typing in somebody's age as 48
format	checks whether data has been entered in the agreed format	typing in the date as 12-12-20 where the format is dd/mm/yyyy	typing in the date as 12/12/2020 where the format is dd/mm/yyyy
length	checks whether data has the required number of characters or numbers	typing in a telephone number as 012 345 678 when it should contain 11 digits	typing in a telephone number as 012 345 678 90 when it should contain 11 digits
presence	checks to make sure a field is not left empty when it should contain data	please enter passport number:.....	please enter passport number: AB 1234567 CD
existence	checks if data in a file or a file name actually exists	data look up for car registration plate A123 BCD which does not exist	data look up for a file called books_in_stock which exists in a database
limit check	Checks only one of the limits (such as the upper limit OR the lower limit)	typing in age as –25 where the data entered should not be negative	typing in somebody's age as 72 where the upper limit is 140
consistency check	checks whether data in two or more fields match up correctly	typing in Mr in the title field and then choosing female in the sex field	typing in Ms in the title field and then choosing female in the sex field
uniqueness check	checks that each entered value is unique	choosing the user name MAXIMUS222 in a social networking site but the user name already exists	choosing the website name Aristooo.com which is not already used

Table 6.1 Common validation

Key terms

Parity byte – additional byte sent with transmitted data to enable vertical parity checking (as

well as horizontal parity checking) to be carried out.

Automatic repeat request (ARQ) – a type of verification check.

Acknowledgement – message sent to a receiver to indicate that data has been received without error.

Timeout – time allowed to elapse before an acknowledgement is received.

6.2.2 Verification

Verification is a way of preventing errors when data is entered manually (using a keyboard, for example) or when data is transferred from one computer to another.

Verification during data entry

When data is manually entered into a computer it needs to undergo verification to ensure there are no errors. There are three ways of doing this: double entry, visual check and check digits.

Double entry

Data is entered twice, using two different people, and then compared (either after data entry or during the data entry process).

Visual check

Entered data is compared with the original document (in other words, what is on the screen is compared to the data on the original paper documents).

Check digits

The **check digit** is an additional digit added to a number (usually in the right-most position). They are often used in barcodes, ISBNs (found on the cover of a book) and VINs (vehicle identification number). The check digit can be used to ensure the barcode, for example, has been correctly inputted. The check digit can catch errors including

- an incorrect digit being entered (such as 8190 instead of 8180)
- a transposition error where two numbers have been swapped (such as 8108 instead of 8180)
- digits being omitted or added (such as 818 or 81180 instead of 8180)
- phonetic errors such as 13 (thirteen) instead of 30 (thirty).

Figure 6.6 shows a barcode with an ISBN-13 code with check digit.

Figure 6.6 Barcode

An example of a check digit calculation is **modulo-11**. The following algorithm is used to generate the check digit for a number with seven digits:

1. Each digit in the number is given a weighting of 7, 6, 5, 4, 3, 2 or 1, starting from the left.
2. The digit is multiplied by its weighting and then each value is added to make a total.
3. The total is divided by 11 and the remainder subtracted from 11.
4. The check digit is the value generated; note if the check digit is 10 then X is used.

For example:

Seven digit number:	4 1 5 6 7 1 0
Weighting values:	7 6 5 4 3 2 1
Sum:	(7 × 4) + (6 × 1) + (5 × 5) + (4 × 6) + (3 × 7) + (2 × 1) + (1 × 0)
	= 28 + 6 + 25 + 24 + 21 + 2 + 0
Total:	= 106
Divide total by 11:	9 remainder 7
subtract remainder from 11:	11 − 7 = 4 (check digit)
final number:	4 1 5 6 7 1 0 4

When this number is entered, the check digit is recalculated and, if the same value is not generated, an error has occurred. For example, if 4 1 5 7 6 1 0 4 was entered, the check digit generated would be 3, indicating an error.

EXTENSION ACTIVITY 6D

1 Find out how the ISBN-13 method works and confirm that the number 978 034 098 382 has a check digit of 9.

2 Find the check digits for the following numbers using both modulo-11 and ISBN-13.

a) 213 111 000 428

b) 909 812 123 544

3 Find a common use for the modulo-11 method of generating check digits.

Verification during data transfer

When data is transferred electronically from one device to another, there is always the possibility of data corruption or even data loss. A number of ways exist to minimise this risk.

Checksums

A **checksum** is a method to check if data has been changed or corrupted following data transmission. Data is sent in blocks and an additional value, the checksum, is sent at the end of the block of data.

To explain how this works, we will assume the checksum of a block of data is 1 byte in length. This gives a maximum value of $2^8 - 1$ (= 255). The value 0000 0000 is ignored in this calculation. The following explains how a checksum is generated.

If the sum of all the bytes in the transmitted block of data is ≤ 255, then the checksum is this value. However, if the sum of all the bytes in the data block > 255, then the checksum is found using the following simple algorithm.

In the example we will assume the value of X is 1185.

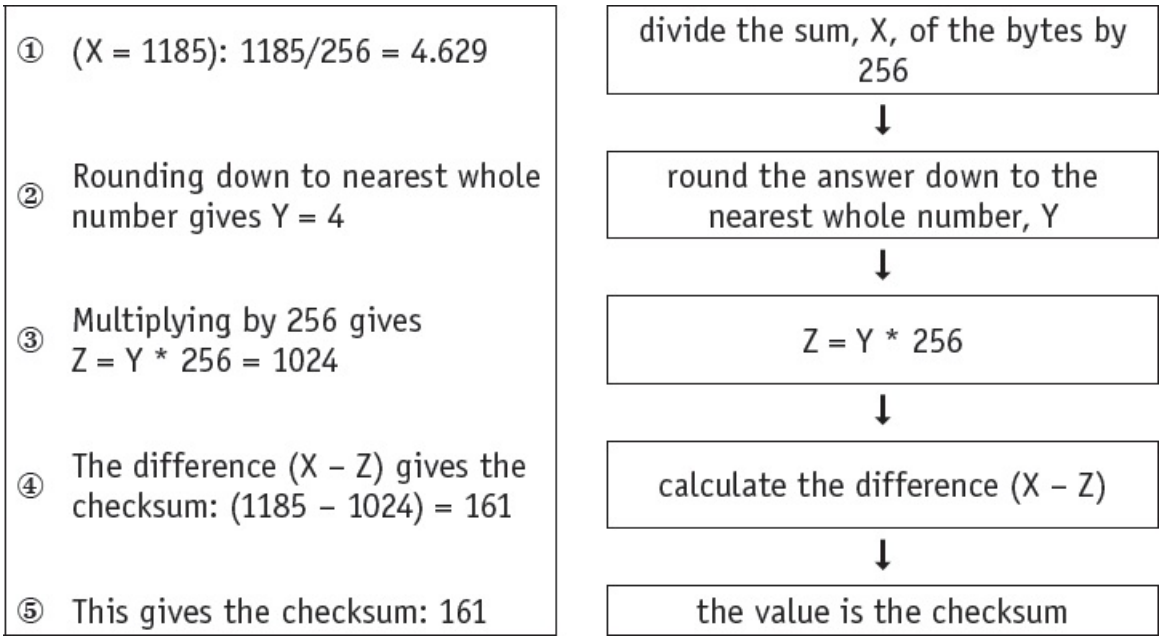


Figure 6.7

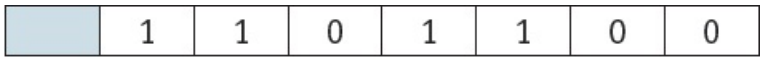
When a block of data is about to be transmitted, the checksum for the bytes is first calculated. This value is transmitted with the block of data. At the receiving end, the checksum is re-calculated from the block of data received. This calculated value is compared to the checksum transmitted. If they are the same, then the data was transmitted without any errors; if they are different, then a request is sent for the data to be re-transmitted.

Parity checks

A **parity check** is another method to check whether data has been changed or corrupted following transmission from one device or medium to another.

A byte of data, for example, is allocated a **parity bit**. This is allocated before transmission. Systems that use **even parity** have an even number of 1-bits; systems that use **odd parity** have an odd number of 1-bits.

Consider the following byte:



parity bit

Figure 6.8

If this byte is using even parity, then the parity bit needs to be 0 since there is already an even number of 1-bits (in this case, four).

If odd parity is being used, then the parity bit needs to be 1 to make the number of 1-bits odd. Therefore, the byte just before transmission would be:

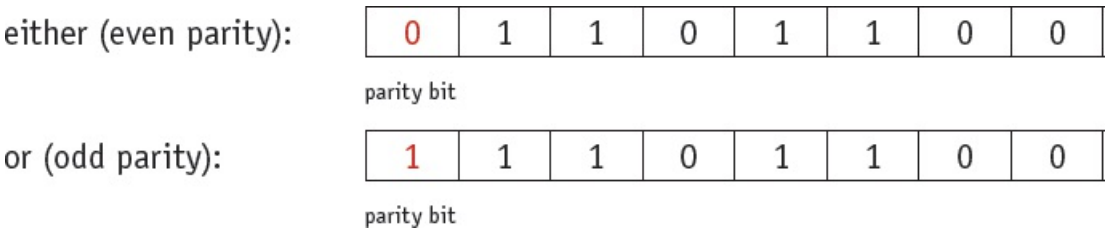


Figure 6.9

Before data is transferred, an agreement is made between sender and receiver regarding which of the two types of parity are used. This is an example of a protocol.

EXTENSION ACTIVITY 6E

Find the parity bits for each of the following bytes:

1 1 1 0 1 1 0 1 even parity used
2 0 0 0 1 1 1 1 even parity used
3 0 1 1 1 0 0 0 even parity used
4 1 1 1 0 1 0 0 odd parity used
5 1 0 1 1 0 1 1 odd parity used

If a byte has been transmitted from 'A' to 'B', and even parity is used, an error would be flagged if the byte now had an odd number of 1-bits at the receiver's end.

For example:

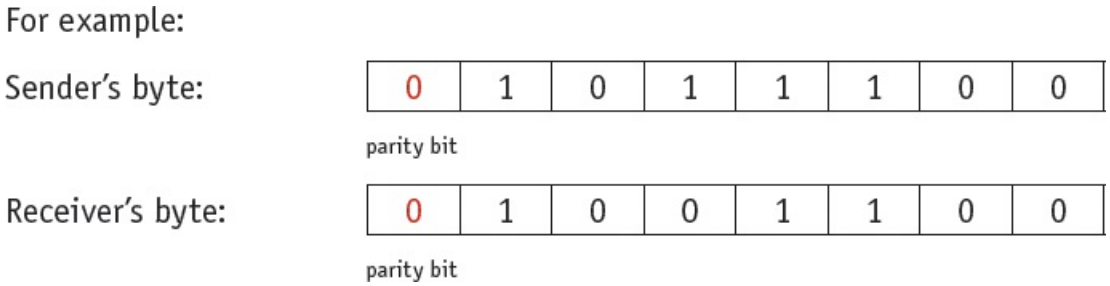


Figure 6.10

In this case, the receiver's byte has three 1-bits, which means it now has odd parity, while the byte from the sender had even parity (four 1-bits). This means an error has occurred during the transmission of the data.

The error is detected by the computer re-calculating the parity of the byte sent. If even parity has been agreed between sender and receiver, then a change of parity in the received byte indicates that a transmission error has occurred.

EXTENSION ACTIVITY 6F

1 Which of the following bytes have an error following data transmission?

a) 1 1 1 0 1 1 0 1 even parity used
b) 0 1 0 0 1 1 1 1 even parity used

c) 0 0 1 1 1 0 0 0 even parity used

d) 1 1 1 1 0 1 0 0 odd parity used

e) 1 1 0 1 1 0 1 1 odd parity used

2 In each case where an error occurs, can you work out which bit is incorrect?

Naturally, any of the bits in the above example could have been changed leading to a transmission error. Therefore, even though an error has been flagged, it is impossible to know exactly which bit is in error.

One of the ways around this problem is to use **parity blocks**. In this method, a block of data is sent and the number of 1-bits are totalled horizontally and vertically (in other words, a parity check is done in both horizontal and vertical directions). As the following example shows, this method not only identifies that an error has occurred but also indicates *where* the error is.

In this example, nine bytes of data have been transmitted. Agreement has been made that even parity will be used. Another byte, known as the **parity byte**, has also been sent. This byte consists entirely of the parity bits produced by the vertical parity check. The parity byte also indicates the end of the block of data.

Table 6.2 shows how the data arrived at the receiving end:

	parity bit	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
byte 1	1	1	1	1	0	1	1	0
byte 2	1	0	0	1	0	1	0	1
byte 3	0	1	1	1	1	1	1	0
byte 4	1	0	0	0	0	0	1	0
byte 5	0	1	1	0	1	0	0	1
byte 6	1	0	0	0	1	0	0	0
byte 7	1	0	1	0	1	1	1	1
byte 8	0	0	0	1	1	0	1	0
byte 9	0	0	0	1	0	0	1	0
parity byte	1	1	0	1	0	0	0	1

Table 6.2

A careful study of the table shows that

- byte 8 (row 8) has incorrect parity (there are three 1-bits)
- bit 5 (column 5) also has incorrect parity (there are five 1-bits).

First, the table shows that an error has occurred following data transmission.

Second, at the intersection of row 8 and column 5, the position of the incorrect bit value (which caused the error) can be found. This means that byte 8 should have been:

0	0	0	1	0	0	1	0

which would also correct column 5 giving an even vertical parity (now has four 1-bits).

This byte could, therefore, be corrected automatically, as shown above, or an error message could be relayed back to the sender asking them to re-transmit the block of data. One final point; if two of the bits change value following data transmission, it may be impossible to locate the error using the above method.

For example, using the above example again:

0	1	0	1	1	1	0	0

This byte could reach the destination as:

0	1	1	1	1	1	0	1

All three are clearly incorrect, but they have retained even parity so will not trigger an error message at the receiving end. Clearly, other methods to complement parity when it comes to error checking of transmitted data are required (such as checksum).

Automatic repeat request (ARQ)

Automatic repeat request (ARQ) is another method to check data following data transmission. This method can be summarised as follows:

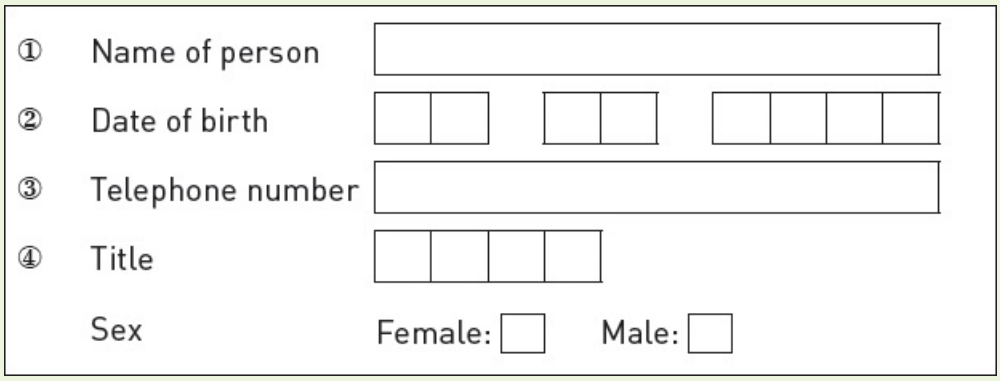
- ARQ uses **acknowledgement** (a message sent to the receiver indicating that data has been received correctly) and **timeout** (the time interval allowed to elapse before an acknowledgement is received).
- When the receiving device detects an error following data transmission, it asks for the data packet to be re-sent.
- If no error is detected, a positive acknowledgement is sent to the sender.
- The sending device will re-send the data package if
  - it receives a request to re-send the data, or
  - a timeout has occurred.
- The whole process is continuous until the data packet received is correct or until the ARQ time limit (timeout) is reached.
- ARQ is often used by mobile phone networks to guarantee data integrity.

ACTIVITY 6B

1 The following block of data was received after transmission from a remote computer; odd parity was being used by both sender and receiver.
One of the bits has been changed during the transmission stage.
Locate where this error is and suggest a corrected byte value:

	parity bit	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
byte 1	0	1	1	0	0	0	1	0
byte 2	1	0	1	1	1	1	1	1
byte 3	1	0	0	1	1	0	0	0
byte 4	0	1	1	0	1	0	1	0
byte 5	1	1	1	0	0	1	1	0
byte 6	1	0	0	0	0	1	0	1
byte 7	0	1	1	1	0	0	0	0
byte 8	0	0	0	0	0	0	0	1
byte 9	0	1	1	1	1	0	1	0
parity byte	1	0	1	1	1	1	0	0

2 a) A company is collecting data about new customers and is using an online form to collect the data, as shown below.
Describe a suitable validation check for each of the four groups of fields.



b) Explain the differences between *validation* and *verification*.
Why are both methods used to maintain the integrity of data?

3 A shopkeeper is populating a database containing information about goods for sale in their shop.
They are entering the data manually, using both validation and verification to ensure the integrity of the entered data.
Here is an example of a record:

A21516BX	25	205.50	03334445556
code of the item NXXXXNN (N = letter; X = digit)	number in stock (1–100)	unit cost in dollars	telephone number of supplier of item

a) Describe how verification could be used to ensure the accuracy of the entered data.

b) Describe suitable validation checks for all four fields and give examples of data which

would fail your chosen validation methods.

End of chapter questions

1 A college is using a local area network (LAN) to access data from a database.

a) Give **two** security measures to protect the data on the college's computer system.

[2]

b) Data regarding new students joining the college is being entered into the database.
Each student has a 7-digit identification number (ID).
A check digit is used as a form of checking to ensure errors have not been made when entering the ID numbers. The verification routine uses modulo-11 with the check digit as the eighth (right-most) digit.
The weightings used to calculate the check digit are: 7, 6, 5, 4, 3, 2 and 1; the value 7 is the multiplier for the left-most digit.
The ID number is: 1 5 6 3 4 1 2
Calculate the check digit.

[4]

c) Name and describe **two** validation checks that could be carried out on the student ID number.

[4]

2 a) Explain what antivirus software is and how it can be used to ensure data security.

[4]

b) Explain how a firewall can be used to identify illegal attempts at accessing a computer system and how they can be used to keep data safe.

[4]

3 The following block of data was received after transmission from a remote computer.
Odd parity was being used by both sender and receiver.
One of the bits has been changed during the transmission stage.
Locate where this error is and suggest a corrected byte value.

[5]

	parity bit	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8
byte 1	0	1	0	0	1	0	0	1
byte 2	0	0	1	1	1	1	1	0
byte 3	1	0	0	0	0	1	1	0
byte 4	1	0	1	1	0	0	0	0
byte 5	1	1	1	0	1	1	1	0
byte 6	0	1	0	0	0	1	0	1
byte 7	1	0	0	1	1	0	1	1
byte 8	1	1	0	1	1	0	0	1
byte 9	0	1	0	1	1	1	1	0
parity byte	0	0	0	0	1	1	0	1

7 Ethics and ownership

In this chapter, you will learn about

- the need for and purpose of ethics as a computer science professional
- the need to act ethically at all times
- the impact of acting ethically or unethically in a given situation
- the need for copyright legislation
- the different types of software licensing, including free software, open source software, shareware and commercial software
- the impact of artificial intelligence (AI) on social, economic and environmental issues.

WHAT YOU SHOULD ALREADY KNOW

Try these four questions before you read this chapter.

1 a) What is meant by an *expert system*?

b) Name **four** components of a typical expert system.

c) Give **three** examples of the use of an expert system.

2 a) What is meant by *copyright*?

b) Why is copyright important?

c) Give examples of items which would be covered by copyright laws.

d) Differentiate between the terms *plagiarism* and *copyright*.

3 a) What impact do computers have on the general public with regards to

i) jobs/employment
ii) the environment
iii) how we shop and bank
iv) human interactions?

b) Describe **three** positive aspects of the impact of computers on society.

4 What is the influence of social media on

a) news reporting

b) world safety

c) personal and private lives of people

d) politics?

7.1 Legal, moral, ethical and cultural implications

Key terms

Legal – relating to, or permissible by, law.

Morality – an understanding of the difference between right and wrong, often founded in personal beliefs.

Ethics – moral principles governing an individual's or organisation's behaviour, such as a code of conduct.

Culture – the attitudes, values and practices shared by a group of people/society.

Intellectual property rights – rules governing an individual's ownership of their own creations or ideas, prohibiting the copying of, for example, software without the owner's permission.

Privacy – the right to keep personal information and data secret and for it to not be unwillingly accessed or shared through, for example, hacking.

Plagiarism – the act of taking another person's work and claiming it as one's own.

BCS – British Computer Society.

IEEE – Institute of Electrical and Electronics Engineers.

ACM – Association for Computing Machinery.

The following definitions are important when considering ethical behaviour:

- **Legal** covers the law, whether or not an action is punishable by law.
- **Morality** concerns questions of right and wrong, and is more often thought of in relation to personal or individual choices.
- **Ethics** also concerns questions of right and wrong, but is more often used in a professional context.
- **Culture** refers to the attitudes, values and practices shared by a society or group of people.

Anything which breaks the law is termed illegal. Examples include copying software and then selling it without the permission of the copyright holders (see Section 7.2).

Morality is the human desire to distinguish between right and wrong. This varies from person to person, and between cultures (something that is considered immoral in one culture, may be acceptable practice in another, for example).

Immoral does not mean something is illegal (and vice versa). Creating a fake news website, for example, is not illegal, but it may be considered immoral if it causes distress to others. If the creator tried to obtain personal and financial data, then it would be become an illegal act.

Similarly, hacking is generally regarded as immoral, but not illegal. However, it becomes illegal

if it compromises national security, or results in financial gain, or reveals personal information, for example.

In short, there is a fine line between an immoral act and an illegal act.

Unethical behaviour is the breaking of a code of conduct. For example, if somebody works for a software company and passes on some ideas to a rival company, this would be regarded as unethical behaviour. If the software is related to national security or is formally copyrighted, then it is also illegal.

It is essential to be clear whether any law has been broken.

The importance of culture is less tangible. When writing computer games, for example, programmers need to be careful that they do not include items which some cultures would find offensive or obscene. Again, this may not be unethical or illegal, but could still cause distress. It is important to realise that boundaries can easily be crossed; in some countries making fun of religion, for example, is illegal.

7.1.1 Computer ethics

Computer ethics is a set of principles set out to regulate the use of computers. Three factors are considered:

- **Intellectual property rights**, for example, copying of software without the permission of the owner.
- **Privacy** issues, for example, hacking or any illegal access to another person's personal data.
- Effect of computers on society, for example, job losses, social impacts, and so on.

Internet use has led to an increase in **plagiarism** – this is when a person takes another person's idea or work and claims it was their own. While it is fine to quote another person's idea, it is essential that some acknowledgement is made so that the originator of the idea or work is known to others. This can be done by a series of references at the end of a document or footnotes on each page where a reference needs to be made. Software exists that can scan text and then look for examples of plagiarism by searching web pages on the internet.

7.1.2 Professional ethical bodies

There are a number of professional bodies representing individuals working in the fields of computing and information technology that have developed their own codes of conduct, to which members are expected to adhere. Belonging to one of these organisations demonstrates your professional integrity by showing that you are committed to upholding the standards they prescribe.

The British Computer Society (BCS)

The **British Computer Society (BCS)** is a professional body set up in the UK, initially to represent the rights and ethical practices of all professionals working in the IT and computing industries. It is now an international body which works in close partnership with other groups to monitor and advise IT practices across the globe.

The BCS Code of Conduct (www.bcs.org/category/6030) covers four main areas:

1. The Public Interest
2. Professional Competence and Integrity
3. Duty to Relevant Authority
4. Duty to the Profession

The Institute of Electrical and Electronics Engineers (IEEE)

The **Institute of Electrical and Electronics Engineers (IEEE)** was set up in the USA with the aims of

- raising awareness of ethical issues
- promoting ethical behaviour among professionals working in the electronics industry
- ensuring engineers and scientists respect the need for ethical behaviour.

To help in this aim, the IEEE has also set out a code of ethics:

IEEE Code of Ethics

We, the members of the IEEE, in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members, and the communities we serve, do hereby commit ourselves to the highest ethical and professional conduct and agree:

1. to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment;
2. to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
3. to be honest and realistic in stating claims or estimates based on available data;
4. to reject bribery in all its forms;
5. to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;
6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;
8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;

9 to avoid injuring others, their property, reputation, or employment by false or malicious action;

10 to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.

Jointly with the **Association for Computing Machinery (ACM)**, the IEEE has also developed a set of eight principles which govern the code of ethics specifically among software engineers. The principles set out to ensure all engineers meet an acceptable and consistent code of ethics. There are certain expectations of the scientists and engineers from the general public as well as from their peers. The actual eight principles behind the code of ethics and professional practice were published way back in 1999.

An abridged version is shown below; a full version can be found at:

www.computer.org/web/education/code-of-ethics

Software Engineering Code of Ethics

1 PUBLIC – Software engineers shall act consistently with the public interest (contains 8 sub-clauses).

2 CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest (contains 9 sub-clauses).

3 PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible (contains 15 sub-clauses).

4 JUDGEMENT – Software engineers shall maintain integrity and independence in their professional judgement (contains 6 sub-clauses).

5 MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance (contains 12 sub-clauses).

6 PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest (contains 13 sub-clauses).

7 COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues (contains 8 sub-clauses).

8 SELF – Software engineers shall participate in life-long learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession (contains 9 sub-clauses).

There are 80 clauses and sub-clauses in total. We shall consider one scenario and see how it fits into a selection of the clauses.

Mikhail works during the day for a software company called *EthicalGamz* developing new software in a number of applications. Mikhail is part of a large team of software engineers writing and testing new code. The team also do market research to help in their development of new software for the future. Much of the work is commercially sensitive and multiple layers of access exist to protect the company from unauthorised sharing of data.

In the evenings and at the weekend, Mikhail works for his own company, *MikhailSoft*, which produces software available to buy on the internet only. To save costs, Mikhail uses coding he helped develop for *EthicalGamz* in his own software. He also outsources some of the work to software engineers in other countries where the wages are much lower and ethics policies are more lax. This saves him a lot of time and money when producing his own software. Mikhail does not pay any licensing fees to *EthicalGamz* and makes no reference to any code used from that company in his own products.

We will now consider the ethical implications of the above scenario using the following sub-clauses from the Software Engineering Code of Ethics.

1.03 approve software only if they have a well-founded belief that it is safe, meets the specification and passes the appropriate tests and does not diminish the quality of life, diminish privacy or harm the environment;

There is an ethical issue here since the software written by personnel from other countries may not meet the specification requirements or appropriate tests. It could lead to any of the three factors being violated, for example, the software may contain spyware of which Mikhail is unaware.

2.02 not knowingly use software that is obtained or retained either illegally or unethically;

Mikhail has no control over the coding being developed by his overseas team, furthermore, using the coding from *EthicalGamz* is illegal use.

3.05 ensure an appropriate method is used for any project on which they work or propose to work;

Using external companies (in his own country or overseas) may be used at various steps in the production of Mikhail's own software. Unless he applies good managerial control, he will be unable to ensure methods used in projects are appropriate or fully ethical in their implementation.

4.02 only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement;

Documentation produced by third party developers is not produced under Mikhail's direct supervision, indeed some of the work done overseas may be outside Mikhail's sphere of knowledge which probably removes his ability to objectively endorse the external work being done.

5.03 ensure that software engineers know the employer's policies and procedures for protecting passwords, files and information that is confidential to the employer or to others;

By using software developed by *EthicalGamz* for his own use, Mikhail may need to give passwords and access to other files to engineers working for his own company, *MikhailSoft*. This would allow non-authorised personnel access to files and information stored on *EthicalGamz* computer systems leading to a potential security breach.

6.05 not protect their own interest at the expense of the profession, client or employer;

By using coding from *EthicalGamz*, Mikhail is enhancing his own interests at the expense of the company and his colleagues at that company.

7.03 credit fully the work of others and refrain from taking undue credit;

By using coding from *EthicalGamz* illegally and unethically, and by making no reference to the source of his 'illegal' code, Mikhail is effectively taking full credit for all the work done by his colleagues.

8.07 do not give unfair treatment to anyone because of any irrelevant prejudices

Mikhail may dismiss overseas workers who do not agree with his own political or religious beliefs and such dismissals would be deemed unfair and break this code of practice.

EXTENSION ACTIVITY 7A

Using the example above, consider the following eight sub-clauses and decide how (or if) Mikhail is breaking the code of ethics in each case.

1.01 accept full responsibility for their own work

2.03 use the property of a client or employer only in ways properly authorised, and with

the client's or employer's knowledge and consent

3.03 identify, define and address ethical, economic, cultural, legal and environmental issues related to work projects

4.04 not engage in deceptive financial practices such as bribery, double billing or other improper financial practices

5.02 ensure that software engineers are informed of standards before being held to them

6.08 take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work

7.02 assist colleagues in their professional development

8.09 recognise that personal violations of this Code are inconsistent with being a professional software engineer

7.1.3 Impact on the public

Figure 7.1 summarises the potential impact of any software or hardware being developed on the general public.

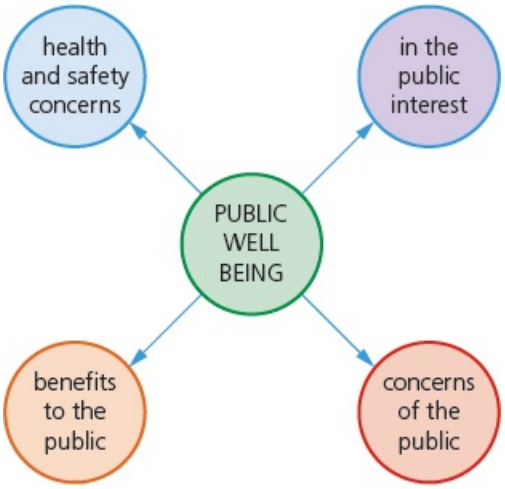


Figure 7.1 Potential impact of software or hardware being developed on the general public

While software engineers and scientists consider the Software Engineering Code of Ethics, the impact on the general public cannot be ignored.

This section begins by considering three instances in which computer hardware or software led to expensive errors, which impacted on the general public.

LA airport shutdown in 2007

In this example, aeroplanes at LA airport (in the USA) were grounded due to a simple software issue: a faulty network card in a device continued to send incorrect data over the airport's network. Eventually, the whole of the USA Customs and Borders Agency came to an abrupt standstill at LA airport. This resulted in all flights leaving and landing at the airport being cancelled for about eight hours until the fault was cleared. It cost several million US dollars in lost revenue to the aeroplane operators. The impact on the general public was cancellation of holidays, loss of business and general frustration.

Exploding laptop computers in 2008

Japan holds an annual trade show displaying the latest in computer technology. In 2008, during the trade show, a number of Dell laptop computers burst into flames under the full view of the visiting public and television cameras. The problem was traced back to faulty batteries in the laptops which had been overheating and eventually exploded and burst into flames. As if this was not enough, the problem escalated when Apple reported similar problems with some of its tablets, laptops and desktop computers. Some 100 million computer devices had to be recalled at an estimated cost of over 300 million US dollars to the manufacturers. The impact on the general public would have been devastating if this problem had not been discovered before the devices were generally available to buy.

Airbus A380 incompatible software issue in 2006

In Europe, Airbus Industries uses a number of factories throughout Europe where the design, development and construction of aeroplanes takes place. During 2006, while the new A380 was being developed, a surprising issue came to life: the software in two factories would not 'talk to each other'. The factory in Hamburg (Germany) was using an old version of CATIA design software while another plant in Toulouse (France) was using the latest version of CATIA software. When a part of the A380 from Hamburg and a part of the A380 from Toulouse were brought together for assembly, the wiring in the two parts did not match up (the cables could not be linked together). This was all due to the fact that the two versions of the software produced different design specifications for the wiring. It cost the company millions of Euros to redo the design and remanufacturing of parts where old software was still in use. Fortunately, this was not a safety issue, but if some other design incompatibility had

occurred after assembly of an A380, the effect could have been catastrophic leading to possible loss of life.

All of these examples are cost-related, but still had – or potentially had – an impact on the general public. Regrettably, there are many other examples. Other issues which can affect the general public and businesses include

- companies selling software systems which do not meet the required standard for security (inadequate protection against hacking, spyware and other security issues)
- the covering up of security issues (such as the XEN security threat which forced several cloud servers to become compromised – an attempt was made to cover up the issue but the affected cloud operators had to come clean)
- the release of private data (such as the celebrity photo leaks, when a cloud server was hacked)
- social media not policing subversive activity, such as hate mail and cyber bullying. Such activity is undergoing close scrutiny by several countries around the world
- search engines giving results at the top of the search due to donations to the search engine operators.

EXTENSION ACTIVITY 7B

Bearing in mind some of the issues raised above, consider these two questions.

1 Should we police the internet to stop certain activities taking place?

2 Should governments have the power to close down websites (such as *Twitter* or *Facebook*) which do not remove hate mail, incitements to violence or unacceptable photographs from their sites?

ACTIVITY 7A

1 Describe why it is necessary to produce a code of ethics to cover the computing and electronics industries.

2 Mariam and Asma were having a discussion about whether or not the internet should be policed.

Mariam was in favour of the argument and put forward two reasons.

① It would prevent illegal material being posted on websites, such as racist comments, pornography, terrorist activities and so on.

② Some form of control would prevent children and other vulnerable groups being subjected to undesirable websites.

Asma was against the argument and put forward two of her own reasons.

① Material published on websites is already available from other sources.

② Policing would go against freedom of information and freedom of speech.

Put forward your own arguments and discuss whether you think Mariam's or Asma's reasons are valid.

3 Describe the main differences between the terms: *legal*, *morality*, *ethics* and *culture*. Give examples of each.

7.2 Copyright issues

Key terms

Piracy – the practice of using or making illegal copies of, for example, software.

Product key – security method used in software to protect against illegal copies or use.

Digital rights management (DRM) – used to control the access to copyrighted material.

Free Software Foundation – organisation promoting the free distribution of software, giving users the freedom to run, copy, change or adapt the coding as needed.

Open Source Initiative – organisation offering the same freedoms as the Free Software Foundation, but with more of a focus on the practical consequences of the four shared rules, such as more collaborative software development.

Freeware – software that can be downloaded free of charge; however, it is covered by the usual copyright laws and cannot be modified; nor can the code be used for another purpose.

Shareware – software that is free of charge initially (free trial period). The full version of the software can only be downloaded once the full fee for the software has been paid.

7.2.1 Software copyright and privacy

Software is protected by copyright laws in much the same way as music CDs, videos and articles from magazines and books are protected.

When software is purchased, there are certain rules that must be obeyed:

- It is illegal to make a software copy and sell it or give it away.
- Software cannot be used on a network or used on multiple computers without a multi-use licence.
- It is illegal to use coding from copyrighted software in your own software – and then pass this software on or sell it as your own – without the permission of the copyright holder.
- It is illegal to rent out a software package without permission to do so.
- It is illegal to use the name of copyrighted software on other software without agreement to do so.

Software **piracy** (making illegal copies of software) is a major issue among software companies. They take many steps to stop the illegal copying of software and to stop illegal copies being used once they have been sold:

- When software is being installed, the user will be asked to key in a unique reference number or **product key** (a string of letters and numbers) which was supplied with the original copy of the software (for example: 4a3c 0efa 65ab a81e).
- The user will be asked to click a button or box which states they agree to the licence agreement before the software continues to install.
- The original software packaging often comes with a sticker informing the purchaser that it is illegal to make copies of the software; the label is often in the form of a hologram indicating that this is a genuine copy.
- Some software will only run if the CD-ROM, DVD-ROM or memory stick is actually in the drive; this stops illegal multiple use and network use of the software.
- Some software will only run if a dongle is plugged into one of the USB ports.

(See also Section 7.2.2 regarding further copyright protection using DRM.)

The Federation Against Software Theft (FAST) was set up in the UK to protect the software industry against piracy. FAST prosecutes organisations and individuals involved in any copyright infringements.

Similar organisations exist in other countries. The following extract from a newspaper article describes a typical example of how strict the anti-piracy laws are in some countries.

TRADERS FINED $100 000

Two eBay traders from the United States of America agreed this week to pay a total of $100 000 in damages after they were caught selling illegal copies of Norton security software.

The SIIA settled the case against the two traders who also agreed to stop selling illegal software and provided SIIA with records identifying their customers and suppliers.

7.2.2 The internet and the World Wide Web (WWW)

Digital rights management (DRM) was originally set up to control what devices a CD could play on. Preventing a CD from playing on a computer, for example, would help stop it being copied illegally. DRM has since been updated to cover more areas; it does this by using protection software to help stop the copying of, for example, music tracks, video files or ebooks. DRM creates restrictions that control what the users can do with the data. For example, allowing a music file to be streamed over the internet but not copied, allowing an ebook to be read on a tablet only, or a game requiring an internet connection to a certain website to work, and so on. The aim of DRM is to ensure that any attempt made to break the copyright protection will produce a defective copy which will not work.

When you buy a product protected by DRM, it may come with a key which licences a single user on one device and this key must be registered. Another example – of which there are many – is Apple Music's use of DRM layers in streamed music to prevent a user downloading all the music in the first month of a subscription and then cancelling their subscription.

7.2.3 Software licensing

Commercial software

Commercial software is available to customers for a fee, providing a licence for one genuine copy to be used on a single device, or a multi-use licence for multiple users. Occasionally, software is offered free of charge if an earlier version was bought by the user. This type of software is fully copyright-protected and none of the code can be used without the prior consent of the copyright owner.

Free software and the Open Source Initiative

The **Free Software Foundation** and the **Open Source Initiative** are non-profit organisations that promote the benefits of giving users the freedom to run, copy, change and adapt software. Examples of software licensed in this way include: F-spot (photographic manager), Scribus (DTP/word processor) and LibreOffice (Office Suite). Users are allowed to follow the four freedoms:

- Run the software for any legal purpose they wish.
- Study the program source code and modify it where necessary to meet their needs.
- Redistribute copies of the software to friends and family.
- Distribute code modified by the user to friends and family.

Users do not need to seek permission to do the above since the software is not protected by copyright restrictions. However, there are still some rules that the user must adhere to. Users cannot

- add source code from another piece of software unless this is also described as free software or open source software
- use the source code to produce software which copies existing software which is subject to copyright laws
- adapt the source code in such a way that it infringes copyright laws protecting other software
- use the source code to produce software which is deemed offensive by third parties.

While the two organisations promote the same four freedoms, they have different basic philosophies.

Free Software Foundation focuses on what the recipient of the software is permitted to do with the software.

Open Source Initiative focuses on the practical consequences offered by the four freedoms; the aims are to provide effective collaboration on software development by the users. There are ten principles that have been developed to ensure the philosophy of the Open Source Initiative is adhered to:

1. **Free Redistribution** The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Source Code** The program must include source code and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source

code there must be a well-publicised means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. **Derived Works** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of The Author's Source Code** The license may restrict source-code from being distributed in modified form only if the license allows the distribution of 'patch files' with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No Discrimination Against Persons or Groups** The license must not discriminate against any person or group of persons.
6. **No Discrimination Against Fields of Endeavor** The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. **Distribution of License** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License Must Not Be Specific to a Product** The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
9. **License Must Not Restrict Other Software** The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
10. **License Must Be Technology-Neutral** No provision of the license may be predicated on any individual technology or style of interface.

Freeware

Freeware is software a user can download from the internet free of charge. Once it has been downloaded, there are no fees associated with using the software (examples include: Adobe Reader, Skype and some media players). Unlike free software, freeware is subject to copyright laws and users are often requested to tick a box to say they understand and agree to the terms and conditions governing the software. This means that a user is not allowed to study or modify the source code in any way.

Shareware

Shareware allows users to try out some software free of charge for a trial period. At the end of the trial period, the author of the software will request that you pay a fee if you wish to continue using it. Once the fee is paid, a user is registered with the originator of the software and free

updates and help are then provided. Often, the trial version of the software is missing some of the features found in the full version, and these do not become available until the fee is paid.

This type of software is protected by copyright laws and users must not use the source code in any of their own software without permission.

ACTIVITY 7B

1 a) What is meant by the term *software piracy*?

b) Describe **three** ways of protecting software against deliberate attempts at making copies to sell or give away.

2 A software company offers a suite of shareware programs. It contains a spreadsheet, word processor, database and drawing package.
What are the benefits to the following two stakeholders of offering software packages as shareware?

- The company
- The customer

7.3 Artificial intelligence (AI)

Key terms

Artificial intelligence (AI) – machine or application which carries out a task that requires some degree of intelligence when carried out by a human counterpart.

7.3.1 What is AI?

Artificial intelligence (AI) is a machine or application which carries out a task that requires some degree of intelligence when carried out by a human being. These tasks could include

- the use of a language
- carrying out a mathematical calculation or function
- recognising a person's face
- the ability to operate machinery, such as a car, an aeroplane or a train
- analysing data to predict the outcome of a future event, such as weather forecasting.

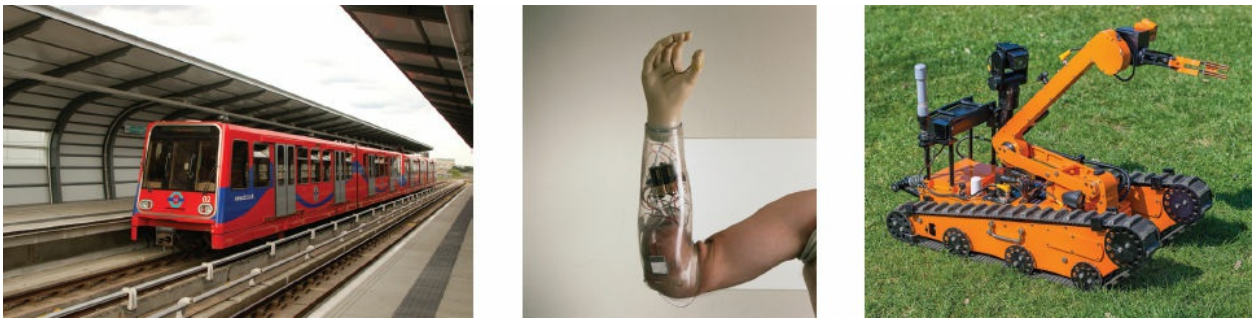


Figure 7.2 Examples of how AI can be used in every-day life

AI duplicates human tasks requiring decision-making and problem-solving skills.

7.3.2 The impact of AI

People often associate AI with science fiction, fantasy and robots. Numerous films and books fuel this association. The science fiction author, Isaac Asimov, went so far as to produce his own three laws of robotics:

1. A robot may not injure a human through action or inaction.
2. A robot must obey orders given by humans without question.
3. A robot must protect itself unless it conflicts with the two laws above.

However, AI goes way beyond robotics. It covers an ever-increasing number of areas, such as

- autonomous (driverless) vehicles
- artificial limb technology
- drones, used to carry out dangerous or unpleasant tasks such as bomb disposal, welding, or entering nuclear disaster areas
- climate change predictions
- medical procedures, such as eye operations where extreme precision is required.

7.3.3 The impacts of AI on society, the economy and the environment

As a result of increasing automation over the next few decades, the human race will need to consider the impacts that AI will have on society, the economy and the environment. So should we all be worried? In this section, we will consider a number of existing AI technologies, plus some predictions for the future, to help stimulate discussions. As mentioned in Section 7.3.2, AI is not just about robots, but covers many areas (this is explored further in Chapter 18, which explores specific AI technologies in more depth).

We will look at some of the areas mentioned in Section 7.3.2 in more depth and consider the implications of using AI (the descriptions that follow will mix up benefits and drawbacks – in Activity 7C you will need to consider the overall impact).

New developments in AI are constantly being announced and you are advised to keep up to date by checking out the many websites that keep an eye on AI development.

Below are some of the developments and impacts that are currently expected to be seen in the near future.

Research has predicted that, by 2030, some 600 million jobs will be lost globally and as many as 400 million people will need to retrain or switch jobs – all caused by the inevitable advances in AI. The most likely jobs to be lost are those doing medium- and low-skilled work, but high-skilled jobs (such as hospital technicians, architects, engineers) are also at risk. This could lead to civil unrest with large numbers of young people out of work, with few or no employment prospects, unless they have a sought-after skill.

History has shown, however, that previous technological advances all ended up creating a net increase in jobs. As automation takes over, jobs on the factory floor are lost, but production becomes much faster and more efficient, thus requiring an increase in the number people doing tasks that the automation process cannot yet do, such as quality control, test driving new vehicles and so on. Technology creates new jobs which are generally more interesting to humans than the manual jobs which are lost. However, history does not always repeat itself, so we need to prepare ourselves for a large reduction in employment and think about how to redistribute wealth so that the overall impact of AI will be positive.

It is predicted that, eventually, 99% of all jobs could be eliminated since the increase in the use of AI is exponential – competition between countries and companies to expand their economies will continue to fuel this growth. One question that might be legitimately asked is, ‘if 99% of jobs disappear, who will build the robots and maintain them?’ To answer that question, let us consider a present-day solution to the question. 3D printers are actually now being designed and made by other 3D printers with no human interaction – the whole process is automatic with AI algorithms in control of the building, design and maintenance of these printers. So, it seems logical that other robots/machines will build and maintain future robots and other AI systems.

An increase in AI will leave people with more time to pursue their hobbies and have a better lifestyle. Previous industrial revolutions have led to steep changes in the economies of countries that embrace the new technology. Being left behind is not an economic option but is it a good environmental option?

Improvements in AI technology can have a positive impact on the environment. Scientists now have more information than ever about what affects the environment. AI can help by finding patterns and interconnections within the thousands of data sets. This helps scientists make informed predictions about the environment and potential climate change. Since this analysis is very complex, the use of AI systems can speed up this process incredibly and allow the human race to take action much faster than they could by present methods. Here are some potential ways in which AI can help:

- AI can help us to conserve natural resources (for example, improve the conservation of water supplies).
- Detection of pollution in the air and in the seas using AI is much more accurate, allowing scientists to pinpoint the source(s) of pollution more accurately and much faster.
- In the future it could be possible to combine weather forecasting and AI to allow for better predictions about renewable energy resources needed for the next few days. This would lead to a more precise automated renewable energy forecast using solar, tide, thermal and wind energy generation.
- AI would allow us to learn from nature's ecosystems by monitoring and modelling, for example, a river's ecosystem. This would enable us to gain a better understanding of what can affect the delicate balance of life in the river. Such real-time environmental monitoring would allow us to quickly take remedial action before the affects became irreversible. AI would make this possible due to the ability to analyse vast amounts of very complex (inter-related) data.

We will now look at three particular areas where AI could have a large impact.

Transport

Some taxi companies are already looking at the introduction of autonomous (driverless) cars. A customer can call up the taxi using an app on their mobile phone, which also automatically handles the payment. Information about the taxi (such as its location and estimated arrival time) would be sent to the mobile phone until the driverless taxi arrives at the exact pick-up point. There would not be any people anywhere in the chain, with AI systems taking total control. Some car manufacturers are on the brink of actually supplying autonomous vehicles (cars, buses and trucks). This would be much more efficient but would put many drivers out of a job.

Criminal justice system

Advances in facial recognition systems is making fingerprinting in forensic science almost obsolete. AI is also being used to automate legal work and some courts in the USA have trialled the use of AI to sentence criminals and even decide if a prisoner is eligible for parole. Is this a bad thing? Here are some questions to think about:

- Does government use of AI need a warrant to allow online data to be searched for all potential criminal activity?
- Can AI be used to listen in to our mobile phone conversations and assess our emails? Social media companies are already coming under pressure in this area – would AI help this or could criminals make use of it to hide criminal activity?
- What about legal malpractice – what would be the mechanism to challenge an AI inspired legal decision?
- How do we ensure no bias creeps into AI decision making processes? The software being

trialled in the USA to determine a prisoner's suitability for parole is already showing bias against black African Americans. How do we ensure such prejudices by governments and individuals when using AI systems is not allowed to occur?

Advertising and use of data

You may remember the Cambridge Analytica scandal in 2018 which hinged around potential misuse of data obtained from a social media company (nearly 90 million profiles had been used by the company leading some people to believe it had influenced the 2016 USA presidential elections). AI could reduce such occurrences by allowing much closer monitoring. It would need to be very sophisticated and act quickly to have any real impact – human beings certainly could not respond fast enough.

Algorithms can now tailor advertising aimed at specific people by using AI machine learning – this is done by building personality profiles of every internet and mobile phone user. Data is picked up from search engines, social media and visits to websites – all this data can be analysed by machine learning algorithms (see Chapter 18 for more details).

ACTIVITY 7C

Look through this chapter on the impacts of AI and produce a short essay or wall display highlighting the pros and cons. Draw a reasoned conclusion and debate the overall impact of AI with your classmates.

ACTIVITY 7D

1 In 2017, Diane Bryant, the chief operating officer of Google Cloud, claimed that AI can:

- help us manage the Earth's very scarce resources
- improve cancer diagnosis using precision medicine leading to customised treatments
- lead to improvements in human rights in many countries due to cloud computing, better connectivity and reduced costs in developing faster computers.

Describe, with examples, why Ms Bryant's claims could help people in the future.

2 Give **three** different examples of AI.
For each of your examples, give **one** benefit and **one** drawback to the general public.

End of chapter questions

1 Nicolae has joined a software company as a new team manager. During his induction he was given a presentation on the company's code of conduct and the company's expected ethical behaviour.
He was given hand-outs after the presentation which included the code of conduct and ethical behaviour.

a) Explain what is meant by the term *611*.

[2]

b) Describe the differences between behaving in an unethical manner and in an illegal manner.

[3]

c) Nicolae joins a team writing new software in a programming language unfamiliar to him. Part of his job will be to visit a client and oversee the team writing the software to meet the client's requirements.
He has little previous experience of working off-site at the client's premises, and has to depend on a junior colleague to help him through the process. This makes Nicolae uncomfortable in his role as project manager.
After six months with the company, Nicolae will have a meeting with his own line manager. The line manager will check Nicolae's progress against the IEEE eight principles and code of practice. Nicolae has decided to raise three issues with his line manager.

i) Describe **three** issues he could legitimately raise.

[3]

ii) State which of the IEEE's eight principles each issue described in part c) i) comes under.

[3]

iii) Describe what actions the line manager should take to address the three issues you raised in part c) i).

[3]

2 a) Name **three** types of software licensing.

[3]

b) For each example, describe **three** features which identify the differences between them.

[3]

c) Describe how copyright issues affect each type of named software licensing.

[3]

3 a) Computers over the years have been described as first to fifth generation.
Identify the generation that is associated with AI.

[1]

A first
B second
C third
D fourth
E fifth

b) AI is involved in problem-solving.
Identify the term that is used to describe the 'common sense' part of problem-solving.

[1]

A analysis
B critical design
C heuristics
D programming
E sampling

c) Identify the statement that best describes AI.

[1]

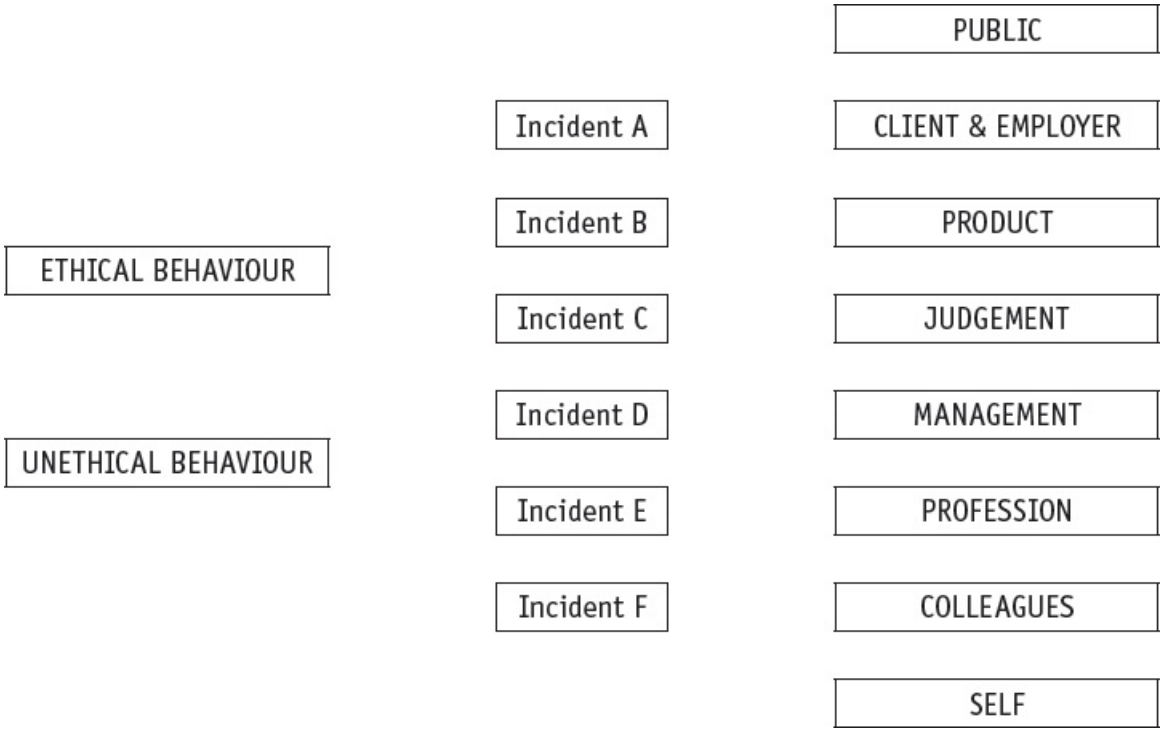
A inputting knowledge into a computer
B programming a computer using an expert's experiences
C playing a strategic game, such as chess
D making a machine behave in an intelligent way
E using a computer to mimic human behaviour

d) Identify the AI process that involves repetition, evaluation and then refinement.

[1]

A diagnostics
B fact finding
C heuristics
D interpretation
E iteration

4 The IEEE Software Engineering Code of Ethics uses eight key principles shown in the right-hand column of the following diagram.
Tom is employed as a tester with a software company. He is keen to become a trainee programmer.
The middle column in the diagram labels six incidents which have happened to Tom this week. The table that follows the diagram describes each incident.



Incident	Description

A	Tom has received some phishing emails. He reported this to the bank they were supposed to come from.
B	Tom has asked his manager if they will pay for him to attend a programming course.
C	Tom is testing beta versions of new games software at work. He copies the software on to CD-Rs and sells them to his friends.
D	Tom has completed the application forms to join the Chartered Institute for IT.
E	Tom finds it difficult to work with one of his colleagues. His way of dealing with this has been to refuse to speak with the colleague.
F	Tom's manager had considered the testing of a new game was completed. Tom reported to his manager that he thought there were still bugs which needed to be rectified.

a) Copy the diagram above and connect each of the six incidents to either ethical behaviour or unethical behaviour.

[2]

b) Consider each incident you have identified as **ethical behaviour**.
Indicate the IEE category each incident maps to.

[4]

Adapted from Cambridge International AS & A Level Computer Science 9608 Paper 12 Q5 November 2017

8 Databases

In this chapter, you will learn about

- the limitations of a file-based approach to storage and retrieval of data
- the features of a relational database that overcome the limitations of a file-based approach
- the terminology associated with a relational database model
- entity-relationship (E-R) diagrams to document database design
- normalisation to third normal form (3NF)
- producing a normalised database design for a given set of data or tables
- the features provided by a database management system (DBMS)
- the software tools provided by a DBMS
- the creation and modification of a database structure using a database definition language (DDL)
- queries and the maintenance of a database using a database manipulation language (DML)
- using SQL as a DDL and as a DML
- how to understand a given SQL script
- how to write an SQL script.

8.1 Database concepts

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you read the first part of this chapter.

1 Databases are commonly used to store large amounts of data in a well organised way. Identify **three** databases that are storing information about you.

2 **a)** Name **three** of the most commonly used database management systems.

b) Give **four** benefits of using a database.

3 Relational databases use their own terminology.

a) Explain what the terms *122* and *122* mean.

b) Identify and explain the meaning of **three** or more terms used with relational databases.

c) What is a normalised relational database?

Key terms

Database – a structured collection of items of data that can be accessed by different applications programs.

Relational database – a database where the data items are linked by internal pointers.

Table – a group of similar data, in a database, with rows for each instance of an entity and columns for each attribute.

Record (database) – a row in a table in a database.

Field – a column in a table in a database.

Tuple – one instance of an entity, which is represented by a row in a table.

Entity – anything that can have data stored about it, for example, a person, place, event, thing.

Attribute (database) – an individual data item stored for an entity, for example, for a person, attributes could include name, address, date of birth.

Candidate key – an attribute or smallest set of attributes in a table where no tuple has the same value.

Primary key – a unique identifier for a table. It is a special case of a candidate key.

Secondary key – a candidate key that is an alternative to the primary key.

Foreign key – a set of attributes in one table that refer to the primary key in another table.

Relationship – situation in which one table in a database has a foreign key that refers to a primary key in another table in the database.

Referential integrity – property of a database that does not contain any values of a foreign key that are not matched to the corresponding primary key.

Index (database) – a data structure built from one or more columns in a database table to speed up searching for data.

Entity-relationship (E-R) model or E-R diagram – a graphical representation of a database and the relationships between the entities.

Normalisation (database) – the process of organising data to be stored in a database into two or more tables and relationships between the tables, so that data redundancy is minimised.

First normal form (1NF) – the status of a relational database in which entities do not contain repeated groups of attributes.

Second normal form (2NF) – the status of a relational database in which entities are in 1NF and any non-key attributes depend upon the primary key.

Third normal form (3NF) – the status of a relational database in which entities are in 2NF and all non-key attributes are independent.

Composite key – a set of attributes that form a primary key to provide a unique identifier for a table.

8.1.1 The limitations of a file-based approach

A file is a collection of items of data. It can be structured as a collection of records, where each record is made up of fields containing data about the same ‘thing’. Individual elements of data can be called data items.

When a program is used for data processing, the organisation of any records used depends on how the program is written. Records can be fixed or variable in length and each record may also contain information about its structure, for example, the number of fields or the length of the record. If these records are to be processed by another program, that program must be written to use the exact same record structure. If the structure is changed by one program, the other program must be rewritten as well. This can cause problems if updating programs is not carefully managed.

For example, a business keeps separate payroll files and sales files. Each file is used by a different application.

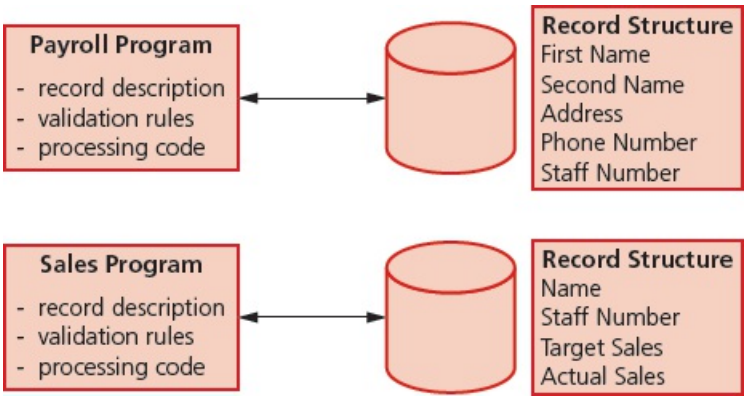


Figure 8.1 File-based approach

Several problems have occurred using this file-based approach. The name of a member of staff and their staff number are stored twice. The way the staff name is stored is different for each program. If the staff number was changed by the payroll program and not by the sales program, these fields may contain different values for the same member of staff. The fields in the two files are also in a different order: the staff number is the fifth field in the payroll record and the second field in the sales file.

A file-based approach is limited because

- storage space is wasted when data items are duplicated by the separate applications and some data is redundant
- data can be altered by one application and not by another; it then becomes inconsistent
- enquiries available can depend on the structure of the data and the software used so the data is not independent.

ACTIVITY 8A

Match the problems with the payroll and sales system to the limitations of a file-based approach set out above.

8.1.2 The advantages of a relational database over a file-based approach

What is a database? There are many different definitions of a database, such as:

> … A (large) collection of data items and links between them, structured in a way that allows it to be accessed by a number of different applications programs. The term is also used loosely to describe any collection of data.
>
> *BCS Glossary of Computing, 14th Edition*
>
> … An electronic filing cabinet which allows the user to perform various tasks including: adding new empty files, inserting data into existing files, retrieving data from existing files, updating data in existing files and cross-referencing data in files.
>
> *An Introduction to Database Systems (sixth edition) by CJ Date*

More straightforwardly, a **database** is a structured collection of items of data that can be accessed by different applications programs. Data stored in databases is structured as a collection of records, where each record is made up of fields containing data about the same 'thing'. A **relational database** is a database in which the data items are linked by internal pointers.

Using the same example as previously, a business keeps a database for payroll and sales data. A payroll application is used for the payroll and a sales processing application is used for sales.

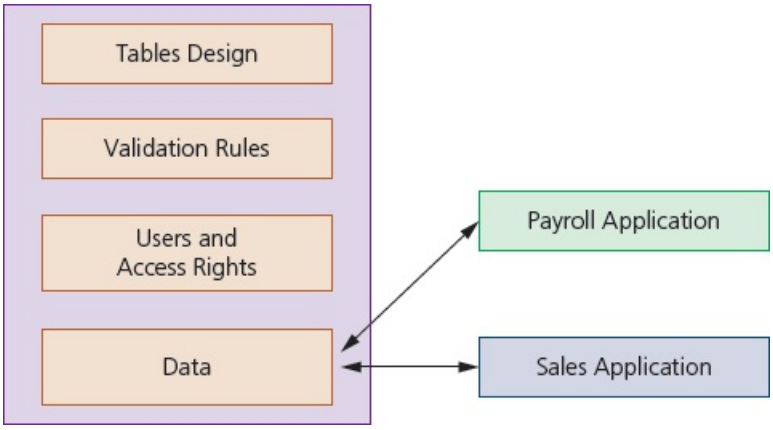


Figure 8.2 Database approach

The problems that occurred using the file-based approach have been solved. The name of a member of staff and their staff number are only stored once. So, any changes made to the data by the payroll application will be seen by the sales processing application and vice versa. The fields are the same and in the same order.

A database approach is beneficial because

- storage space is not wasted as data items are only stored once, meaning little or no redundant data
- data altered in one application is available in another application, so the data is consistent
- enquiries available are not dependent on the structure of the data and the software used, so the data is independent.

8.1.3 Relational database model terminology

In order to rigorously define the structure of a relational database we need to be able to understand and use the terminology associated with a relational database.

A relational database data structure can look similar to a file-based structure as it also consists of records and fields. A **table** is a group of similar data, in a database, with rows for each instance of an entity and columns for each attribute. A **record** is a row in a table in a database. A **field** is a column in a table in a database.

For example, a database of students in a school could contain the table **Student** with a record for each student that contains the fields **First Name**, **Second Name**, **Date of Birth** and **Class ID**.

First Name	Second Name	Date Of Birth	Class ID
Noor	Baig	09/22/2010	7A
Ahmed	Sayed	06/11/2010	7B
Tahir	Hassan	01/30/2011	7A

← row is a record

↑
column is a field

Table 8.1 Part of a student table

Now data is independent of the program processing it. The terms record and field are also used in file processing, so there is more rigorous terminology used specifically for relational databases. Files of data are replaced by tables, with each row of a table representing a record (a **tuple**, sometimes called a logical record or an occurrence of an **entity**). Each column of the table is an **attribute** that can also be referred to as a field.

An entity is anything that can have data stored about it, such as a person, place, event or object. An attribute is an individual data item stored for an entity; to use the same example as before, for a student attributes could include first name, second name, date of birth and class. As stated before, a table is a group of similar data, in a database, with rows for each instance of an entity and columns for each attribute. A tuple is one instance of an entity, which is represented by a row in a table.

First Name	Second Name	Date Of Birth	Class ID
Noor	Baig	09/22/2010	7A
Ahmed	Sayed	06/11/2010	7B
Tahir	Hassan	01/30/2011	7A

← each row is a tuple

↑
each column is an attribute

Table 8.2 Part of a table for student entity

Data is shared between applications using the database. In order to ensure the consistency of data updating is controlled or automatic, so that any copies of a data item are changed to the new value. Also, in order to reduce the number of copies of a data item to a minimum, a relational database uses pointers between tables. These pointers are keys that provide relationships between tables.

There are different types of keys.

- A **candidate key** is an attribute or smallest set of attributes in a table where no tuple has the same value.
- A **primary key** is a unique identifier for a table, it is a special case of a candidate key.
- A **secondary key** is a candidate key that is an alternative to the primary key.
- A **foreign key** is a set of attributes in one table that refer to the primary key in another table.

For example, a database of chemical elements contains a table **Elements** with attributes **Symbol**, **Name** and **Atomic Weight**. As all these attributes are unique to each element, all are candidate keys. One of these could be chosen as the primary key, for example Symbol. Then the other two attributes, Name and Atomic Weight, would be secondary keys.

all attributes are candidate keys

Symbol	Name	Atomic Weight
H	Hydrogen	1.008
Li	Lithium	6.94
Na	Sodium	22.990

Symbol is the primary key

Name and Atomic Weight are secondary keys

Table 8.3 Part of a table of elements

Most tables have only one candidate key, which is used as the primary key. For example, the student table could have an extra attribute Student ID, which is unique to each student.

Student ID	First Name	Second Name	Date Of Birth	Class ID
S1276	Noor	Baig	09/22/2010	7A
S1277	Ahmed	Sayed	06/11/2010	7B
S2199	Tahir	Hassan	01/30/2011	7A

Student ID is the primary key and the candidate key

Table 8.4 Part of a table for student entity

Relationships

A **relationship** is formed when one table in a database has a foreign key that refers to a primary key in another table in the database. In order to ensure **referential integrity** the database must not contain any values of a foreign key that are not matched to the corresponding primary key.

Most databases include more than one table. For example, a school database could contain the table **Student** and another table **Class** that contains the **Class ID**, the **Teacher Name** and **Location** of classroom. Only values for **Class ID** that are stored in the **Class** table can be used as the foreign key in the **Student** table.

Student ID	First Name	Second Name	Date Of Birth	Class ID
S1276	Noor	Baig	09/22/2010	7A
S1277	Ahmed	Sayed	06/11/2010	7B
S2199	Tahir	Hassan	01/30/2011	7A

↑
Class ID is the foreign key

Table 8.5 Part of a table for student entity

Class ID	Teacher Name	Location
7A	Mr Khan	Floor 2 Room 3
7B	Miss Malik	Floor 2 Room 4
7C	Miss Gill	Floor 2 Room 5

↑
Class ID is the primary key

Table 8.6 Part of a table for class entity

Relationships can take several forms

- one-to-one, 1:1
- one-to-many, 1:m
- many-to-one, m:1
- many-to-many, m:m.

The relationship between **Student** and **Class** is many-to-one, as one value of the attribute **Class ID** may appear many times in the **Student** table but only once in the **Class** table.

In order to speed up searching for data, an **index** can be used. This is a data structure built from one or more columns in a database table. The **Student** table could be indexed on **Class**, **Second Name** and **First Name** to provide class lists in alphabetical order of **Second Name**.

8.1.4 Entity-relationship (E-R) diagrams

An **E-R diagram** can be used to document the design of a database. This provides an easily understandable visual representation of how the entities in a database are related.

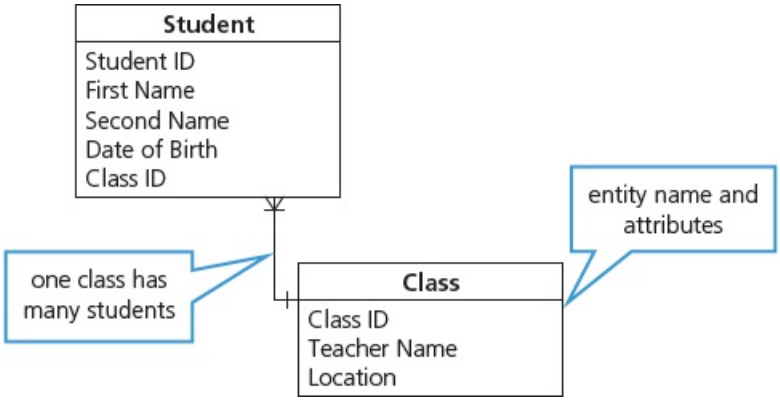


Figure 8.3 E-R diagram for school database

Relationships may be mandatory or optional. For example, in a workroom with desks, each employee has one desk, but there could be spare desks. The relationship between desk and employee is zero or one, so this relationship is optional. The relationship between mother and child is mandatory because every mother must have at least one child, so the relationship is one or many. The type of relationship and whether it is mandatory or optional gives the cardinality of the relationship. The cardinality of relationships is shown in Figure 8.4.

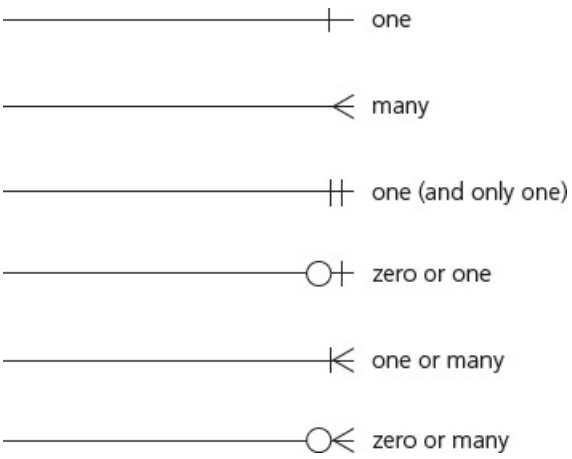


Figure 8.4 Cardinality of relationships

ACTIVITY 8B

The School database will also include the following details about each teacher:

- teaching licence number
- date of birth
- address.

A teacher can have more than one class. A table **Teacher** is to be added.

List the attributes for this new table. Show the change that should be made to the attributes in the **Class** table.

Draw the new E-R diagram for the three tables in the database.

EXTENSION ACTIVITY 8A

In small groups, identify suitable entity relationships for each example of cardinality shown above. Explain your findings to another group or the whole class.

8.1.5 The normalisation process

Normalisation is used to construct a relational database that has integrity and in which data redundancy is reduced. Tables that are not normalised will be larger. As more data is stored, it will be harder to update the database when changes are made and more difficult to extract the required data to answer queries.

For example, if the **School** database is held in a single table it could look like this:

Student ID	First Name	Second Name	Date Of Birth	Class ID	Location	Teacher Name	Licence Number	Address	Teacher Date Of Birth
S1276	Noor	Baig	09/22/2010	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985
S1277	Ahmad	Sayed	06/11/2010	7B	Floor 2 Room 4	Miss Malik	68943	School House 2	12/14/1988
S1299	Tahir	Hassan	01/30/2011	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985

Table 8.7

This could cause problems when alterations are made to the records. Every time a new student is added, the teacher's name, address, licence number, date of birth, and the location of the classroom need to be added as well. If Mr Khan leaves the school and is replaced by another teacher, then every record containing his name and other details needs to be changed. If all the students from Class 7B leave, then all the details about Class 7B will be lost.

The rules for normalisation are set out as follows.

1. **First normal form (1NF)** – entities do not contain repeated groups of attributes.
2. **Second normal form (2NF)** – entities are in 1NF and any non-key attributes depend upon the primary key. There are no partial dependencies.
3. **Third normal form (3NF)** – entities are in 2NF and all non-key attributes are independent. The table contains no non-key dependencies.

When the database is in 3NF, all attributes in a table depend upon the key, the whole key and nothing but the key.

The School database also includes subject choices for each student. For this database to be normalised, the process is:

Student ID	First Name	Second Name	Date Of Birth	Subject Name	Subject Teacher	Class ID	Location	Teacher Name	Licence Number	Address	Teacher Date Of Birth
S1276	Noor	Baig	09/22/2010	Maths, History, Geography	Mr Yee, Miss Wu, Mr Khan	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985
S1277	Ahmad	Sayed	06/11/2010	Maths, Science, Geography	Mr Yee, Miss Yo, Mr Khan	7B	Floor 2 Room 4	Miss Malik	68943	School House 2	12/14/1988
S1299	Tahir	Hassan	01/30/2011	Maths, Science, History	Mr Yee, Miss Yo, Miss Wu	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985

Table 8.8

First normal form (1NF)

The un-normalised School database can be represented as follows.

STUDENT(StudentID, FirstName, SecondName, DateOfBirth, SubjectName, SubjectTeacher, SubjectName, SubjectTeacher, SubjectName, SubjectTeacher, ClassID, Location, TeacherName, LicenceNumber, Address, TeacherDateOfBirth).

STUDENT is the table name; the attributes are listed in order and the primary key is underlined.

The student's subjects and the subject teacher are the repeating attributes. For the database to be in first normal form, these need to be removed to a separate table and linked to the original table with a foreign key.

Student ID	First Name	Second Name	Date Of Birth	Class ID	Location	Teacher Name	Licence Number	Address	Teacher Date Of Birth
S1276	Noor	Baig	09/22/2010	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985
S1277	Ahmad	Sayed	06/11/2010	7B	Floor 2 Room 4	Miss Malik	68943	School House 2	12/14/1988
S1299	Tahir	Hassan	01/30/2011	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985

Student ID	Subject Name	Subject Teacher
S1276	Maths	Mr Yee
S1276	History	Miss Wu
S1276	Geography	Mr Khan
S1277	Maths	Mr Yee
S1277	Science	Miss Yo
S1277	Geography	Mr Khan
S1299	Maths	Mr Yee
S1299	Science	Miss Yo
S1299	History	Miss Wu

Table 8.9 School database in 1NF

The School database can now be represented in 1NF as follows.

STUDENT(StudentID, FirstName, SecondName, DateOfBirth, ClassID, Location, TeacherName, LicenceNumber, Address, TeacherDateOfBirth).

STUDENTSUBJECT(StudentID, SubjectName, SubjectTeacher).

The primary key for the **STUDENTSUBJECT** table is a **composite key** formed from the two attributes **StudentID** and **SubjectName**; the attribute **StudentID** is also a foreign key that links to the **STUDENT** table.

Second normal form (2NF)

There are now two tables; in the **STUDENTSUBJECT** table the primary key is a composite key and the **SubjectTeacher** is only dependent on the **SubjectName** part of the primary key. This is a partial dependence and needs to be removed by introducing a third table, **SUBJECT**.

Student ID	First Name	Second Name	Date Of Birth	Class ID	Location	Teacher Name	Licence Number	Address	Teacher Date Of Birth
S1276	Noor	Baig	09/22/2010	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985
S1277	Ahmad	Sayed	06/11/2010	7B	Floor 2 Room 4	Miss Malik	68943	School House 2	12/14/1988
S1299	Tahir	Hassan	01/30/2011	7A	Floor 2 Room 3	Mr Khan	37952	School House 1	03/27/1985

Student ID	Subject Name
S1276	Maths
S1276	History
S1276	Geography
S1277	Maths
S1277	Science
S1277	Geography
S1299	Maths
S1299	Science
S1299	History

Subject Name	Subject Teacher
Maths	Mr Yee
History	Miss Wu
Geography	Mr Khan
Science	Miss Yo

Table 8.10 School database in 2NF

The School database can now be represented in 2NF as follows.

STUDENT(StudentID, FirstName, SecondName, DateOfBirth, ClassID, Location, TeacherName, LicenceNumber, Address, TeacherDateOfBirth)

STUDENTSUBJECT(StudentID, SubjectName)

SUBJECT(SubjectName, SubjectTeacher)

Third normal form (3NF)

There are now three tables. In the **STUDENT** table, the attributes **Location** and **TeacherName** depend upon the attribute **ClassID** and the attributes **LicenceNumber**, **Address** and **TeacherDateOfBirth** depend upon the attribute **TeacherName**. These are non-key dependencies that need to be removed to ensure that the database is in 3NF.

At this stage it is also worth inspecting the database and its contents to consider any other problems that could arise, such as the following:

- Teacher names might not be unique; therefore, it is better to use the licence number as a primary key.
- Teachers can be both class teachers and subject teachers; these need to be combined in one table.

Student ID	First Name	Second Name	Date Of Birth	Class ID
S1276	Noor	Baig	09/22/2010	7A
S1277	Ahmad	Sayed	06/11/2010	7B
S1299	Tahir	Hassan	01/30/2011	7A

Licence Number	Teacher Name	Address	Teacher Date Of Birth
37952	Mr Khan	School House 1	03/27/1985
68943	Miss Malik	School House 2	12/14/1988
35859	Mr Yee	School House 1	10/07/1985
77248	Miss Yo	School House 2	05/05/1987
72691	Miss Wu	School House 2	11/21/1989
37952	Mr Khan	School House 1	03/27/1985

Class ID	Location	Licence Number
7A	Floor 2 Room 3	37952
7B	Floor 2 Room 4	68943

Student ID	Subject Name
S1276	Maths
S1276	History
S1276	Geography
S1277	Maths
S1277	Science
S1277	Geography

S1299	Maths
S1299	Science
S1299	History

Subject Name	Licence Number
Maths	35859
History	72691
Geography	37952
Maths	77248

Table 8.11 School database in 3NF

The improved School database can now be represented in 3NF as follows.

STUDENT(StudentID, FirstName, SecondName, DateOfBirth,)

CLASS(ClassID, Location, LicenceNumber)

TEACHER(LicenceNumber, TeacherName, Address, TeacherDateOfBirth)

STUDENTSUBJECT(StudentID, SubjectName)

SUBJECT(SubjectName, LicenceNumber).

ACTIVITY 8C

Construct an E-R diagram to represent the database structure of the fully normalised school database shown above.

EXTENSION ACTIVITY 8B

Discuss any other possible problems that could occur with this database.

Hint: look at the subject table and think about subjects that could have more than one teacher or different levels. Identify an improved database structure that could solve the problem.

The School database example showed at each stage why the database was not normalised. Here is another example for you to try.

A database has been set up as a single table to store employees of a business and their contacts. Part of the database is shown below.

Employee Number	Employee Name	Position	Contact Number	Contact Name	Contact Email Address
7001	James Tey	Financial Director	28	Mary Jones	mary@xyz.com
			31	James Smith	james@pqr.com
			17	Mishal Hussani	mh@xyz.com
7002	Paul Leigh	Accountant	19	Mary Cheung	mch@abc.com
			27	Dean Knight	knd@swz.com
7011	Suzy Mey	Personnel Manager	28	Mary Jones	mary@xyz.com

Table 8.12 Un-normalised employee database

This table is not in 1NF because there are repeating attributes and the table is not in 3NF because there are non-key dependencies. The employee database can be represented as:

EMPLOYEE(EmployeeNumber, EmployeeName, Position, ContactNumber, ContactName, ContactEmailAddress).

Where **EmployeeNumber** is the primary key **ContactNumber**, **ContactName** and **ContactEmailAddress** may be repeated as often as required.

ACTIVITY 8D

Normalise the Employee database and show the new tables. Draw the E-R diagram for the normalised database.

ACTIVITY 8E

1 a) i) Describe the limitations of a file-based approach to storage and retrieval of data.

ii) Give **two** benefits of using a database management system.

b) A new relational database is to be developed. The developer needs to produce a normalised database design.

i) Explain what is meant by *normalisation*.

ii) Describe the process of normalisation.

2 A warehouse stores parts for cars for several manufacturers. A database stores the following data for each part:
Part number, part description, date last ordered, minimum order level, manufacturer name, manufacturer address, manufacturer contact details, position in warehouse, number in stock

a) Design a fully normalised database for the parts.

b) Draw the E-R diagram.

8.2 Database management systems (DBMSs)

WHAT YOU SHOULD ALREADY KNOW

Try these two questions before you read the second part of this chapter.

1 a) Name a database management system (DBMS) you have used.

b) Describe **three** tasks that you have used the DBMS for.

2 Most DBMSs include back-up procedures and access rights to keep the data secure.

a) Describe what is meant by back-up.

b) Describe what is meant by access rights.

c) How do these features help to keep data secure?

Key terms

Database management system (DBMS) – systems software for the definition, creation and manipulation of a database.

Data management – the organisation and maintenance of data in a database to provide the information required.

Data dictionary – a set of data that contains metadata (data about other data) for a database.

Data modelling – the analysis and definition of the data structures required in a database and to produce a data model.

Logical schema – a data model for a specific database that is independent of the DBMS used to build that database.

Access rights (database) – the permissions given to database users to access, modify or delete data.

Developer interface – feature of a DBMS that provides developers with the commands required for definition, creation and manipulation of a database.

Structured query language (SQL) – the standard query language used with relational databases for data definition and data modification.

Query processor – feature of a DBMS that processes and executes queries written in structured query language (SQL).

8.2.1 How a DBMS addresses the limitations of a file-based approach

Data redundancy issue

This is solved by storing data in separate linked tables, which reduces the duplication of data as most items of data are only stored once. Items of data used to link tables by the use of foreign keys are stored more than once. The DBMS will flag any possible errors when any attempt is made to accidentally delete this type of item.

Data inconsistency issue

This is also solved by storing most items of data only once, allowing updated items to be seen by all applications. As data is not inconsistent, the integrity of the data stored is improved. Consistent data is easier to maintain as an item of data will only be changed once, not multiple times, by different applications.

Data dependency issue

Data is independent of the applications using the database, so changes made to the structure of the data will be managed by the DBMS and have little or no effect on the applications using the database. Any fields or tables added to or removed from the database will not affect the applications that do not use those fields/tables, as each application only has access to the fields/tables it requires.

Information from a database is more easily available in a form that is required so it is not dependent on the structure of the data and the application used. A DBMS usually includes facilities to query the data stored using a defined query language or a query-by-example facility.

The DBMS approach

A **DBMS** uses a more structured approach to the **management**, organisation and maintenance of data in a database. An already-defined data structure can be used to set up and create the database. The entry of new data, the storage of data, the alteration and deletion of data are all managed by the DBMS.

A DBMS uses a **data dictionary** to store the metadata, including the definition of tables, attributes, relationships between tables and any indexing. The data dictionary can also define the validation rules used for the entry of data and contain data about the physical storage of the data. The use of a data dictionary improves the integrity of the data stored, helping to ensure that it is accurate, complete and consistent.

Data modelling is an important tool used to show the data structure of a database. An E-R diagram is an example of a data model. A **logical schema** is a data model for a specific database that is independent of the DBMS used to build the database.

A DBMS helps to provide data security to prevent the unwanted alteration, corruption, deletion or sharing of data with others that have no right to access it.

Security measures taken by a DBMS can include

- using usernames and passwords to prevent unauthorised access to the database
- using access rights to manage the actions authorised users can take, for example, users could read/write/delete, or read only, or append only
- using **access rights** to manage the parts of the database they have access to, for example, the provisions of different views of the data for different users to allow only certain users access to some tables
- automatic creation and scheduling of regular back-ups
- encryption of the data stored
- automatic creation of an audit trail or activity log to record the actions taken by users of the database.

8.2.2 The use and purpose of DBMS software tools

Developer interface

The **developer interface** allows a developer to write queries in **structured query language (SQL)** rather than using query-by-example. These queries are then processed and executed by the **query processor**. This allows the construction of more complex queries to interrogate the database.

Query processor

The query processor takes a query written in SQL and processes it. The query processor includes a DDL interpreter, a DML compiler and a query evaluation engine. Any DDL statements are interpreted and recorded in the database's data dictionary. DML statements are compiled into low level instructions that are executed by the query evaluation engine. The DML compiler will also optimise the query.

ACTIVITY 8F

1 a) Describe how a DBMS overcomes the limitations of a file-based approach to the storage and retrieval of data.

b) Describe how a DBMS ensures that data stored in a database is secure.

2 a) Describe **three** features provided by a DBMS.

b) A school stores timetabling data for all pupils and classes.
Which features could a DBMS use to ensure that the administrators, teachers and pupils can only see the information available to them?

8.3 Data definition language (DDL) and data manipulation language (DML)

WHAT YOU SHOULD ALREADY KNOW

Try this exercise before you read the third part of this chapter.
Using a DBMS with a graphical user interface (GUI), create the student database used in Section 8.1.5.
Write the following queries using a query-by-example form.

1. A list of all the teachers and their subjects.
2. A list of the pupils in class 7A in alphabetical order of second name.
3. A list of the students studying each subject.

You may want to save this database to practise your SQL commands.

Key terms

Data definition language (DDL) – a language used to create, modify and remove the data structures that form a database.

Data manipulation language (DML) – a language used to add, modify, delete and retrieve the data stored in a relational database.

SQL script – a list of SQL commands that perform a given task, often stored in a file for reuse.

8.3.1 Industry standard methods for building and modifying a database

DBMSs use a **data definition language (DDL)** to create, modify and remove the data structures that form a relational database. DDL statements are written as a script that uses syntax similar to a computer program.

DBMSs use a **data manipulation language (DML)** to add, modify, delete and retrieve the data stored in a relational database. DML statements are written in a script that is similar to a computer program.

These languages have different functions: DDL is used for working on the relational database structure, whereas DML is used to work with the data stored in the relational database.

Most DBMSs use structured query language (SQL) for both data definition and data manipulation. SQL was developed in the 1970s and since then it has been adopted as an industry standard.

8.3.2 SQL (DDL) commands and scripts

In order to be able to understand and write SQL, you should have practical experience of writing **SQL scripts**. There are many applications that allow you to do this. For example, MySQL and SQLite are freely available ones. When using any SQL application it is important that you check the commands available to use as these may differ slightly from those listed below.

You will need to be able to understand and use the following DDL commands.

SQL (DDL) command	Description
CREATE DATABASE	Creates a database
CREATE TABLE	Creates a table definition
ALTER TABLE	Changes the definition of a table
PRIMARY KEY	Adds a primary key to a table
FOREIGN KEY … REFERENCES …	Adds a foreign key to a table

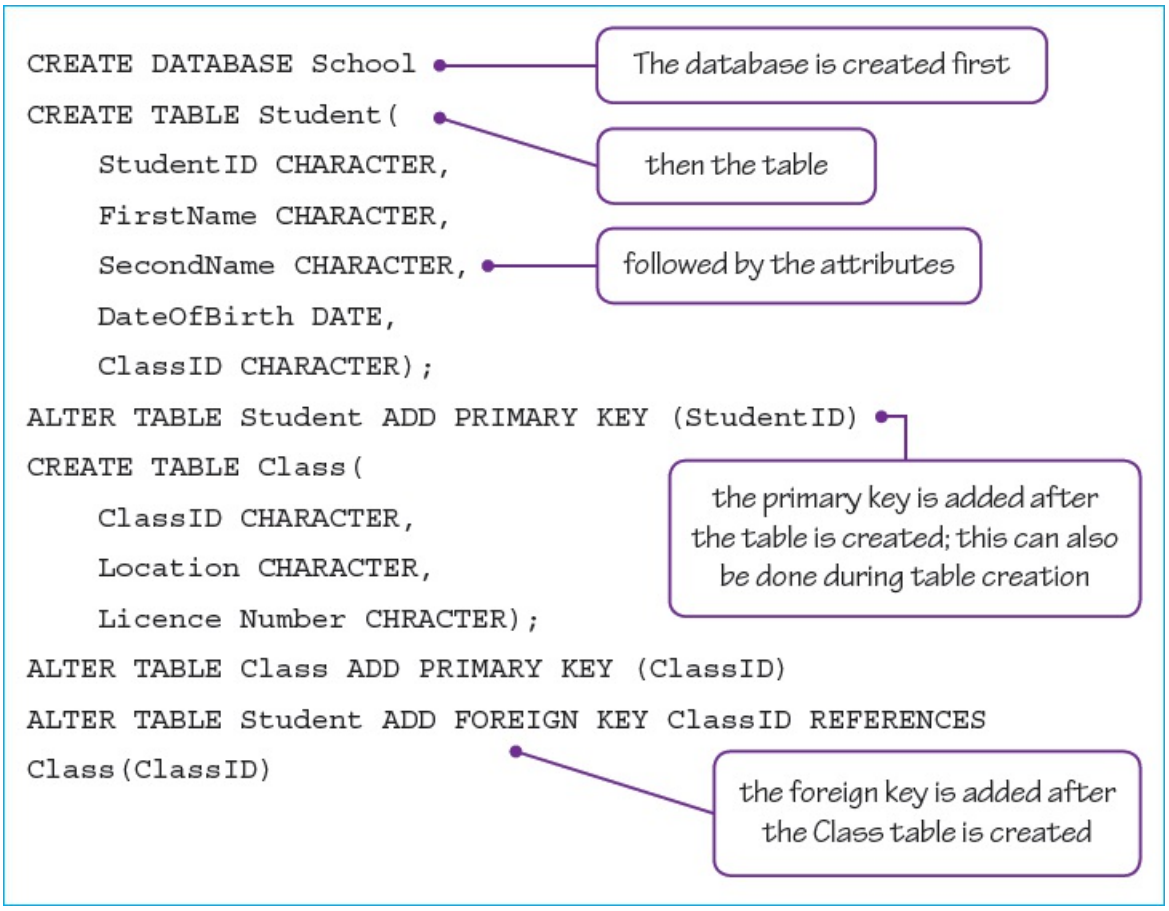
Table 8.13 DDL commands

You also need to be familiar with the following data types used for attributes in SQL.

Data types for attributes	Description
CHARACTER	Fixed length text
VARCHAR(n)	Variable length text
BOOLEAN	True or False; SQL uses the integers 1 and 0
INTEGER	Whole number
REAL	Number with decimal places
DATE	A date usually formatted as YYYY-MM-DD
TIME	A time usually formatted as HH:MM:SS

Table 8.14 Data types for attributes

Here are some examples of DDL that could have been used when the school database was created.



ACTIVITY 8G

Create the Teacher table and add the **Licence Number** as a foreign key to the Class table.

8.3.3 SQL (DML) commands and scripts

In order to be able to understand and write SQL, you should have practical experience of writing SQL scripts and queries. There are many applications that allow you to do this. Again, MySQL and SQLite are freely available ones. You can also write SQL commands in Access. When using any SQL application, it is important that you check the commands available to use as these may differ slightly from those listed below.

You will need to be able to understand and use the following DML commands.

SQL (DML) query command	Description
SELECT FROM	Fetches data from a database. Queries always begin with SELECT.
WHERE	Includes only rows in a query that match a given condition
ORDER BY	Sorts the results from a query by a given column either alphabetically or numerically
GROUP BY	Arranges data into groups
INNER JOIN	Combines rows from different tables if the join condition is true
SUM	Returns the sum of all the values in the column
COUNT	Counts the number of rows where the column is not NUL
AVG	Returns the average value for a column with a numeric data type

SQL (DML) maintenance commands	Description
INSERT INTO	Adds new row(s) to a table
DELETE FROM	Removes row(s) from a table
UPDATE	Edits row(s) in a table

Table 8.15 DML commands

Here are some examples of DML that could have been used to query and update the school database.

This query will show, in alphabetical order of second name, the first and second names of all students in class 7A:

```
SELECT FirstName, SecondName
FROM Student
WHERE ClassID = '7A'
ORDER BY SecondName
```

This query will show the teacher's name and the subject taught:

```
SELECT Teacher.TeacherName AND Subject.SubjectName
FROM Teacher INNER JOIN Subject ON Teacher.
LicenceNumber = Subject.LicenceNumber
```

ACTIVITY 8H

Create a query to show each student's First Name, Second Name and the subjects studied by each student.

This statement will insert a row into the Student table:

```
INSERT INTO Student VALUES(S1301, Peter, Probert,
06/06/2011, 7A)
```

If the values for all the columns are not known, then the table columns need to be specified before the values are inserted:

```
INSERT INTO Student(StudentID, FirstName, SecondName)
VALUES(S1301, Peter, Probert)
```

These statements will delete the specified row(s) from the Student table (take care: `DELETE FROM Student` will delete the whole table!):

```
DELETE FROM Student
WHERE StudentID = 'S1301'
```

The values for any column can be counted, totalled or averaged.

For example, if an extra column was added to the **STUDENTSUBJECT** table showing each student's exam mark in that subject, the following query could be used to total all of the students' exam marks:

```
SELECT SUM (ExamMark)
FROM STUDENTSUBJECT
```

ACTIVITY 8I

Use the SQL statements AVG and COUNT to find the average mark and count how many marks have been recorded.

End of chapter questions

1 A database has been designed to store data about programmers and the programs they have developed.

These facts help to define the structure of the database:

– Each programmer works in a particular team.

– Each programmer has a unique first name.

– Each team has one or more programmer.

– Each program is for one customer only.

– Each programmer can work on any program.

– The number of days that each programmer has worked on a program is recorded.

The table ProgDev was the first attempt at designing the database.

FirstName	Team	ProgramName	NoOfDays	Customer
Alice	WC	TV control	3	SKM
		Ice alert	2	WZP
		Digital camera	6	HNC
Charles	PC	Oil flow	1	GEB
		Rescue Pack	8	BGF
Ahmad	QR	TV control	2	SKM
		Accounts	8	ARC
		Digital camera	4	HNC
		Test Pack	3	GKN

a) State why the table is **not** in first normal form (1NF).

[1]

b) The database design is changed to:

```
Programmer (FirstName, Team)
Program (FirstName, ProgramName, NoOf Days,
Customer)
```

Table: Programmer	
FirstName	Team

Table: Program			
FirstName	ProgramName	NoOfDays	Customer

c) **i)** A relationship between the two tables has been implemented.
Explain how this has been done.

[2]

ii) Explain why the `Program` table is **not** in third normal form (3NF).

[2]

iii) Write the table definitions to give the database in 3NF.

[2]

2 A school stores a large amount of data. This includes student attendance, qualification, and contact details. The school's software uses a file-based approach to store this data.

a) The school is considering changing to a DBMS.

i) State what DBMS stands for.

[1]

ii) Describe **two** ways in which the Database Administrator (DBA) could use the DBMS software to ensure the security of the student data.

[4]

iii) A feature of the DBMS software is a query processor.
Describe how the school secretary could use this software.

[2]

iv) The DBMS has replaced software that used a file-based approach with a relational database.
Describe how using a relational database has overcome the previous problems associated with a file-based approach.

[3]

b) The database design has three tables to store the classes that students attend.

STUDENT(StudentID, FirstName, LastName, Year, TutorGroup)
CLASS(ClassID, Subject)
CLASS-GROUP(StudentID, ClassID)

Primary keys are not shown.

There is a one-to-many relationship between **CLASS** and **CLASS-GROUP**.

i) Describe how this relationship is implemented.

[2]

ii) Describe the relationship between **CLASS-GROUP** and **STUDENT**.

[1]

iii) Write an SQL script to display the StudentID and FirstName of all students who are in the tutor group 10B.
Display the list in alphabetical order of LastName.

[4]

iv) Write an SQL script to display the LastName of all students who attend the class whose ClassID is CS1.

[4]

Cambridge International AS & A Level Computer Science 9608
Paper 12 Q8 June 2016

9 Algorithm design and problem solving

In this chapter, you will learn about
- computational thinking skills (abstraction and decomposition)
- how to write algorithms that provide solutions to problems using structured English, flowcharts and pseudocode
- the process of stepwise refinement.

In order to design a computer system that performs a specific task, or solves a given problem, the task or problem has to be rigorously defined and set out, showing *what* is going to be computed and *how* it is going to be computed.

This chapter introduces tools and techniques that can be used to design a software solution to work with associated computer hardware to form a computer system.

Practice is essential to develop skills in computational thinking. Designs shown with pseudocode or flowcharts can be traced to check if the proposed solution works, but the best way to actually test that a computer system works is to code it and use it or, even better, get somebody else to use it. Therefore, practical programming activities, alongside other activities, will be suggested at every stage to help reinforce the skills being learnt and develop the skill of programming.

The programming languages to use are:
- Java
- Python
- VB.NET.

WHAT YOU SHOULD ALREADY KNOW

Can you answer these six questions and complete the following activity?

1 What is a procedure?
2 What is a function?
3 What is an algorithm?
4 What is structured English?
5 What is a flowchart?
6 What is pseudocode?

Write an algorithm using a flowchart to find the average of a number of integers. Both the number of values and each integer are to be input, and the average is to be output.

Use the flowchart of your algorithm to write the algorithm in pseudocode.

Use your pseudocode to write and test a program that includes a function to solve the problem.

9.1 Computational thinking skills

Key terms

Abstraction – the process of extracting information that is essential, while ignoring what is not relevant, for the provision of a solution.

Decomposition – the process of breaking a complex problem into smaller parts.

Pattern recognition – the identification of parts of a problem that are similar and could use the same solution.

Computational thinking is used to study a problem and formulate an effective solution that can be provided using a computer. There are several techniques used in computational thinking, including abstraction, decomposition, algorithms and pattern recognition.

9.1.1 Using abstraction

Abstraction is an essential part of computational thinking. It enables computer scientists to develop clear models for the solution to complex problems. Abstraction involves extracting information that is essential while ignoring what is not relevant for the provision of a solution, only including what is necessary to solve that problem.

Abstraction encourages the development of simplified models that are suited to a specific purpose by eliminating any unnecessary characteristics from that model. Many everyday items use abstraction, such as maps, calendars and timetables.

Maps use abstraction to show what is required for a specific purpose, for example, a road map should only show the necessary detail required to drive from one place to another. The road map in Figure 9.1 has reduced the complexity by only showing the essential details needed, such as roads, road numbers and towns, and removing other information about the terrain that would not be helpful (as shown in the satellite view).

The benefits of eliminating any unnecessary characteristics from the model include

- the time required to develop the program is reduced so the program can be delivered to the customer more quickly
- the program is smaller in size so takes up less space in memory and download times are shortened
- customer satisfaction is greater as their requirements are met without any extraneous features.

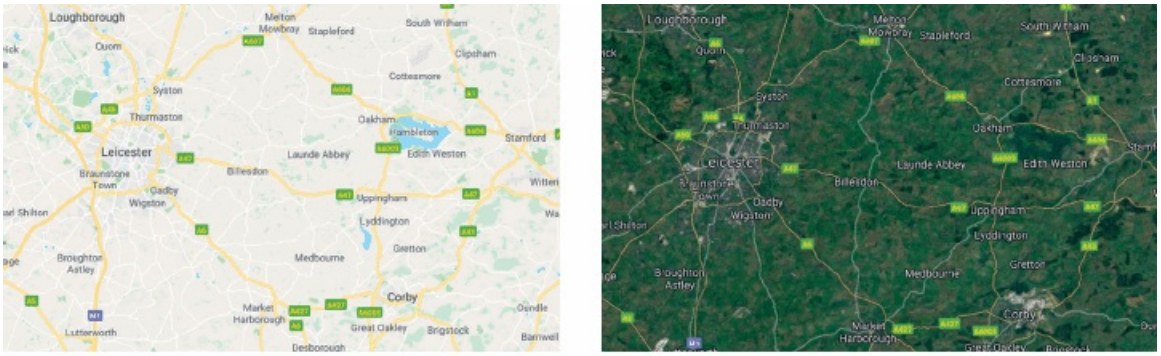


Figure 9.1 Road map and satellite view

The first stage of abstraction is to identify the purpose of the model of the situation that is to be built. The situation could be one that occurs in real life, an imaginary one, or a future event such as modeling the route of a deep space probe.

Once the purpose has been identified, sources of information need to be identified. These can include observations, views of potential users, and evidence from other existing models.

The next stage is to use the information gathered from appropriate sources to identify what details need to be included in the model, how these details should be presented and what details are extraneous and need to be removed from the model.

For example, maps are used for many different purposes and can take different forms depending

on the identified use. The purpose of the road map model in Figure 9.1 is to allow a driver to plan a journey, therefore, it includes towns and roads with their numbers. The roads depicted are a scaled down version of the actual road to help the driver visualise the route.

A rail map model has another purpose and, therefore, looks very different, only showing rail lines, stations and perhaps details about accessibility for wheelchair users at different stations. A train passenger has no need to visualise the route, so the rail lines are simplified for clarity.

9.1.2 Using decomposition

Decomposition is also an essential part of computational thinking. It enables computer scientists to break a complex problem into smaller parts that can be further subdivided into even smaller parts until each part is easy to examine and understand, and a solution can be developed for it. When a rigorous decomposition is undertaken, many simple problems are found to be more complex than at first sight.

Pattern recognition is used to identify those parts that are similar and could use the same solution. This leads to the development of reusable program code in the form of subroutines, procedures and functions. When writing a computer program, each final part is defined as a separate program module that can be written and tested as a separate procedure or function, as shown in Figure 9.2. Program modules already written and tested can also be identified and reused, thus saving development time. See Chapter 12 for further details.

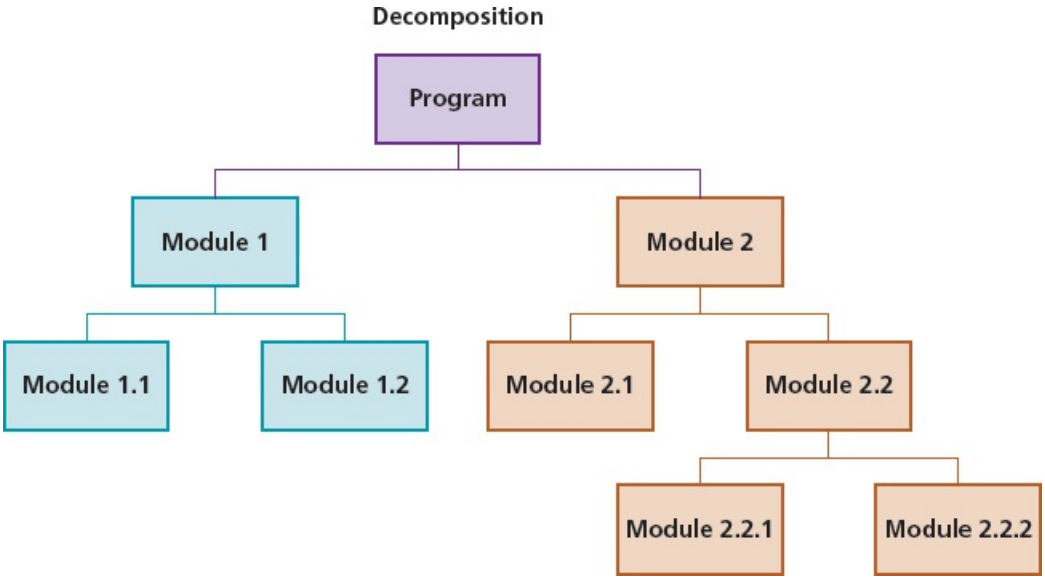


Figure 9.2 Decomposition of a program into modules

9.2 Algorithms

Key terms

Structured English – a method of showing the logical steps in an algorithm, using an agreed subset of straightforward English words for commands and mathematical operations.

Flowchart – a diagrammatic representation of an algorithm.

Algorithm – an ordered set of steps to be followed in the completion of a task.

Pseudocode – a method of showing the detailed logical steps in an algorithm, using keywords, identifiers with meaningful names, and mathematical operators.

Stepwise refinement – the practice of subdividing each part of a larger problem into a series of smaller parts, and so on, as required.

9.2.1 Writing algorithms that provide solutions to problems

There are several methods of writing algorithms before attempting to program a solution. Here are three frequently used methods.

- **Structured English** is a method of showing the logical steps in an algorithm, using an agreed subset of straightforward English words for commands and mathematical operations to represent the solution. These steps can be numbered.
- A **flowchart** shows diagrammatically, using a set of symbols linked together with flow lines, the steps required for a task and the order in which they are to be performed. These steps, together with the order, are called an **algorithm**. Flowcharts are an effective way to show the structure of an algorithm.
- **Pseudocode** is a method of showing the detailed logical steps in an algorithm, using keywords, identifiers with meaningful names and mathematical operators to represent a solution. Pseudocode does not need to follow the syntax of a specific programming language, but it should provide sufficient detail to allow a program to be written in a high-level language.

Below, you will see the algorithm from the What you should already know section on page 217 written using each of these three methods.

Structured English

1 Ask for the number of values

2 Loop that number of times

3 Enter a value in loop

4 Add the value to the Total in loop

5 Calculate and output average

Pseudocode

```
Total ← 0
PRINT "Enter the number of values to average"
INPUT Number
FOR Counter ← 1 TO Number
    PRINT "Enter value"
    INPUT Value
    Total ← Total + Value
NEXT Counter
Average ← Total / Number
PRINT "The average of ", Number, " values is ", Average
```

ACTIVITY 9A

You have been asked to write an algorithm for drawing regular polygons of any size.

In pairs, divide the problem into smaller parts, identifying those parts that are similar.

Write down your solution as an algorithm in structured English.

Swap your algorithm with another pair.

Test their algorithm by following their instructions to draw a regular polygon. Discuss any similarities and differences between your solutions.

Flowchart

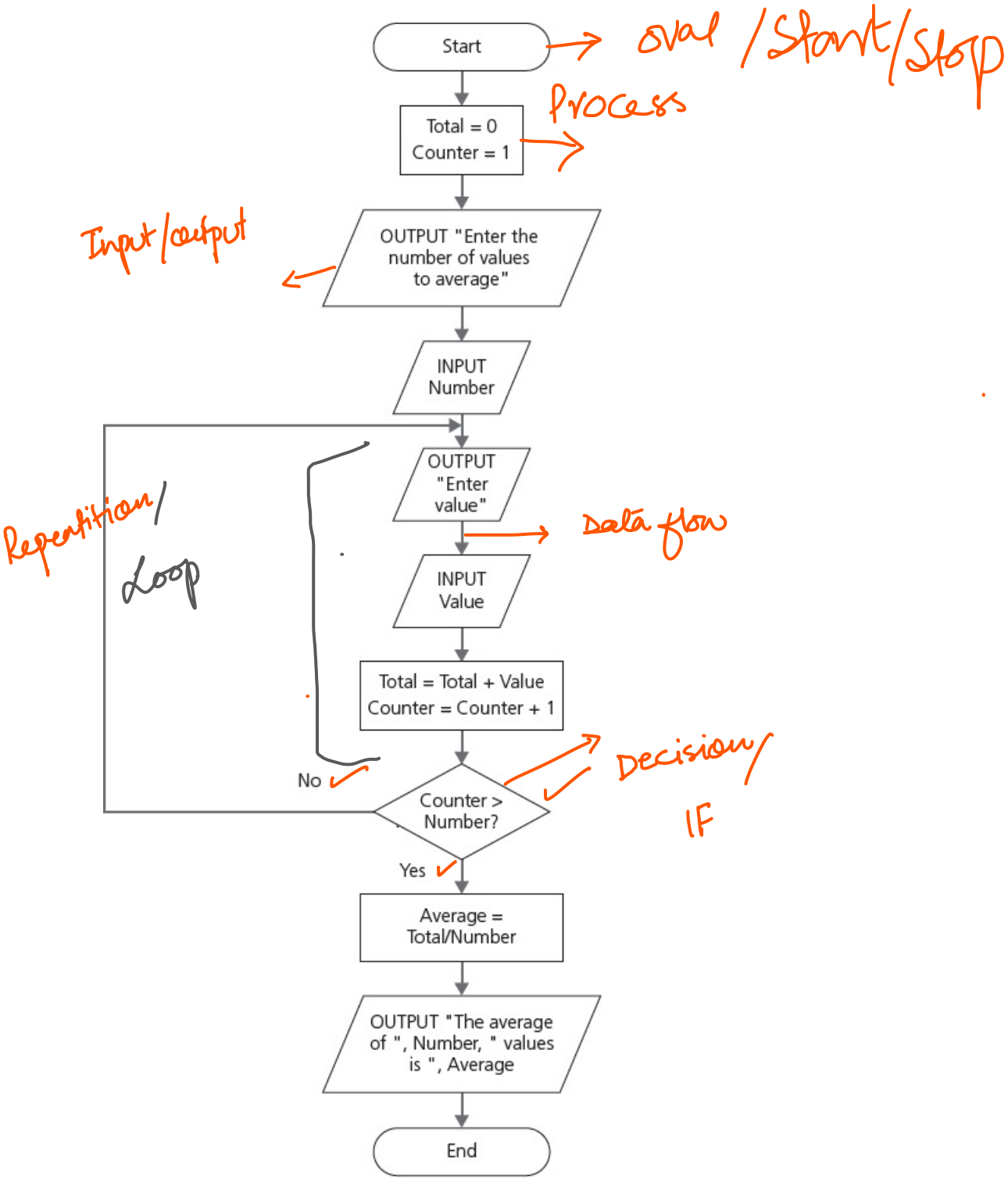

Start
Total = 0
Counter = 1
OUTPUT "Enter the number of values to average"
INPUT Number
OUTPUT "Enter value"
INPUT Value
Total = Total + Value
Counter = Counter + 1
Counter > Number?
No
Yes
Average = Total/Number
OUTPUT "The average of ", Number, " values is ", Average
End
oval /start/stop
Process
Input/output
Repetition/ Loop
data flow
Decision/ IF

Figure 9.3

9.2.2 Writing simple algorithms using pseudocode

Each line of pseudocode is usually a single step in an algorithm. The pseudocode used in this book follows the rules in the *Cambridge International AS & A Level Computer Science Pseudocode Guide for Teachers* and is set out using a fixed width font and indentation, where required, of four spaces, except for THEN, ELSE and CASE clauses that are only indented by two spaces.

All identifier names used in pseudocode should be meaningful; for example, the name of a person could be stored in the variable identified by Name. They should also follow some basic rules: they should only contain the characters A–Z, a–z and 0–9, and should start with a letter. Pseudocode identifiers are usually considered to be case insensitive, unlike identifiers used in a programming language.

It is good practice to keep track of any identifiers used in an identifier table, such as Table 9.1.

Identifier name	Description
`StudentName`	Store a student name
`Counter`	Store a loop counter
`StudentMark`	Store a student mark

Table 9.1

Pseudocode statements to use for writing algorithms.

To input a value:

```
INPUT StudentName
```

To output a message or a value or a combination:

```
OUTPUT "You have made an error"
OUTPUT StudentName
OUTPUT "Student name is ", StudentName
```

To assign a value to a variable (the value can be the result of a process or a calculation):

```
Counter ← 1
Counter ← Counter + 1
MyChar ← "A"
LetterValue ← ASC(MyChar)
StudentMark ← 40
Percentage ← (StudentMark / 80) * 100
Oldstring ← "Your mark is"
NewString ← OldString & " ninety-seven"
```

Operators used in pseudocode assignment statements:

+	Addition
-	Subtraction
*	Multiplication
/	Division
&	String concatenation
←	Assignment

ACTIVITY 9B

Identify the values stored in the variables when the assignment statements in the example above have all been completed. The function ASC returns the ASCII value of a character.

To perform a selection using IF statements for a single choice or a choice and an alternative, and CASE statements when there are multiple choices or multiple choices and an alternative:

IF – single choice

```
IF MyValue > YourValue
  THEN
    OUTPUT "I win"
ENDIF
```

IF – single choice with alternative

```
IF MyValue > YourValue
  THEN
    OUTPUT "I win"
  ELSE
    OUTPUT "You win"
ENDIF
```

CASE – multiple choices

```
CASE OF Direction
  "N": Y ← Y + 1
  "S": Y ← Y - 1
  "E": X ← X + 1
  "W": X ← X - 1
ENDCASE
```

CASE – multiple choices with alternative

```
CASE OF Direction
  "N": Y ← Y + 1
  "S": Y ← Y - 1
  "E": X ← X + 1
  "W": X ← X - 1
  OTHERWISE : OUTPUT "Error"
ENDCASE
```

Relational operators used in pseudocode selection statements:

= Equal to

<> Not equal to

> Greater than

> Less than

>= Greater than or equal to

<= Less than or equal to

Programming languages may not always have the same selection constructs as pseudocode, so it is important to be able to write a program that performs the same task as a solution given in pseudocode.

Here are three programs, one in each of the three prescribed programming languages, to demonstrate the single choice IF statement. Note the construction of the IF statement, as it is different from the pseudocode.

While the Cambridge International AS Level syllabus does not require you to be able to write program code, the ability to do so will increase your understanding, and will be particularly beneficial if you are studying the full Cambridge International A Level course.

Python

```
# IF - single choice Python
myValue = int(input("Please enter my value "))
yourValue = int(input("Please enter your value "))
if myValue > yourValue:
            print ("I win")
```

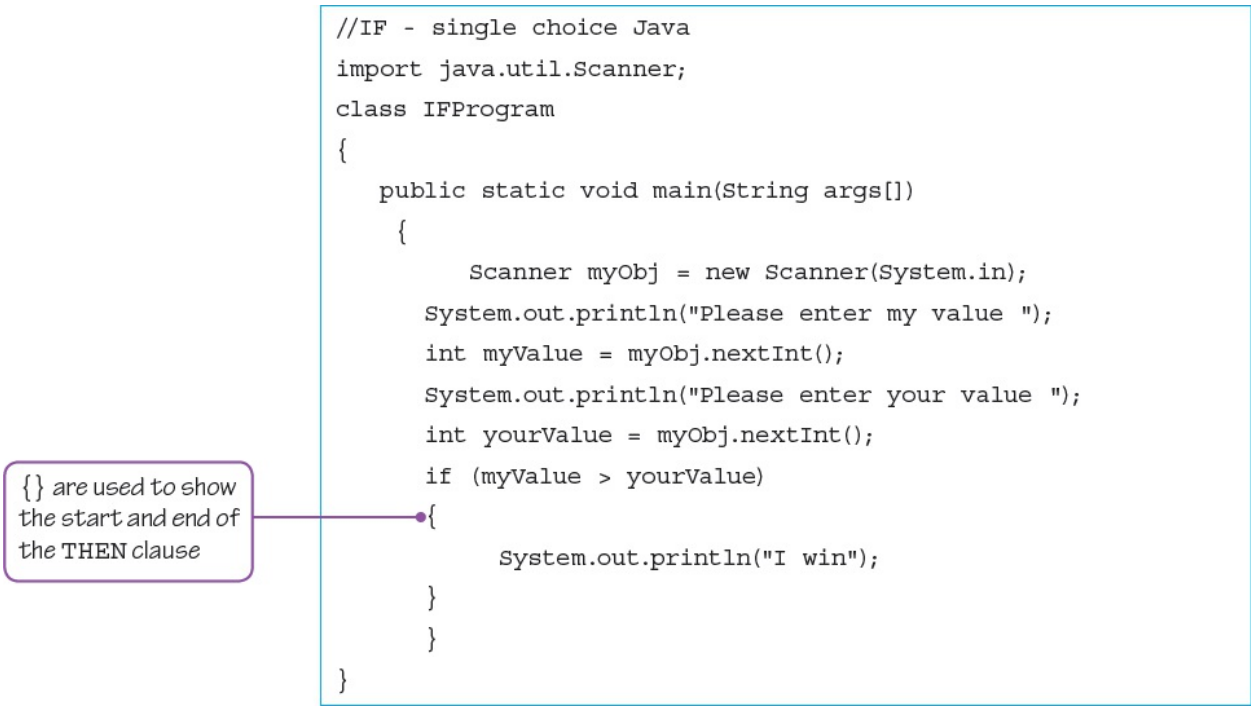
The colon indicates the start of the THEN clause. All statements in the THEN clause are indented as shown

VB

```
'IF - single choice VB
Module Module1
    Sub Main()
        Dim myValue, yourValue As Integer
              Console.Write("Please enter my value ")
        myValue = Integer.Parse(Console.ReadLine())
              Console.Write("Please enter your value ")
        yourValue = Integer.Parse(Console.ReadLine())
              If myValue > yourValue Then
                  Console.WriteLine("I win")
                  Console.ReadKey() 'wait for keypress
              End If
    End Sub
End Module
```

Use of THEN and END IF

Java




```
//IF - single choice Java
import java.util.Scanner;
class IFProgram
{
   public static void main(String args[])
    {
         Scanner myObj = new Scanner(System.in);
      System.out.println("Please enter my value ");
      int myValue = myObj.nextInt();
      System.out.println("Please enter your value ");
      int yourValue = myObj.nextInt();
      if (myValue > yourValue)
      {
            System.out.println("I win");
      }
      }
}
```

ACTIVITY 9C

1. In the programming language you have chosen to use, write a short program to input MyValue and YourValue and complete the single choice with an alternative IF statement shown on page 224. Note any differences in the command words you need to use and the construction of your programming statements compared with the pseudocode.
2. In the programming language you have chosen to use, write a short program to set X and Y to zero, input Direction and complete the multiple choice with an alternative CASE statement shown on page 224 and output X and Y. Note any differences in the command words you need to use and the construction of your programming statements compared to the pseudocode.

To perform iteration using FOR, REPEAT–UNTIL and WHILE loops:

```
Total ← 0
FOR Counter ← 1 TO 10
    OUTPUT "Enter a number "
    INPUT Number
    Total ← Total + Number
NEXT Counter
OUTPUT "The total is ", Total
```

```
FOR Counter ← 1 TO 10 STEP 2
    OUTPUT Counter
NEXT Counter
```

A FOR loop has a fixed number of repeats, the STEP increment is an optional expression that must be a whole number.

```
REPEAT
    OUTPUT "Please enter a positive number "
    INPUT Number
UNTIL Number > 0
```

Statements in a REPEAT loop are always executed at least once.

```
Number ← 0
WHILE Number >= 0 DO
    OUTPUT "Please enter a negative number "
    INPUT Number
ENDWHILE
```

Statements in a WHILE loop may sometimes not be executed.

Programming languages may not always use the same iteration constructs as pseudocode, so it is important to be able to write a program that performs the same task as a solution given in pseudocode.

Here are three programs to demonstrate a simple FOR loop, one in each of the three prescribed programming languages. Note the construction of the FOR statement, as it is different from the pseudocode.

Python

```
# FOR - simple loop Python
for Counter in range (1,10,2):
    print(Counter)
```

The colon indicates the start of the FOR loop. All statements in the FOR loop are indented as shown

VB

```
'FOR - simple loop VB
 Module Module1
    Sub Main()
        Dim Counter As Integer
            For Counter = 1 To 10 Step 2
                Console.WriteLine(Counter)
            Next
        Console.ReadKey() 'wait for keypress
    End Sub
End Module
```

Use of STEP and NEXT

Java

```
//FOR - simple loop Java
class FORProgram
{
  public static void main(String args[])
  {
    for (int Counter = 1; Counter <= 10; Counter = Counter + 2)
    {
        System.out.println(Counter);
    }
  }
}
```

{} are used to show the start and end of the FOR loop

WHILE and REPEAT loops and IF statements make use of comparisons to decide whether statements within a loop are repeated or a statement or group of statements are executed. The comparisons make use of relational operators and the logic operators AND, OR and NOT. The outcome of these comparisons is always either true or false.

ACTIVITY 9D

In the programming language you have chosen to use, write a short program to perform the same tasks as the other three loops shown in pseudocode. Note any differences in the command words you need to use, and the construction of your programming statements compared to the pseudocode.

```
REPEAT
    OUTPUT "Please enter a positive number less than fifty"
    INPUT Number
UNTIL (Number > 0) AND (Number < 50)
```

A simple algorithm can be clearly documented using these statements. A more realistic algorithm to find the average of a number of integers input would include checks that all the values input are whole numbers and that the number input to determine how many integers are input is also positive.

This can be written in pseudocode by making use of the function INT(x) that returns the integer part of x:

ACTIVITY 9E

In pseudocode, write statements to check that a number input is between 10 and 20 or over 100. Make use of brackets to ensure that the order of the comparisons is clear.

```
Total ← 0
REPEAT
    PRINT "Enter the number of values to average"
    INPUT Number
UNTIL (Number > 0) AND (Number = INT(Number))
FOR Counter ← 1 TO Number
    REPEAT
        PRINT "Enter an integer value "
        INPUT Value
    UNTIL Value = INT(Value)
    Total ← Total + Value
NEXT Counter
Average ← Total / Number
PRINT "The average of ", Number, " values is ", Average
```

The identifier table for this algorithm is presented in Table 9.2.

Identifier name	Description
Total	Running total of integer values entered

Number	Number of integer values to enter
Value	Integer value input
Average	Average of all the integer values entered

Table 9.2

Here are three programs to find the average of a number of integers input, one in each of the three prescribed programming languages. Note the construction of the loops, as they are different from the pseudocode. All the programming languages check for an integer value.

Python

```
# Find the average of a number of integers input Python
Total = 0
Number = int(input("Enter the number of values to average "))
while Number <= 0:
        Number = int(input("Enter the number of values to average "))
for Counter in range (1, Number + 1):
        Value = int(input("Enter an integer value "))
        Total = Total + Value
Average = Total / Number
print ("The average of ", Number, " values is ", Average)
```


VB

```
'Find the average of a number of integers input VB
 Module Module1
    Public Sub Main()
        Dim Total, Number, Counter, Value As Integer
            Dim Average As Decimal
            Do
                  Console.Write("Enter the number of values to average ")
                  Number = Integer.Parse(Console.ReadLine())
            Loop Until Number > 0
        For Counter = 1 To Number
                  Console.Write("Enter an integer value ")
                  Value = Integer.Parse(Console.ReadLine())
            Total = Total + Value
        Next
        Average = Total / Number
            Console.WriteLine("The average of " & Number & " values is " & Average)
            Console.ReadKey()
     End Sub
End Module
```

Use of DO and LOOP UNTIL

Java

```
//Find the average of a number of integers input Java
import java.util.Scanner;
class AverageAlg
{
   public static void main(String args[])
    {
         Scanner myObj = new Scanner(System.in);
         int Number;
         int Total = 0;
      do
        {
            System.out.println("Enter the number of values to average ");
            Number = myObj.nextInt();
        }
        while (Number < 0);
        for (int Counter = 1; Counter <= Number; Counter ++)
        {
            System.out.println("Enter an integer value ");
            int Value = myObj.nextInt();
            Total = Total + Value;
        }
      float Average = (float)Total / Number;
      System.out.println("The average of " + Number + " values is " + Average);
      }
}
```

Use of DO WHILE loop

Java automatically performs integer division when two integers are used

ACTIVITY 9F

In pseudocode, write an algorithm to set a password for a user when they have to input the same word twice. Then allow the user three attempts to enter the correct password. Complete an identifier table for your algorithm.

Finally, check your pseudocode algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

9.2.3 Writing pseudocode from a structured English description

There are no set rules for writing structured English – the wording just needs to be unambiguous and easily understandable. Pseudocode is more precise and usually follows an agreed set of rules.

From a structured English description, the following things need to be possible:

- Any variables that need to be used can be identified and put in an identifier table – these can be items input or output as the results of calculations.
- Input and output can be identified from the wording used, for example, Enter, Read, Print, Write.
- Selection can be identified from the wording used, for example, If, Then, Choose.
- Any iteration required can be identified from the wording used, for example, Loop, Repeat.
- Any processes needed can be identified from the wording used, for example, Set, Calculate.

When the identifier table is complete, each structured English statement can be used to write one or more pseudocode statements, keeping the same order as the structured English.

Here is an example of an algorithm to calculate a runner's marathon time in seconds, using structured English.

```
1  Enter time taken to run marathon in hours, minutes and seconds
2  Calculate and store marathon time in seconds
3  Output marathon time in seconds
```

This can be used to identify the variables required and complete the identifier table (Table 9.3).

Identifier name	Description
MarathonHours	The hours part of the marathon time
MarathonMinutes	The minutes part of the marathon time
MarathonSeconds	The seconds part of the marathon time
TotalMarathonTimeSeconds	Total marathon time in seconds

Table 9.3

Using these identifiers, each step of the structured English algorithm can be converted to pseudocode, as demonstrated below.

```
1  Enter time taken to run marathon in hours, minutes and seconds
```

There are three variables used: MarathonHours, MarathonMinutes and MarathonSeconds. This is explicitly input and implicitly output as the user needs to understand what input is required. The pseudocode required is as follows.

```
OUTPUT "Enter the time you took to run the marathon"
OUTPUT "Enter hours"
INPUT MarathonHours
OUTPUT "Enter minutes"
INPUT MarathonMinutes
OUTPUT "Enter seconds"
INPUT MarathonSeconds
```

2 Calculate and store marathon time in seconds

This is a process using the variables MarathonHours, MarathonMinutes and MarathonSeconds and using an assignment statement to store the result in TotalMarathonTimeSeconds. The pseudocode required is as follows.

```
TotalMarathonTimeSeconds ← (MarathonHours * 3600
  + MarathonMinutes) * 60 + MarathonSeconds
```

3 Output marathon time in seconds

This is output using the variable TotalMarathonTimeSeconds. The pseudocode required is as follows.

```
OUTPUT "Time for marathon in seconds ",
TotalMarathonTimeSeconds
```

ACTIVITY 9G

The structured English description has been extended below to check the runner's time against their personal best.

1 Enter time taken to run marathon in hours, minutes and seconds
2 Calculate and store marathon time in seconds
3 Output marathon time in seconds
4 Enter personal best time in seconds
5 If marathon time in seconds is shorter than the personal best time then
6 Reset personal best time in seconds
7 Output the personal best time

Extend the identifier table and write the extra pseudocode to complete the algorithm. Then

check your algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

9.2.4 Writing pseudocode from a flowchart

Flowcharts are diagrams showing the structure of an algorithm using an agreed set of symbols, as shown in Table 9.4.

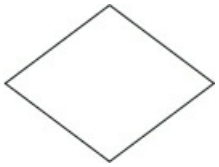
Pseudocode	Flowchart symbol
INPUT or OUTPUT	
IF or CASE Part of FOR, REPEAT and WHILE	
FOR, REPEAT and WHILE	 Returning flow line
Assignment ← using a calculation or a pre-defined process, for example, INT	

Table 9.4

Flowcharts can be used to identify any variables required and you can then complete an identifier table. Each flowchart symbol can be used to identify and write one or more pseudocode statements.

Here is an example of a flowchart of an algorithm that can be used to check an exam grade:

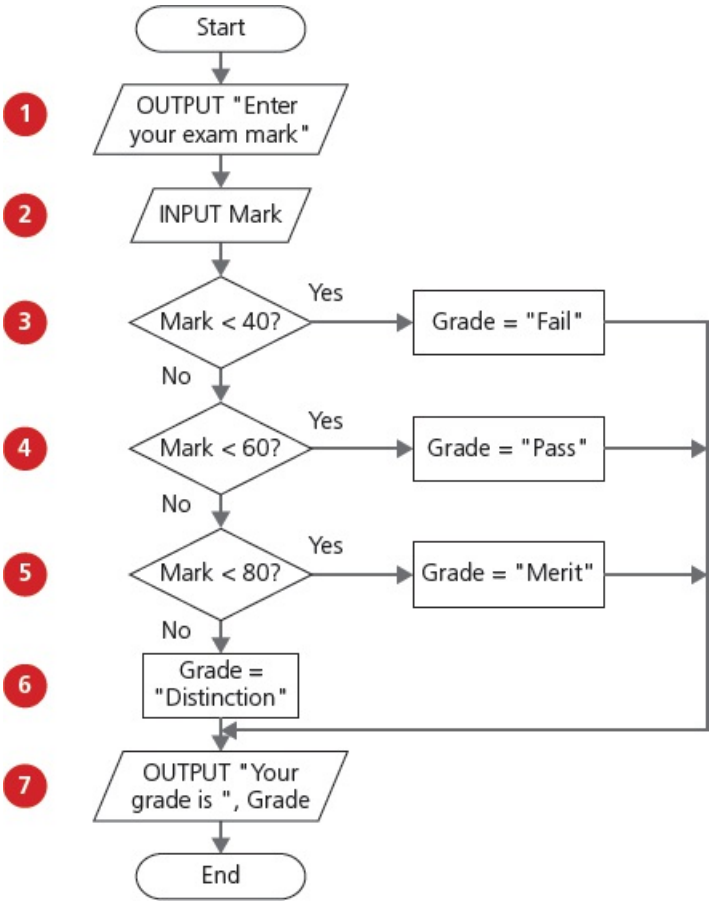


Figure 9.4

The same algorithm is presented in pseudocode on the left. Below is the identifier table:

Identifier name	Description
Mark	Exam mark
Grade	Exam grade

Table 9.5

3 4 5 and 6 form a nested selection (IF) structure, as each following statement is part of the ELSE clause. It is only at 7 that the selection is complete. The flowchart shows this clearly and the pseudocode uses indentation to show the nesting.

```
1    OUTPUT "Enter your exam mark"
2    INPUT Mark
3    IF Mark < 40
        THEN
          Grade ← "Fail"
        ELSE
4        IF Mark < 60
            THEN
              Grade ← "Pass"
            ELSE
5            IF Mark < 80
                THEN
                  Grade ← "Merit"
                ELSE
6                 Grade ← "Distinction"
              ENDIF
          ENDIF
      ENDIF
7    OUTPUT "Your grade is ", Grade
```

9.2.5 Stepwise refinement

The algorithms looked at so far have been short and simple. When an algorithm is written to solve a more complex problem, decomposition is used to break the problem down into smaller and more manageable parts. These parts then need to be written as a series of steps where each step can be written as a statement in a high-level programming language, this process is called **stepwise refinement**.

Many problems are more complex than they seem if a robust solution is to be developed. Look at the first step of the structured English to calculate a time in seconds.

1 **Enter time taken to run marathon in hours, minutes and seconds**
2 Calculate and store marathon time in seconds
3 Output marathon time in seconds

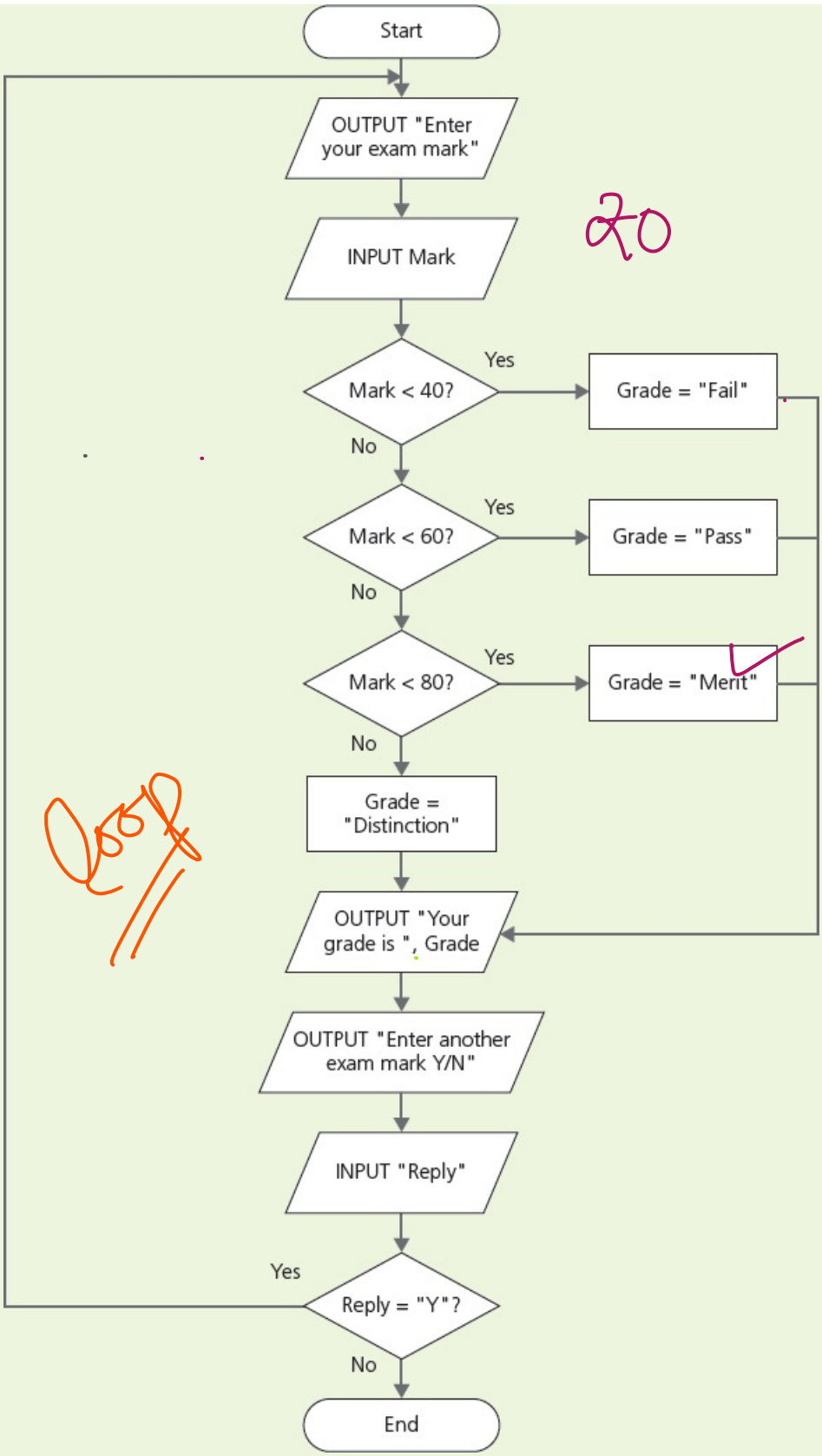
The first step can be further broken down, as follows:

1.1 Enter the hours
1.2 Enter the minutes
1.3 Enter the seconds

ACTIVITY 9H

The flowchart on page **232** has been extended to allow more than one mark to be input.

Test data: 30, 95, 75, 20



20

marks
30
95
75
20

Trace Table

Mark	Grade
30	Fail
95	Distinction
75	Merit
20	Fail

Extend the identifier table and write the extra pseudocode to complete the algorithm. Then

check your algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

Each of these steps can be broken down further:

1.1.1 Input value for hours

1.1.2 Check input in the range 2 to 8

1.1.3 Reject if out of range or not a whole number and re-input value step 1.1.1

1.1.4 Accept and store value in hours

1.2.1 Input value for minutes

1.2.2 Check input in the range 0 to 59

1.2.3 Reject if out of range or not a whole number and re-input value step 1.2.1

1.2.4 Accept and store value in minutes

1.3.1 Input value for seconds

1.3.2 Check input in the range 0 to 59

1.3.3 Reject if out of range or not a whole number and re-input value step 1.3.1

1.3.4 Accept and store value in seconds

These steps can now be written in pseudocode. For example, the input routine for the seconds:

```
REPEAT
    OUTPUT "Enter seconds"
    INPUT Value
UNTIL (Value >= 0) AND (Value <= 59) AND (Value = INT(Value))
MarathonSeconds ← Value
```

ACTIVITY 9I

Look at the algorithm to calculate the area of a chosen shape written in structured English below. Use stepwise refinement to break each step into more manageable parts then rewrite the algorithm using pseudocode.

1 Choose the shape (square, triangle, circle)
2 Enter the length(s)
3 Calculate the area
4 Output the area

Then check your pseudocode algorithm works by writing a short program from your pseudocode statements using the same names for your identifiers.

End of chapter questions

1 Algorithms can be shown as structured English, flowcharts and pseudocode.
Explain what is meant by

a) structured English

[2]

b) a flowchart

[2]

c) pseudocode.

[2]

2 Several techniques are used in computational thinking.
Explain what is meant by

a) abstraction

[2]

b) decomposition

[2]

c) pattern recognition.

[2]

3 Describe, using an example, the process of stepwise refinement.

[2]

4 Computer programs have to evaluate expressions.

– Study the sequence of pseudocode statements.
– Write down the value assigned to each variable.

```
DECLARE h, w, r, Perimeter, Area : REAL
DECLARE A, B, C, D, E            : BOOLEAN
h ← 13.6 w ← 6.4
Perimeter ← (h + w) * 2
r ← 10
Area 3.142 * r^2
Z ← 11 + r / 5 + 3
A ← NOT(r > 10)
```

a) Perimeter

[1]

b) Area

[1]

c) Z

[1]

d) A

[1]

5 Study the pseudocode and answer the following questions. Line numbers have been added to help you.

```
01  REPEAT
02  OUTPUT "Menu Temperature Conversion"
03  OUTPUT "Celsius to Fahrenheit          1"
04  OUTPUT "Fahrenheit to Celsius          2"
05  OUTPUT "Exit                           3"
06  OUTPUT "Enter choice"
07  IF Choice = 1 OR Choice = 2
08    THEN
09      OUTPUT "Enter temperature"
10      INPUT Temperature
11      IF Choice = 1
12        THEN
13          ConvertedTemperature ← 1.8*Temperature + 32
14        ELSE
15          ConvertedTemperature ← (Temperature - 32) * 5 / 9
16      ENDIF
17      OUTPUT "Converted temperature is ", ConvertedTemperature
18    ELSE
19      IF Choice <> 3
20        THEN
21          OUTPUT "Error in choice"
22      ENDIF
23  ENDIF
24  UNTIL Choice = 3
```

a) Give the line number of:

i) an assignment statement

[1]

ii) a selection

[1]

iii) an iteration.

[1]

b) Complete an identifier table for the algorithm.

[3]

c) Extend the algorithm to only allow four tries for a correct choice.

[3]

10 Data types and structures

In this chapter, you will learn about
- basic data types, and how to select and use them
- two different data structures: records and arrays
- how to handle text files consisting of many lines, using pseudocode
- three different abstract data types (ADTs): stacks, queues and linked lists.

WHAT YOU SHOULD ALREADY KNOW

Try this activity to see if you can use one-dimensional arrays before you read the first part of this chapter.

Write an algorithm, using pseudocode, to find the largest and smallest of five numbers. The numbers are to be input with appropriate prompts, stored in an array, and the largest and smallest are to be output with appropriate messages. If you haven't used an array before store the values in five separate variables.

Key terms

Data type – a classification attributed to an item of data, which determines the types of value it can take and how it can be used.

Identifier – a unique name applied to an item of data.

Record (data type) – a composite data type comprising several related items that may be of different data types.

Composite data type – a data type constructed using several of the basic data types available in a particular programming language.

Array – a data structure containing several elements of the same data type.

Index (array) – a numerical indicator of an item of data's position in an array.

Lower bound – the index of the first element in an array, usually 0 or 1.

Upper bound – the index of the last element in an array.

Linear search – a method of searching in which each element of an array is checked in order.

Bubble sort – a method of sorting data in an array into alphabetical or numerical order by comparing adjacent items and swapping them if they are in the wrong order.

File – a collection of data stored by a computer program to be used again.

Abstract data type (ADT) – a collection of data and a set of operations on that data.

Stack – a list containing several items operating on the last in, first out (LIFO) principle.

Queue – a list containing several items operating on the first in, first out (FIFO) principle.

Linked list – a list containing several items in which each item in the list points to the next item in the list.

10.1 Data types and records

Any computer system that is reusable needs to store data in a structured way so that it can be reused in the future. One of the most powerful tools in computer science is the ability to search large amounts of data and obtain results very quickly. This chapter introduces data structures that enable effective and efficient computer-based storage and searching to take place.

10.1.1 Data types

Data types allow programming languages to provide different classifications for items of data, so they can be used for different purposes. For example, integers are discrete whole numbers used for counting and indexing, whereas real numbers can be used to provide accurate measurements.

You need to be able to understand and make appropriate use of data types when writing pseudocode or programs to provide a solution to a problem.

Programming languages have a number of built in data types. Table 10.1 lists the basic data types that you need to know and how they are referred to in pseudocode and different programming languages.

Data type	Description	Pseudocode	Python	Java	VB.NET
Boolean	Logical values, True (1) and False (2)	BOOLEAN	bool	boolean	Boolean
char	Single alphanumerical character	CHAR	*Not used*	char	Char
date	Value to represent a date	DATE	class datetime	class Date	Date
integer	Whole number, positive or negative	INTEGER	int	byte short int long	Integer
real	Positive or negative number with a decimal point	REAL	float	float double	single
string	Sequence of alphanumerical characters	STRING	str	class String	String

Table 10.1 Basic data types

ACTIVITY 10A

Decide which data type would be the best one to use for each item.

a) Your name
b) The number of children in a class
c) The time taken to run a race
d) Whether a door is open or closed
e) My birthday

In pseudocode and some programming languages, before data can be used, the type needs to be decided. This is done by declaring the data type for each item to be used. Each data item is identified by a unique name, called an **identifier**.

In pseudocode a declaration statement takes this form:

```
DECLARE <identifier> : <data type>
```

For example:

```
DECLARE myBirthday : DATE
```

ACTIVITY 10B

Write declaration statements in pseudocode for each item.

a) Your name

b) The number of children in a class

c) The time taken to run a race

d) Whether a door is open or closed

Write these declaration statements in your chosen programming language. If this is Python, you may need to write assignment statements.

10.1.2 Records

Records are **composite data types** formed by the inclusion of several related items that may be of different data types. This allows a programmer to refer to these items using the same identifier, enabling a structured approach to using related items. A record will contain a fixed number of items. For example, a record for a book could include title, author, publisher, number of pages, and whether it is fiction or non-fiction.

A record data type is one example of a composite user-defined data type. A composite data type references other existing data types when it is defined. A composite data type must be defined before it can be used. Any data type not provided by a programming language must be defined before it can be used.

In pseudocode, a record data type definition takes the following form:

```
TYPE
<Typename>
    DECLARE <identifier> : <data type>
    DECLARE <identifier> : <data type>
    DECLARE <identifier> : <data type>
    ::
    ::
ENDTYPE
```

For example, the book record data type could be defined like this:

```
TYPE
TbookRecord
    DECLARE title : STRING
    DECLARE author : STRING
    DECLARE publisher : STRING
    DECLARE noPages : INTEGER
    DECLARE fiction : BOOLEAN
ENDTYPE
```

The data type, TbookRecord, is now available for use and an identifier may now be declared in the usual way:

```
DECLARE Book : TbookRecord
```

Items from the record are now available for use and are identified by:

```
<identifier>.<item identifier>
```

For example:

```
Book.author ← "David Watson"
Book.fiction ← FALSE
```

ACTIVITY 10C

1 Write definition statements in pseudocode for a student record type containing these items.
   a) Name
   b) Date of birth
   c) Class
   d) Gender
2 Use this record type definition to declare a record myStudent and set up and output a record for a male student Ahmad Sayed, in Class 5A, who was born on 21 March 2010.
3 In your chosen programming language, write a short program to complete this task.

10.2 Arrays

An **array** is a data structure containing several elements of the same data type; these elements can be accessed using the same identifier name. The position of each element in an array is identified using the array's **index**. The index of the first element in an array is the **lower bound** and the index of the last element is the **upper bound**.

The lower bound of an array is usually set as zero or one. Some programming languages can automatically set the lower bound of an array.

Arrays can be one-dimensional or multi-dimensional. In this chapter, we will look at one-dimensional (1D) and two-dimensional (2D) arrays.

10.2.1 1D arrays

A 1D array can be referred to as a list. Here is an example of a list with nine elements and a lower bound of zero.

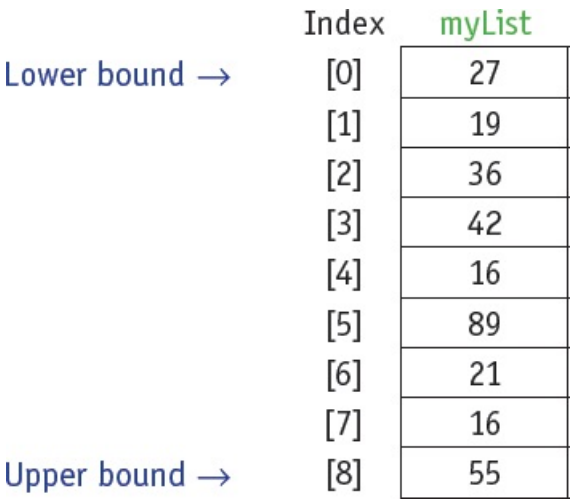


Figure 10.1 Example of a 1D array

When a 1D array is declared in pseudocode, the lower bound (LB), upper bound (UB) and data type are included:

```
DECLARE <identifier> : ARRAY[LB:UB] OF <data type>
```

For example:

```
DECLARE myList : ARRAY[0:8] OF INTEGER
```

The declared array can then be used, as follows:

```
myList[7] ← 16
```

ACTIVITY 10D

1 Write statements in pseudocode to populate the array myList, as shown in Figure 10.1, using a FOR … NEXT loop.

2 In your chosen programming language, write a short program to complete this task, then output the contents of the array. Before writing your program find out how your programming language sets up array bounds.

10.2.2 2D arrays

A 2D array can be referred to as a table, with rows and columns. Here is an example of a table with nine rows and three columns (27 elements) and lower bounds of zero.

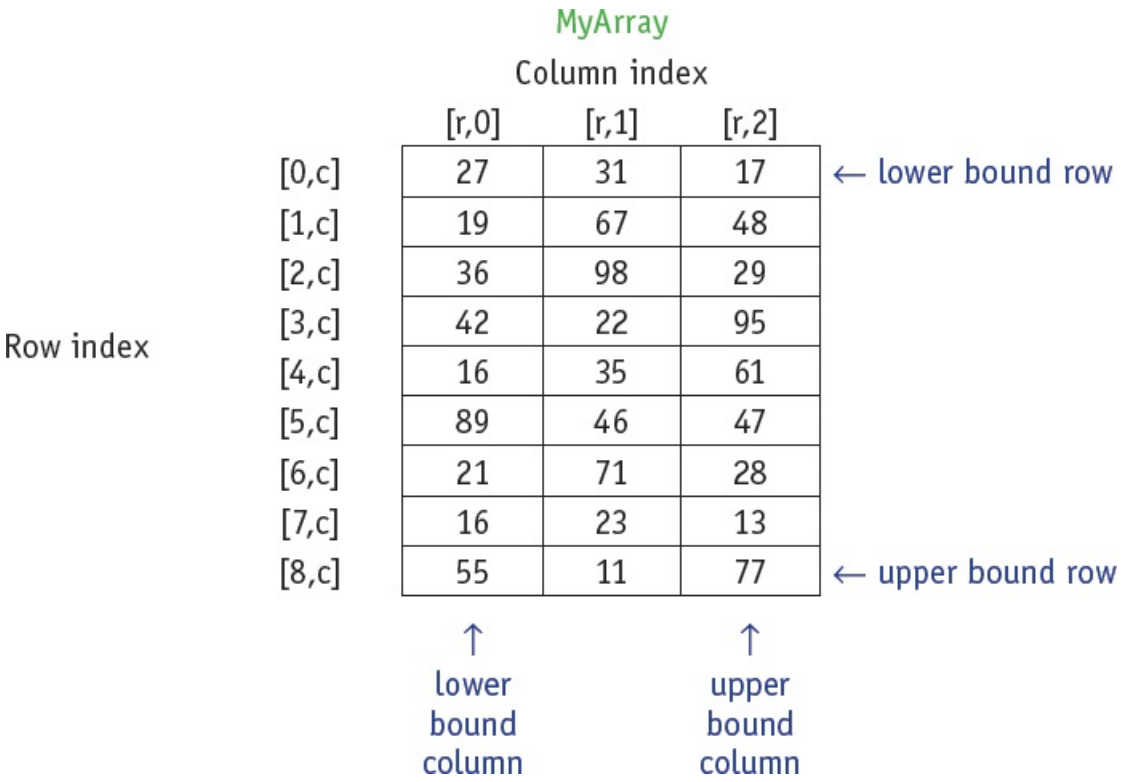


Figure 10.2 Example of a 2D array

When a 2D array is declared in pseudocode, the lower bound for rows (LBR) and the upper bound for rows (UBR), the lower bound for columns (LBC) and the upper bound for columns (UBC), and data type are included:

```
DECLARE <identifier> : ARRAY[LBR:UBR, LBC:UBC] OF <data type>
```

For example:

```
DECLARE myArray : ARRAY[0:8,0:2] OF INTEGER
```

The declared array can then be used, as follows:

```
myArray[7,0] ← 16
```

ACTIVITY 10E

1 Write statements in pseudocode to populate the array myArray, as shown in Figure 10.2,

using a nested FOR … NEXT loop.

2 In your chosen programming language, write a short program to complete this task, then output the contents of the array.

Arrays are used to store multiple data items in a uniformly accessible manner. All the data items use the same identifier and each data item can be accessed separately by the use of an index. In this way, lists of items can be stored, searched and put into an order. For example, a list of names can be ordered alphabetically, or a list of temperatures can be searched to find a particular value.

EXTENSION ACTIVITY 10A

Write a program to populate a three-dimensional (3D) array.

ACTIVITY 10F

In small groups of three or four, identify at least three uses for a 1D array and three uses for a 2D array. Compare array structures with record structures, decide if any of your uses would be better structured as records.

10.2.3 Using a linear search

To find an item stored in an array, the array needs to be searched. One method of searching is a **linear search**. Each element of the array is checked in order, from the lower bound to the upper bound, until the item is found or the upper bound is reached.

For example, the search algorithm to find if an item is in the populated 1D array myList could be written in pseudocode as:

```
DECLARE myList : ARRAY[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE item : INTEGER
DECLARE found : BOOLEAN
upperBound ← 8
lowerBound ← 0
OUTPUT "Please enter item to be found"
INPUT item
found ← FALSE
index ← lowerBound
REPEAT
    IF item = myList[index]
      THEN
        found ← TRUE
    ENDIF
    index ← index + 1
UNTIL (found = TRUE) OR (index > upperBound)
IF found
  THEN
    OUTPUT "Item found"
  ELSE
    OUTPUT "Item not found"
ENDIF
```

This algorithm uses the variables upperBound and lowerBound so that the algorithm is easier to adapt for different lengths of list. The REPEAT … UNTIL loop makes use of two conditions, so that the algorithm is more efficient, terminating as soon as the item is found in the list.

As stated in Chapter 9, it is good practice to provide an identifier table to keep track of and explain the use of each identifier in an algorithm. This allows the programmer to keep track of the identifiers used and provides a useful summary of identifiers and their uses if the algorithm requires modification at a later date. Table 10.2 is the identifier table for the linear search

algorithm.

Identifier	Description
item	The integer to be found
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
found	Flag to show when item has been found

Table 10.2

ACTIVITY 10G

Extend the pseudocode algorithm to output the value of the index if the item is found. In your chosen programming language write a short program to complete this task. You will need to populate the array myList before searching for an item. Use the sample data shown in myList in Figure 10.1 and search for the values 89 and 77.

EXTENSION ACTIVITY 10B

Extend your program created in Activity 10G to find any repeated items in a list and print out how many items were found.

10.2.4 Using a bubble sort

Lists can be more useful if the items are sorted in a meaningful order. For example, names could be sorted in alphabetical order, or temperatures could be sorted in ascending or descending order. There are several sorting algorithms available. One method of sorting is a **bubble sort**. Each element of the array is compared with the next element and swapped if the elements are in the wrong order, starting from the lower bound and finishing with the element next to the upper bound. The element at the upper bound is now in the correct position. This comparison is repeated with one less element in the list, until there is only one element left or no swaps are made.

For example, the bubble sort algorithm to sort the populated 1D array myList could be written in pseudocode as:

```
DECLARE myList : ARRAY[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE swap : BOOLEAN
DECLARE temp : INTEGER
DECLARE top : INTEGER
upperBound ← 8
lowerBound ← 0
top ← upperBound
REPEAT
    FOR index = lowerBound TO top - 1
        Swap ← FALSE
        IF myList[index] > myList[index + 1]
          THEN
            temp ← myList[index]
            myList[index] ← myList[index + 1]
            myList[index + 1] ← temp
            swap ← TRUE
        ENDIF
    NEXT
    top ← top -1
UNTIL (NOT swap) OR (top = 0)
```

Table 10.3 is the identifier table for the bubble sort algorithm.

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element

swap	Flag to show when swaps have been made
top	Index of last element to compare
temp	Temporary storage location during swap

Table 10.3

The following eight tables show the changes to the 1D array myList as the bubble sort is completed. After each iteration of the FOR … NEXT loop, the highest value in the list is correctly placed and the lower values of the list are swapped and move towards the start of the list, until no more swaps are made.

First pass of bubble sort

All nine elements compared and five swaps:

	index	myList								
	[0]	27	19	19	19	19	19	19	19	19
	[1]	19	27	27	27	27	27	27	27	27
	[2]	36	36	36	36	36	36	36	36	36
	[3]	42	42	42	42	16	16	16	16	16
	[4]	16	16	16	16	42	42	42	42	42
	[5]	89	89	89	89	89	89	21	21	21
	[6]	21	21	21	21	21	21	89	16	16
	[7]	16	16	16	16	16	16	16	89	55
top →	[8]	55	55	55	55	55	55	55	55	89

Figure 10.3

Second pass of bubble sort

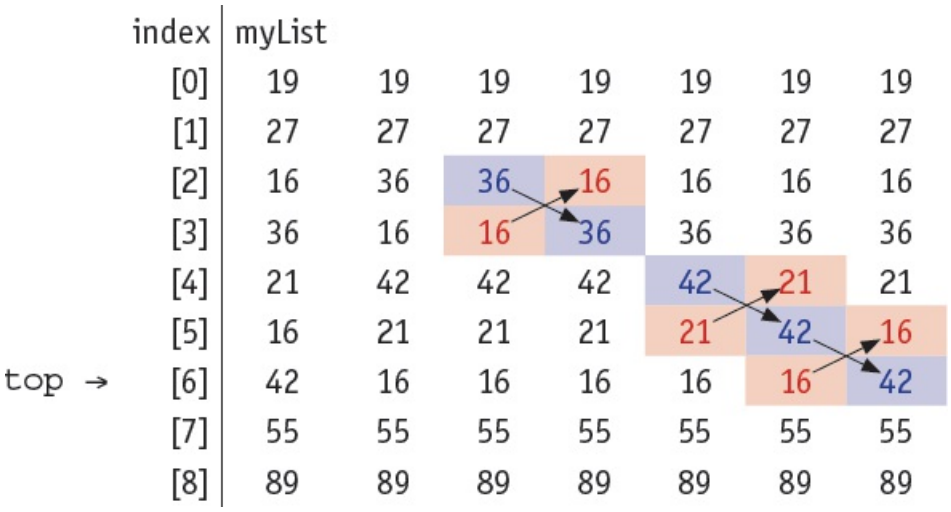
Eight elements compared and three swaps:

	index	myList							
	[0]	19	19	19	19	19	19	19	19
	[1]	27	27	27	27	27	27	27	27
	[2]	36	36	36	16	16	16	16	16
	[3]	16	16	16	36	36	36	36	36
	[4]	42	42	42	42	42	21	21	21
	[5]	21	21	21	21	21	42	16	16
	[6]	16	16	16	16	16	16	42	42
top →	[7]	55	55	55	55	55	55	55	55
	[8]	89	89	89	89	89	89	89	89

Figure 10.4

Third pass of bubble sort

Seven elements compared and three swaps:

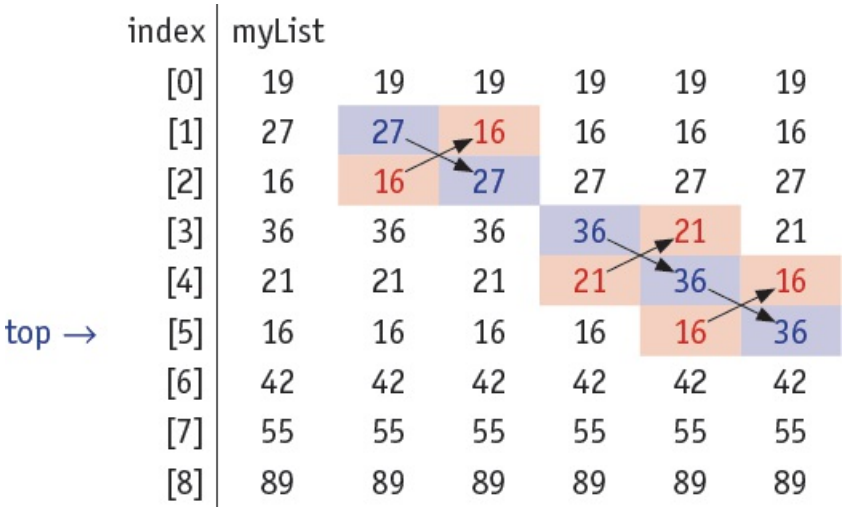


	index	myList						
	[0]	19	19	19	19	19	19	19
	[1]	27	27	27	27	27	27	27
	[2]	16	36	36	16	16	16	16
	[3]	36	16	16	36	36	36	36
	[4]	21	42	42	42	42	21	21
	[5]	16	21	21	21	21	42	16
top →	[6]	42	16	16	16	16	16	42
	[7]	55	55	55	55	55	55	55
	[8]	89	89	89	89	89	89	89

Figure 10.5

Fourth pass of bubble sort

Six elements compared and three swaps:

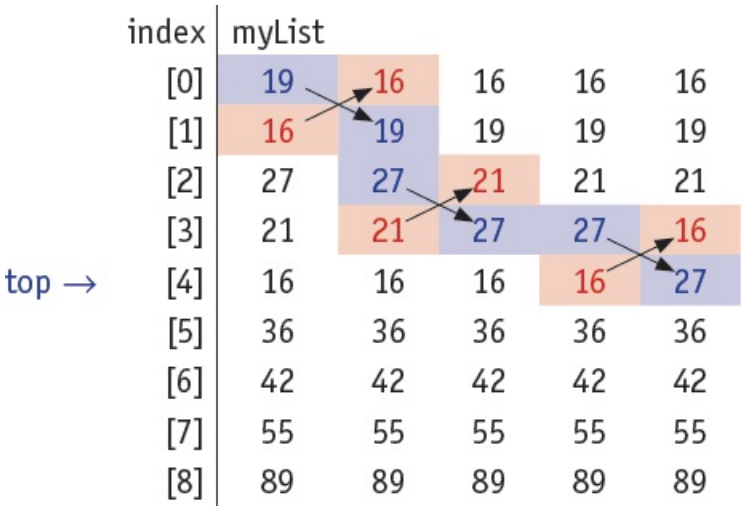


	index	myList					
	[0]	19	19	19	19	19	19
	[1]	27	27	16	16	16	16
	[2]	16	16	27	27	27	27
	[3]	36	36	36	36	21	21
	[4]	21	21	21	21	36	16
top →	[5]	16	16	16	16	16	36
	[6]	42	42	42	42	42	42
	[7]	55	55	55	55	55	55
	[8]	89	89	89	89	89	89

Figure 10.6

Fifth pass of bubble sort

Five elements compared and three swaps:

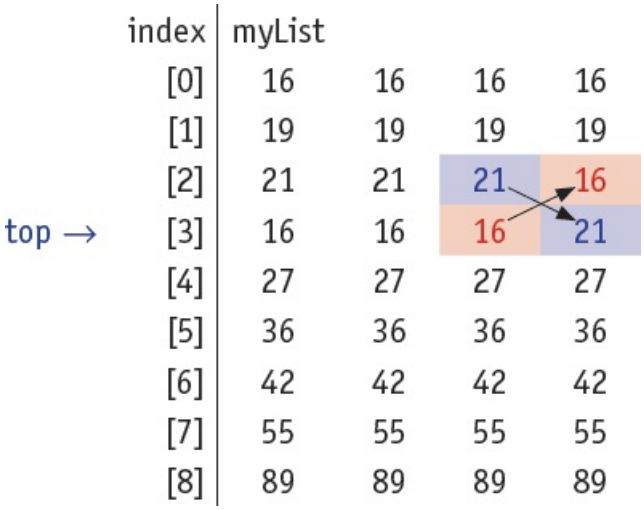


	index	myList				
	[0]	19	16	16	16	16
	[1]	16	19	19	19	19
	[2]	27	27	21	21	21
	[3]	21	21	27	27	16
top →	[4]	16	16	16	16	27
	[5]	36	36	36	36	36
	[6]	42	42	42	42	42
	[7]	55	55	55	55	55
	[8]	89	89	89	89	89

Figure 10.7

Sixth pass of bubble sort

Four elements compared and one swap:



	index	myList			
	[0]	16	16	16	16
	[1]	19	19	19	19
	[2]	21	21	21	16
top →	[3]	16	16	16	21
	[4]	27	27	27	27
	[5]	36	36	36	36
	[6]	42	42	42	42
	[7]	55	55	55	55
	[8]	89	89	89	89

Figure 10.8

Seventh pass of bubble sort

Three elements compared and one swap:

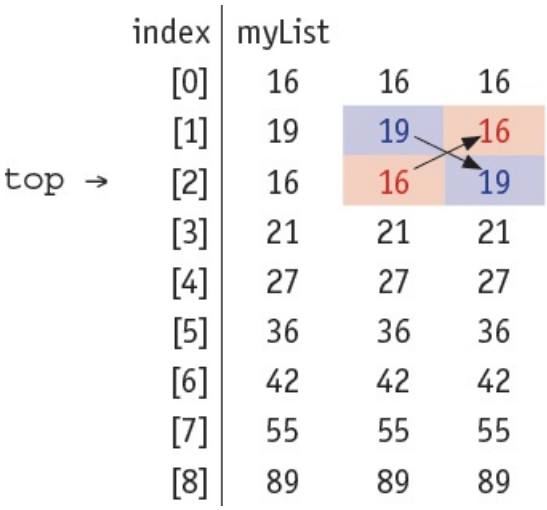


Figure 10.9

Eighth pass of bubble sort

Two elements compared and no swaps:

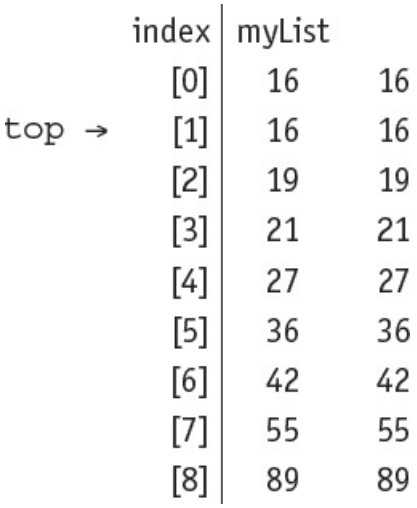


Figure 10.10

ACTIVITY 10H

In your chosen programming language, write a short program to complete a bubble sort on the array myList. Use the sample data shown in myList in Figure 10.1 to populate the array before sorting. Output the sorted list once the bubble sort is completed.

10.3 Files

Computer programs store data that will be required again in a **file**. Every file is identified by its filename. In this chapter, we are going to look at how to use text files. Text files contain a sequence of characters. Text files can include an end of line character that enables the file to be read from and written to as lines of characters.

In pseudocode, text files are handled using the following statements.

To open a file before reading from it or writing to it:

```
OPEN <file identifier> FOR <file mode>
```

Files can be opened in one of the following modes:

READ	reads data from the file
WRITE	writes data to the file, any existing data stored in the file will be overwritten
APPEND	adds data to the end of the file

Once the file is opened in READ mode, it can be read from a line at a time:

```
READFILE <file identifier>, <variable>
```

Once the file is opened in WRITE or APPEND mode, it can be written to a line at a time:

```
WRITEFILE <file identifier>, <variable>
```

In both cases, the variable must be of data type STRING.

The function EOF is used to test for the end of a file. It returns a value TRUE if the end of a file has been reached and FALSE otherwise.

```
EOF(<file identifier>)
```

When a file is no longer being used it should be closed:

```
CLOSEFILE <file identifier>
```

This pseudocode shows how the file myText.txt could be written to and read from:

```
DECLARE textLn : STRING
DECLARE myFile : STRING
myFile ← "myText.txt"
OPEN myFile FOR WRITE
REPEAT
    OUTPUT "Please enter a line of text"
    INPUT textLn
    IF textLn <> ""
      THEN
        WRITEFILE, textLn
      ELSE
        CLOSEFILE(myFile)
    ENDIF
UNTIL textLn = ""
OUTPUT "The file contains these lines of text:"
OPEN myFile FOR READ
REPEAT
    READFILE, textLn
    OUTPUT textLn
UNTIL EOF(myFile)
CLOSEFILE(myFile)
```

ACTIVITY 10I

Extend this pseudocode to append further lines to the end of myFile. In your chosen programming language write a short program to complete this file handling routine.

Identifier name	Description
textLn	Line of text
myFile	File name

Table 10.4

10.4 Abstract data types (ADTs)

An **abstract data type (ADT)** is a collection of data and a set of operations on that data. For example, a stack includes the items held on the stack and the operations to add an item to the stack (push) or remove an item from the stack (pop). In this chapter we are going to look at three ADTs: stack, queue and linked list.

Table 10.5 lists some of the uses of stacks, queues and linked lists mentioned in this book.

Stacks	Queues	Linked lists
Memory management (see Section 16.1)	Management of files sent to a printer (see Section 5.1)	Using arrays to implement a stack (see Section 19.1)
Expression evaluation (see Section 16.3)	Buffers used with keyboards (see Section 5.1)	Using arrays to implement a queue (see Section 19.1)
Backtracking in recursion (see Section 19.2)	Scheduling (see Section 16.1)	Using arrays to implement a binary tree (see Section 19.1)

Table 10.5 Uses of stacks, queues and linked lists

- **Stack** – a list containing several items operating on the last in, first out (LIFO) principle. Items can be added to the stack (push) and removed from the stack (pop). The first item added to a stack is the last item to be removed from the stack.
- **Queue** – a list containing several items operating on the first in, first out (FIFO) principle. Items can be added to the queue (enqueue) and removed from the queue (dequeue). The first item added to a queue is the first item to be removed from the queue.
- **Linked list** – a list containing several items in which each item in the list points to the next item in the list. In a linked list a new item is always added to the start of the list.

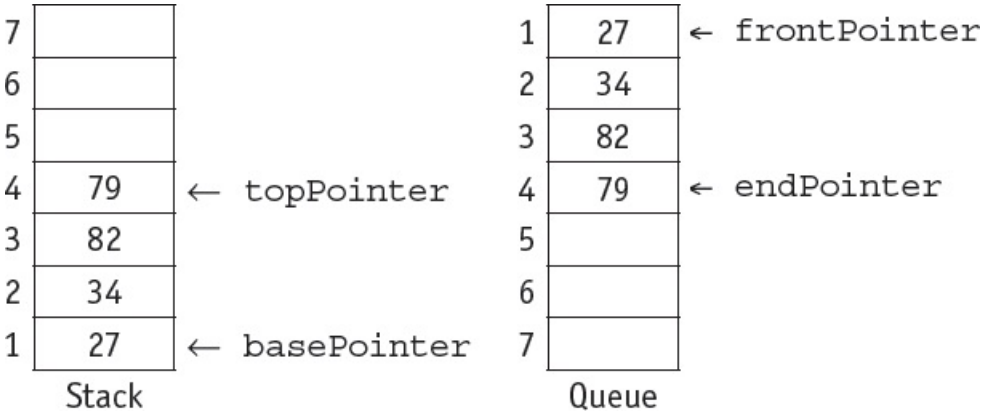


Figure 10.11 Stack and queue

startPointer

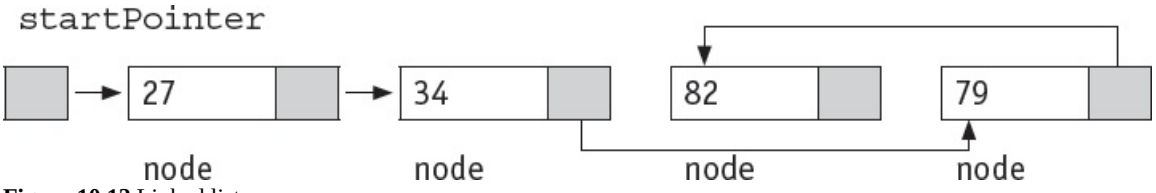


Figure 10.12 Linked list

Stacks, queues and linked lists all make use of pointers to manage their operations. Items stored in stacks and queues are always added at the end. Linked lists make use of an ordering algorithm for the items, often ascending or descending.

A stack uses two pointers: a base pointer points to the first item in the stack and a top pointer points to the last item in the stack. When they are equal there is only one item in the stack.

A queue uses two pointers: a front pointer points to the first item in the queue and a rear pointer points to the last item in the queue. When they are equal there is only one item in the queue.

A linked list uses a start pointer that points to the first item in the linked list. Every item in a linked list is stored together with a pointer to the next item. This is called a node. The last item in a linked list has a null pointer.

10.4.1 Stack operations

The value of the basePointer always remains the same during stack operations:

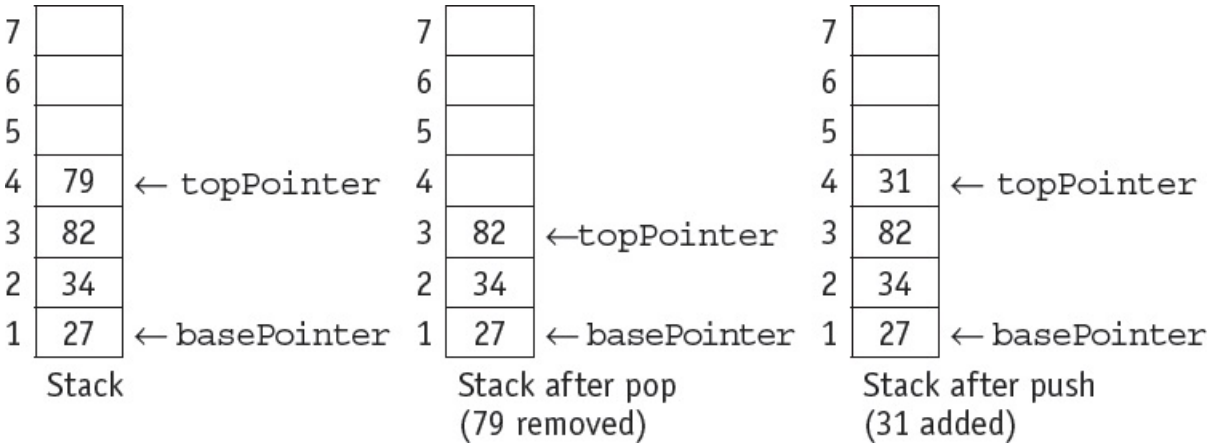


Figure 10.13

A stack can be implemented using an array and a set of pointers. As an array has a finite size, the stack may become full and this condition must be allowed for.

In pseudocode, stack operations are handled using the following statements. Please note that you are not expected to be able to write the pseudocode statements included in this section at Cambridge AS Level. However, you may find them useful to refer back to if you are studying the full Cambridge A Level syllabus.

To set up a stack

```
DECLARE stack ARRAY[1:10] OF INTEGER
DECLARE topPointer : INTEGER
DECLARE basePointer : INTEGER
DECLARE stackful : INTEGER
basePointer ← 1
topPointer ← 0
stackful ← 10
```

To push an item, stored in item, *onto a stack*

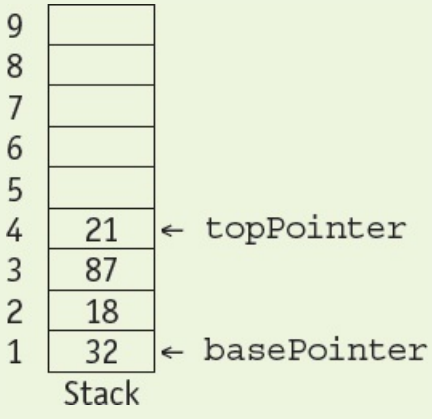
```
IF topPointer < stackful
  THEN
    topPointer ← topPointer + 1
    stack[topPointer] ← item
  ELSE
    OUTPUT "Stack is full, cannot push"
ENDIF
```

To pop an item, stored in item, *from the stack*

```
  IF topPointer = basePointer - 1
    THEN
      OUTPUT "Stack is empty, cannot pop"
    ELSE
      Item ← stack[topPointer]
      topPointer ← topPointer - 1
  ENDIF
```

ACTIVITY 10J

Look at this stack.



Show the stack and the value of topPointer and basePointer when an item has been popped off the stack and 67 followed by 92 have been pushed onto the stack.

10.4.2 Queue operations

The value of the frontPointer changes after dequeue but the value of the rearPointer changes after enqueue:

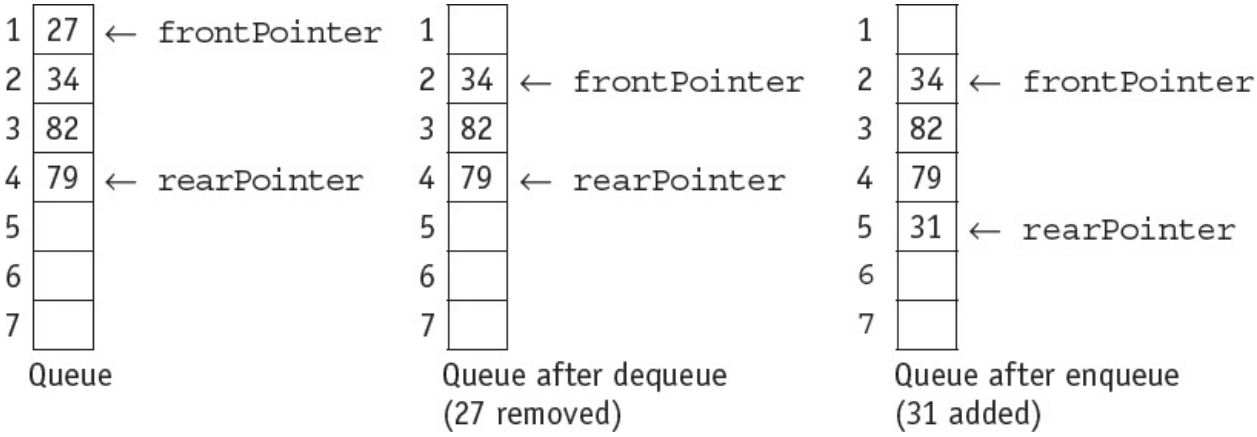


Figure 10.14

A queue can be implemented using an array and a set of pointers. As an array has a finite size, the queue may become full and this condition must be allowed for. Also, as items are removed from the front and added to the end of a queue, the position of the queue in the array changes. Therefore, the queue should be managed as a circular queue to avoid moving the position of the items in the array every time an item is removed.

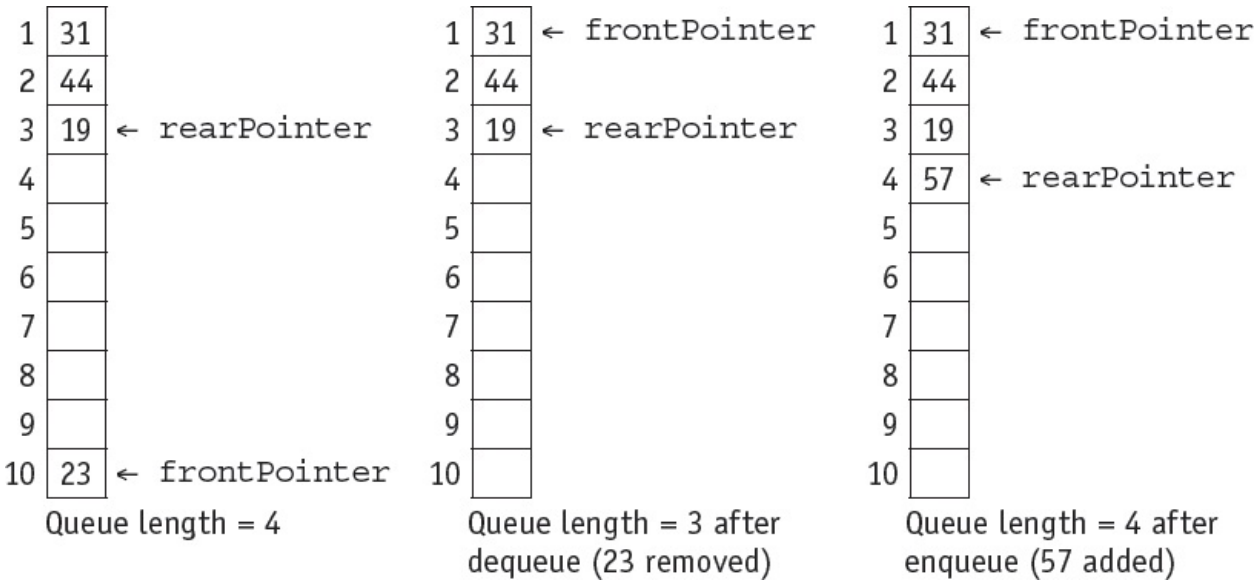


Figure 10.15 Circular queue operation

When a queue is implemented using an array with a finite number of elements, it is managed as a circular queue. Both pointers, frontPointer and rearPointer, are updated to point to the first element in the array (lower bound) after an operation where that pointer was originally pointing to the last element of the array (upper bound), providing the length of the queue does not exceed

the size of the array.

In pseudocode, queue operations are handled using the following statements.

To set up a queue

```
DECLARE queue ARRAY[1:10] OF INTEGER
DECLARE rearPointer : INTEGER
DECLARE frontPointer : INTEGER
DECLARE queueful : INTEGER
DECLARE queueLength : INTEGER
frontPointer ← 1
endPointer ← 0
upperBound ← 10
queueful ← 10
queueLength ← 0
```

To add an item, stored in item, onto a queue

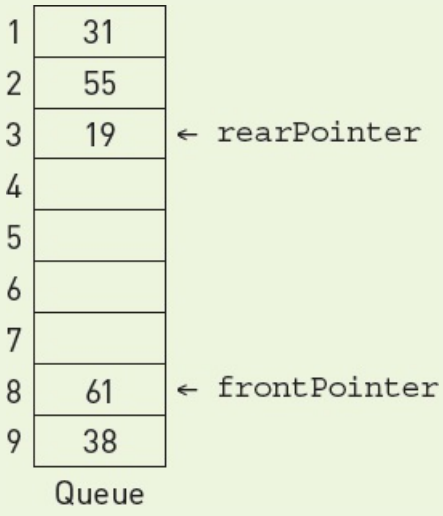
```
IF queueLength < queueful
  THEN
    IF rearPointer < upperBound
      THEN
        rearPointer ← rearPointer + 1
      ELSE
        rearPointer ← 1
    ENDIF
    queueLength ← queueLength + 1
    queue[rearPointer] ← item
  ELSE
    OUTPUT "Queue is full, cannot enqueue"
ENDIF
```

To remove an item from the queue and store in item

```
IF queueLength = 0
  THEN
    OUTPUT "Queue is empty, cannot dequeue"
  ELSE
    Item ← queue[frontPointer]
    IF frontPointer = upperBound
      THEN
        frontPointer ← 1
      ELSE
        frontPointer ← frontPointer + 1
    ENDIF
    queueLength ← queueLength - 1
ENDIF
```

ACTIVITY 10K

Look at this queue.



Show the circular queue and the value of the length of the queue, frontPointer and rearPointer when three items have been removed from the queue and 25 followed by 75 have been added to the queue.

10.4.3 Linked list operations

A linked list can be implemented using two 1D arrays, one for the items in the linked list and another for the pointers to the next item in the list, and a set of pointers. As an array has a finite size, the linked list may become full and this condition must be allowed for. Also, as items can be removed from any position in the linked list, the empty positions in the array must be managed as an empty linked list, usually called the heap.

The following diagrams demonstrate the operations of linked lists.

The startPointer = –1, as the list has no elements. The heap is set up as a linked list ready for use.

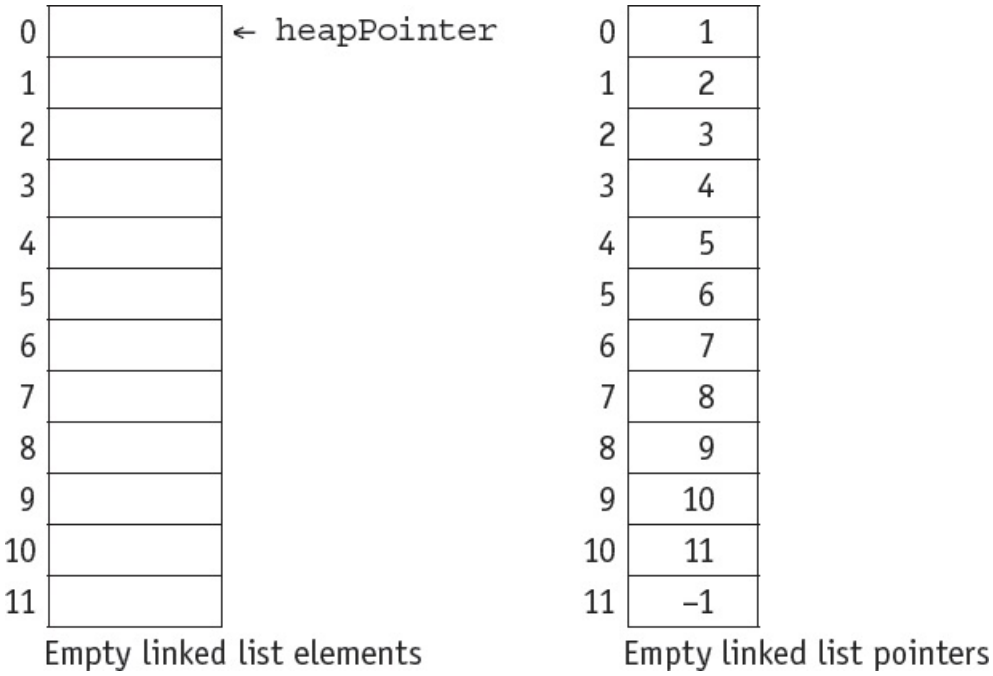


Figure 10.16

The startPointer is set to the element pointed to by the heapPointer where 37 is inserted. The heapPointer is set to point to the next element in the heap by using the value stored in the element with the same index in the pointer list. Since this is also the last element in the list the node pointer for it is reset to –1.

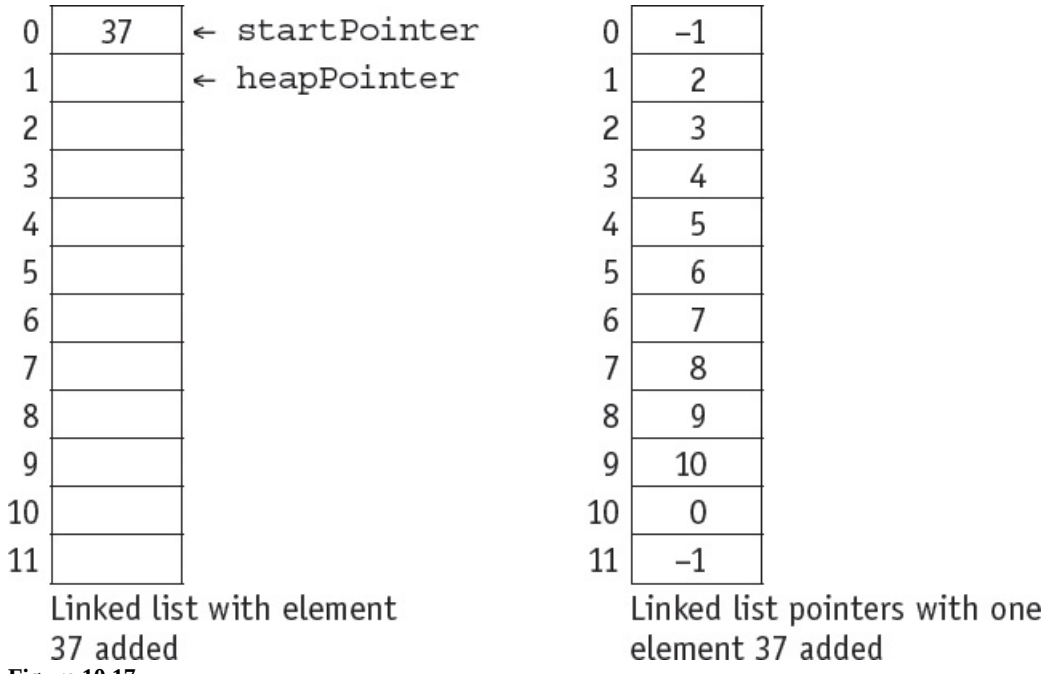


Figure 10.17

The startPointer is changed to the heapPointer and 45 is stored in the element indexed by the heapPointer. The node pointer for this element is set to the old startPointer. The node pointer for the heapPointer is reset to point to the next element in the heap by using the value stored in the element with the same index in the pointer list.

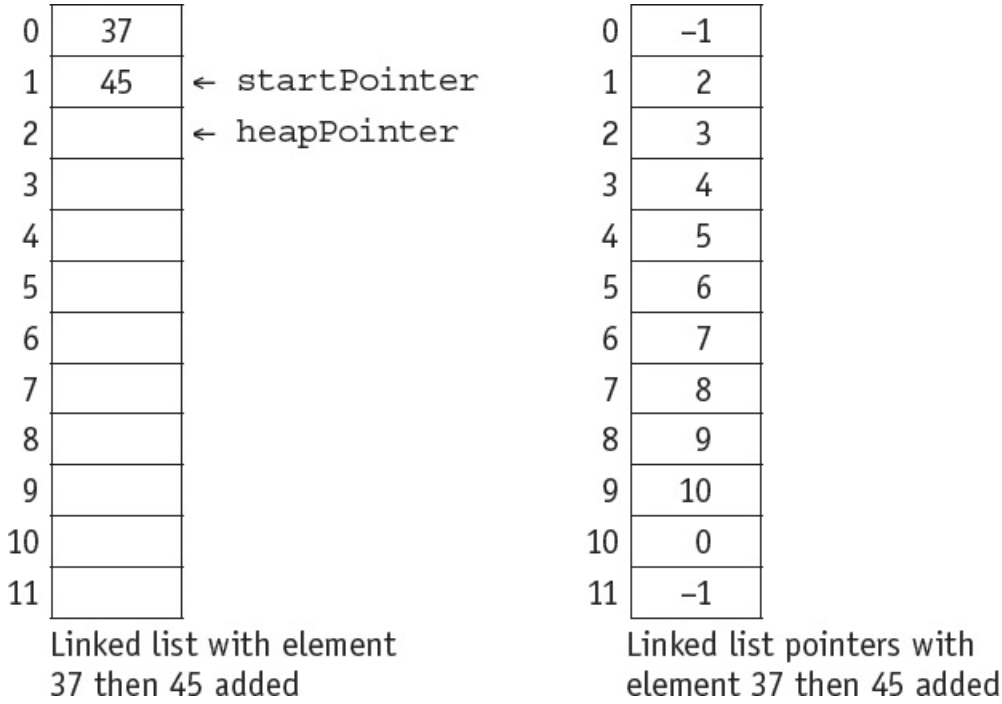
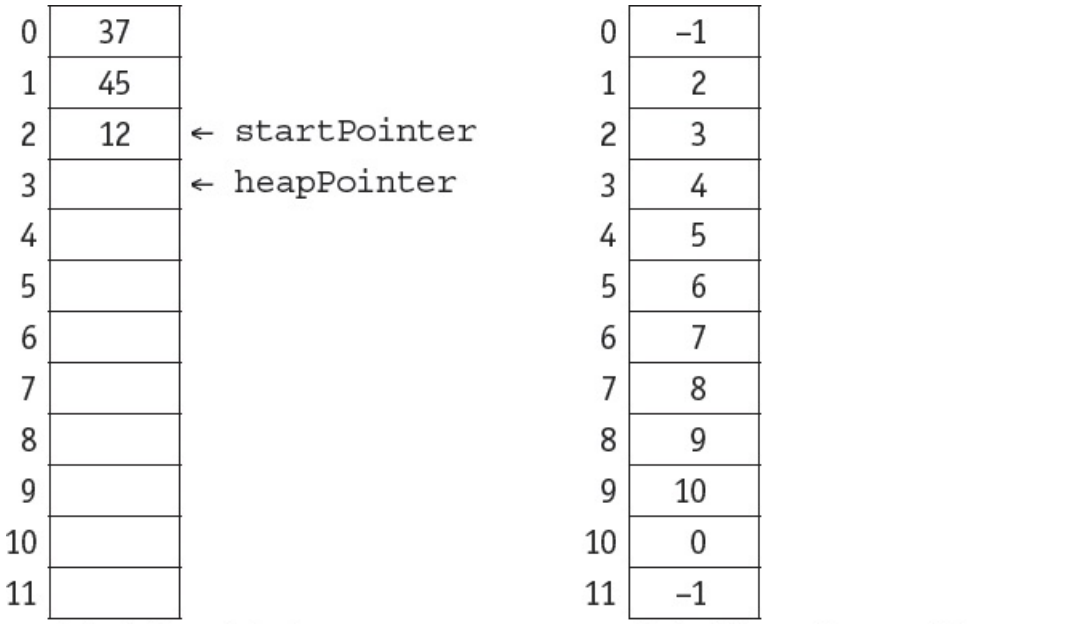


Figure 10.18

The process is repeated when 12 is added to the list.



Linked list with elements 37, 45 then 12 added

Linked list pointers with elements 37, 45 then 12 added

Figure 10.19

To set up a linked list

```
DECLARE mylinkedList ARRAY[0:11] OF INTEGER
DECLARE myLinkedListPointers ARRAY[0:11] OF INTEGER
DECLARE startPointer : INTEGER
DECLARE heapStartPointer : INTEGER
DECLARE index : INTEGER
heapStartPointer ← 0
startPointer ← -1 // list empty
FOR index ← 0 TO 11
    myLinkedListPointers[index] ← index + 1
NEXT index
// the linked list heap is a linked list of all the
spaces in the linked list, this is set up when the
linked list is initialised
myLinkedListPointers[11] ← -1
// the final heap pointer is set to -1 to show no
further links
```

The above code sets up a linked list ready for use. Below is the identifier table.

Identifier	Description
myLinkedList	Linked list to be searched
myLinkedListPointers	Pointers for linked list
startPointer	Start of the linked list
heapStartPointer	Start of the heap
index	Pointer to current element in the linked list

Table 10.6

The table below shows an empty linked list and its corresponding pointers.

heapstartPointer

	myLinkedList	myLinkedListPointers
[0]		1
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		7
[7]		8
[8]		9
[9]		10
[10]		11
[11]		–1

startPointer = –1

Table 10.7 Empty myLinkedList and myLinkedListPointers

You will not be expected to write pseudocode to implement and use these structures, but you will need to be able to show how data can be added to and deleted from these ADTs.

ACTIVITY 10L

Look at this linked list.

0	37		0	−1
1	45		1	2
2	12	← startPointer	2	3
3		← heapPointer	3	4
4			4	5
5			5	6
6			6	7
7			7	8
8			8	9
9			9	10
10			10	0
11			11	

Show the linked list and the value of startPointer and heapPointer when 37 has been removed from the linked list and 18 followed by 75 have been added to the linked list.

EXTENSION ACTIVITY 10C

Write programs to set up and manage a stack and a queue using a 1D array. Use the data in the examples to test your programs.

End of chapter questions

1 Abstract data types (ADTs) are collections of data and the operations used on that data. Explain what is meant by

a) stack

[2]

b) queue

[2]

c) linked list.

[2]

2 Explain, using an example, what is meant by a composite data type.

[2]

3 Explain, using diagrams, the process of reversing a queue using a stack.

[4]

4 a) Write pseudocode to set up a text file to store records like this, with one record on every line.

[4]

```
TYPE
TstudentRecord
    DECLARE name : STRING
    DECLARE address : STRING
    DECLARE className : STRING
ENDTYPE
```

b) Write pseudocode to append a record.

[4]

c) Write pseudocode to find and delete a record.

[4]

d) Write pseudocode to output all the records.

[4]

5 Data is stored in the array NameList[1:10]. This data is to be sorted using a bubble sort:

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 9
        IF NameList[Pointer] > NameList[Pointer + 1]
         THEN
           Temp ← NameList[Pointer]
           NameList[Pointer] ← NameList[Pointer + 1]
           NameList[Pointer + 1] ← Temp
         ENDIF
    NEXT
NEXT
```

a) A special case is when NameList is already in order. The algorithm above is applied to this special case.
Explain how many iterations are carried out for each of the loops.

[2]

b) Rewrite the algorithm using **pseudocode**, to reduce the number of unnecessary comparisons.
Use the same variable names where appropriate.

[5]

Adapted from Cambridge International AS & A Level Computer Science 9608 Paper 41 Q5 part (b) June 2015

6 A queue Abstract Data Type (ADT) has these associated operations:

– create queue
– add item to queue
– remove item from queue

The queue ADT is to be implemented as a linked list of nodes.
Each node consists of data and a pointer to the next node.

a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Copy the diagram below and add appropriate labels to show the final state of the queue.
Use the space on the left as a workspace.
Show your final answer in the node shapes on the right:

[3]

b) Using pseudocode, a record type, Node, is declared as follows:

```
TYPE Node
   DECLARE Name    : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
```

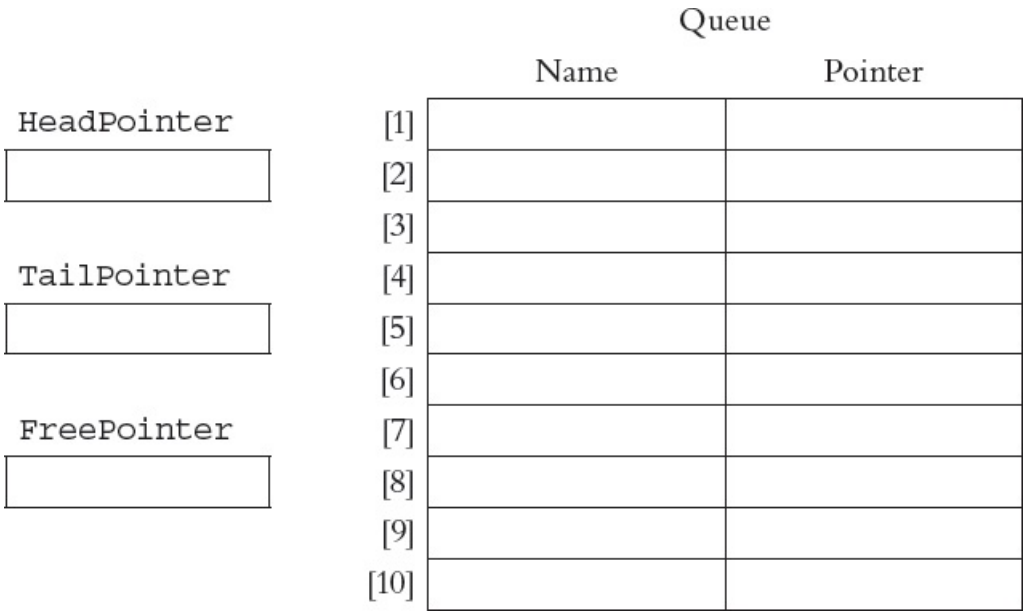
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array Queue.

i) The CreateQueue operation links all nodes and initialises the three pointers that need to be used: HeadPointer, TailPointer and FreePointer.
Copy and complete the diagram to show the value of all pointers after CreateQueue has been executed.

[4]



ii) The algorithm for adding a name to the queue is written, using pseudocode, as a procedure with the header:

```
PROCEDURE AddName(NewName)
```

where NewName is the new name to be added to the queue.
The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
Queue	Array[1:10] OF Node	Array to store node data
NewName	STRING	Name to be added
FreePointer	INTEGER	Pointer to next free node in array
HeadPointer	INTEGER	Pointer to first node in queue
TailPointer	INTEGER	Pointer to last node in queue

CurrentPointer	INTEGER	Pointer to current node

→

```
PROCEDURE AddName(BYVALUE NewName : STRING)
 // Report error if no free nodes remaining
 IF FreePointer = 0
   THEN
     Report Error
     ELSE
     // new name placed in node at head of free list
     CurrentPointer ← FreePointer
     Queue[CurrentPointer].Name ← NewName
     // adjust free pointer
     FreePointer ← Queue[CurrentPointer].Pointer
     // if first name in queue then adjust head pointer
     IF HeadPointer = 0
       THEN
         HeadPointer ← CurrentPointer
     ENDIF
     // current node is new end of queue
     Queue[CurrentPointer].Pointer ← 0
     TailPointer ← CurrentPointer
 ENDIF
ENDPROCEDURE
```

Copy and complete the **pseudocode** for the procedure RemoveName. Use the variables listed in the identifier table.

[6]

```
PROCEDURE RemoveName()
 // Report error if Queue is empty
   ..........
   ..........
   ..........
   ..........
OUTPUT Queue[..........].Name
 // current node is head of queue
   ..........
 // update head pointer
   ..........
 // if only one element in queue then update tail
    pointer
   ..........
   ..........
   ..........
   ..........
 // link released node to free list
   ..........
   ..........
   ..........
ENDPROCEDURE
```

Cambridge International AS & A Level Computer Science 9608
Paper 41 Q6 June 2015

11 Programming

In this chapter, you will learn about

- declaring and assigning values to variables and constants
- using programming constructs, including iteration with IF and CASE structures
- using programming constructs, including selection with three different types of loop: count controlled, post-condition and pre-condition
- writing structured programs defining and using procedures and functions.

WHAT YOU SHOULD ALREADY KNOW

Try this activity before you read the first part of this chapter.

Write an algorithm, using pseudocode, to sort a list of ten numbers. The numbers are to be input with appropriate prompts, stored in an array, sorted in ascending order and then searched for the number 27. The sorted list is to be output, as well as a message stating whether 27 was found or not.

Write and test your algorithm using your chosen programming language. Ensure that you test your program with test data that includes 27 and test data without 27.

11.1 Programming basics

Key terms

Constant – a named value that cannot change during the execution of a program.

Variable – a named value that can change during the execution of a program.

Function – a set of statements that can be grouped together and easily called in a program whenever required, rather than repeating all of the statements each time. Unlike a procedure, a function always returns a value.

Library routine – a tested and ready-to-use routine available in the development system of a programming language that can be incorporated into a program.

Procedure – a set of statements that can be grouped together and easily called in a program whenever required, rather than repeating all of the statements each time.

In order to write a program that performs a specific task or solves a given problem, the solution to the task or problem needs to be designed and then coded. This chapter demonstrates the programming tools used when coding a solution, including basic statements, constructs and structures.

11.1.1 Constants and variables

A **constant** is a named value that cannot change during the execution of a program. A **variable** is a named value that can change during the execution of a program. All variables and constants should be declared before use. Constants will always be assigned a value when declared. It is good practice to assign a value to any variables that are used, so that the programmer is certain of the starting value.

It is good practice to create an identifier list and check that every variable on that list has been declared and a value assigned before any data manipulation is programmed.

Note that some programming languages (for example, Python) do not support the declaration of variables and the concept of a constant; in such cases, the assignment of a value at the start of a program will ensure that a variable of the correct data type can be used. An identifier that is to be used as a constant can also be assigned a value. However, it is important that the programming concepts of data declaration and the difference between variables and constants are clearly understood and can be demonstrated using pseudocode.

Example 11.1

Write an algorithm using pseudocode to calculate and output the volume and surface area of a sphere for any radius that is input.

Solution

First create an identifier table.

Identifier name	Description
radius	Stores radius input
volume	Stores result of volume calculation
surfaceArea	Stores result of surface area calculation
pi	Constant set to 3.142

Then declare constants and variables in pseudocode.

```
DECLARE radius : REAL
DECLARE volume : REAL
DECLARE surfaceArea : REAL
CONSTANT pi ← 3.142
```

Provide pseudocode for input, process and output.

Input usually includes some output in the form of prompts stating what should be input.

```
OUTPUT "Please enter the radius of the sphere "
INPUT radius
```

Check the value of the radius, to ensure that it is suitable.

```
WHILE radius <= 0 DO
    OUTPUT "Please enter a positive number "
    INPUT radius
ENDWHILE
```

Calculate the volume and the surface area; this is the processing part of the algorithm.

```
volume ← (4 / 3) * pi * radius * radius * radius
surfaceArea ← 4 * pi * radius * radius
```

Finally, the results of the calculations need to be output.

```
OUTPUT "Volume is ", volume
OUTPUT "Surface area is ", surfaceArea
```

Table 11.1 shows the declaration of some of the constants and variables from Example 11.1 in the three prescribed programming languages.

While the Cambridge International AS Level syllabus does not require you to be able to write program code, the ability to do so will increase your understanding, and will be particularly beneficial if you are studying the full Cambridge International A Level course.

Declarations of constants and variables	**Language**
pi = 3.142	Python does not require any separate declarations and makes no difference between constants and variables
Dim radius As Decimal Dim volume As Decimal Dim surfaceArea As Decimal Const pi As Decimal = 3.142 Or Dim radius, volume, surfaceArea As	In VB, constants and variables are declared before use. Declarations can be single statements or can contain multiple declarations in a single statement. Constants can be explicitly typed as shown or implicitly typed, for example: Const pi = 3.142

Decimal Public Const pi As Decimal = 3.142	

final double PI = 3.142;
:
:
double volume = (4 / 3) * PI * radius * radius * radius;
	In Java, constant values are declared as variables with a final value so no changes can be made. These final variable names are usually capitalised to show they cannot be changed. Variables are often declared as they are used rather than at the start of the code

Table 11.1

Table 11.2 below gives examples of the input statements in the three prescribed programming languages.

Input statements	**Language**
radius = float(input("Please enter the radius of the sphere "))	Python combines the prompt with the input statement and the type of the input
Console.Write("Please enter the radius of the sphere ") radius = Decimal.Parse(Console.ReadLine())	VB uses a separate prompt and input. The input specifies the type
import java.util.Scanner; : Scanner myObj = new Scanner(System.in); : System.out.println("Please enter the radius of the sphere "); double radius = myObj.nextDouble();	In Java, the input library has to be imported at the start of the program and an input object is set up. Java uses a separate prompt and input. The input specifies the type and declares the variable of the same type

Table 11.2

Table 11.3 below shows how to check the value of the radius in each of the three programming languages.

Check that the value of the radius is a positive number	Language
while radius <= 0:	Python uses a check at the start of the loop
Do : Loop Until radius > 0	VB uses a check at the end of the loop

do
{
:
}
while (radius <= 0);
	Java uses a check at the end of the loop

Table 11.3

Table 11.4 below shows how to calculate the volume and the surface area in each of the three programming languages.

Calculate the volume and the surface area	Language
volume = (4 / 3) * pi * radius * radius * radius surfaceArea = 4 * pi * radius * radius	Python
volume = (4 / 3) * pi * radius * radius * radius surfaceArea = 4 * pi * radius * radius	VB
double volume = (4 / 3) * PI * radius * radius * radius; double surfaceArea = 4 * PI * radius * radius;	Java (variables declared as used)

Table 11.4

Table 11.5 below shows how to output the results in each of the three programming languages.

Output the results	Language
print("Volume is ", volume) print("Surface area is ", surfaceArea)	Python uses a comma
Console.WriteLine("Volume is " & volume) Console.WriteLine ("Surface area is " & surfaceArea)	VB uses &

System.out.println("Volume is " + volume); System.out.println("Surface area is " + surfaceArea);	Java uses +

Table 11.5

The complete programs are shown below:

Python

```
pi = 3.142
radius = float(input("Please enter the radius of the sphere "))
while radius <= 0:
   radius = float(input("Please enter the radius of the sphere "))
volume = (4 / 3) * pi * radius * radius * radius
surfaceArea = 4 * pi * radius * radius
print("Volume is ", volume)
print("Surface area is ", surfaceArea)
```

VB

Every console program in VB must contain a main module. These statements are shown in red

```
Module Module1
   Public Sub Main()
      Dim radius As Decimal
      Dim volume As Decimal
      Dim surfaceArea As Decimal
      Const pi As Decimal = 3.142
      Do
         Console.Write("Please enter the radius of the sphere ")
         radius = Decimal.Parse(Console.ReadLine())
      Loop Until radius > 0
      volume = (4 / 3) * pi * radius * radius * radius
      surfaceArea = 4 * pi * radius * radius
      Console.WriteLine("Volume is " & volume)
      Console.WriteLine ("Surface area is " & surfaceArea)
      Console.ReadKey()
   End Sub
End Module
```

Java

Every console program in Java must contain a class with the file name and a main procedure. These statements are shown in red

```
import java.util.Scanner;
class Activity11A
{
  public static void main(String args[])
    {
      Scanner myObj = new Scanner(System.in);
      final double PI = 3.142;
      double radius;
    do
    {
      System.out.println("Please enter the radius of the sphere ");
      radius = myObj.nextDouble();
    }
    while (radius <= 0);
    double volume = (4 / 3) * PI * radius * radius * radius;
    double surfaceArea = 3 * PI * radius * radius;
    System.out.println("Volume is " + volume);
    System.out.println("Surface area is " + surfaceArea);
    }
}
```

ACTIVITY 11A

Write and test the algorithm in your chosen programming language. Extend the algorithm to allow for more than one calculation, ending when –1 is input, and test it with this test data: 4.7, 34, –11, 0 and –1.

Many programming languages contain built-in **functions** ready to be used. For example, DIV returns the integer part of a division and MOD returns the remainder. For example, DIV(10,3) will return 3, and MOD(10,3) will return 1. See Section 11.3.2 for guidance on how to define a function.

Functions are used for string manipulation. Strings are one of the best ways of storing data. The comparison of data stored in a string has many uses, for example, checking passwords, usernames and other memorable data.

Example 11.2

Write an algorithm using pseudocode to check that the length of a password and the first and last letter of a password input is correct.

You can use these string manipulation functions:

LENGTH(anyString : STRING) RETURNS INTEGER returns the integer value representing

the length of anyString.

RIGHT(anyString: STRING, x : INTEGER) RETURNS STRING returns rightmost x characters from anyString.

LEFT(anyString: STRING, x : INTEGER) RETURNS STRING returns leftmost x characters from anyString.

MID(anyString: STRING, x : INTEGER, y : INTEGER) RETURNS STRING returns y characters starting at position x from anyString.

Solution

First create an identifier table.

Identifier name	Description
storedPassword	Stores password
inputPassword	Stores password to be checked
size	Stores length of password to be checked

`LENGTH(anyString : STRING) RETURNS INTEGER` returns the integer value representing the length of anyString.

`RIGHT(anyString: STRING, x : INTEGER) RETURNS STRING` returns rightmost x characters from anyString.

`LEFT(anyString: STRING, x : INTEGER) RETURNS STRING` returns leftmost x characters from anyString.

`MID(anyString: STRING, x : INTEGER, y : INTEGER) RETURNS STRING` returns y characters starting at position x from anyString.

Table 11.6 below shows the string manipulation functions for Example 11.2 in the three prescribed programming languages.

String manipulation functions	Language
len(inputPassword) inputPassword[0] inputPassword[-1:]	Python first character of string last character of string
Len(inputPassword) Left(inputPassword, 1)	VB first character of string

Right(inputPassword, 1)	last character of string
int size =inputPassword.length(); inputPassword.charAt(0) inputPassword.charAt(size - 1)	Java first character of string last character of string

Table 11.6

ACTIVITY 11B

Write the algorithm in Example 11.2 in your chosen programming language and test it with this test data: "Sad", "Cheese", "Secret" and "secret". Find out how you could extend your program to match upper or lower-case letters.

11.1.2 Library routines

Many programming language development systems include **library routines** that are ready to incorporate into a program. These routines are fully tested and ready for use. A programming language IDE usually includes a standard library of functions and **procedures** as well as an interpreter and/or a compiler. These standard library routines perform tasks such as input/output that are required by most programs.

ACTIVITY 11C

Find out about the standard libraries that are included with the programming language you use. Identify at least six routines included in the library.

11.2 Programming constructs

11.2.1 CASE and IF

The algorithm in Example 11.2 uses a nested IF statement, as there are two different choices to be made. Where there are several different choices to be made, a CASE statement should be used for clarity.

Figure 11.1 shows that choices can be made using a condition based on

- the value of the identifier being considered, for example < 10
- a range of values that the identifier can take, for example 1:10
- the exact value that the identifier can take, for example 10.

And a final catch all for an identifier that met none of the conditions given OTHERWISE.

For each criterion there can be one or many statements to execute.

For each criterion there can be one or many statements to execute.

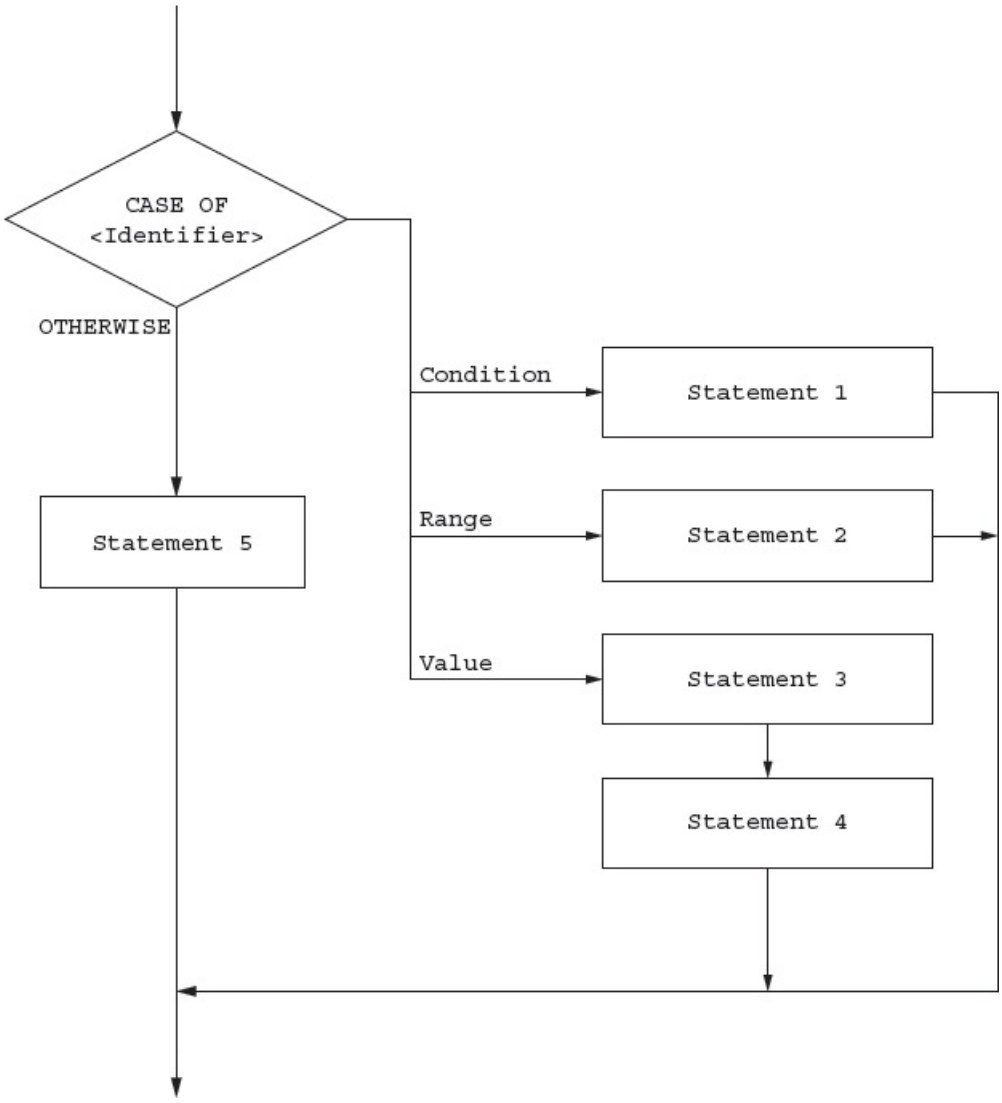


Figure 11.1

For example, choices from a menu can be managed by the use of a CASE statement.

Menu
1 Routine 1
2 Routine 2
3 Routine 3
4 Routine not written
5 Routine not written
6 Routine not written
10 Exit

```
DECLARE choice : INTEGER
OUTPUT "Please enter your choice "
INPUT choice
CASE choice OF
  1 : OUTPUT "Routine 1 "
  2 : OUTPUT "Routine 2 "
  3 : OUTPUT "Routine 3 "
  4 … 6 : OUTPUT "Routine not written"
  10 : Exit
  OTHERWISE OUTPUT "Incorrect choice"
```

Table 11.7 below shows the case statements in VB and Java. Python does not use this construct.

```
Select choice
Case 1
  Console.Writeline ("Routine 1")
Case 2
  Console.Writeline ("Routine 2")
Case 1
  Console.Writeline ("Routine 3")
Case 4, 5, 6
  Console.Writeline ("Routine not written")
Case 10
  Console.Writeline ("Exit")
Case Else
  Console.Writeline ("Incorrect choice")
End Select
```

Table 11.7

ACTIVITY 11D

1 Write an algorithm using pseudocode to either add, subtract, multiply or divide two numbers and output the answer. The two numbers are to be input with appropriate prompts followed by +, −, * or /. Any other character is to be rejected.

2 Check that your pseudocode algorithm works by writing a short program in your chosen programming language from your pseudocode statements using the same names for your identifiers.

11.2.2 Loops

Loops enable sections of code to be repeated as required. They can be constructed in different ways to meet the requirements of an algorithm.

1. a count-controlled loop FOR … NEXT
2. a post-condition loop REPEAT … UNTIL
3. a pre-condition loop WHILE … DO … ENDWHILE

In order to program efficiently, it is important to select the appropriate loop structure to efficiently solve the problem. For example, it is better to use a REPEAT … UNTIL loop rather than a WHILE … DO … ENDWHILE loop for a validation check. The statements inside the loop must always be executed at least once and there is no need for a second input statement.

For example, to check that a value is between 0 and 10 inclusive.

```
REPEAT … UNTIL
REPEAT
    OUTPUT "Enter value "
    INPUT value
UNTIL value < 0 OR value > 10
```

Table 11.8 below shows post-condition loops in VB and Java. Python only uses pre-condition loops.

```
WHILE … DO … ENDWHILE
OUTPUT "Enter value "
INPUT value
WHILE value < 0 OR value > 10 DO
    OUTPUT "Enter value "
    INPUT value
ENDWHILE
```

Table 11.8

Table 11.9 below shows pre-condition loops in each of the three programming languages.

Pre-condition loops	**Language**
`while <condition>:` `  <statements>`	Python
	VB

While <condition> <statements> End While	

while (<condition>)
{
  <statements>;
}
	Java

Table 11.9

Where the number of repetitions is known, a FOR … NEXT loop is the best choice, as the loop counter does not have to be managed by the programmer.

ACTIVITY 11E

Write and test a short program in your chosen programming language to check that an input value is between 0 and 10 inclusive.

1.3 Structured programming

Key terms

Parameter – a variable applied to a procedure or function that allows one to pass in a value for the procedure to use.

By value – a method of passing a parameter to a procedure in which the value of the variable cannot be changed by the procedure.

By reference – a method of passing a parameter to a procedure in which the value of the variable can be changed by the procedure.

Header (procedure or function) – the first statement in the definition of a procedure or function, which contains its name, any parameters passed to it, and, for a function, the type of the return value.

Argument – the value passed to a procedure or function.

11.3.1 Procedures

When writing an algorithm, there are often similar tasks to perform that make use of the same groups of statements. Instead of repeating these statements every time they are required, many programming languages make use of subroutines or named procedures. A procedure is defined once and can be called many times within a program.

Different terminology is used by some programming languages. Procedures are

- void functions in Python
- subroutines in VB
- methods in Java.

To be consistent, we will use the term procedure – you will need to check what to do in your chosen programming language.

A procedure can be defined in pseudocode, as follows:

```
PROCEDURE <identifier>
    <statements>
ENDPROCEDURE
```

The procedure can then be called many times:

```
CALL <identifier>
```

For example, a procedure to print a line of stars would be defined as follows:

```
PROCEDURE stars (Number : INTEGER)
    FOR Counter  1 TO Number
        OUTPUT"*"
    NEXT Counter
ENDPROCEDURE
```

And used like this:

```
CALL stars
```

ACTIVITY 11F

Using the procedure definition and call given for your chosen programming language, write a short program to define and call a procedure to write a line of stars.

Table 11.10 below shows how to define this procedure in each of the three prescribed

programming languages.

```
def stars():
    print("*************")
```

Table 11.10

Table 11.11 shows how it can be used.

Procedure – call	Language
stars()	Python
stars()	VB
stars();	Java

Table 11.11

It is often useful to pass a value to a procedure that can be used to modify the action(s) taken. For example, to decide how many stars would be output. This is done by passing a **parameter** when the procedure is called.

A procedure with parameters can be defined in pseudocode, as follows:

```
Sub stars()
    Console.WriteLine("*************")
End Sub
```

The procedure can then be called many times:

```
static void stars()
{
    System.out.println("************");
}
```

The procedure to print a line with a given number stars would be defined as follows:

```
PROCEDURE stars (Number : INTEGER)
    OUTPUT "*************" ENDPROCEDURE
```

And used like this, to print seven stars:

```
CALL stars (7)
```

The interface between a procedure and a program must match the procedure definition. When a procedure is defined with parameters, the parameters in the procedure call must match those in the procedure definition.

Table 11.12 below shows how to define a procedure with a parameter in each of the three

programming languages.

```
myNumber ← 7
CALL stars (myNumber)
```

Table 11.12

ACTIVITY 11G

Extend your short program in your chosen programming language to define and use a procedure that accepts a parameter to write a line with a given number of stars.

There are two methods of passing a parameter to a procedure: **by value** and **by reference**. When a parameter is passed by value, if a variable is used, the value of that variable cannot be changed within the procedure. When a parameter is passed by reference the value of the variable passed as the parameter can be changed by the procedure.

A procedure with parameters passed by reference can be defined in pseudocode as follows:

```
PROCEDURE <identifier> (BYREF <parameter1>:<datatype>, <parameter2>:<datatype>...)
    <statements>
ENDPROCEDURE
```

For example, a procedure to convert a temperature from Fahrenheit to Celsius could be defined as follows:

```
PROCEDURE celsius(BYREF temperature : REAL)
     temperature ← (temperature - 32) / 1.8
ENDPROCEDURE
```

And used as follows:

```
CALL celsius(myTemp)
```

Table 11.13 below shows how to pass parameters in the three programming languages.

Passing parameters	Language
`Def celsius(temperature):`	Python all data is passed by value

`Sub celsius(temperature As Decimal)`
`Sub celsius(ByVal temperature As Decimal)`
`Sub celsius(ByRef temperature As Decimal)`
	VB
	parameter passed implicitly by value
	parameter passed by value

	parameter passed by reference
`static void celsius(double temperature)`	Java all data is passed by value

Table 11.13

ACTIVITY 11H

Write an algorithm in pseudocode to use a procedure, with a parameter passed by reference, to convert a temperature from Celsius to Fahrenheit.

11.3.2 Functions

When writing an algorithm, there are often similar calculations or tasks to perform that make use of the same groups of statements and ***always*** produce an answer. Instead of repeating these statements every time they are required, many programming languages make use of subroutines or named functions. A function always returns a value; it is defined once and can be called many times within a program. Functions can be used on the right-hand side of an expression.

Different terminology is used by some programming languages. Functions are

- fruitful functions in Python
- functions in VB
- methods with returns in Java.

A function without parameters is defined in pseudocode as follows:

```
FUNCTION <identifier> RETURNS <data type>
    <statements>
ENDFUNCTION
```

A function with parameters is defined in pseudocode as follows:

```
FUNCTION <identifier>(<parameter1>:<datatype>, <parameter2>:<datatype>...)
  RETURNS <data type>
    <statements>
ENDFUNCTION
```

The keyword RETURN is used as one of the statements in a function to specify the value to be returned. This is usually the last statement in the function definition. There can be more than one RETURN used in a function if there are different paths through its statements. This technique needs to be used with great care. For example, a function to find a substring of a given length starting at a given place in a string that returns a null string rather than an error could be:

```
FUNCTION substring (myString : STRING, start : INTEGER, length : INTEGER)
  RETURNS STRING
    IF LENGTH (myString) >= length + start
      THEN
        RETURN MID (myString, start, length)
      ELSE
        RETURN ""
    ENDIF
ENDFUNCTION
```

Functions are used as part of an expression. The value returned by the function is used in the expression.

For example, the procedure used previously to convert a temperature from Fahrenheit to Celsius could be written as a function:

```
FUNCTION celsius (temperature : REAL) RETURNS REAL
    RETURN (temperature - 32) / 1.8
ENDFUNCTION
```

And used as follows:

```
myTemp ← celsius(myTemp)
```

The interface between a function and a program must match the function definition. When a function is defined with parameters, the parameters in the function call must match those in the function definition.

Table 11.14 below shows how to write this function in each of the three programming languages.

```
def celsius(temperature):
    return (temperature - 32) / 1.8
```

Table 11.14

ACTIVITY 11I

Re-write the algorithm you wrote in Activity 11H as a function, with a parameter, to convert a temperature from Celsius to Fahrenheit. Test your algorithm by writing a short program in your chosen programming language to define and use this function. Note the differences in your programs and state, with reasons, which is the better structure to use for this algorithm: a procedure or a function.

When procedures and functions are defined, the first statement in the definition is a **header**, which contains

- the name of the procedure or function
- any parameters passed to the procedure or function
- the type of the return value for a function.

When procedures or functions are called, the parameters or **arguments** (the values passed to the procedure or function) must be in the same order as the parameters in the declaration header and each argument must be of the same type as the parameter given in the header. Procedure calls are single stand-alone statements and function calls form part of an expression on the right-hand side.

End of chapter questions

1 Use pseudocode to declare these variables and constants.
You will need to decide which identifiers are variables and which are constants.

[6]

Identifier name	Description
height	Stores value input for length of height
maxHeight	Maximum height, 25
width	Stores value input for length of width
maxWidth	Maximum width, 30
hypotenuse	Stores calculated value of hypotenuse
area	Stores calculated value of area

2 Write a pseudocode algorithm to input the height and width of a right-angled triangle and check that these values are positive and less than the maximum values given in question 1.

[4]

3 **a)** For this question, you will need to use this function, which returns the real value of the square root of anyPosVal:

```
SQUAREROOT(anyPosVal : REAL) RETURNS REAL
```

Extend your algorithm for Question 2 to

i) calculate the hypotenuse

[2]

ii) calculate the area

[2]

iii) calculate the perimeter.

[2]

b) Provide a menu to choose which calculation to perform.

[3]

c) Check that all the above work by writing and testing a program in your chosen programming language.

[6]

4 Explain what is meant by the term *library routine*.
Give **two** examples of uses of library routines.

[4]

5 Procedures and functions are subroutines.
Explain what is meant by

a) a procedure

[2]

b) a function

[2]

c) a parameter

[2]

d) a procedure or function header.

[2]

6 Explain the difference between

a) a procedure and a function

[2]

b) passing parameters by value and by reference

[2]

c) defining a procedure and calling a procedure.

[2]

7 A driver buys a new car.
The value of the car reduces each year by a percentage of its current value.
The percentage reduction is:
– in the first year, 40%
– in each following year, 20%
The driver writes a program to predict the value of the car in future years.
The program requirements are:
– enter the cost of the new car (to nearest $)
– calculate and output the value of the car at the end of each year
– the program will end when either the car is nine years old, or when the value is less than $1000.

a) Study the incomplete pseudocode which follows in part b) and copy and complete this identifier table.

[3]

Identifier	Data type	Description

b) Copy and complete the pseudocode for this design.

[6]

```
OUTPUT "Enter purchase price"
INPUT PurchasePrice
CurrentValue ← ..........................................
YearCount ← 1
WHILE ..................... AND ....................
   IF ..................................................
     THEN
     CurrentValue ← CurrentValue * (1 - 40 / 100)
   ELSE
     CurrentValue ← ..................................
 ENDIF
 OUTPUT YearCount, CurrentValue
  ..........................................................
ENDWHILE
```

Cambridge International AS & A Level Computer Science 9608
Paper 21 Q5 November 2015

12 Software development

In this chapter, you will learn about

- the purpose, types and stages of the program development lifecycle
- how to document program design using structure charts and state-transition diagrams
- avoiding syntax, logic and run-time errors in programs
- different methods of testing programs to identify and correct such errors
- the types of maintenance used as part of the program development lifecycle.

12.1 Program development lifecycle

WHAT YOU SHOULD ALREADY KNOW

You may have studied or heard about the systems (or program) development lifecycle.

Try this activity before you read the first part of this chapter. A program development lifecycle goes through the same stages for a program.

Name and describe the stages of the program/ systems development lifecycle. There are

different development lifecycles used depending upon the system and the type of program being developed. Identify at least four of these. Work in small groups to research one of these and share your findings with the other groups.

Key terms

Program development lifecycle – the process of developing a program set out in five stages: analysis, design, coding, testing and maintenance.

Analysis – part of the program development lifecycle; a process of investigation, leading to the specification of what a program is required to do.

Design – part of the program development lifecycle; it uses the program specification from the analysis stage to show how the program should be developed.

Coding – part of the program development lifecycle; the writing of the program or suite of programs.

Testing – part of the program development lifecycle; the testing of the program to make sure that it works under all conditions.

Maintenance – part of the program development lifecycle; the process of making sure that the program continues to work during use.

Waterfall model – a linear sequential program development cycle, in which each stage is completed before the next is begun.

Iterative model – a type of program development cycle in which a simple subset of the requirements is developed, then expanded or enhanced, with the development cycle being repeated until the full system has been developed.

Rapid application development (RAD) – a type of program development cycle in which different parts of the requirements are developed in parallel, using prototyping to provide early user involvement in testing.

12.1.1 The purpose of a program development lifecycle

In order to develop a successful program or suite of programs that is going to be used by others to perform a specific task or solve a given problem, the development needs to be well ordered and clearly documented, so that it can be understood and used by other developers.

This chapter introduces the formal stages of software (program) development that are set out in the **program development lifecycle**.

A program that has been developed may require alterations at any time in order to deal with new circumstances or new errors that have been found, so the stages are referred to as a lifecycle as this continues until the program is no longer used.

12.1.2 Stages in the program development lifecycle

The stages of development in the program development lifecycle are shown in this chapter. The coding, testing and maintenance stages are looked at in depth and include appropriate tools and techniques to be used at each stage. Therefore, practical activities will be suggested for these stages to help reinforce the skills being learnt.

Here is a brief overview of the program development lifecycle, divided into the five stages, as shown in Figure 12.1.

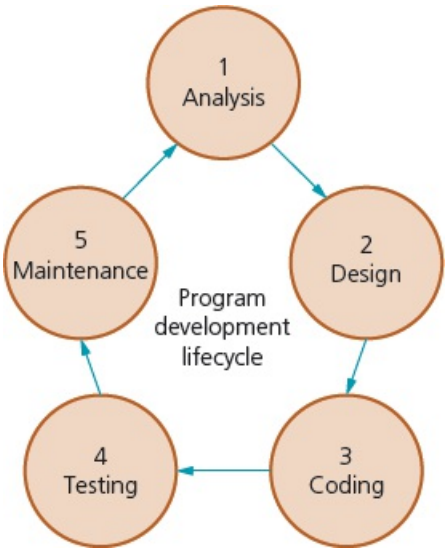


Figure 12.1 The program development lifecyle

Analysis

Before any problem can be solved, it needs to be clearly defined and set out so everyone working on the solution understands what is needed. This is called the requirements specification. The **analysis** stage often starts with a feasibility study, followed by investigation and fact finding to identify exactly what is required from the program.

Design

The program specification from the analysis stage is used to show how the program should be developed. When the **design** stage is complete, the programmer should know what is to be done, all the tasks that need to be completed, how each task is to be performed and how the tasks work together. This can be formally documented using structure charts, state-transition diagrams and pseudocode.

Coding

The program or set of programs is written using a suitable programming language.

Testing

The program is run many times with different sets of test data, to test that it does everything it is supposed to do in the way set out in the program design.

Maintenance

The program is maintained throughout its life, to ensure it continues to work effectively. This involves dealing with any problems that arise during use, including correcting any errors that come to light, improving the functionality of the program, or adapting the program to meet new requirements.

12.1.3 Different development lifecycles

Each program development methodology has its own strength. Different models have been developed based on the lifecycle for developers to use in practice. The models we will consider will be divided into the five stages set out above: analysis, design, **coding**, **testing** and **maintenance**.

In this section of the chapter, we will look at three models: The **waterfall model**, the **iterative model**, and **rapid application development (RAD)**.

The waterfall model

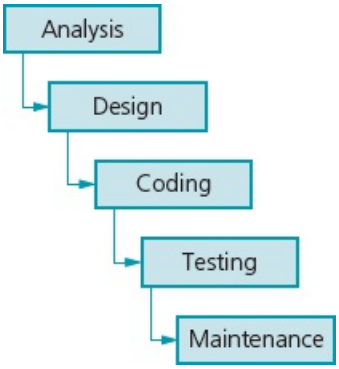


Figure 12.2 The waterfall model

This linear sequential development cycle is one of the earliest models used, where each stage is completed and signed off before the next stage is begun. This model is suitable for smaller projects with a short timescale, for which the requirements are well known and unlikely to change.

Principles	linear, as each stage is completed before the next is begun
	well documented as full documentation is completed at every stage
	low customer involvement; only involved at the start and end of the process
Benefits	easy to manage, understand and use
	stages do not overlap and are completed one at a time
	each stage has specific deliverables
	works well for smaller programs where requirements are known and understood
Drawbacks	difficult to change the requirements at a later stage
	not suitable for programs where the requirements could be subject to change
	working program is produced late in the lifecycle
	not suitable for long, complex projects

Table 12.1 Principles, benefits and drawbacks to the waterfall model

The iterative model

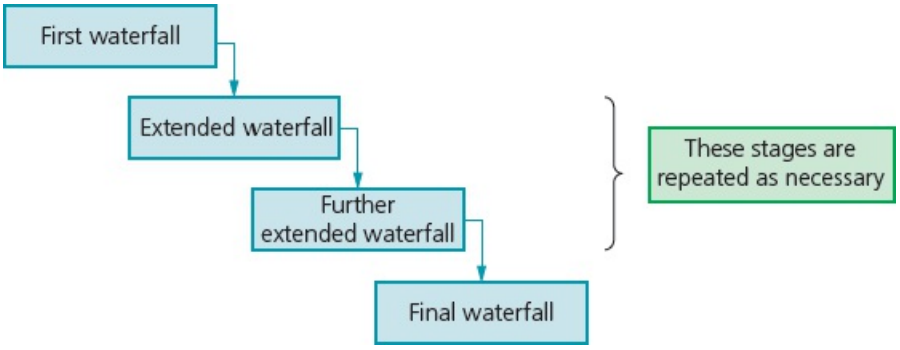


Figure 12.3 The iterative model

This development cycle first develops a simple subset of the requirements, then expands or enhances the model and runs the development cycle again. These program development cycles are repeated until the full system has been developed. This model is suitable for projects for which the major requirements are known but some details are likely to change or evolve with time.

Principles	incremental development as the program development lifecycle is repeated
	working programs are produced for part of the system at every iteration
	high customer involvement, as part of the system can be shown to the customer after every iteration
Benefits	some working programs developed quickly at an early stage in the lifecycle
	easier to test and debug smaller programs
	more flexible as easier to alter requirements
	customers involved at each iteration therefore no surprises when final system delivered
Drawbacks	whole system needs to be defined at start, so it can be broken down into pieces to be developed at each iteration
	needs good planning overall and for every stage
	not suitable for short simple projects

Table 12.2 Principles, benefits and drawbacks to the iterative model

Rapid application development (RAD)

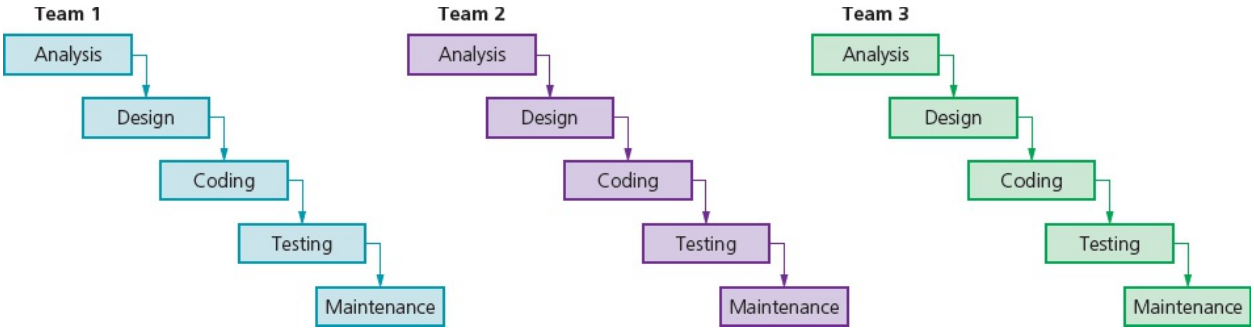


Figure 12.4 Rapid application development (RAD)

This development cycle develops different parts of the requirements in parallel, using prototyping to provide early user involvement in testing. Program development cycles are run in parallel for each part of the requirement, using a number of different teams. Prototyping is often used to show initial versions to customers to obtain early feedback. This model is suitable for complicated projects that need developing in a short timeframe to meet the evolving needs of a business.

Principles	minimal planning
	reuses previously written code where possible, makes use of automated code generation where possible
	high customer involvement, as customers can use the prototypes during development
Benefits	reduced overall development time
	rapid frequent customer feedback informs the development
	very flexible as requirements evolve from feedback during development
	as parts of the system are developed side by side, modification is easier because each part must work independently
Drawbacks	system under development needs to be modular
	needs strong teams of skilled developers
	not suitable for short simple projects

Table 12.3 Principles, benefits and drawbacks to rapid application development (RAD)

EXTENSION ACTIVITY 12A

Find out about four more program development methodologies.

12.2 Program design

WHAT YOU SHOULD ALREADY KNOW

In this chapter, you will need to be able to write more complicated pseudocode, as described by a structure chart. It is essential that you consolidate your knowledge before you attempt to do this.

Make sure that you have read and understood Chapter 11 and you are able to write pseudocode that passes parameters to procedures and functions.

Key terms

Structure chart – a modelling tool used to decompose a problem into a set of sub-tasks. It shows the hierarchy or structure of the different modules and how they connect and interact with each other.

Finite state machine (FSM) – a mathematical model of a machine that can be in one state of a fixed set of possible states; one state is changed to another by an external input; this is known as a transition.

State-transition diagram – a diagram showing the behaviour of a finite state machine (FSM).

State-transition table – a table showing every state of a finite state machine (FSM), each possible input and the state after the input.

12.2.1 Purpose and use of structure charts

A **structure chart** is a modelling tool used in program design to decompose a problem into a set of sub-tasks. The structure chart shows the hierarchy or structure of the different modules and how they connect and interact with each other. Each module is represented by a box and the parameters passed to and from the modules are shown by arrows pointing towards the module receiving the parameter. Each level of the structure chart is a refinement of the level above.

Figure 12.5 shows a structure chart for converting a temperature from Fahrenheit to Celsius. The top level shows the name for the whole task that is refined into three sub-tasks or modules shown on the next level.

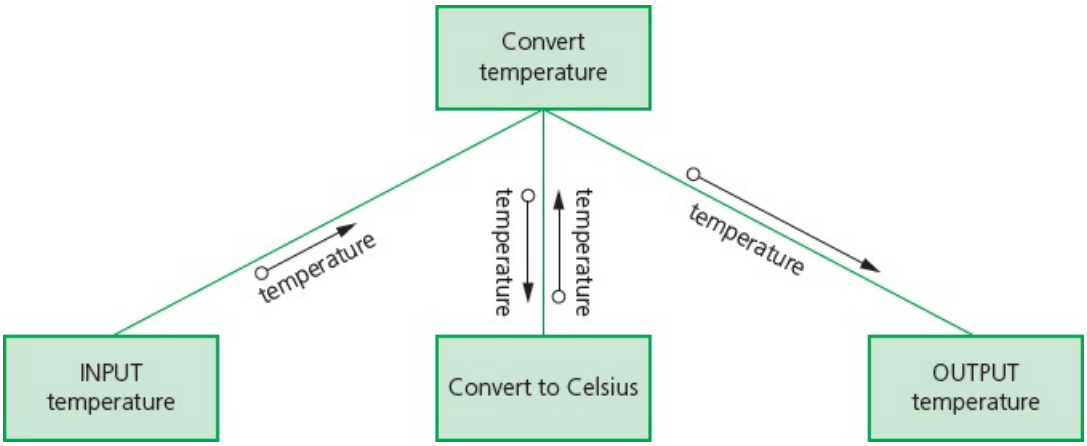


Figure 12.5 A structure chart for converting a temperature from Fahrenheit to Celsius

ACTIVITY 12A

Draw a structure chart to input the height and width of a right-angled triangle, calculate output and output the length of the hypotenuse.

Structure charts can also show selection. The temperature conversion task above could be extended to either convert from Fahrenheit to Celsius or Celsius to Fahrenheit using the diamond shaped box to show a condition that could be true or false, as shown in Figure 12.6.

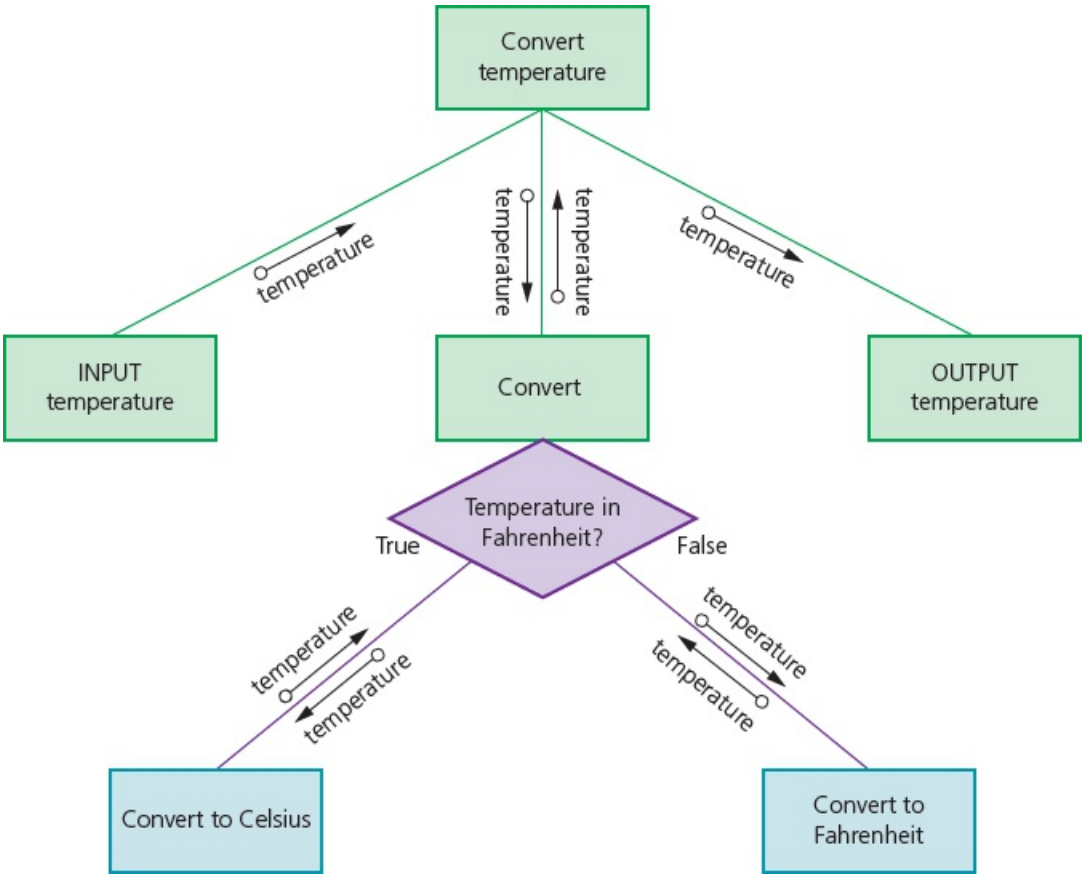


Figure 12.6

ACTIVITY 12B

Draw a structure chart to input the radius of a sphere, calculate output and output either the volume or the surface area.

Structure charts can also show repetition. The temperature conversion task above could be extended to repeat the conversion until the number 999 is input. The repetition is shown by adding a labelled semi-circular arrow above the modules to be completed (Figure 12.7).

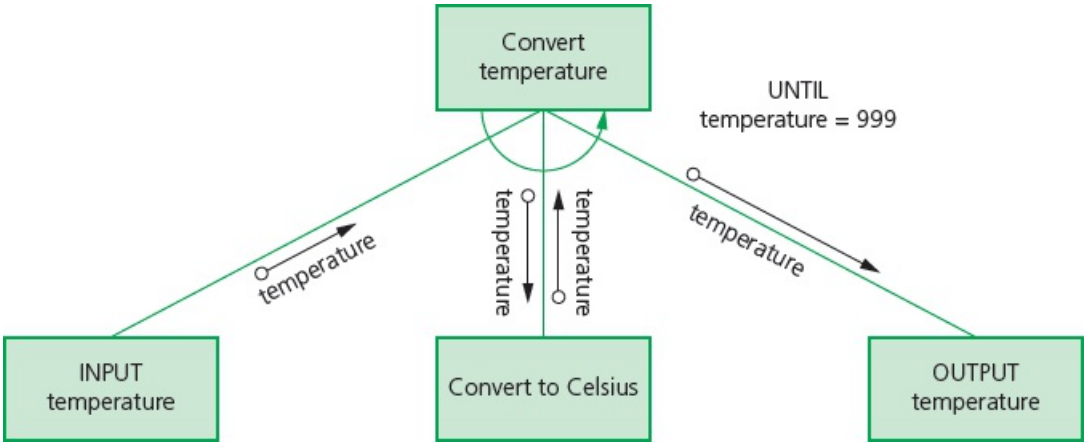


Figure 12.7

Once a structure chart has been completed, it can be used to derive a pseudocode algorithm.

ACTIVITY 12C

Amend your structure chart to input the radius of a sphere, calculate and output either the volume or the surface area. The algorithm should repeat until a radius of zero is entered.

Figure 12.8 shows a possible structure chart for Activity 12C.

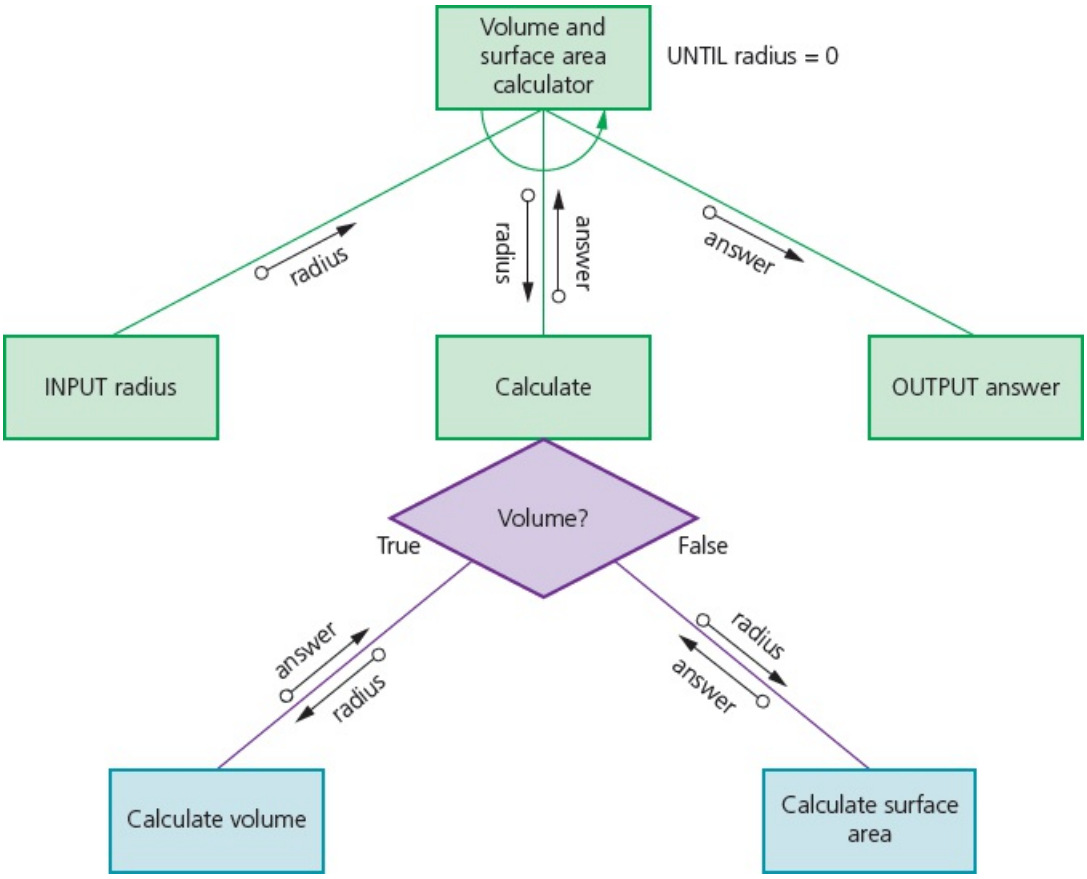


Figure 12.8

To derive the pseudo code first, you will need to create an identifier table.

Identifier name	Description
radius	Stores radius input
answer	Stores result of calculation
pi	Constant set to 3.142

Table 12.4

Then declare constants and variables in pseudocode. You can identify two of the variables

required from the parameters shown in the structure diagram.

```
DECLARE radius : REAL
DECLARE answer : REAL
CONSTANT pi ← 3.142
```

Provide pseudocode for the modules shown in the structure diagram. As Calculate volume and Calculate surface area provide the answer to a calculation, these can be defined as functions.

```
FUNCTION calculateVolume (radius:real) RETURNS real
    RETURN (4 / 3) * pi * radius * radius * radius
ENDFUNCTION
FUNCTION calculateSurfaceArea (radius:real) RETURNS real
    RETURN 4 * pi * radius * radius
ENDFUNCTION
```

The input and output modules could be defined as procedures.

```
PROCEDURE inputRadius
    OUTPUT "Please enter the radius of the sphere "
    INPUT radius
    WHILE radius < 0 DO
        OUTPUT "Please enter a positive number "
        INPUT radius
    ENDWHILE
ENDPROCEDURE
PROCEDURE outputAnswer
    OUTPUT answer
ENDPROCEDURE
```

The pseudocode for the whole algorithm, including the selection and repetition, would be as follows.

```
DECLARE radius : REAL
DECLARE answer : REAL
CONSTANT pi ← 3.142
FUNCTION calculateVolume (radius:real) RETURNS real
    RETURN (4 / 3) * pi * radius * radius * radius
ENDFUNCTION
FUNCTION calculateSurfaceArea (radius:real) RETURNS real
    RETURN 4 * pi * radius * radius
ENDFUNCTION
PROCEDURE inputRadius
    OUTPUT "Please enter the radius of the sphere "
    INPUT radius
    WHILE radius < 0 DO
        OUTPUT "Please enter a positive number "
        INPUT radius
    ENDWHILE
ENDPROCEDURE
PROCEDURE outputAnswer
    OUTPUT answer
ENDPROCEDURE
CALL inputRadius
WHILE radius <> 0
    OUTPUT "Do you want to calculate the Volume (V) or Surface Area (S)"
    INPUT reply
    IF reply = "V"
      THEN
        answer ← calculateVolume(radius)
        OUTPUT "Volume "
      ELSE
        answer ← calculateSurfaceArea(radius)
        OUTPUT "Surface Area "
    ENDIF
    CALL outputAnswer
    CALL inputRadius
ENDWHILE
```

ACTIVITY 12D

Draw a structure chart to extend the temperature conversion algorithm to both convert from Fahrenheit to Celsius and Celsius to Fahrenheit, and repeat until a temperature of 999 is input.

Use your structure chart to create an identifier table and write the pseudocode for this algorithm.

12.2.2 Purpose and use of state-transition diagrams to document algorithms

A **finite state machine (FSM)** is a mathematical model of a machine that can be in one of a fixed set of possible states. One state is changed to another by an external input, this is called a transition. A diagram showing the behaviour of an FSM is called a **state-transition diagram**.

State-transition diagrams show the conditions needed for an event or events that will cause a transition to occur, and the outputs or actions carried out as the result of that transition.

State-transition diagrams can be constructed as follows:

- States are represented as nodes (circles).
- Transitions are represented as interconnecting arrows.
- Events are represented as labels on the arrows.
- Conditions can be specified in square brackets after the event label.
- The initial state is indicated by an arrow with a black dot.
- A stopped state is indicated by a double circle.

The algorithm for unlocking a door using a three-digit entry code can be represented by a state-transition diagram. If the door is unlocked with a three-digit entry code, the lock can be in four states

- locked and waiting for the input of the first digit
- waiting for the input of the second digit
- waiting for the input of the third digit
- unlocked.

If an incorrect digit is input, then the door returns to the locked state. The algorithm halts when the door is unlocked. A **state-transition table** shows every state, each possible input and the state after the input. The state-transition table for a door with the entry code 259 is shown below.

Current state	Event	Next state
locked	2 input	waiting for input of 2nd digit
locked	not 2 input	locked
waiting for input of 2nd digit	5 input	waiting for input of 3rd digit
waiting for input of 2nd digit	not 5 input	locked
waiting for input of 3rd digit	9 input	unlocked and stopped
waiting for input of 3rd digit	not 9 input	locked

Table 12.5 The state-transition table for a door with the entry code 259

The state-transition diagram for a door with the entry code 259 is shown in Figure 12.9.

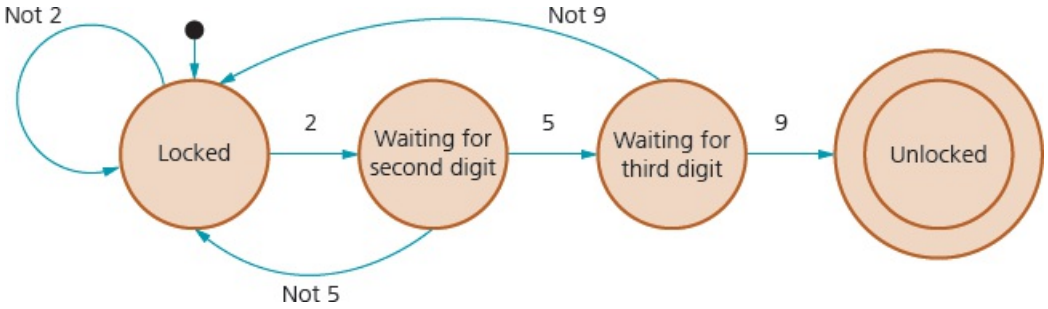


Figure 12.9 State-transition diagram for a door with the entry code 259

ACTIVITY 12E

Draw a state-transition diagram for the operation of a television with a single on/off button. The television can be in three states: on, off and standby. If the button is pressed once (single press) in standby or off, the television switches on; if the button is pressed once (single press) when the television is on, the television goes to standby; if the button is pressed twice (double press) when the television is on, the television goes to off. Double presses in standby or off are ignored.

Copy and complete the state-transition table and draw the state-transition diagram for the television operation.

Current state	**Event**	**Next state**
Off	Single press	
Off	Double press	
Standby	Single press	
Standby	Double press	
On	Single press	
On	Double press	

12.3 Program testing and maintenance

WHAT YOU SHOULD ALREADY KNOW

You will need to be able to know how to thoroughly test any programs that you write. Have a go at the activity below.

Take a program that you have written recently and explain to another student how you tested the program and which data sets you chose for your testing.

Key terms

Trace table – a table showing the process of dry-running a program with columns showing the values of each variable as it changes.

Run-time error – an error found in a program when it is executed; the program may halt unexpectedly.

Test strategy – an overview of the testing required to meet the requirements specified for a particular program; it shows how and when the program is to be tested.

Test plan – a detailed list showing all the stages of testing and every test that will be performed for a particular program.

Dry run – a method of testing a program that involves working through a program or module from a program manually.

Walkthrough – a method of testing a program. A formal version of a dry run using pre-defined test cases.

Normal test data – test data that should be accepted by a program.

Abnormal test data – test data that should be rejected by a program.

Extreme test data – test data that is on the limit of that accepted by a program.

Boundary test data – test data that is on the limit of that accepted by a program or data that is just outside the limit of that rejected by a program.

White-box testing – a method of testing a program that tests the structure and logic of every path through a program module.

Black-box testing – a method of testing a program that tests a module's inputs and outputs.

Integration testing – a method of testing a program that tests combinations of program modules that work together.

Stub testing – the use of dummy modules for testing purposes.

Alpha testing – the testing of a completed or nearly completed program in-house by the development team.

Beta testing – the testing of a completed program by a small group of users before it is

released.

Acceptance testing – the testing of a completed program to prove to the customer that it works as required.

Corrective maintenance – the correction of any errors that appear during use.

Perfective maintenance – the process of making improvements to the performance of a program.

Adaptive maintenance – the alteration of a program to perform new tasks.

12.3.1 Ways of avoiding and exposing faults in programs

Most programs written to perform a real task will contain errors, as programmers are human and do make mistakes. The aim is to avoid making as many mistakes as possible and then find as many mistakes as possible before the program goes live. Unfortunately, this does not always happen and many spectacular failures have occurred. More than one large bank has found that its customers were locked out of their accounts for some time when new software was installed. Major airlines have had to cancel flights because of programming errors. One prison service released prisoners many days earlier than required for about 15 years because of a faulty program.

Faults in an executable program are frequently faults in the design of the program. Fault avoidance starts with the provision of a comprehensive and rigorous program specification at the end of the analysis phase of the program development lifecycle, followed by the use of formal methods such as structure charts, state-transition diagrams and pseudocode at the design stage. At the coding stage, the use of programming disciplines such as information hiding, encapsulation and exception handling, as described in Chapter 20, all help to prevent faults.

Faults or bugs in a program are then exposed at the testing stage. Testing will show the presence of faults to be corrected, but cannot guarantee that large, complex programs are fault free under all circumstances. Faults can appear during the lifetime of a program and may be exposed during live running. The faults are then corrected as part of the maintenance stage of the program lifecycle.

EXTENSION ACTIVITY 12B

In small groups, research recent spectacular software failures. Choose one failure per group and find out what went wrong and how it affected the organisation and their customers. Summarise and present your group’s findings to the rest of the class.

12.3.2 Location, identification and correction of errors

Syntax errors are errors in the grammar of a source program. In the coding phase of the program development lifecycle, programs are either compiled or interpreted so they can be executed. During this operation, the syntax of the program is checked, and errors need to be corrected before the program can be executed. Figure 12.10 shows an example of a syntax error being found, and the IDE offering a possible reason for the error.

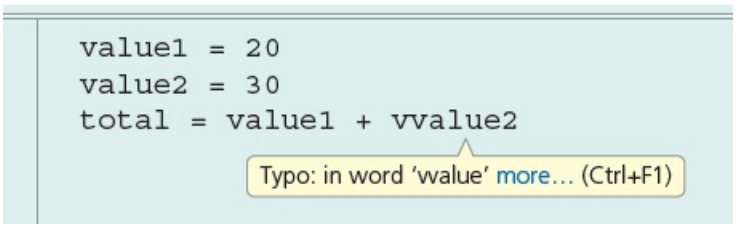


Figure 12.10

Many IDEs will offer suggestions about what syntax errors are and how to correct them.

Logic errors are errors in the logic of a program, meaning the program does not do what it is supposed to do. These errors are usually found when the program is being tested. For example, the program in Figure 12.11 will run, but the results will not be as expected.

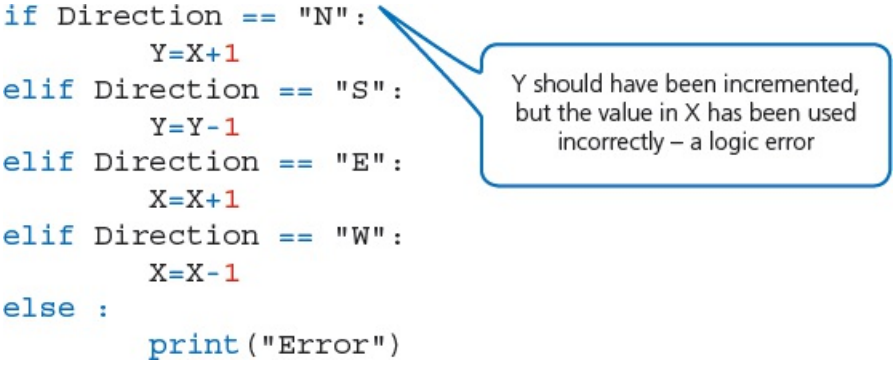


Figure 12.11

Many IDEs will allow you to single step through a program to find errors (see Chapter 5, Section 5.2.4). You can also manually work through a program to check that it works as it should, using tools such as a **trace table**. Trace tables show the process of dry-running a program with columns showing the values of each variable as it changes.

Run-time errors happen when the program is executed. The program may halt unexpectedly or go into an infinite loop and need to be stopped by brute force. If a program is being tested in an IDE, then this type of error may be managed, and a suitable error message given, as shown below.

Program with divide by zero error

```
1  value1 = 10
2  value2 = 0
3  value3 = value1 + value2
4  value4 = value1 / value2
Traceback (most recent call last):
  File "ErrorTest.py", line 4, in <module>
    value4 = value1 /value2
ZeroDivisionError: division by zero
Process finished with exit code 1
```

If the program has already been released for use and a run-time error occurs, the developer should be informed so that the program can be updated and re-released or a patch can be sent out to all customers to solve the problem. A patch is a small program, released by the developers, to run with an existing program to correct an error or provide extra functionality. The Windows operating system is frequently patched and the process of downloading patches and updating the program has been automated.

12.3.3 Program testing

Programs need to be rigorously tested before they are released. Tests begin from the moment they are written; they should be documented to show that the program is robust and ready for general use.

There needs to be a **test strategy** set out in the analysis stage of the program development lifecycle showing an overview of the testing required to meet the requirements specified. This shows how and when the program is to be tested.

In order to clarify what tests need to be performed, a **test plan** is drawn up showing all the stages of testing and every test that will be performed. As the testing is carried out, the results of the tests can be added to the plan showing that the program has met its requirements.

There are several formal methods of testing used to ensure that a program is robust and works to the standard required. Although there is a testing stage in the program development lifecycle, testing in some form occurs at every stage, from design to maintenance.

During the program design stage, pseudocode is written. This can be tested using a **dry run**, in which the developer works through a program or module from a program manually and documents the results using a trace table.

For example, a procedure to perform a calculation could be tested as follows.

```
PROCEDURE calculation(number1, number2, sign)
CASE sign OF
   '+' : answer ← number1 + number2
   '-' : answer ← number1 + number2
   '*' : answer ← number1 * number2
   '/' : answer ← number1 / number2
  OTHERWISE answer ← 0
ENDCASE
IF answer <> 0
  THEN
     OUTPUT answer
ENDIF
ENDPROCEDURE
```

The test data used could include 20 10 +, 20 10 −, 20 10 *, 20 10 /, 20 10 ? and 20 0 /.

The trace table below shows the value of each variable and any output.

number1	number2	sign	answer	OUTPUT
20	10	+	30	30
20	10	–	30	30
20	10	*	200	200
20	10	/	2	2
20	10	?	0	
20	0	/	undefined	

Table 12.6

The errors found in the routine by performing the dry run have been highlighted in red. These can now be corrected before this routine is coded.

ACTIVITY 12F

Correct the pseudocode and perform the dry run again to ensure that your corrections work.

ACTIVITY 12G

Using the pseudocode algorithm for the volume and surface area of a sphere in Section 12.2.1, devise some test data and a trace table to test the algorithm.

Swap your test data and trace table with another student then perform the dry run and complete the trace table.

Discuss any differences or problems you find.

A **walkthrough** is a formalised version of a dry run using pre-defined test cases. This is where another member of the development team independently dry runs the pseudocode, or the developer takes the team members through the dry run process. This is often done as a demonstration.

During the program development and testing, each module is tested as set out in the test plan. Test plans are often set out as a table; an example for the calculation procedure is shown below.

Test	Purpose	Test data	Expected outcome	Actual outcome
to test the + calculation	to ensure that the + calculation works as expected	normal data 20 10 +	30	30
		abnormal data twenty ten +	error message	incorrect calculation
to test the – calculation	to ensure that the – calculation works as expected	normal data 20 10 –	10	30
		abnormal data twenty ten –	error message	incorrect calculation

Table 12.7 An example of a calculation procedure set out as a table

The results from this testing show that the error in the subtraction calculation has not been fixed and the routine is not trapping any abnormal data in the variables used by the calculation. These errors will need correcting and then the routine will need to be retested.

EXTENSION ACTIVITY 12C

In the programming language you have chosen to use, write the procedure for calculations and any other code necessary. Use and extend the test plan above to ensure that the calculation module works as it should.

Several types of test data need to be used during testing:

- **Normal test data** that is to be accepted by a program and is used to show that the program is working as expected.
- **Abnormal test data** that should be rejected by a program as it is unsuitable or could cause problems.
- **Extreme test data** that is on the limit of that accepted by a program; for example, when testing a validation rule such as number >= 12 AND number <= 32 the extreme test data would be 12 at the lower limit and 32 at the upper limit; both these values should be accepted.
- **Boundary test data** that is on the limit of that accepted by a program or data that is just outside the limit of that rejected by a program; for example, when testing a validation rule such as number >= 12 AND number <= 32 the boundary test data would be 12 and 11 at the lower limit and 32 and 33 at the upper limit; 12 and 32 should be accepted, 11 and 33 should be rejected.

ACTIVITY 12H

An algorithm is to be written to test whether a password is eight characters or more and 15 characters or less in length. The password must contain at least one digit, at least one capital letter and no characters other than letters or digits.

Devise a set of test data to be used to test the password checking algorithm.

Discuss any problems there may be in devising a complete set of test data.

EXTENSION ACTIVITY 12D

In the programming language you have chosen to use, write a procedure to check that the password conforms to the rules in Activity 12H. Use your test data from the previous activity to write a test plan and test that your procedure works as it should.

As the program is being developed the following types of testing are used:

- **White-box testing** is the detailed testing of how each procedure works. This involves testing the structure and logic of every path through a program module.
- **Black-box testing** tests a module's inputs and outputs.
- **Integration testing** is the testing of any separately written modules to ensure that they work together, during the testing phase of the program development lifecycle. If any of the modules have not been written yet, this can include **stub testing**, which makes use of dummy modules for testing purposes.

When the program has been completed, it is tested as a whole:

- **Alpha testing** is used first. The completed, or nearly completed, program is tested in-house by the development team.
- **Beta testing** is then used. The completed program is tested by a small group of users before it is generally released.
- **Acceptance testing** is then used for the completed program to prove to the customer that it works as required in the environment in which it will be used.

12.3.4 Program maintenance

Program maintenance is not like maintaining a piece of equipment by replacing worn out parts. Programs do not wear out, but they might not work correctly in unforeseen circumstances. Logic or run-time errors that require correction may occur from time to time, or users may want to use the program in a different way.

Program maintenance can usually be divided into three categories:

- **Corrective maintenance** is used to correct any errors that appear during use, for example trapping a run-time error that had been missed during testing.
- **Perfective maintenance** is used to improve the performance of a program during its use, for example improving the speed of response.
- **Adaptive maintenance** is used to alter a program so it can perform any new tasks required by the customer, for example working with voice commands as well as keyboard entry.

ACTIVITY 12I

Analyse the pseudocode algorithm for the volume and surface area of the sphere in Section 12.2.1 and identify at least three improvements you could make to the functionality of this algorithm.

ACTIVITY 12J

1 Explain, using examples, the difference between syntax and logic errors.

2 A programmer wants to test that a range check for values over 10 and under 100 works. Identify **three** types of test data that should be used. Provide an example of each type of test data and describe how the program should react to the data.

3 Identify **three** types of program maintenance and describe the role of each type of maintenance.

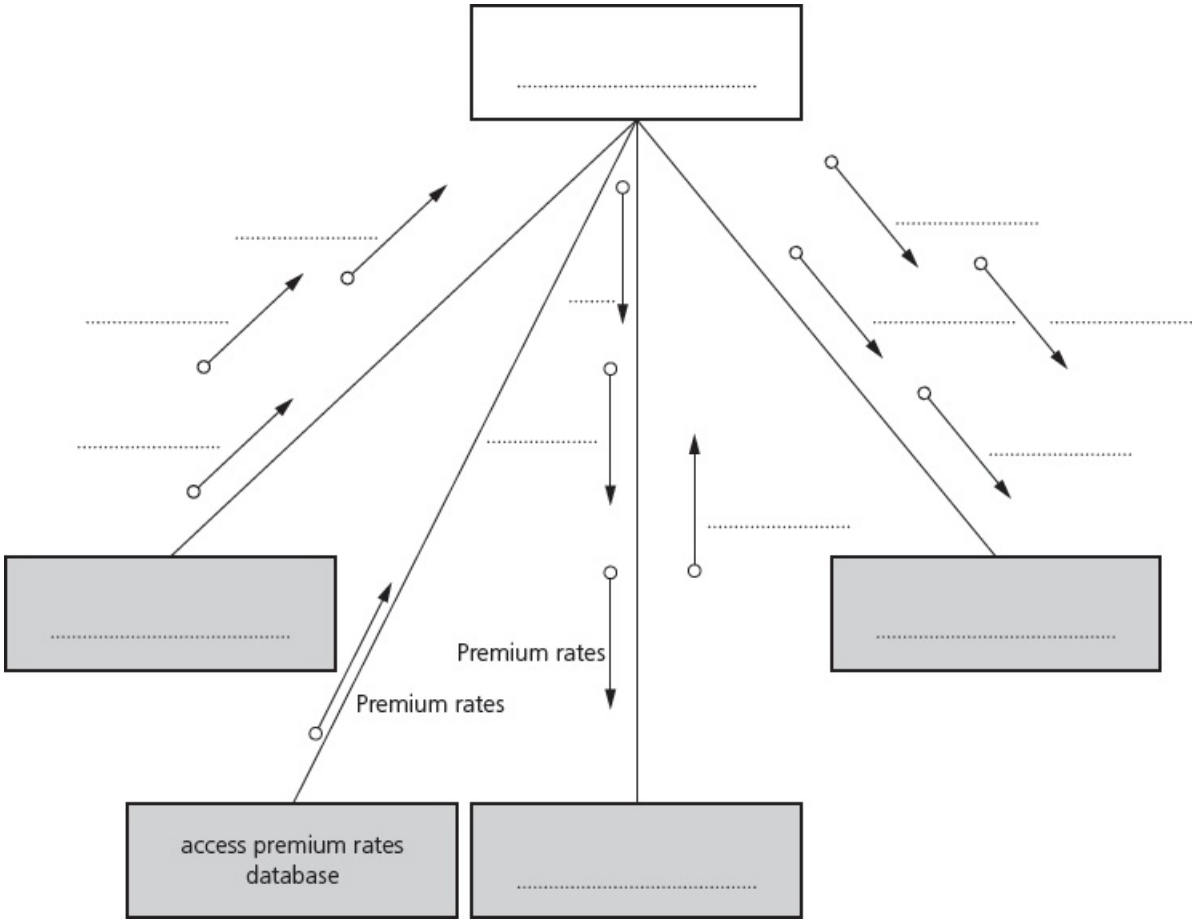
End of chapter questions

1 When the guarantee on a computer runs out, the owner can take out insurance to cover breakdown and repairs.
The price of the insurance is calculated from:

- the model of the computer
- the age of the computer
- the current insurance rates

Following an enquiry to the insurance company, the customer receives a quotation letter with the price of the insurance. A program is to be produced.

The structure chart below shows the modular design for this process.



a) Copy the chart above and, using the letters **A** to **D**, add the labelling to the chart boxes.

[2]

Modules	
A	Send quotation letter
B	Calculate price
C	Produce insurance quotation
D	Input computer details

b) Using the letters **E** to **J**, complete the labelling on the chart.

[4]

Some of these letters will be used more than once.

Data items	
E	CustomerName
F	CustomerEmail

G	Model
H	Age
I	PolicyCharge
J	PolicyNumber

Cambridge International AS & A Level Computer Science 9608 Paper 22 Q3 June 2015

2 A 1D array, Product, of type STRING is used to store information about a range of products in a shop. There are 100 elements in the array. Each element stores one data item.

The format of each data item is as follows:

<ProductID><ProductName>
– ProductID is a four-character string of numerals
– ProductName is a variable-length string

The following pseudocode is an initial attempt at defining a procedure, ArraySort, which will perform a bubble sort on Product. The array is to be sorted in ascending order of ProductID. Line numbers have been added for identification purposes only.

```
01  PROCEDURE SortArray
02    DECLARE Temp : CHAR
03    DECLARE FirstID, SecondID : INTEGER
04    FOR I ← 1 TO 100
05      FOR J ← 2 TO 99
06          FirstID ← MODULUS(LEFT(Product[J], 6))
07          SecondID ← MODULUS(LEFT(Product[J + 1],6))
08          IF FirstID > SecondID

09              THEN
10                Temp ← Product[I]
11                Product[I] Product[J + 1]
12                Product[J + 1] Temp
13     ENDFOR
14      ENDIF
15    ENDFOR
16  ENDPROCEDURE
```

The pseudocode contains a number of errors.

Copy and complete the following table to show:

- the line number of the error
- the error itself
- the correction that is required.

[8]

Note:

- If the same error occurs on more than one line, you should only refer to it ONCE.
- Lack of optimisation should **not** be regarded as an error.

Line number	**Error**	**Correction**
01	Wrong procedure name –"SortArray"	PROCEDURE ArraySort

Cambridge International AS & A Level Computer Science 9608 Paper 22 Q3 November 2017

3 A company creates two new websites, Site X and Site Y, for selling bicycles.

Various programs are to be written to process the sales data.

These programs will use data about daily sales made from Site X (using variable SalesX) and Site Y (using variable SalesY).

Data for the first 28 days is shown below.

	SalesDate	SalesX	SalesY
1	03/06/2015	0	1
2	04/06/2015	1	2
3	05/06/2015	3	8
4	06/06/2015	0	0
5	07/06/2015	4	6
6	08/06/2015	4	4
7	09/06/2015	5	9
8	10/06/2015	11	9
9	11/06/2015	4	1
...			
28	01/07/2015	14	8

a) Name the data structure to be used in a program for SalesX.

[2]

b) The programmer writes a program from the following pseudocode design.

```
x ← 0
FOR  DayNumber ← 1 to 7
     IF SalesX[DayNumber] + SalesY[DayNumber] >= 10
       THEN
          x ← x + 1
          OUTPUT SalesDate[DayNumber]
     ENDIF
ENDFOR
OUTPUT x
```

i) Trace the execution of this pseudocode by copying and completing this trace table.

[4]

x	DayNumber	OUTPUT
0		

ii) Describe, in detail, what this algorithm does.

[3]

Cambridge International AS & A Level Computer Science 9608 Paper 22 Q5 parts (a) and (b) June 2015

13 Data representation

In this chapter, you will learn about

- user-defined data types
- the definition and use of non-composite and composite data types
- the choice and design of an appropriate user-defined data type for a given problem
- methods of file organisation, such as serial, sequential and random
- methods of file access, such as sequential and direct access
- hashing algorithms
- binary floating-point real numbers
- converting binary floating-point real numbers into denary numbers
- converting denary numbers into binary floating-point real numbers
- the normalisation of binary floating-point numbers
- how underflow and overflow can occur
- how binary representation can lead to rounding errors.

13.1 User-defined data types

WHAT YOU SHOULD ALREADY KNOW

Try these two questions before you read the first part of this chapter.

1 Select an appropriate data type for each of the following.
   - **a)** A name
   - **b)** A student's mark
   - **c)** A recorded temperature
   - **d)** The start date for a job
   - **e)** Whether an item is sold or not

2 Write pseudocode to define a record structure to store the following data for an animal in a zoo.
   - Name
   - Species
   - Date of birth
   - Location
   - Whether the animal was born in the zoo or not
   - Notes

Programmers use specific data types that exactly match a program's requirements. They define their own data types based on primitive data types provided by a programming language, or data types that they have defined previously in a program. These are called **user-defined data types**. User-defined data types can be divided into non-composite and composite data types.

Key terms

User-defined data type – a data type based on an existing data type or other data types that have been defined by a programmer.

Non-composite data type – a data type that does not reference any other data types.

Enumerated data type – a non-composite data type defined by a given list of all possible values that has an implied order.

Pointer data type – a non-composite data type that uses the memory address of where the data is stored.

Set – a given list of unordered elements that can use set theory operations such as intersection and union.

13.1.1 Non-composite data types

A **non-composite data type** can be defined without referencing another data type. It can be a primitive type available in a programming language or a user-defined data type. Non-composite user-defined data types are usually used for a special purpose.

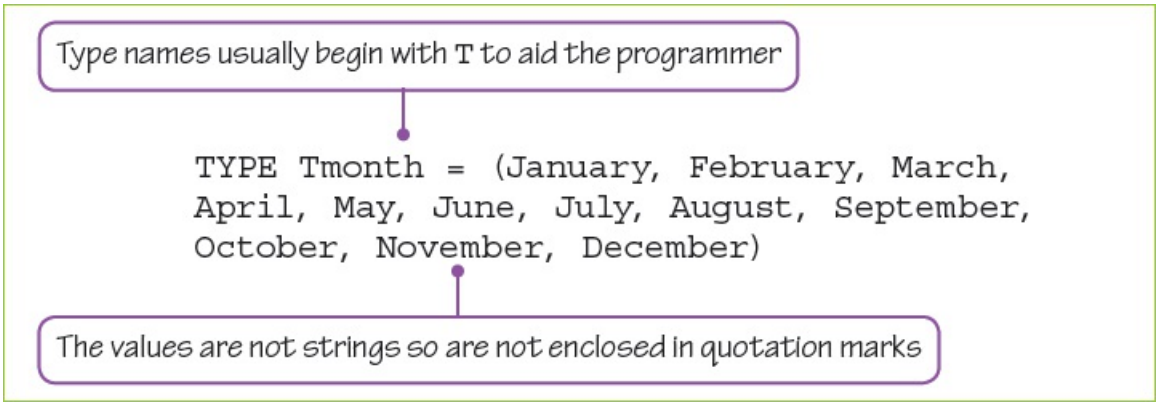
We will consider enumerated data types for lists of items and pointers to data in a computer's memory.

Enumerated data type

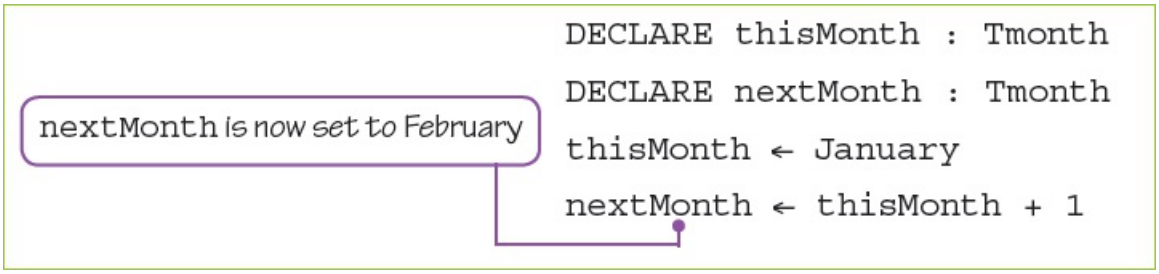
An enumerated data type contains no references to other data types when it is defined. In pseudocode, the type definition for an **enumerated data type** has this structure:

```
TYPE <identifier> = (value1, value2, value3, ... )
```

For example, a data type for months of the year could be defined as:



Then the variables thisMonth and nextMonth of type Tmonth could be defined as:



ACTIVITY 13A

Using pseudocode, declare an enumerated data type for the days of the week. Then declare two variables today and yesterday, assign a value of Wednesday to today, and write a suitable assignment statement for tomorrow.

Pointer data type

A **pointer data type** is used to reference a memory location. This data type needs to have information about the type of data that will be stored in the memory location. In pseudocode the type definition has the following structure, in which ^ shows that the type being declared is a pointer and <Typename> is the type of data to be found in the memory location, for example INTEGER or REAL, or any user-defined data type.

```
TYPE <pointer> = ^<Typename>
```

For example, a pointer for months of the year could be defined as follows:

```
TYPE TmonthPointer = ^Tmonth
DECLARE monthPointer : TmonthPointer
```

Tmonth is the data type in the memory location that this pointer can be used to point to

It could then be used as follows:

```
monthPointer ← ^thisMonth
```

If the contents of the memory location are required rather than the address of the memory location, then the pointer can be dereferenced. For example, myMonth can be set to the value stored at the address monthPointer is pointing to:

```
DECLARE myMonth : Tmonth
myMonth ← monthPointer^
```

monthPointer has been dereferenced

ACTIVITY 13B

Using pseudocode for the enumerated data type for days of the week, declare a suitable pointer to use. Set your pointer to point at today. Remember, you will need to set up the pointer data type and the pointer variable.

13.1.2 Composite data types

A data type that refers to any other data type in its type definition is a composite data type. In Chapter 10, the data type for record was introduced as a composite data type because it refers to other data types.

```
TYPE
   TbookRecord
        DECLARE title : STRING
        DECLARE author : STRING
        DECLARE publisher : STRING
        DECLARE noPages : STRING
        DECLARE fiction : STRING
   ENDTYPE
```

Other data types

Other composite data types include **sets** and classes.

Sets

A set is a given list of unordered elements that can use set theory operations such as intersection and union. A set data type includes the type of data in the set. In pseudocode, the type definition has this structure:

```
TYPE <set-identifier> = SET OF <Basetype>
```

The variable definition for a set includes the elements of the set.

```
DEFINE <identifier> (value1, value2, value3, ... ) :
<set-identifier>
```

A set of vowels could be declared as follows:

```
TYPE Sletter = SET OF CHAR
DEFINE vowel ('a', 'e', 'i', 'o', 'u') : letters
```

EXTENSION ACTIVITY 13A

Many programming languages offer a set data type. Find out about how set operations are implemented in the programming language you are using.

Classes

A class is a composite data type that includes variables of given data types and methods (code routines that can be run by an object in that class). An object is defined from a given class; several objects can be defined from the same class. Classes and objects will be considered in more depth in Chapter 20.

ACTIVITY 13C

1 Explain, using examples, the difference between composite and non-composite data types.

2 Explain why programmers need to define user-defined data types.
Use examples to illustrate your answers.

3 Choose an appropriate data type for the following situations.
Give the reason for your choice in each case.

a) A fixed number of colours to choose from.

b) Data about each house that an estate agent has for sale.

c) The addresses of integer data held in main memory.

13.2 File organisation and access

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you read the second part of this chapter.

1 Describe **three** different modes that files can be opened in.

2 Write pseudocode to carry out the following operations on a text file.

   a) Create a text file.

   b) Write several lines of text to the file.

   c) Read the text that you have written to the file.

   d) Append a line of text at the end of the file.

3 Write a program to test your pseudocode.

Key terms

Serial file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in the order they were added to the file.

Sequential file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in a given order.

Random file organisation – a method of file organisation in which records of data are physically stored in a file in any available position; the location of any record in the file is found by using a hashing algorithm on the key field of a record.

Hashing algorithm (file access) – a mathematical formula used to perform a calculation on the key field of the record; the result of the calculation gives the address where the record should be found.

File access – the method used to physically find a record in the file.

Sequential access – a method of file access in which records are searched one after another from the physical start of the file until the required record is found.

Direct access – a method of file access in which a record can be physically found in a file without physically reading other records.

13.2.1 File organisation and file access

File organisation

Computers are used to access vast amounts of data and to present it as useful information. Millions of people expect to be able to retrieve the information they need in a useful form when they ask for it. This information is all stored as data in files, everything from bank statements to movie collections. In order to be able to find data efficiently it needs to be organised. Data of all types is stored as records in files. These files can be organised using different methods.

Serial file organisation

The **serial file organisation** method physically stores records of data in a file, one after another, in the order they were added to the file.

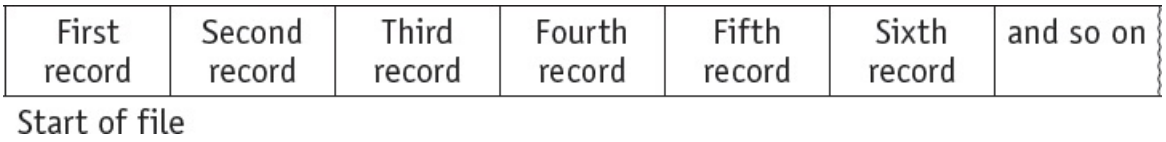


Figure 13.1

New records are appended to the end of the file. Serial file organisation is often used for temporary files storing transactions to be made to more permanent files. For example, storing customer meter readings for gas or electricity before they are used to send the bills to all customers. As each transaction is added to the file in the order of arrival, these records will be in chronological order.

Sequential file organisation

The **sequential file organisation** method physically stores records of data in a file, one after another, in a given order. The order is usually based on the key field of the records as this is a unique identifier. For example, a file could be used by a supplier to store customer records for gas or electricity in order to send regular bills to each customer. All records are stored in ascending customer number order, where the customer number is the key field that uniquely identifies each record.

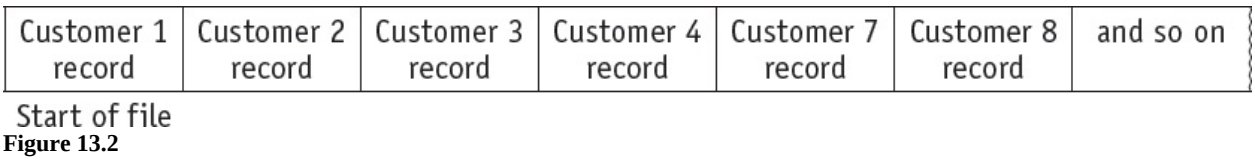


Figure 13.2

New records must be added to the file in the correct place; for example, if Customer 5 is added to the file, the structure becomes:

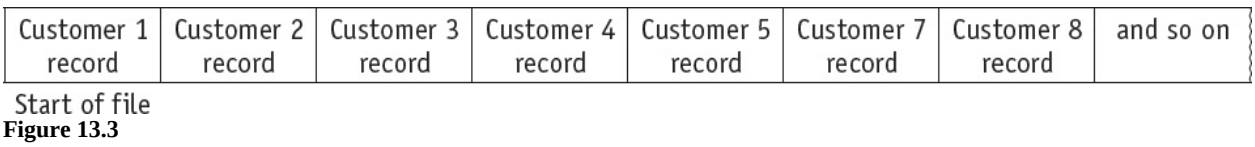


Figure 13.3

Random file organisation

The **random file organisation** method physically stores records of data in a file in any available position. The location of any record in the file is found by using a **hashing algorithm** (see Section 13.2.2) on the key field of a record.

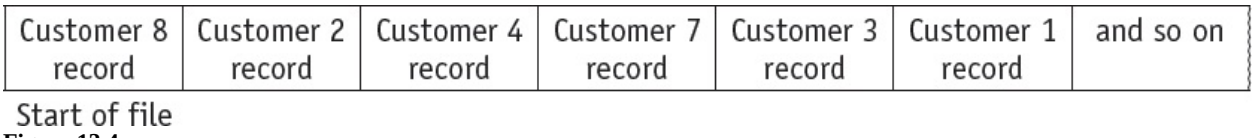


Figure 13.4

Records can be added at any empty position.

File access

There are different methods of **file access** (the method used to physically find a record in the file). We will consider two of them: **sequential access** and **direct access**.

Sequential access

The sequential access method searches for records one after another from the physical start of the file until the required record is found, or a new record can be added to the file. This method is used for serial and sequential files.

For a **serial** file, if a particular record is being searched for, every record needs to be checked until that record is found or the whole file has been searched and that record has not been found. Any new records are appended to the end of the file.

For a sequential file, if a particular record is being searched for, every record needs to be checked until the record is found or the key field of the current record being checked is greater than the key field of the record being searched for. The rest of the file does not need to be searched as the records are sorted on ascending key field values. Any new records to be stored are inserted in the correct place in the file. For example, if the record for Customer 6 was requested, each record would be read from the file until Customer 7 was reached. Then it would be assumed that the record for Customer 6 was not stored in the file.

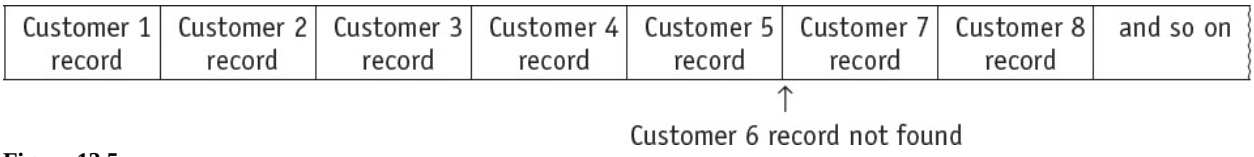


Figure 13.5

Sequential access is efficient when every record in the file needs to be processed, for example, a monthly billing or payroll system. These files have a high hit rate during the processing as nearly every record is used when the program is run.

Direct access

The direct access method can physically find a record in a file without other records being physically read. Both sequential and random files can use direct access. This allows specific records to be found more quickly than using sequential access.

Direct access is required when an individual record from a file needs to be processed. For example, when a single customer record needs to be updated when the customer's phone number is changed. Here, the file being processed has a low hit rate as only one of the records in the file is used.

For a sequential file, an index of all the key fields is kept and used to look up the address of the file location where a given record is stored. For large files, searching the index takes less time than searching the whole file.

For a random access file, a hashing algorithm is used on the key field to calculate the address of the file location where a given record is stored.

13.2.2 Hashing algorithms

In the context of storing and accessing data in a file, a hashing algorithm is a mathematical formula used to perform a calculation on the key field of the record. The result of the calculation gives the address where the record should be found. More complex hashing algorithms are used in the encryption of data.

Here is an example of a simple hashing algorithm:

If a file has space for 2000 records and the key field can take any values between 1 and 9999, then the hashing algorithm could use the remainder when the value of key field is divided by 2000, together with the start address of the file and the size of the space allocated to each record.

In the simplest case, where the start address is 0 and each record is stored in one location.

To store a record identified by a key field with value 3024, the hashing algorithm would give address 1024 as the location to store the record.

Key field	Remainder	Address
3024	1024	1024 = 0 + 1 * 1024

Table 13.1

Unfortunately, storing another record with a key field 5024 would result in trying to use the same file location and a collision would occur.

Key field	Same remainder	Same address
5024	1024	1024 = 0 + 1 * 1024

Table 13.2

This often happens with hashing algorithms for direct access to records in a file.

There are two ways of dealing with this:

1. An open hash where the record is stored in the next free space.
2. A closed hash where an overflow area is set up and the record is stored in the next free space in the overflow area.

When reading a record from a file using direct access, the address of the location to read from is calculated using the hashing algorithm and the key field of the record stored there is read. But, before using that record, the key field must be checked against the original key field to ensure that they match. If the key fields do not match, then the following records need to be read until a match is found (open hash) or the overflow area needs to be searched for a match (closed hash).

ACTIVITY 13D

A file of records is stored at address 500. Each record takes up five locations and there is space for 1000 records. The key field for each record can take the value 1 to 9999.

The hashing algorithm used to calculate the address of each record is the remainder when the

value of key field is divided by 1000 together with the start address of the file and the size of the space allocated to each record.

Calculate the address to store the record with key field 9354.

If this location has already been used to store a record and an open hash is used, what is the address of the next location to be checked?

Hashing algorithms can also be used to calculate addresses from names. For example, adding up the ASCII values for every character in a name and dividing this by the number of locations in the file could be used as the basis for a hashing algorithm.

EXTENSION ACTIVITY 13B

Write a program to

- find the ASCII value for each character in a name of up to 10 characters
- add the values together
- divide by 1000 and find the remainder
- multiply this value by 20 and add it to 2000
- display the result.

If this program simulates a hashing algorithm for a file, what is the start address of the file and the size of each record?

ACTIVITY 13E

1 Explain, using examples, the difference between serial and sequential files.

2 Explain the process of direct access to a record in a file using a hashing algorithm.

3 Choose an appropriate file type for the following situations.
Give the reason for your choice in each case.

a) Borrowing books from a library.

b) Providing an annual tax statement for employees at the end of the year.

c) Recording daily rainfall readings at a remote weather station to be collected every month.

13.3 Floating-point numbers, representation and manipulation

WHAT YOU SHOULD ALREADY KNOW

Try these six questions before you read the third part of this chapter.

1 Convert these denary numbers into binary.

a) +48

b) +122

c) −100

d) −55

e) −2

2 Convert these binary numbers into denary.

a) 00110011

b) 01111110

c) 10110011

d) 11110010

e) 11111111

3 Use two's complement to find the negative values of these binary numbers.

a) 00110100

b) 00011101

c) 01001100

d) 00111111

e) 01111110

4 Carry out these binary additions, showing all your working.

a) 00110001 + 00011110

b) 01000001 + 00111111

c) 00111100 + 01000101

d) 01111101 + 01011100

e) 11101100 + 01100000

f) 10001111 + 10011111

g) 01000101 + 10111100

h) 01111110 + 01111110

i) 11111100 + 11100011

j) 11001100 + 00011111

5 Write the following numbers in standard form

a) 123 000 000

b) 2 505 000 000 000 000

c) –1200

d) 0.000000002341

e) −0.0000124005

6 a) Standard form is sometimes used to put denary improper fractions into proper fractions. For example, $\frac{14}{5}$ can be written as $\frac{1.4}{5} \times 10^1$, and $\frac{112}{3}$ can be written as $\frac{1.12}{3} \times 10^2$.
Change the following improper fractions into proper fractions using this format:

i) $\frac{21}{5}$

ii) $\frac{117}{4}$

iii) $\frac{558}{20}$

b) When using binary, we can convert improper binary fractions into proper fractions. For example, $\frac{7}{2}$ can be written as $\frac{7}{8} \times 4$ (where $4 \equiv 2^2$), and $\frac{23}{2}$ can be written as $\frac{23}{32} \times 16$ (where $16 \equiv 2^4$).
Change the following improper binary fractions into proper binary fractions using this format.

i) $\frac{11}{2}$

ii) $\frac{41}{4}$

iii) $\frac{52}{4}$

Key terms

Mantissa – the fractional part of a floating point number.

Exponent – the power of 2 that the mantissa (fractional part) is raised to in a floating-point number.

Binary floating-point number – a binary number written in the form $M \times 2^E$ (where M is the mantissa and E is the exponent).

Normalisation (floating-point) – a method to improve the precision of binary floating-point numbers; positive numbers should be in the format 0.1 and negative numbers in the format 1.0.

Overflow – the result of carrying out a calculation which produces a value too large for the computer's allocated word size.

Underflow – the result of carrying out a calculation which produces a value too small for the computer's allocated word size.

13.3.1 Floating-point number representation

In Chapter 1, we learnt about how binary numbers can be stored in a fixed-point representation. The magnitude of the numbers stored depends on the number of bits used. For example, 8 bits allowed a range of −128 to +127 (using two's complement representation) whereas 16 bits increased this range to −16 384 to +16 383.

However, this type of representation limits the range of numbers and does not allow for fractional values. To increase the range, and to allow for fractions, we can look to the method used in the denary number system.

For example, 312 110 000 000 000 000 000 000 can be written as 3.1211×10^{23} using scientific notation. If we adopt this system in binary, we get:

$$\mathbf{M \times 2^{E}}$$

M is the **mantissa** and E is the **exponent**.

This is known as **binary floating-point representation**.

In our examples, we will assume a computer is using 8 bits to represent the mantissa and 8 bits to store the exponent (a binary point is assumed to exist between the first and second bits of the mantissa). Again, using denary as our example, a number such as 0.31211×10^{24} means:

	$\frac{1}{10}$	$\frac{1}{100}$	$\frac{1}{1000}$	$\frac{1}{10\,000}$	$\frac{1}{100\,000}$		10	1
•	3	1	2	1	1	×	2	4
	mantissa values						exponent	

Figure 13.6

We thus get the binary floating-point equivalent (using 8 bits for the mantissa and 8 bits for the exponent with the assumed binary point between –1 and $\frac{1}{2}$ in the mantissa):

–1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$		–128	64	32	16	8	4	2	1

Figure 13.7

Converting binary floating-point numbers into denary

Example 13.1

Convert this binary floating-point number into denary.

−1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
0		1	0	1	1	0	1	0

−128	64	32	16	8	4	2	1
0	0	0	0	0	1	0	0

Solution

Method 1

Add up the mantissa values where a 1 bit appears:

$$M = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{64} \equiv \frac{32}{64} + \frac{8}{64} + \frac{4}{64} + \frac{1}{64} = \frac{45}{64}$$

Add up the exponent values where a 1 bit appears:

$E = 4$

Use $M \times 2^E$:

$$\frac{45}{64} \times 2^4 = \frac{45}{64} \times 16$$
$$= 0.703125 \times 16$$
$$= 11.25 \text{ (the denary value)}$$

Method 2

Write the mantissa as 0.1011010.

The exponent is 4, so move the binary point four places to the right (to match the exponent value). This gives 01011.010.

whole number part						**fraction part**		
−16	8	4	2	1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$
0	1	0	1	1		0	1	0

This gives 11.25 (the same result as method 1).

Example 13.2

Convert this binary floating-point number into denary.

−1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
0		0	1	0	1	0	0	0

−128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	1

Solution

Method 1

Add up the mantissa values where a 1 bit appears:

$$M = \frac{1}{4} + \frac{1}{16} \equiv \frac{4}{16} + \frac{1}{16} = \frac{5}{16}$$

Add up the exponent values where a 1 bit appears:

$$E = 2 + 1 = 3$$

Use $M \times 2^E$:

$$\frac{5}{16} \times 2^3 = \frac{5}{16} \times 8$$
$$= 0.3125 \times 8$$
$$= 2.5 \text{ (the denary value)}$$

Method 2

Write the mantissa as 0.0101000.

The exponent is 3, so move the binary point three places to the right (to match the exponent value). This gives 0010.1000.

whole number part					fraction part			
−8	4	2	1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
0	0	1	0		1	0	0	0

This gives 2.5 (the same result as method 1).

Now we shall consider negative values.

Example 13.3

Convert this binary floating-point number into denary.

−1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
1		1	0	0	1	1	0	0

−128	64	32	16	8	4	2	1
0	0	0	0	1	1	0	0

Solution

Method 1

Add up the mantissa values where a 1 bit appears:

$$M = -1 + \frac{1}{2} + \frac{1}{16} + \frac{1}{32} \equiv -1 + \frac{16}{32} + \frac{2}{32} + \frac{1}{32} \equiv -1 + \frac{19}{32} \equiv -\frac{32}{32} + \frac{19}{32} = -\frac{13}{32}$$

Add up the exponent values where a 1 bit appears:

$E = 8 + 4 = 12$

Use $M \times 2^E$:

$$-\frac{13}{32} \times 2^{12} = -\frac{13}{32} \times 4096$$
$$= -0.40625 \times 4096$$
$$= -1664 \text{ (the denary value)}$$

Method 2

Since the mantissa is negative, first convert the value using two's complement.

So, write the mantissa as 00110011 + 1 = 00110100.

This gives −0.0110100.

The exponent is 12, so move the binary point 12 places to the right (to match the exponent value). This gives −0011010000000.0.

whole number part														fraction part
−4096	2048	1024	512	256	128	64	32	6	8	4	2	1	•	$\frac{1}{2}$
0	0	1	1	0	1	0	0	0	0	0	0	0		0

This gives −(1024 + 512 + 128) = −1664 (the same result as method 1).

Example 13.4

Convert this binary floating-point number into denary.

−1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
1	•	1	0	0	1	1	0	0

−128	64	32	16	8	4	2	1
1	1	1	1	1	1	0	0

Solution

Method 1

Add up the mantissa values where a 1 bit appears:

$$M = -1 + \frac{1}{2} + \frac{1}{16} + \frac{1}{32} \equiv -1 + \frac{16}{32} + \frac{2}{32} + \frac{1}{32} \equiv -1 + \frac{19}{32} \equiv -\frac{32}{32} + \frac{19}{32} = -\frac{13}{32}$$

Add up the exponent values where a 1 bit appears:

$E = -128 + 64 + 32 + 16 + 8 + 4 = -4$

Use $M \times 2^E$:

$$-\frac{13}{32} \times 2^{-4} = -\frac{13}{32} \times 0.0625$$
$$= -0.40625 \times 0.0625$$
$$= -0.025390625 \text{ (the denary value)}$$

Method 2

Since the mantissa is negative, first convert the value using two's complement.

So, write the mantissa as 00110011 + 1 = 00110100.

This gives −0.0110100.

The exponent is −4, so move the binary point four places to the **left** (to match the negative exponent value). This gives −0.00000110100.

whole number part		fraction part										
−1	•	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$	$\frac{1}{2048}$
0		0	0	0	0	0	1	1	0	1	0	0

This gives $-\left(\frac{1}{64}+\frac{1}{128}+\frac{1}{512}\right)=-\frac{13}{512}$

$= -0.025390625$ (the same result as method 1).

ACTIVITY 13F

Convert these binary floating-point numbers into denary numbers (the mantissa is 8 bits and the exponent is 8 bits in all cases).

a)	0 1 0 0 1 1 1 0	0 0 0 0 0 1 0 1
b)	0 0 1 0 1 0 0 1	0 0 0 0 0 1 1 1
c)	0 1 1 1 0 0 0 0	1 1 1 1 1 0 1 1
d)	0 0 0 1 1 1 1 0	1 1 1 1 1 1 0 0
e)	0 1 1 1 0 0 0 0	0 0 0 0 0 0 1 1

f)

1	0	0	1	1	0	0	0

0	0	0	0	0	0	1	0

g)

1	1	1	1	0	1	0	0

0	0	0	0	0	1	0	0

h)

1	0	1	1	0	0	0	0

0	0	0	0	0	1	0	1

i)

1	0	1	1	0	0	0	0

1	1	1	1	1	1	0	1

j)

1	1	1	0	0	0	0	0

1	1	1	1	1	0	1	0

Converting denary numbers into binary floating-point numbers

Example 13.5

Convert +4.5 into a binary floating-point number.

Solution

Method 1

Turn the number into an improper fraction:

$4.5 = \frac{9}{2}$

The fraction needs to be < 1, which means the numerator < denominator; we can do this by dividing successively by 2 until the denominator > numerator.

$\frac{9}{2} \rightarrow \frac{9}{4} \rightarrow \frac{9}{8} \rightarrow \frac{9}{16}$ The numerator (9) is now < denominator (16).

So, $\frac{9}{2}$ can be written as $\frac{9}{16} \times 8 \equiv \frac{9}{16} \times 2^3$ and the original fraction is now written in the correct format, $M \times 2^E$.

$\frac{9}{16} = \frac{1}{2} + \frac{1}{16}$, which gives the mantissa as 0.1001.

And the exponent is 2^3, which is represented as 11 in our binary floating point format.

Filling in the gaps with 0s gives:

0	1	0	0	1	0	0	0

0	0	0	0	0	0	1	1

Method 2

4 = 0100 and .5 = .1 which gives: 0100.1

Now move the binary point as far as possible until 0.1 can be formed:

0100.1 becomes 0.1001 by moving the binary point **three** places left.

So, the exponent must **increase** by three:

0.1001×11

Filling in the gaps with 0s gives:

4 = 0100 and .5 = .1 which gives: 0100.1

This is the same result as method 1.

Example 13.6

Convert +0.171875 into a binary floating-point number.

Solution

Method 1

Remember, the fraction needs to be < 1, which means the numerator < denominator.

0.1001×11, so this fraction is already in the correct form.

0	1	0	0	1	0	0	0

0	0	0	0	0	0	1	1

, which gives the mantissa as 0.0010110 and exponent as 0.

Filling in the gaps with 0s gives:

$$\frac{11}{64} = \frac{1}{8} + \frac{1}{32} + \frac{1}{64}$$

Method 2

0 = 0 and .171875 = .001011

0	0	0	1	0	1	1	0

0	0	0	0	0	0	0	0

, which gives: 0.001011.

Now move the binary point as far as possible until 0.1 can be formed:

0.001011 becomes 0.1011 by moving the binary point **two** places **right**.

So, the exponent must **increase** by two (in other words, −2).

The number 2, using eight bits is 00000010.

Applying two's complement gives us 11111101 + 1 = 11111110

Thus, we have:

0.1011×11111110

Filling in the gaps with 0s gives:

0.1011×11111110

This is exactly the same result as method 1.

EXTENSION ACTIVITY 13C

Show why:

0	1	0	1	1	0	0	0		1	1	1	1	1	1	1	0

is the same as:

0	0	0	1	0	1	1	0		0	0	0	0	0	0	0	0

Example 13.7

Convert −10.375 into a binary floating-point number.

Solution

Method 1

Turn the number into an improper fraction:

0	1	0	1	1	0	0	0		1	1	1	1	1	1	1	0

Now make the fraction < 1.

$0.375 \equiv \frac{3}{8}$, so $-10.375 = -\frac{83}{8}$

Now make the fraction < 1.

$$-\frac{83}{8} \equiv -\frac{83}{128} \times 16 \equiv -\frac{83}{128} \times 2^4$$

$-\frac{83}{128} = -1 + \frac{45}{128}$, which gives the mantissa as 1.0101101

$\left(\frac{45}{128} = \frac{1}{4} + \frac{1}{16} + \frac{1}{32} + \frac{1}{128}\right)$.

1	0	1	0	1	1	0	1		0	0	0	0	0	1	0	0

, which gives the mantissa as 1.0101101

$\frac{1}{4} + \frac{1}{8}$

And the exponent is 2^4, which is represented as 100 in our binary floating point format.

Filling in the gaps with 0s gives:

−01010.011

Method 2

−10 = −01010 and $\frac{1}{4}+\frac{1}{8}\equiv$.375 = .011, which gives: −01010.011.

Using two's complement (on 01010011) we get: 10101100 + 1 = 10101101 (= 10101.101).

Now move the binary point as far as possible until 1.0 can be formed:

10101.101 becomes 1.0101101 by moving the binary point **four** places **left**.

So, the exponent must **increase** by four.

1.0101101 × 100

Filling in the gaps with 0s gives:

1.0101101 × 100

This is the same result as method 1.

ACTIVITY 13G

1 Write these into binary floating-point format using an 8-bit mantissa and 8-bit exponent.

a)

1	0	1	0	1	1	0	1		0	0	0	0	0	1	0	0

b) $\frac{11}{32}\times 2^{7}$

c) $\frac{19}{64}\times 2^{3}$

d) $\frac{21}{128}\times 2^{-5}$

e) $\frac{15}{16}\times 2^{-3}$

f) $\frac{21}{8}\times 2^{3}$

g) $-\frac{11}{64}\times 2^{4}$

h) $-\frac{9}{16}\times 2^{-1}$

i) $-\frac{5}{16}\times 2^{5}$

j) $-\frac{1}{4}\times 2^{-6}$

2 Convert these denary numbers into binary floating-point numbers using an 8-bit mantissa and 8-bit exponent.

a) +3.5

b) 0.3125

c) 15.375

d) $-\frac{5}{8}\times 2^{-2}$

e) 9.125

f) $\frac{41}{64}$

g) −3.5

h) −10.25

i) $-\frac{15}{32}$

j) $-1.046875\left(\equiv -1\frac{3}{64}\right)$

Potential rounding errors and approximations

All the problems up to this point have involved fractions which are linked somehow to the number 2 (such as $-3\frac{11}{32}$). We will now consider numbers which can only be represented as an approximate value (the accuracy of which will depend on the number of bits that make up the mantissa).

The representation of the following example (denary number 5.88), using an 8-bit mantissa and 8-bit exponent, will lead to an inevitable rounding error since it is impossible to convert the denary number into an exact binary equivalent.

This error could be reduced by increasing the size of the mantissa; for example, a 16-bit mantissa would allow the number 5.88 to be represented as 5.875, which is a better approximation.

We will consider how to represent the denary number 5.88 using an 8-bit mantissa and 8-bit exponent.

To convert this into binary, we will use a method similar to that used in Chapter 1.

.88 × 2 = 1.76 so we will use the 1 value to give 0.1

.76 × 2 = 1.52 so we will use the 1 value to give 0.11

.52 × 2 = 1.04 so we will use the 1 value to give 0.111

.04 × 2 = 0.08 so we will use the 0 value to give 0.1110

.08 × 2 = 0.16 so we will use the 0 value to give 0.11100

.16 × 2 = 0.32 so we will use the 0 value to give 0.111000

.32 × 2 = 0.64 so we will use the 0 value to give 0.1110000

.64 × 2 = 1.28 so we will use the 1 value to give 0.11100001

We have to stop here since our system uses a maximum of 8 bits. Now the value of 5 (in binary) is 0101; this gives:

5.88 ≡ 0101.11100001

Moving the binary point as far to the left as possible gives:

0.1011100×2^3 (2^3 since the binary point was moved three places).

5.88 ≡ 0101.11100001

So, 5.88 is stored as 5.75 in our floating-point system.

EXTENSION ACTIVITY 13D

Using 8-bit mantissa and exponent, show how the following numbers would be approximated

a) 1.63
b) 8.13
c) 12.32
d) 5.90
e) 7.40.

Now consider this set of numbers.

Thus, we get 0.1011100 00000011
(mantissa) (exponent)

$$= \frac{23}{32} \times 2^3 = \frac{23}{4} = 5.75$$

Figure 13.8

As shown, there are several ways of representing the number 2. Using this sequence, if we kept shifting to the right, we would end up with:

0.0000000 00001001	= 2

Figure 13.9

This could lead to problems. To overcome this, we use a method called **normalisation**.

With this method, for a **positive number**, the mantissa must start with 0.1 (as in our first representation of 2 above). The bits in the mantissa are shifted to the left until we arrive at 0.1; for each shift left, the exponent is reduced by 1. Look at the examples above to see how this works (starting with 0.0001000 we shift the bits 3 places to the left to get to 0.100000 and we reduce the exponent by 3 to now give 00000010, so we end up with the first representation!).

For a **negative number** the mantissa must start with 1.0. The bits in the mantissa are shifted until we arrive at 1.0; again, the exponent must be changed to reflect the number of shifts.

Example 13.8

Normalise 0.0011100 00000101 $\left(\equiv \frac{7}{32} \times 2^5 = 7\right)$.

Solution

Shift the bits left to get 0.1110000.

Since the bits were shifted two places left, the exponent must reduce by two to give 00000011.

This gives 0.1110000 00000011, which is now normalised.

Note: 0.1110000 00000011 $0.1110000\ \ 00000011 \equiv \frac{7}{8} \times 2^3 = 7$, so the normalised form still represents the correct value.

Example 13.9

Normalise 1.1101100 00001010 $\left(\equiv -\frac{5}{32} \times 2^{10} = -160\right)$

Solution

Shift the bits left until to get 1.0110000.

Since the bits were shifted two places left, the exponent must reduce by two to give 00001000.

This gives 1.011000 00001000, which is now normalised.

Note: 1.011000 00001000 $1.011000\ \ 00001000 \equiv -\frac{5}{8} \times 2^8 = -5 \times 32 = -160$, so the normalised form still represents the same value.

ACTIVITY 13H

Normalise these binary floating-point numbers.

a) 0. 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0
b) 0. 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1
c) 0. 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0
d) 0. 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1
e) 0. 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
f) 1. 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0
g) 1. 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0
h) 1. 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1
i) 0. 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
j) 1. 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0

Precision versus range

The following values relate to an 8-bit mantissa and an 8-bit exponent (using two's complement):

The **maximum positive number** which can be stored is:

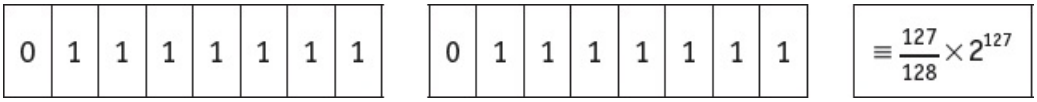


Figure 13.10

The **smallest positive number** which can be stored is:

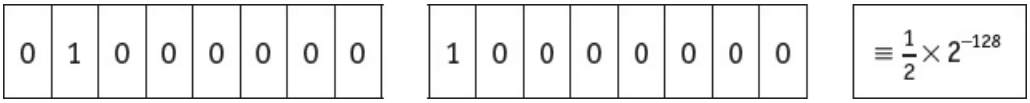


Figure 13.11

The **smallest magnitude negative number** which can be stored is:

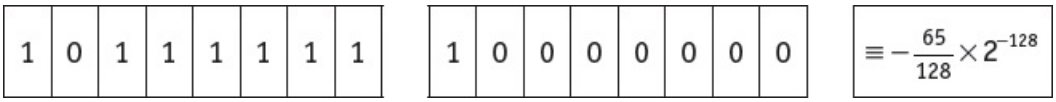


Figure 13.12

The **largest magnitude negative number** which can be stored is:

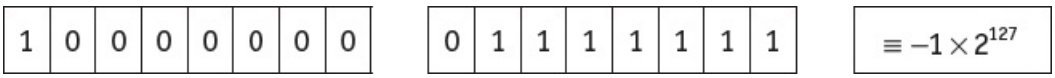


Figure 13.13

- The accuracy of a number can be increased by increasing the number of bits used in the mantissa.
- The range of numbers can be increased by increasing the number of bits used in the exponent.
- Accuracy and range will always be a trade-off between mantissa and exponent size.

Consider the following three cases.

1

Figure 13.14

The mantissa is 12 bits and the exponent is 4 bits.

This gives a largest positive value of $\frac{2047}{2048} \times 2^{7}$; which gives high accuracy but small range.

2

Figure 13.15

The mantissa is 8 bits and the exponent is 8 bits.

This gives a largest positive value of $\frac{127}{128} \times 2^{127}$, which gives reduced accuracy but increased

range.

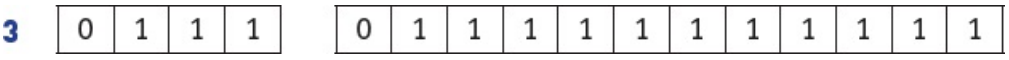


Figure 13.16

The mantissa is 4 bits and the exponent is 12 bits.

This gives a largest possible value of $\frac{7}{8} \times 2^{2047}$, which gives poor accuracy but extremely high range.

Floating-point problems

The storage of certain numbers is an approximation, due to limitations in the size of the mantissa. This problem can be minimised when using programming languages that allow for double precision and quadruple precision.

EXTENSION ACTIVITY 13E

Look at this coding, written in pseudocode.

```
number ← 0.0
FOR loop ← 0 TO 50
    number ← number + 0.1
    OUTPUT number
ENDFOR
```

a) When running this program the expected output would be 0.1, 0.2, 0.3, …, 5.0.
Explain why the output from this program gave values such as 0.399999 rather than 0.4.

b) Run the program on your own computer using a language such as Pascal.
Does it confirm the above statement?

c) Discuss ways of overcoming the error(s) described in your answer to part a).

There are additional problems:

- If a calculation produces a number which exceeds the maximum possible value that can be stored in the mantissa and exponent, an **overflow** error will be produced. This could occur when trying to divide by a very small number or even 0.
- When dividing by a very large number this can lead to a result which is less than the smallest number that can be stored. This would lead to an **underflow** error.
- One of the issues of using normalised binary floating-point numbers is the inability to store the number zero. This is because the mantissa must be 0.1 or 1.0 which does not allow for a zero value.

EXTENSION ACTIVITY 13F

Find out how computer systems deal with the value 0 when using normalised binary floating-point numbers.

ACTIVITY 13I

1 What is the largest positive and smallest magnitude number which can be stored in a computer using 10-bit mantissa and 6-bit exponent.

2 A computer uses 32 bits to store the mantissa and exponent.
Discuss the precision and range of numbers which can be stored in this computer.

3 **a)** A calculation carried out on a computer produced the result 1.21×10^{100}
The computer's largest possible value which can be stored is 10^{99}.
Discuss the problems this result would cause.

b) A calculation involving $\frac{x}{y}$ is being carried out on a computer.
One of the potential values of y is 0.
Discuss the problems this might cause for the computer.

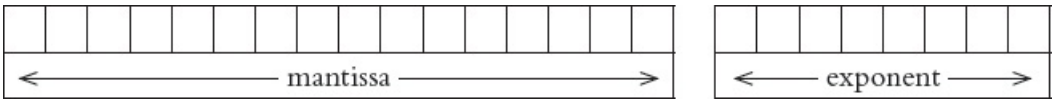
4 A computer uses a 10-bit mantissa and a 6-bit exponent.
What approximate values would be stored for the following numbers?

a) 2.88

b) −5.38

End of chapter questions

1 A computer holds binary floating-point numbers in two's complement form with the binary point immediately after the left-most bit.
A 24-bit word is used as follows:



a) Three words are held in floating-point representations:

Ⓐ 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 | 1 1 1 1 1 1 1 1

Ⓑ 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 1 1

Ⓒ 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 1 0 1

i) State the values being represented by A, B and C.

[3]

ii) Identify the value that is not normalised.

[1]

iii) Explain why it is normal for floating-point numbers to be normalised.

[1]

b) Comment on the accuracy and range of numbers stored in this computer.

[3]

c) Discuss the problems of representing the number zero in normalised floating-point format.

[2]

2 A computer uses 12 bits for the mantissa and 6 bits for the exponent.

a) Convert these binary floating-point numbers into denary.

i)

0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1

[3]

ii)

1	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0	0

[3]

b) Convert these denary numbers into binary floating-point numbers.

i) +4.75

[3]

ii) −8.375

[3]

3 In a particular computer system, real numbers are stored using floating-point representation with:

– 8 bits for the mantissa

– 8 bits for the exponent

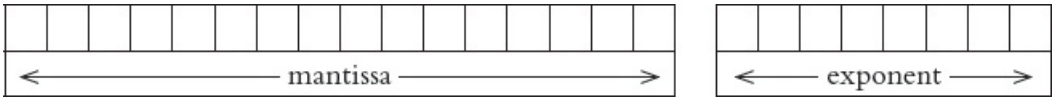
– two's complement form for both mantissa and exponent.

a) Calculate the floating-point representation of +3.5 in this system.
Show your working.

[3]

b) Calculate the floating-point representation of −3.5 in this system.
Show your working.

[3]



Cambridge International AS & A Level Computer Science 9608
Paper 32 Q1 parts (a) and (b) November 2016

4 a) Using the pseudocode declarations below, identify

i) an enumerated data type

[1]

ii) a composite data type

[1]

iii) a non-composite data type

[1]

iv) a user-defined data type.

[1]

```
TYPE Tseason = (Spring, Summer, Autumn, Winter)
TYPE
TJournalRecord
    DECLARE title : STRING
    DECLARE author : STRING
    DECLARE publisher : STRING
    DECLARE noPages : INTEGER
    DECLARE season : Tseason
ENDTYPE
```

b) Write pseudocode to declare a variable Journal of type TJournalRecord and assign the following values to the variable.

[3]

Title – Spring Flowers

Author – H Williams

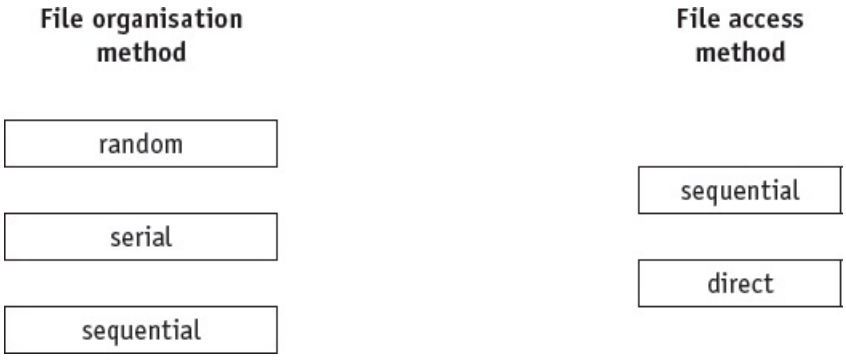
Publisher – XYZ Press

Number of pages – 40

Season – Spring

5 a) Three file organisation methods and two file access methods are shown below. Copy the diagram below and connect each file organisation method to its appropriate file access method(s).

[4]



b) An energy company supplies electricity to a large number of customers. Each customer has a meter that records the amount of electricity used. Customers submit meter readings

using their online account.
The company's computer system stores data about its customers.
This data includes:
– account number
– personal data (name, address, telephone number)
– meter readings
– username and encrypted password.
The computer system uses three files:

File	Content	Use
A	Account number and meter readings for the current month.	Each time a customer submits their reading, a new record is added to the file.
B	Customer's personal data.	At the end of the month to create a statement that shows the electricity supplied and the total cost.
C	Usernames and encrypted passwords.	When customers log in to their accounts to submit meter readings.

For each of the files A, B and C, state an appropriate file organisation method for the use given in the table.
All three file organisation methods must be different.

Justify your choice.

[9]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q4 June 2017

14 Communication and internet technologies

In this chapter, you will learn about

- the need for protocols during communication
- the implementation of protocols such as a stack
- TCP/IP protocols, including the four layers (Application, Transport, Internet and Link), the purpose and function of the four layers, and application when a message is sent from one host to another on the internet
- HTTP, FTP, POP3/4, IMAP, SMTP and BitTorrent protocols (such as BitTorrent provides peer-to-peer file sharing)
- circuit switching (including benefits and drawbacks)
- the benefits and drawbacks of packet switching
- the function of a router in packet switching
- the use of packet switching to pass messages across the network (including the internet).

14.1 Protocols

WHAT YOU SHOULD ALREADY KNOW

In Chapter 2, you learned about networks. Try these five questions before you read the first part of this chapter.

1 a) Explain the terms *IP* and *TCP*.

b) What are *protocols* and why are they used?

2 a) Explain how peer-to-peer networks operate.

b) What are the pros and cons of peer-to-peer networks?

3 a) Explain the term *stack*.

b) Explain the term *queue*.

c) Give examples of the use of a stack and the use of a queue.

4 a) Explain what is meant by *Ethernet*.

b) What is meant by *IP conflicts* when using Ethernet? Explain how they can be overcome.

5 a) What is a *DNS*?

b) What is meant by *HTTP*?

c) Explain the role of *status flags*.

Key terms

Protocol – a set of rules governing communication across a network: the rules are agreed by both sender and recipient.

HTTP – hypertext transfer protocol.

Packet – a message/data is split up into smaller groups of bits for transmission over a network.

Segment (transport layer) – this is a unit of data (packet) associated with the transport layer protocols.

FTP – file transfer protocol.

SMTP – simple mail transfer protocol.

Push protocol – protocol used when sending emails, in which the client opens the connection to the server and keeps the connection active all the time, then uploads new emails to the server.

Binary file – a file that does not contain text only. The file is machine-readable but not human-readable.

MIME – multi-purpose internet mail extension. A protocol that allows email attachments containing media files as well as text to be sent.

POP – post office protocol.

IMAP – internet message access protocol.

TCP – transmission control protocol.

Pull protocol – protocol used when receiving emails, in which the client periodically connects to a server, checks for and downloads new emails from a server and then closes the connection.

Host-to-host – a protocol used by TCP when communicating between two devices.

Host – a computer or device that can communicate with other computers or devices on a network.

BitTorrent – protocol used in peer-to-peer networks when sharing files between peers.

Peer – a client who is part of a peer-to-peer network/file sharing community.

Metadata – a set of data that describes and gives information about other data.

Pieces – splitting up of a file when using peer-to-peer file sharing.

Tracker – central server that stores details of all other computers in the swarm.

Swarm – connected peers (clients) that share a torrent/tracker.

Seed – a peer that has downloaded a file (or pieces of a file) and has then made it available to other peers in the swarm.

Leech – a peer with negative feedback from swarm members.

Lurker – user/client that downloads files but does not supply any new content to the community.

14.1.1 The need for protocols

When communicating over networks, it is essential that some form of **protocol** is used by the sender and receiver of the data. Both parties need to agree the protocol being used to ensure successful communication takes place. In Chapter 6, we discussed parity checking as a way of determining whether data was transmitted correctly. With this method, it was essential to agree the protocol: even or odd parity. Without agreeing this protocol, it would be impossible to use parity checking. Many different protocols exist since there are several activities taking place over the internet.

The next section considers one of the most common sets of protocols, which are implemented by using a stack structure with several layers.

14.1.2 TCP/IP protocols

This is the four-layer structure for TCP/IP protocols:

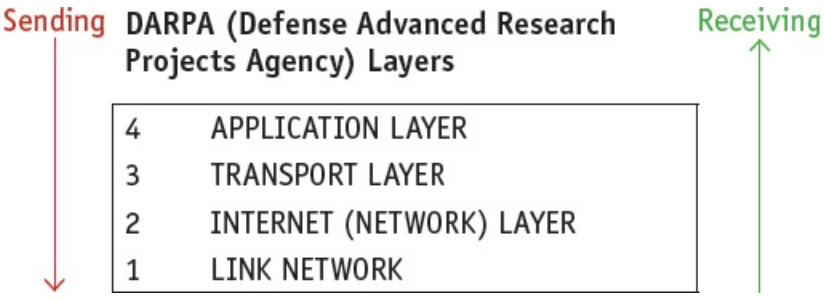


Figure 14.1 Four layer structure for TCP/IP

Using layers breaks the process down into manageable self-contained modules (this process is known as **decomposition**), making it easier to develop and easier to make software and hardware compatible.

When sending data across the internet (network), the layers are used in the order layer 4 to layer 1; when receiving data across the internet (network), the layers are used in the order layer 1 to layer 4. Each of the layers is implemented using software.

Application layer

The application layer contains all the programs that exchange data, such as web browsers or server software; it sends files to the transport layer. This layer allows applications to access the services used in other layers and also defines the protocols that any app uses to allow the exchange of data.

There are several protocols associated with the application layer:

HTTP	hypertext transfer protocol; this is a protocol responsible for correct transfer of files that make up web pages on the world wide web
SMTP	simple mail transfer protocol; this handles the sending of emails
POP3/4	post office protocol; this handles the receiving of emails
IMAP	internet message access protocol; this handles the receiving of emails
DNS	domain name service; protocol used to find the IP address, for example, when sending emails
FTP	file transfer protocol; this is a protocol used when transferring messages and attachments
RIP	routing information protocol; this is the protocol routers use to exchange routing information over an IP network
SNMP	simple network management protocol; protocol used when exchanging network

	management information between network management and network devices (such as routers, servers and other network devices)

Table 14.1 Protocols associated with the application layer

It is worth re-visiting the terms *packet* and *router*.

Messages are split up into small groups of bits called *packets* (for example, a web page would be split up into a number of packets before sending over the network).

A router is used to transmit packets of data; routers contain connections to many other routers; when packets arrive at a router it decides where next to send them.

Important terminology: packets are known as *frames* at the data-link layer, *datagrams* at the internet layer and **segments** at the transport layer. Different names are used as each layer adds its own header to the packet.

Do not confuse 'frames' in this context with 'frames' when discussing paging memory management in Chapter 16.

Hypertext transfer protocol (HTTP)

HTTP is probably the most important application layer protocol. Essentially, this protocol underpins the world wide web. It is used when, for example, fetching an HTML document from a web server (Figure 14.2).

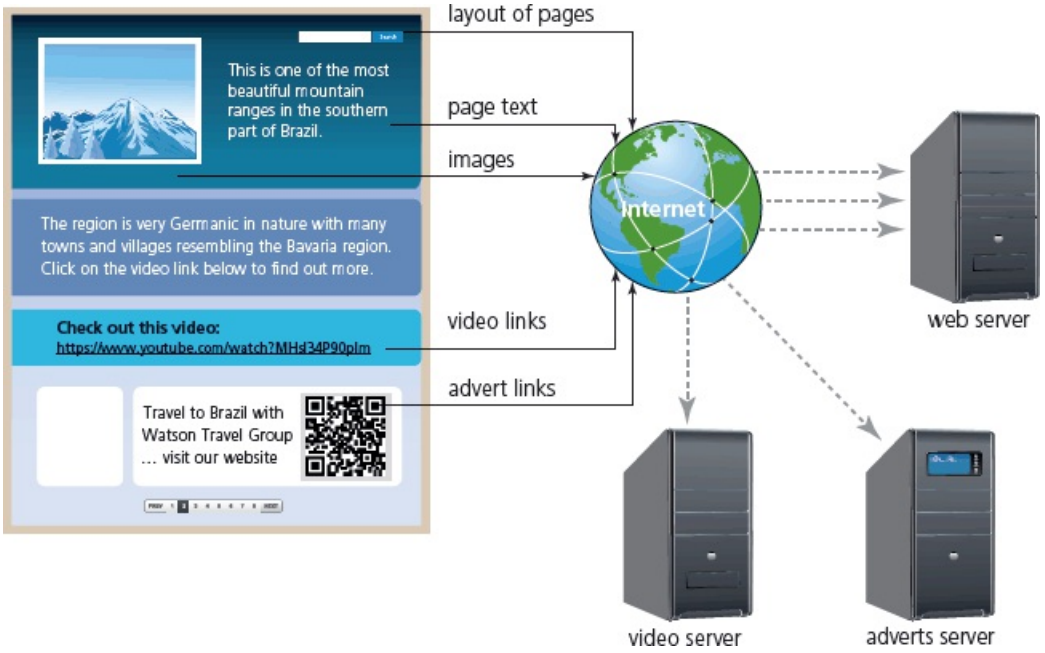


Figure 14.2 Fetching an HTML document from a web server

This makes use of hyperlinks (rules for the transferring of data over the internet). HTTP is a client/server protocol: request messages are sent out to the web servers which then respond.

HTTP protocols define the format of the messages sent and received. The web browser (which is part of the application layer) initiates the web page request and also converts HTML into a

format which can be displayed on the user's screen or can be played through their media player.

The following summarises what happens when a user requests a web page from a website.

- The user keys the URL into their browser.
- HTTP(s) transmits the request from the application layer to the transport layer (TCP).
- The TCP creates data packets and sends them (via port 80) to the destination port(s).
- The DNS server stores a database of URLs and matching IP addresses.
- The DNS server uses the domain name typed into the browser to look up the IP address of the appropriate website.
- The server TCP sends back an acknowledgement (see the section on host-to-host communication on page **333**).
- Once communication has been established, the web server sends the web page back in HTML format to the browser.
- The browser interprets the page and displays it or sends the data in the correct format to the media player.

File transfer protocol (FTP)

The **FTP file transfer protocol (FTP)** is a network protocol used when transferring files from one computer/device to another via the internet or other networks. It is similar to HTTP and SMTP, but FTP's only task is the application protocol for the transfer of files over a network. Web browsers can be used to connect to an FTP address in a way similar to HTTP, for example, ftp://username@ftp.example.gov/

Additional features of FTP include

- **anonymous ftp** – this allows a user to access files without the need to identify who they are to the ftp server; for example, '331 Anonymous access allowed' would be a message received to confirm anonymous access
- **ftp commands** – a user is able to carry out actions that can change files stored on the ftp server; for example, delete, close, rename, cd (change directory on a remote machine), lcd (change directory on a local machine)
- **ftp server** – this is where the files, which can be downloaded as required by a user, are stored.

A session would be started by typing in the ftp host_name (of remote system), followed by a user id and password. The user would then be able to use ftp commands to carry out a number of actions.

Simple mail transfer protocol (SMTP)

Simple mail transfer protocol (SMTP) is a text-based (and connection-based) protocol used when sending emails. It is sometimes referred to as a **push protocol** (in other words, a client opens a connection to a server and keeps the connection active all the time; the client then uploads a new email to the server).

Since SMTP is text-based only, it doesn't handle **binary files** (a binary file is a file containing media/images as well as text and is regarded as being computer-readable only). If an email contains attachments made up of, for example, images, video, music then it is necessary to use

the **multi-purpose internet mail extension (MIME)** protocol instead. A MIME header is used at the beginning of the transmission; clients use this header to select which media player is needed when the attachment is opened.

POP3/4 and IMAP (post office protocol and internet message access protocol)

POP Post office protocol (POP3/4) and **internet message access protocol (IMAP)** are protocols used when receiving emails from the email server. These are known as **pull protocols** (the client periodically connects to a server; checks for and downloads new emails from the server – the connection is then closed; this process is repeated to ensure the client is updated). IMAP is a more a recent protocol than POP3/4, but both have really been superseded by the increasing use of HTTP protocols. However, SMTP is still used when transferring emails between email servers.

Figures 14.3 and 14.4 give an overall view and a more detailed view of the email protocol set up.

Figure 14.3 Overview of email protocol set up

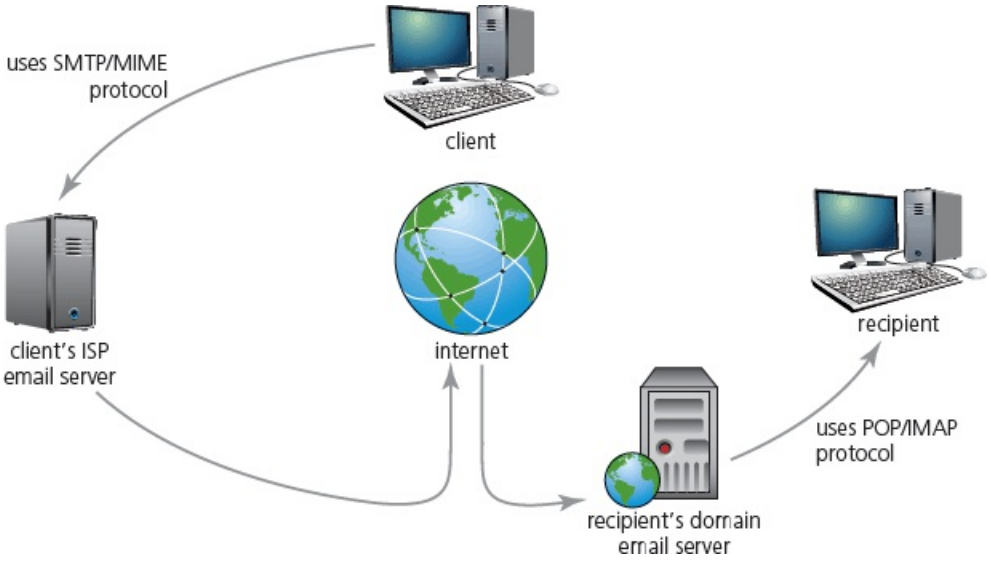


Figure 14.4 Detailed view of email protocol set up

The main difference between POP3/4 and IMAP is synchronisation:

POP3/4	**IMAP**
POP3/4 does not keep the server and client in synchronisation; when emails are downloaded by the client, they are then deleted from the server which means it is not further updated.	IMAP keeps the server and client in synchronisation; only a copy of the email is downloaded with the original remaining on the server until the client manually deletes it.

Table 14.2

Transport layer

The transport layer regulates the network connections; this is where data is broken up into packets which are then sent to the internet/network layer (IP protocol). The transport layer ensures that packets arrive in sequence, without errors, by swapping acknowledgements and retransmitting packets if they become lost or corrupted. The main protocols associated with the transport layer are **transmission control protocol (TCP)**, user datagram protocol (UDP) and SCTP. We will only consider TCP.

Transmission control protocol (TCP)

TCP is responsible for the safe delivery of a message by creating sufficient packets for transmission. It uses positive acknowledgement with re-transmission (PAR) which means it automatically re-sends a data packet if it has not received a positive acknowledgement. TCP is also connection-orientated since it establishes an end-to-end connection between two host computers using handshakes. For this last reason, TCP is often referred to as a **host-to-host** transmission protocol.

The term **host** has been used previously; this refers to a computer or device that can communicate with another computer/device (host). Hosts can include clients and servers that send/receive data, provide services or apps.

These are the steps taken when host computer ‘X’ communicates with another host ‘Y’ (this is an expansion of what happens during TCP involvement shown in the HTTP algorithm earlier on):

- Host ‘X’ will first of all send host ‘Y’ a segment (packet) which will include synchronisation sequence bits so that segments will be received in the correct order.
- Host ‘Y’ will now respond by sending back its own segment (containing an acknowledgement together with its own synchronisation sequence bits).
- Host ‘X’ now sends out its own acknowledgement that the segment from ‘Y’ was received.
- Transmission of data between ‘X’ and ‘Y’ can now take place.

Internet/network layer and network/data-link layer

The internet layer identifies the intended network and host. The common protocol is IP (internet protocol). The concept of IPv4 and IPv6 was covered in depth in Chapter 2.

The network/data-link layer identifies and moves traffic across local segments, encapsulates IP packets into frames for transmission, maps IP addresses to MAC (physical) addresses and ensures correct protocols are followed. The physical network layer specifies requirements of the hardware to be used for the network. The data-link layer identifies network protocols in the packet header (TCP/IP in the case here) and delivers packets to the network.

This is a summary of the IP functions:

- Ensure correct routing of packets of data over the internet/network.
- Responsible for protocols when communicating between networks.
- Take a packet from the transport layer and add its own header which will include the IP addresses of both sender and recipient.
- The IP packet (datagram) is sent to the data-link layer where it is assembles the datagrams into

frames for transmission.

Ethernet protocols

Ethernet is a system that connects a number of computers or devices together to form a LAN. It uses protocols to control the movement of frames between computers or devices and to avoid simultaneous transmission by two or more devices. It is a local protocol and does not provide any means to communicate with external devices; this requires the use of IP which sits on top of the Ethernet protocol.

Figure 14.5 shows the make-up of a typical frame used by the Ethernet protocol.

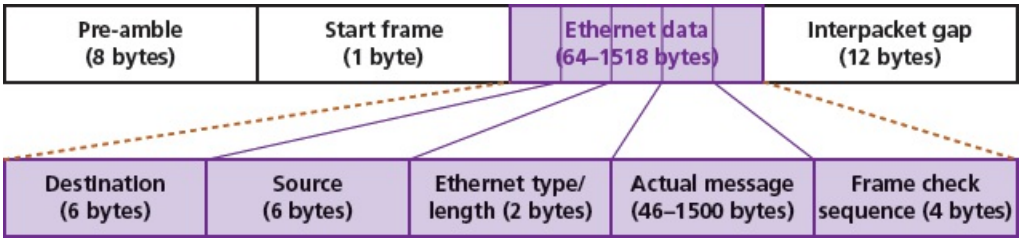


Figure 14.5 A typical frame used by the Ethernet protocol

If VLAN is used, the Ethernet data size increases from 1539 bytes to around 9000 bytes per frame.

The components that make up Ethernet data are

- **destination** – this is the MAC address of the destination computer or device (it is possible to use the value FF:FF:FF:FF:FF:FF as the MAC address if the sender wishes to target every device (for example, to advertise services) or if they do not know the MAC address of the destination device)
- **source** – this is the MAC address of the source computer (using the usual MAC address format of 6 bytes)
- **Ethernet type or length** – if the frame length ≤ 1539 then the value here is the length of the Ethernet frame; if the frame length > 1539 then the value here is the Ethernet type (IPv4 or IPv6 in our example)
- **frame check** – this will include a checksum to provide a method of checking data integrity following transmission of the frame.

Wireless (WiFi) protocols

Wireless LANs (standard IEEE 802.11 protocol) use a MAC protocol called carrier sense multiple access with collision avoidance (CSMA/CA) (not to be confused with CSMA/CD considered in Chapter 2, since this is a totally different concept).

CSMA/CA uses distributed control function (DCF) to ensure a WiFi device can only transmit when there is a free channel available. Since all transmissions are acknowledged when using DCF, if a device does not receive an acknowledgement it will assume a collision will occur and waits for a random time interval before trying again. This is an important protocol to ensure the security and integrity of data being sent over a wireless network (such as WLAN).

Bluetooth protocols

Bluetooth was considered in Chapter 2; it uses the standard IEEE 802.15 protocol for short-range data transmission/communication. There are numerous additional Bluetooth protocols due to the many applications that may use this wireless connectivity; this is outside the scope of this textbook.

WiMax

Worldwide interoperability for microwave access (WiMax) runs under IEEE 802.16 protocol. This connectivity was designed originally for wireless MANs (WMAN). Fixed WiMax networks are based on the IEEE 802.16-2004 protocol, whereas mobile WiMax is based on IEEE 802.16-2005 protocol.

Peer-to-peer file sharing/BitTorrent protocol

The **BitTorrent** is a protocol which is based on the peer-to-peer networking concept (this was covered in Chapter 2). This allows for very fast sharing of files between computers (known as **peers**). While peer-to-peer networks only work well with very small numbers of computers, the concept of sharing files using BitTorrent can be used by thousands of users who connect together over the internet. Because user computers are sharing files directly with each other (rather than using a web server) they are sharing files in a way similar to that used in a peer-to-peer network; the main difference is that the BitTorrent protocol allows many computers (acting as peers) to share files.

Suppose computer ‘A’ wishes to share a file with a number of other interested peers. How can we use the BitTorrent protocol to allow this file sharing?

Initially, to share a file, the peer (computer ‘A’) creates a small file called a torrent (for example, MyVideoFile.torrent). The torrent contains **metadata** about the file about to be shared.

The actual file is broken up into equal segments known as **pieces** (typically a 20 MiB file may be broken up into 20 × 1 MiB pieces).

Other peers who wish to download this file must first obtain the torrent and connect to the appropriate **tracker** – a central server that contains data about all of the computers connected to it.

As each peer receives a piece of file they then become a source for that piece of file. Other peers connected to the tracker will, therefore, know where to find the piece of file they need.

Once a peer has downloaded a file completely and they make the file (or required pieces of the file) available to other peers in the **swarm** (a group of peers connected together), they become a **seed**. The more seeds in the swarm, the faster the file downloading process between peers.

Logging off once the full file download has been completed is frowned upon by the swarm community; such a peer is termed a **leech**.

Usually, once a file is fully downloaded, a peer is requested to remain online so they can become part of the seeding process until all peers have received the whole file. Note that file pieces may not be downloaded sequentially and have to be rearranged in the correct order by the BitTorrent protocol to produce the final file (quite important if the file is a video!).

At the time of writing, BitTorrent was responsible for about 12% of the video file sharing, for example, being carried out over the internet. This is only a fraction of the video file activity

which uses YouTube (which is about 50% of all of the video file sharing over the internet), but is still a considerable amount of data.

Here is a summary of some of the terms used when discussing BitTorrent:

- **Swarm** – a group of peers connected together is known as a swarm; one of the most important facts when considering whether or not a swarm can continue to allow peers to complete a torrent is its availability; availability refers to the number of complete copies of torrent contents that are distributed amongst a swarm. Note: a torrent is simply the name given to a file being shared on the peer-to-peer network.
- **Seed** – a peer that has downloaded a file (or pieces of a file) and has then made it available to other peers in the swarm.
- **Tracker** – this is a central server that stores details about other computers that make up the swarm; it will store details about all the peers downloading or uploading the file, allowing the peers to locate each other using the stored IP addresses.
- **Leech** – a peer that has a negative impact on the swarm by having a poor share ratio, that is, they are downloading much more data than they are uploading to the others; the ratio is determined using the formula:

  $$\text{ratio} = \frac{\text{amount of data the peer has uploaded}}{\text{amount of data the peer has downloaded}}$$

  If the ratio > 1 then the peer has a positive impact on the swarm; if the ratio < 1 then the peer has a negative effect on the swarm.
- **Lurker** – a peer that downloads many files but does not make available any new content for the community as a whole.

In Figure 14.6, we will assume 12 peers have connected to the tracker. One peer has begun to upload a video file and six peers are acting as seeds. Two peers are behaving as leeches and three peers have just joined and have requested a download of the video file. The arrangement would look something like this:

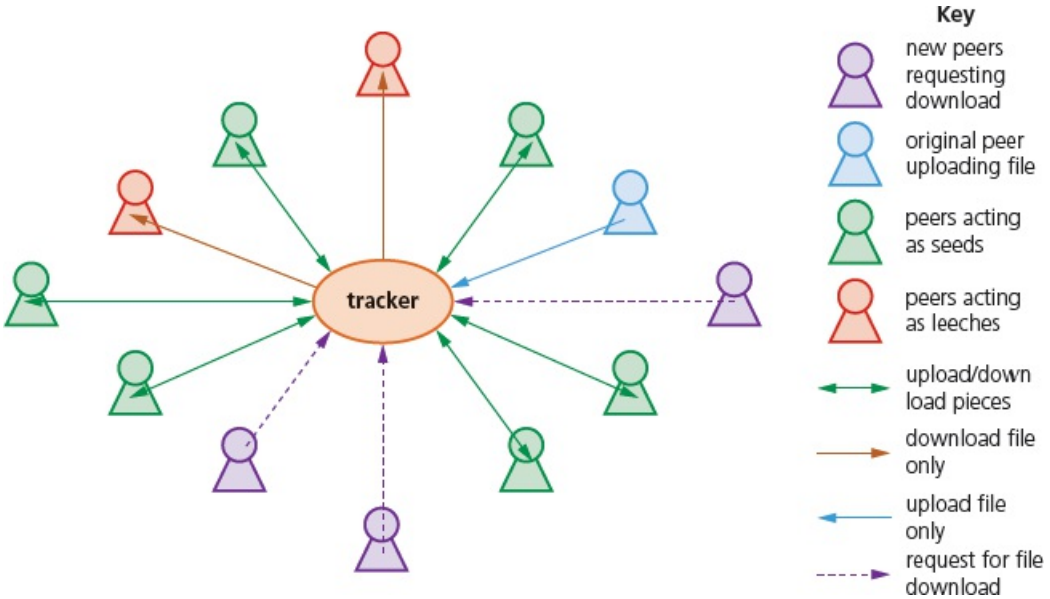


Figure 14.6 The arrangement of peers during the download and upload of file (pieces)

14.2 Circuit switching and packet switching

WHAT YOU SHOULD ALREADY KNOW

In Chapter 2, you learnt about communications. Try these two questions before you read the second part of this chapter.

1 a) Explain how PSTN is used when making a phone call.

b) Explain how it is possible to use VoIP to make a video call over the internet.

2 Draw a diagram to show how a message containing three packets of data could be routed from computer 'A' to computer 'B'.

Key terms

Circuit switching – method of transmission in which a dedicated circuit/channel lasts throughout the duration of the communication.

Packet switching – method of transmission where a message is broken into packets which can be sent along paths independently from each other.

Hop number/hopping – number in the packet header used to stop packets which never reach their destination from 'clogging up' routes.

Header (data packet) – part of a data packet containing key data such as destination IP address, sequence number, and so on.

Routing table – a data table that contains the information necessary to forward a package along the shortest or best route to allow it to reach its destination.

14.2.1 Circuit switching

The concept of **circuit switching** was introduced in Chapter 2 during the description of how a public switched telephone network (PSTN) was used to make a phone call. Circuit switching uses a dedicated channel/circuit which lasts throughout the connection: the communication line is effectively 'tied up'. When sending data across a network, there are three stages:

1. First, a circuit/channel between sender and receiver must be established.
2. Data transfer then takes place (which can be analogue or digital); transmission is usually bi-directional.
3. After the data transfer is complete, the connection is terminated.

The pros and cons of circuit switching are summarised in this table. Figure 14.7 shows an example of circuit switching.

Pros	Cons
the circuit used is dedicated to the single transmission only	it is not very flexible (for example, it will send empty frames and it has to use a single, dedicated line)
the whole of the bandwidth is available	nobody else can use the circuit/channel even when it is idle
the data transfer rate is faster than with packet switching	the circuit is always there whether or not it is used
the packets of data (frames) arrive at the destination in the same order as they were sent	if there is a failure/fault on the dedicated line, there is no alternative routing available
a packet of data cannot get lost since all packets follow on in sequence along the same single route	dedicated channels require a greater bandwidth
it works better than packet switching in real-time applications	prior to actual transmission, the time required to establish a link can be long

Table 14.3 Pros and cons of circuit switching

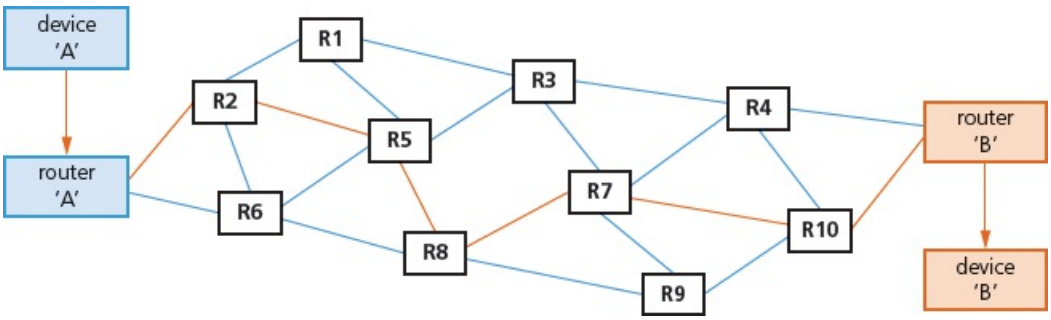


Figure 14.7 An example of circuit switching

The dedicated route from ‘A’ to ‘B’ is first of all established (shown in orange on the diagram). The following connections are then partially implemented: A–R2, R2–R5, R5–R8, R8–R7, R7–R10 and finally R10–B. All packets (frames) follow this single route and communication will take place, provided ‘B’ is not busy.

The main uses of circuit switching include public telephone networks, private telephone networks and private data networks.

14.2.2 Packet switching

Packet switching was introduced in Chapter 2 when describing VoIP, together with a diagram to show how the individual packets are routed from client to client.

Packet switching is a method of transmission in which a message is broken up into a number of packets that can be sent independently to each other from start point to end point. The data packets will need to be reassembled into their correct order at the destination. Figure 14.8 shows an example of packet switching.

Note that

- each packet follows its own path
- routing selection depends on the number of datagram packets waiting to be processed at each node (router)
- the shortest path available is selected
- packets can reach the destination in a different order to that in which they are sent.

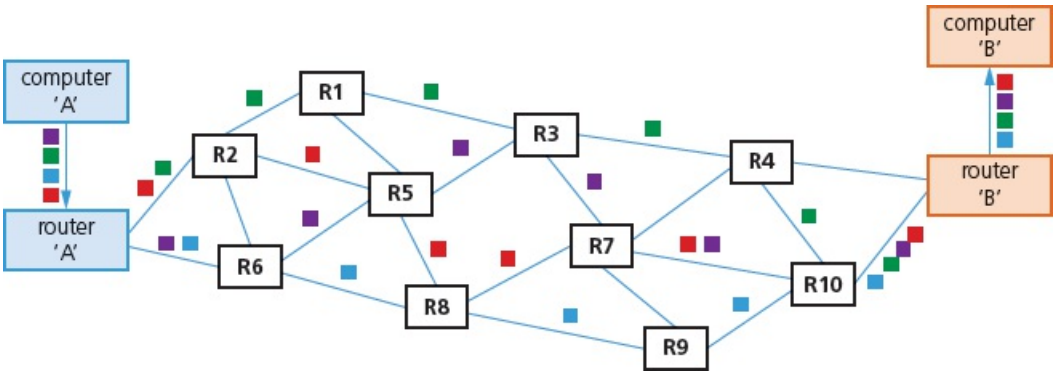


Figure 14.8 An example of packet switching

As Figure 14.8 shows, the message sent by computer 'A' was split into four packets. The original packet order was: ■■■■ and they arrived in the order: ■■■■ which means they need to be reassembled in the correct order at the destination.

The pros and cons of packet switching are summarised in this table.

Pros	Cons
no need to tie up a communication line	the protocols for packet switching can be more complex than those for circuit switching
it is possible to overcome failed or faulty lines by simply re-routing packages	if a packet is lost, the sender must re-send the packet (which wastes time)
it is easy to expand the traffic usage	does not work well with real-time data streams

circuit switching charges the user on the distance and duration of a connection, but packet switching charges users only for the duration of the connectivity	the circuit/channel has to share its bandwidth with other packets
high data transmission is possible with packet switching	there is a delay at the destination while packets are reassembled
packet switching always uses digital networks which means digital data is transmitted directly to the destination	needs large amounts of RAM to handle the large amounts of data

Table 14.4 Pros and cons of packet switching

Comparison of circuit switching and packet switching

Feature	Circuit switching	Packet switching
actual route used needs to be set up before transmission can begin		
a dedicated transmission path is required		
each packet uses the same route		
packets arrive at destination in the correct order		
all the bandwidth of the channel is required		
is bandwidth wasted?		

Table 14.5 Comparison of circuit switching and packet switching

Sometimes it is possible for packets to get lost and keep 'bouncing' around from router to router and never actually get to their destination. Eventually, the network could grind to a halt as the number of 'lost' packets mounts up and clogs up the system. To overcome this, a method called **hopping** is used. A **hop number** is added to the header of each packet. Each packet is only allowed to hop a finite number of times (this number is determined by the network protocol and routing table being used). Each time a packet passes through a router, the hop number is decreased by 1. If the packet has not reached its destination and the hop number = 0, then it will be deleted when it reaches the next router.

Each packet also contains an error checking technique such as a checksum or parity check. If a checksum is used, this value is calculated for each packet and is added to the header. The checksum for each package is recalculated at the destination to ensure no errors have occurred. If the checksum values are different, then a request is made to re-send the packet. A priority value is sometimes also added to a header. A high priority value indicates which packet queue should be used.

This is the make-up of a packet **header** (together with the data in the message being sent) if TCP/IP protocol is being used when sending packets:

IP address of source computer	IP address of destination computer	current hop number of data packet	length of packet in bytes	number of packets in the message	sequence number to allow reassembly of packets	checksum value

Figure 14.9

More generally, packet headers contain the following information (the information used with TCP/IP protocol headers is highlighted in green)

- 4 bits to identify protocol version (such as IPv4, IPv6 – in the example above we assume IP)
- 4 bits to identify header length (in multiples of four; for example, a value of six implies 6 × 4 = 24 bytes)
- 8 bits to represent packet priority
- 16 bits to identify the length of the packet in bytes
- 3 bits are used for fragmentation; the DF flag indicates whether a packet can be fragmented or not (DF = do not fragment) and the MF flag indicates whether there are more fragments of a packet to follow (MF = more fragments)

0	**DF**	MF

- 13 bits to show fragmentation offset to identify the position of the fragments within the original packet
- 8 bits that show the current hop number of the packet
- 16 bits to show the number of packets in the message
- 16 bits to represent the sequence number of the packet
- 8 bits that contain the transmission protocol being used (TCP, UDP)
- 16 bits that contain the header checksum value
- 32 bits that contain the source IP address
- 32 bits that contain the destination IP address.

Routing tables

Routing tables contain the information necessary to forward a package along the shortest/best route to allow it to reach its destination. As soon as the packet reaches a router, the packet header is examined and compared with the routing table. The table supplies the router with instructions to send the packet (hop) to the next available router.

Routing tables include

- number of hops
- MAC address of the next router where the packet is to be forwarded to (hopped)
- metrics (a cost is assigned to each available route so that the most efficient route/path is found)
- network destination (network ID) or pathway
- gateway (the same information as the next hop; it points to the gateway through which target network can be reached)
- netmask (used to generate network ID)

- interface (indicates which locally available interface is responsible for reaching the gateway).

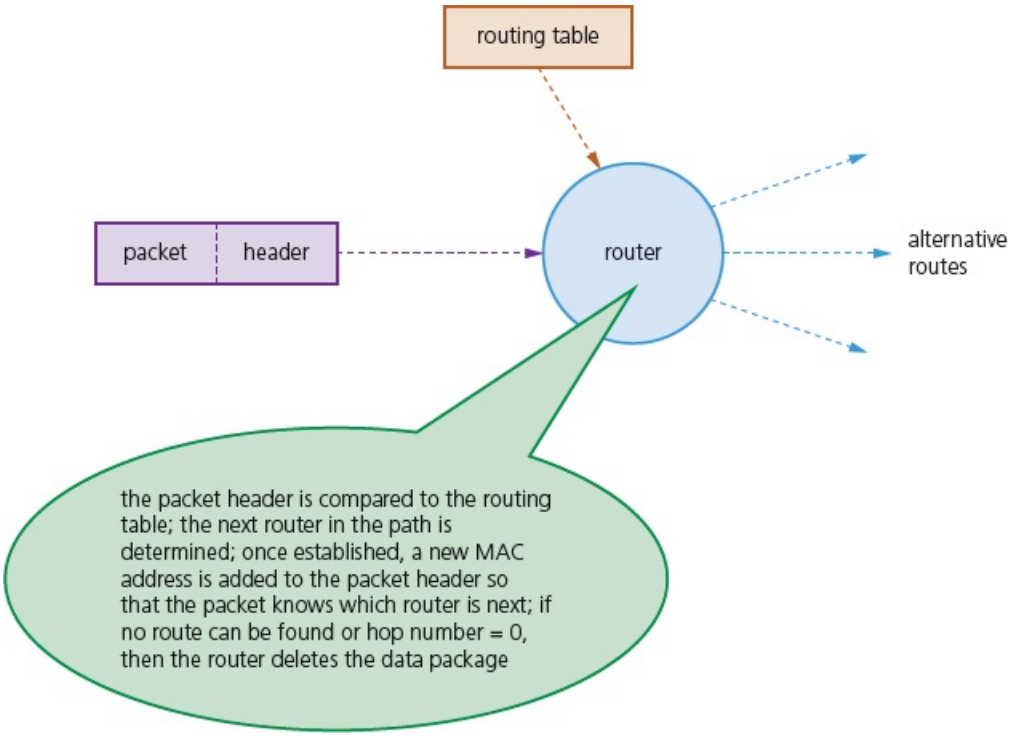


Figure 14.10

Example 14.1

Suppose we are carrying out video conferencing using packet switching as the method of routing data. The performance of the communication is not very good.

a) Describe some of the poor performance you might expect.
Give a reason for this poor performance.

b) What features of circuit switching could potentially improve the performance?

Solution

a) Answers might include: picture and sound may not be in synchronisation (packets arriving at different times); video not continuous – pauses (time delay in reassembling the packets of data); degraded quality of sound and video (possibly caused by competing traffic in communication lines); possible drop out (packets take different routes, so it is possible for packets to be lost).

b) Answers might include: only one route used in circuit switching; therefore, all packets arrive in correct order; dedicated communication channel with circuit switching; therefore, full bandwidth available; there is no loss of synchronisation with circuit switching.

Example 14.2

How would packet switching be used to download a page from a website?

Solution

Answers might include

- the web page is divided up into data packets
- each packet has a header, which includes the IP address of the destination
- the router checks the header against values stored in the routing table …
- … to determine which router the packet needs to be sent to next (hopped) …
- … the MAC address of the next router is added to the package header
- the hop value is checked to see if it equals zero
- each packet may go by a different route
- the destination computer reassembles the packets building up the final web page.

ACTIVITY 14A

1 a) Name the **four** layers which show the TCP/IP protocols.

b) Name **one** protocol associated with each layer.

c) i) Describe the use of protocols when sending and receiving emails.

ii) What is the difference between SMTP and MIME when sending emails?

2 a) What is an *Ethernet*?

b) Describe the contents of an *Ethernet frame*.

c) Ethernet protocols do not provide a means to communicate with devices outside a LAN. How can external devices be communicated with when using an Ethernet?

3 a) Explain the following terms used in peer-to-peer BitTorrent.

i) peer

ii) swarm

iii) tracker

iv) leech

v) seed

b) Explain how it could be possible to deal with peers acting as leeches.

4 a) Describe the difference between a *packet header* and a *routing table*.

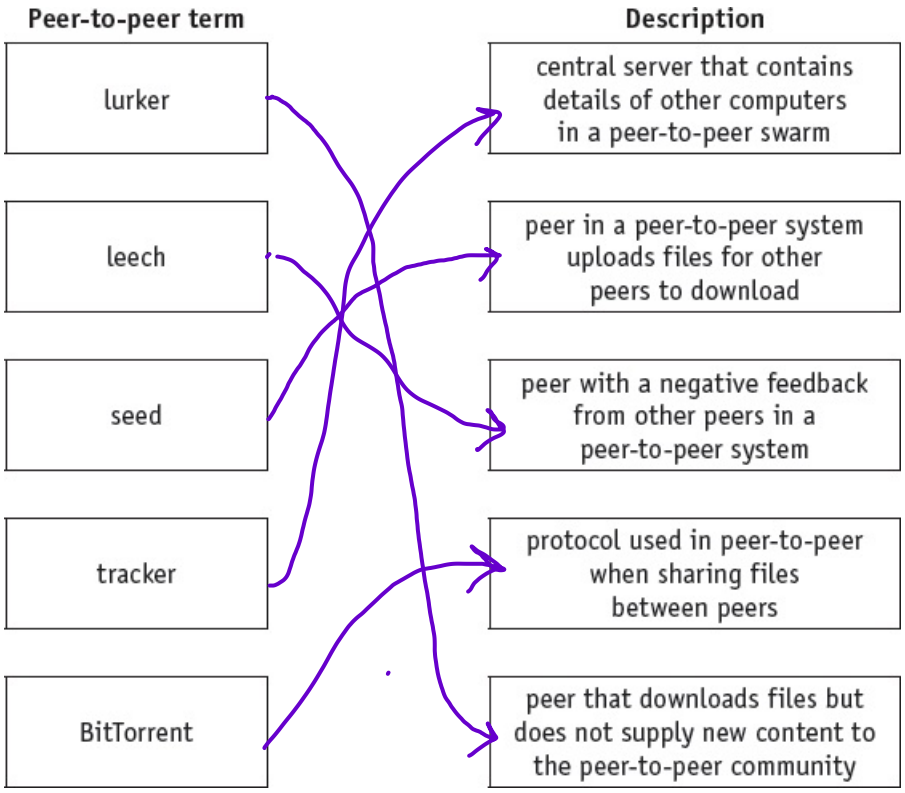
b) How are the packet header and the routing table used to route a package?

5 A person is making a video call using VoIP software.
Explain how packet switching could be used and describe any problems that might occur.

End of chapter questions

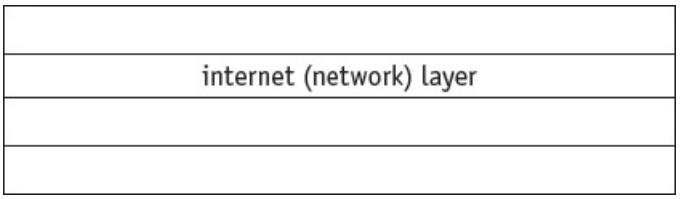
1 a) Copy the diagram below and connect each peer-to-peer term to its correct description.

[5]



b) Copy and complete the diagram to show the layers in a TCP/IP protocol.

[3]

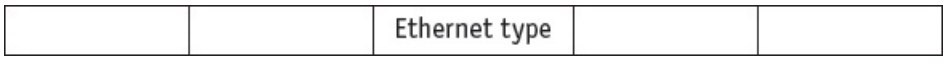


c) Describe the protocols used when sending and receiving emails.

[4]

2 a) An Ethernet frame contains a section called Ethernet data.
Copy and complete this diagram to show the other four items missing from the Ethernet data section.

[4]



b) State what is meant by the term *metadata*.

[1]

c) Describe how files can be shared using the BitTorrent protocol.

[4]

3 a) Explain what is meant by circuit switching.

[2]

b) There are many applications in which digital data are transferred across a network. Video conferencing is one of these.
For this application, circuit switching is preferable to the use of packet switching.
Explain why this is so.

[6]

c) A web page is transferred from a web server to a home computer using the Internet.
Explain how the web page is transferred using packet switching.

[3]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q3 November 2015

4 a) This table shows some statements about circuit switching and packet switching.
Copy the table and indicate which statements are true ( ) and which are false ( ).

[5]

statements	**circuit switching**	**packet switching**
a dedicated circuit/path is needed at all times		
the same route/circuit is used for every packet in the message		
bandwidth is shared with other packets of data		
none of the bandwidth available is wasted during transmission		
packets arrive at the destination in the correct order		

b) Explain the following terms and why they are used when sending packets across a network.

i) hop number/hopping

[2]

ii) checksum

[2]

c) Describe how headers and routing tables are used to route packets efficiently from a sender to recipient.

[5]

15 Hardware

In this chapter, you will learn about

- the differences between RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) processors
- the importance and use of pipelining and registers in RISC processors
- SISD, SIMD, MISD and MIMD basic computer architectures
- the characteristics of massively parallel computers
- interrupt handling on CISC and RISC computers
- Boolean algebra including De Morgan's Laws
- the simplification of logic circuits and expressions using Boolean algebra
- producing truth tables from common logic circuits
- half adder and full adder logic circuits
- the construction and role of SR and JK flip-flop circuits
- using Karnaugh maps in the solution of logic problems.

15.1 Processors and parallel processing

WHAT YOU SHOULD ALREADY KNOW

In Chapter 4, you learnt about processor fundamentals. Try the following three questions to refresh your memory before you start to read the first part of this chapter.

1 A computer uses the following status registers when carrying out the addition of two binary numbers

- a carry flag (C)
- an overflow flag (V)
- a negative flag (N).

Describe what happens to the above status registers when the following pairs of 8-bit binary numbers are added together and explain the significance of the flag values in both sets of calculation

a) 0 0 1 1 1 1 0 0 and 0 1 0 0 0 1 1 0

b) 1 1 0 0 0 1 0 0 and 1 0 1 1 1 0 1 0.

2 Describe the stages in the fetch-execute cycle.

3 **a)** A processor contains three buses: data bus, address bus and control bus.

i) What factors determine the width of a bus?

ii) Which of the three buses will have the smallest width?

iii) An address bus is increased from 16-bit to 64-bit. What would be the result of this upgrade to the processor?

b) Explain the role of

i) the clock

ii) interrupts in a typical processor.

Key terms

CISC – complex instruction set computer.

RISC – reduced instruction set computer.

Pipelining – allows several instructions to be processed simultaneously without having to wait for previous instructions to finish.

Parallel processing – operation which allows a process to be split up and for each part to be executed by a different processor at the same time.

SISD – single instruction single data, computer architecture which uses a single processor and one data source.

SIMD – single instruction multiple data, computer architecture which uses many processors and different data inputs.

MISD – multiple instruction single data, computer architecture which uses many processors but the same shared data source.

MIMD – multiple instruction multiple data, computer architecture which uses many processors, each of which can use a separate data source.

Cluster – a number of computers (containing SIMD processors) networked together.

Super computer – a powerful mainframe computer.

Massively parallel computers – the linking together of several computers effectively forming one machine with thousands of processors.

15.1.1 RISC and CISC processors

Early computers made use of the Von Neumann architecture (see Chapter 4). Modern advances in computer technology have led to much more complex processor design. Two basic philosophies have emerged over the last few years

- developers who want the emphasis to be on the hardware used: the hardware should be chosen to suit the high-level language development
- developers who want the emphasis to be on the software/instruction sets to be used: this philosophy is driven by ever faster execution times.

The first philosophy is part of a group of processor architectures known as **CISC (complex instruction set computer)**. The second philosophy is part of a group of processor architectures known as **RISC (reduced instruction set computer)**.

CISC processors

CISC processor architecture makes use of more internal instruction formats than RISC. The design philosophy is to carry out a given task with as few lines of assembly code as possible. Processor hardware must therefore be capable of handling more complex assembly code instructions. Essentially, CISC architecture is based on single complex instructions which need to be converted by the processor into a number of sub-instructions to carry out the required operation.

For example, suppose we wish to add the two numbers A and B together, we could write the following assembly instruction:

ADD A, B – this is a single instruction that requires several sub-instructions (multi-cycle) to carry out the *ADD*ition operation

This methodology leads to shorter coding (than RISC) but may actually lead to more work being carried out by the processor.

RISC processors

RISC processors have fewer built-in instruction formats than CISC. This can lead to higher processor performance. The RISC design philosophy is built on the use of less complex instructions, which is done by breaking up the assembly code instructions into a number of simpler single-cycle instructions. Ultimately, this means there is a smaller, but more optimised set of instructions than CISC. Using the same example as above to carry out the addition of two numbers A and B (this is the equivalent operation to ADD A, B):

LOAD X, A – this loads the value of A into a register X

LOAD Y, B – this loads the value of B into a register Y

ADD A, B – this takes the values for A and B from X and Y and adds them

STORE Z – the result of the addition is stored in register Z

Each instruction requires one clock cycle (see Chapter 4). Separating commands such as LOAD and STORE reduces the amount of work done by the processor. This leads to faster processor performance since there are ultimately a smaller number of instructions than CISC. It is worth noting here that the optimisation of each of these simpler instructions is done through the use of pipelining (see below).

Table 15.1 shows the main differences between CISC and RISC processors.

CISC features	RISC features
Many instruction formats are possible	Uses fewer instruction formats/sets
There are more addressing modes	Uses fewer addressing modes
Makes use of multi-cycle instructions	Makes use of single-cycle instructions
Instructions can be of a variable length	Instructions are of a fixed length
Longer execution time for instructions	Faster execution time for instructions
Decoding of instructions is more complex	Makes use of general multi-purpose registers
It is more difficult to make pipelining work	Easier to make pipelining function correctly
The design emphasis is on the hardware	The design emphasis is on the software
Uses the memory unit to allow complex instructions to be carried out	Processor chips require fewer transistors

Table 15.1 The differences between CISC and RISC processors

EXTENSION ACTIVITY 15A

Find out how some of the newer technologies, such as EPIC (Explicitly Parallel Instruction Computing) and VLIW (Very Long Instruction Word) processor architectures, are used in computer systems.

Pipelining

One of the major developments resulting from RISC architecture is **pipelining**. This is one of the less complex ways of improving computer performance. Pipelining allows several instructions to be processed simultaneously without having to wait for previous instructions to be completed. To understand how this works, we need to split up the execution of a given instruction into its five stages

1 instruction fetch cycle (IF)

2 instruction decode cycle (ID)
3 operand fetch cycle (OF)
4 instruction execution cycle (IE)
5 writeback result process (WB).

To demonstrate how pipelining works, we will consider a program which has six instructions (A, B, C, D, E and F). Figure 15.1 shows the relationship between processor stages and the number of required clock cycles when using pipelining. It shows how pipelining would be implemented with each stage requiring one clock cycle to complete.

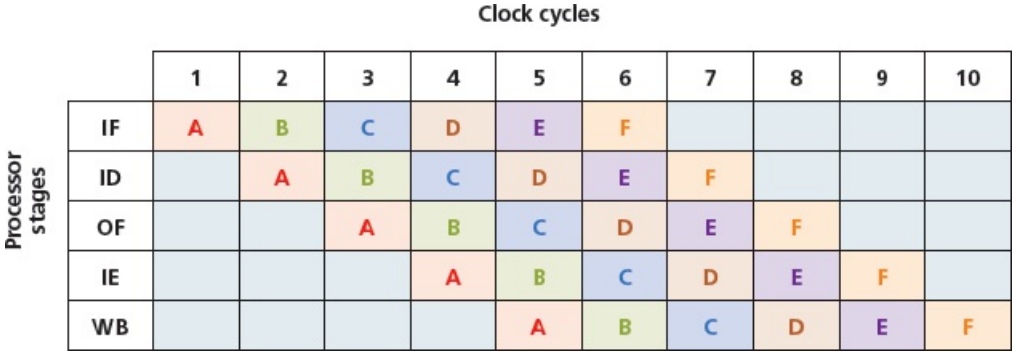


Figure 15.1

This functionality clearly requires processors with several registers to store each of the stages.

Execution of an instruction is split into a number of stages. During clock cycle 1, the first stage of instruction 1 is implemented. During clock cycle 2, the second stage of instruction 1 and the first stage in instruction 2 are implemented. During clock cycle 3, the third stage of instruction 1, second stage of instruction 2 and first stage of instruction 3 are implemented. This continues until all instruction are processed.

In this example, by the time instruction ‘A’ has completed, instruction ‘F’ is at the first stage and instructions ‘B’ to ‘E’ are at various in-between stages in the process. As Figure 15.1 shows, a number of instructions can be processed at the same time, and there is no need to wait for an instruction to go through all five cycles before the next one can be implemented. In the example shown, the six instructions require 10 clock cycles to go to completion. Without pipelining, it would require 30 (6 × 5) cycles to complete (since each of the six instructions requires five stages for completion).

Interrupts

In Chapter 4, we discussed interrupt handling in processors where each instruction is handled sequentially before the next one can start (five stages for instruction ‘A’, then five stages for instruction ‘B’, and so on).

Once the processor detects the existence of an interrupt (at the end of the fetch-execute cycle), the current program would be temporarily stopped (depending on interrupt priorities), and the status of each register stored. The processor can then be restored to its original status before the interrupt was received and serviced.

However, with pipelining, there is an added complexity; as the interrupt is received, there could

be a number of instructions still in the pipeline. The usual way to deal with this is to discard all instructions in the pipeline except for the last instruction in the write-back (WB) stage.

The interrupt handler routine can then be applied to this remaining instruction and, once serviced, the processor can restart with the next instruction in the sequence. Alternatively, although much less common, the contents of the five stages can be stored in registers. This allows all current data to be stored, allowing the processor to be restored to its previous status once the interrupt has been serviced.

15.1.2 Parallel processing

Parallel processor systems

There are many ways that **parallel processing** can be carried out. The four categories of basic computer architecture presently used are described below.

SISD (single instruction single data)

SISD (single instruction single data) uses a single processor that can handle a single instruction and which also uses one data source at a time. Each task is processed in a sequential order. Since there is a single processor, this architecture does not allow for parallel processing. It is most commonly found in applications such as early personal computers.

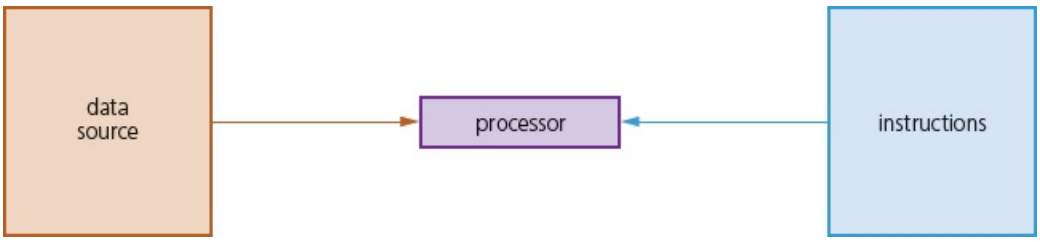


Figure 15.2 SISD diagram

SIMD (single instruction multiple data)

SIMD (single instruction multiple data) uses many processors. Each processor executes the same instruction but uses different data inputs – they are all doing the same calculations but on different data at the same time.

SIMD are often referred to as array processors; they have a particular application in graphics cards. For example, suppose the brightness of an image made up of 4000 pixels needs to be increased. Since SIMD can work on many data items at the same time, 4000 small processors (one per pixel) can each alter the brightness of each pixel by the same amount at the same time. This means the whole of the image will have its brightness increased consistently.

Other applications include sound sampling – or any application where a large number of items need to be altered by the same amount (since each processor is doing the same calculation on each data item).

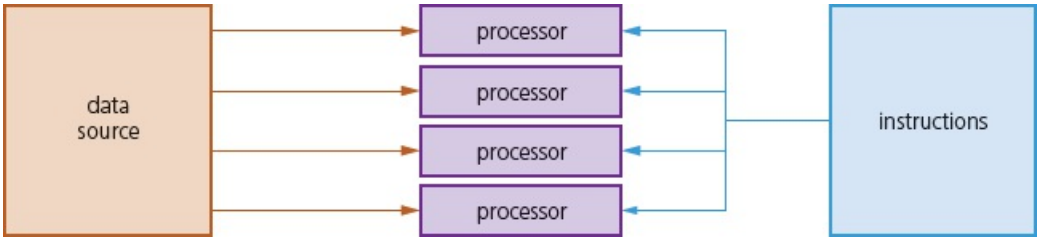


Figure 15.3 SIMD diagram

MISD (multiple instruction single data)

MISD (multiple instruction single data) uses several processors. Each processor uses different

instructions but uses the same shared data source. MISD is not a commonly used architecture (MIMD tends to be used instead). However, the American Space Shuttle flight control system did make use of MISD processors.

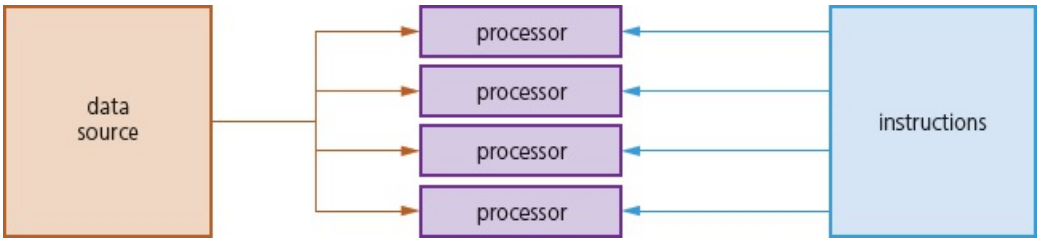


Figure 15.4 MISD diagram

MIMD (multiple instruction multiple data)

MIMD (multiple instruction multiple data) uses multiple processors. Each one can take its instructions independently, and each processor can use data from a separate data source (the data source may be a single memory unit which has been suitably partitioned). The MIMD architecture is used in multicore systems (for example, by super computers or in the architecture of multi-core chips).

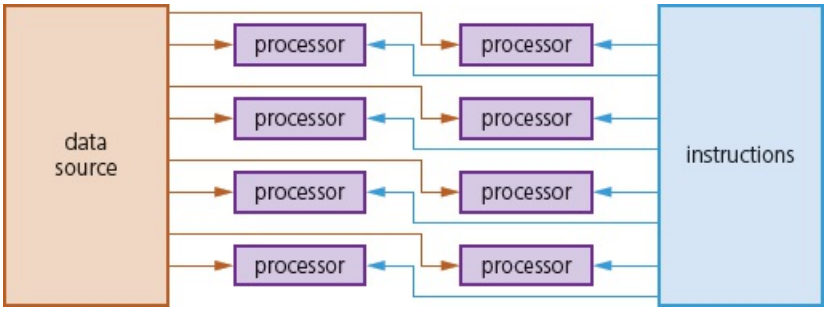


Figure 15.5 MIMD diagram

There are a number of factors to consider when using parallel processing.

When carrying out parallel processing, processors need to be able to communicate. The data which has been processed needs to be transferred from one processor to another.

When software is being designed, or programming languages are being chosen, they must be capable of processing data from multiple processors at the same time.

It is a much faster method for handling large volumes of *independent data*; any data which relies on the result of a previous operation (*dependent data*) would not be suitable in parallel processing. Data used will go through the same processing, which requires this independence from other data.

Parallel processing overcomes the Von Neumann ‘bottleneck’ (in this type of architecture, data is constantly moving between memory and processor, leading to *latency*; as processor speeds have increased, the amount of time they remain idle has also increased since the processor’s performance is limited to the internal data transfer rate along the buses). Finding a way around this issue is one of the driving forces behind parallel computers in an effort to greatly improve processor performance.

However, parallel processing requires more expensive hardware. When deciding whether or not to use this type of processor, it is important to take this factor into account.

Parallel computer systems

SIMD and MIMD are the most commonly used processors in parallel processing. A number of computers (containing SIMD processors) can be networked together to form a **cluster**. The processor from each computer forms part of a larger pseudo-parallel system which can act like a **super computer**. Some textbooks and websites also refer to this as grid computing.

Massively parallel computers have evolved from the linking together of a number of computers, effectively forming one machine with several thousand processors. This was driven by the need to solve increasingly complex problems in the world of science and mathematics. By linking computers (processors) together in this way, it massively increases the processing power of the 'single machine'. This is subtly different to cluster computers where each computer (processor) remains largely independent. In massively parallel computers, each processor will carry out part of the processing and communication between computers is achieved via interconnected data pathways. Figure 15.6 shows this simply.

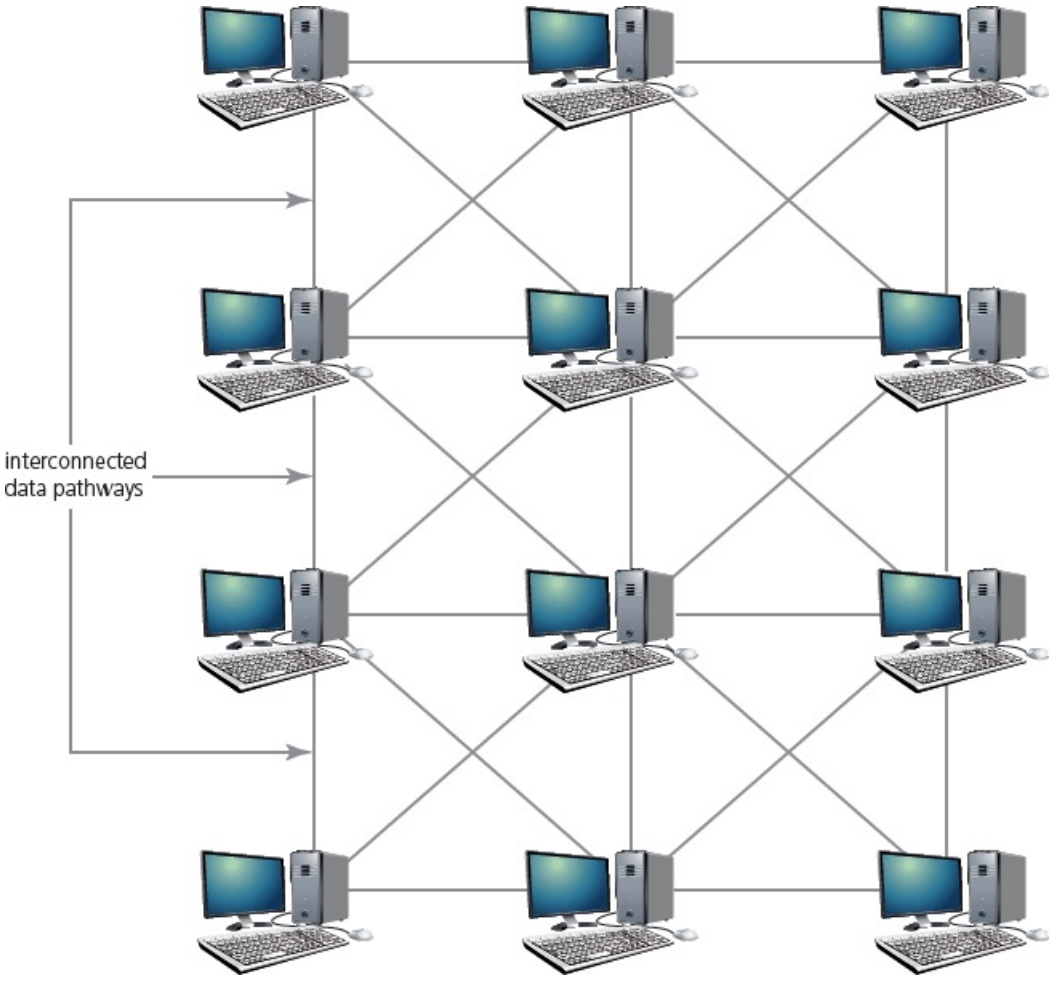


Figure 15.6 Typical massively parallel computer (processor) system showing interconnected pathways

EXTENSION ACTIVITY 15B

1 Find out more about the applications of multi-computer systems (cluster and massively parallel computers). In particular, research their uses in seismology, astronomy, climate modelling, nuclear physics and weather forecasting models.

2 Look at Figure 15.7. Determine, from research, the main reasons for the almost linear expansion in the processing speed of computers over the last 25 years. The data in the graph compares *Number of calculations per second* against *Year*.

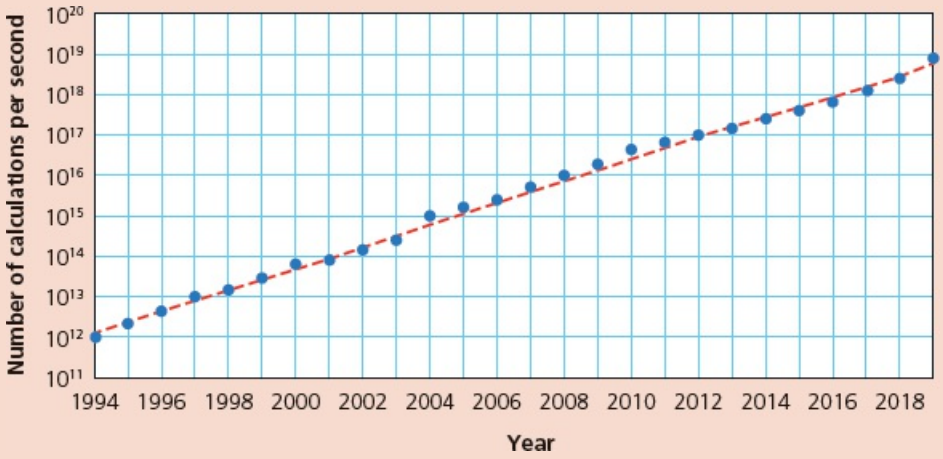


Figure 15.7

ACTIVITY 15A

1 a) Describe why RISC is an important development in processor technology.

b) Describe the main differences between RISC and CISC technologies.

2 a) What is meant by the Von Neumann bottleneck?

b) How does the Von Neumann bottleneck impact on processor performance?

3 a) What are the main differences between cluster computers and massively parallel computers?

b) Describe one application which uses massively parallel computers. Justify your choice of answer.

4 A processor uses pipelining. The following instructions are to be input

1 LOAD A
2 LOAD B
3 LOAD C
4 ADD A,B,C
5 STORE D
6 OUT D

Draw a diagram to show how many clock cycles are needed for these six instructions to be carried out. Compare your answer to the number of clock cycles needed for a processor using sequential processing.

15.2 Boolean algebra and logic circuits

Key terms

Half adder circuit – carries out binary addition on two bits giving sum and carry.

Full adder circuit – two half adders combined to allow the sum of several binary bits.

Combination circuit – circuit in which the output depends entirely on the input values.

Sequential circuit – circuit in which the output depends on input values produced from previous output values.

Flip-flop circuits – electronic circuits with two stable conditions using sequential circuits.

Cross-coupling – interconnection between two logic gates which make up a flip-flop.

Positive feedback – the output from a process which influences the next input value to the process.

Sum of products (SoP) – a Boolean expression containing AND and OR terms.

Gray codes – ordering of binary numbers such that successive numbers differ by one bit value only, for example, 00 01 11 10.

Karnaugh maps (K-maps) – a method used to simplify logic statements and logic circuits – uses Gray codes.

WHAT YOU SHOULD ALREADY KNOW

In Chapter 3, you learnt about logic gates and logic circuits. Try the following three questions to refresh your memory before you start to read the second part of this chapter.

1 Produce a truth table for the logic circuit shown in Figure 15.8.

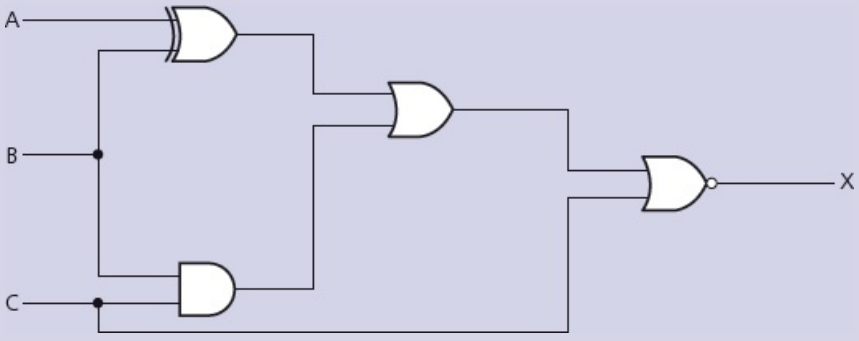


Figure 15.8

2 Draw a simplified version of the logic circuit shown in Figure 15.9 and write the Boolean expressions to represent Figure 15.9 and your simplified version.

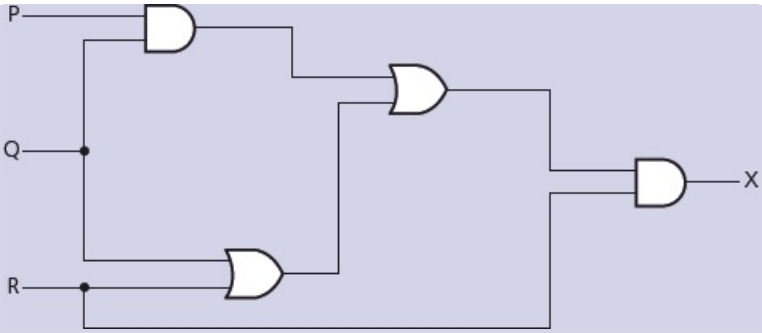


Figure 15.9

3 The warning light on a car comes on (= 1) if either one of three conditions occur

- sensor1 **AND** sensor2 detect a fault (give an input of 1) **OR**
- sensor2 **AND** sensor3 detect a fault (give an input of 1) **OR**
- sensor1 **AND** sensor3 detect a fault (give an input of 1).

a) Write a Boolean expression to represent the above problem.

b) Give the logic circuit to represent the above system.

c) Produce a truth table and check your answers to parts a) and b) agree.

15.2.1 Boolean algebra

Boolean algebra is named after the mathematician George Boole. It is a form of algebra linked to logic circuits and is based on the two statements:

TRUE (1)

FALSE (0)

The notation used in this book to represent these two Boolean operators is:

$\bar{A}$	which is also written as NOT A
A.B	which is also written as A AND B
A + B	which is also written as A OR B

Table 15.2 summarises the rules that govern Boolean algebra. It also includes De Morgan's Laws. Also note that, in Boolean algebra, 1 + 1 = 1, 1 + 0 = 1, and $\overline{\bar{A}} = A$ (remember your logic gate truth tables in Chapter 3).

Commutative Laws	A + B = B + A	A.B = B.A
Associative Laws	A + (B + C) = (A + B) + C	A.(B.C) = (A.B).C
Distributive Laws	A.(B + C) = (A.B) + (A.C) (A + B).(A + C) = A + B.C	A + (B.C) = (A + B).(A + C)
Tautology/Idempotent Laws	A.A = A	A + A = A
Tautology/Identity Laws	1.A = A	0 + A = A
Tautology/Null Laws	0.A = 0	1 + A = 1
Tautology/Inverse Laws	A.$\bar{A}$ = 0	A + $\bar{A}$ = 1
Absorption Laws	A.(A + B) = A A + A.B = A	A + (A.B) = A A + $\bar{A}$.B = A + B
De Morgan's Laws	$\overline{(A.B)} = \bar{A} + \bar{B}$	$\overline{(A + B)} = \bar{A}.\bar{B}$

Table 15.2 The rules that govern Boolean algebra

Table 15.3 shows proof of De Morgan's Laws. Since the last two columns in each section are identical, then the two De Morgan's Laws hold true.

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$	$\overline{A.B}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Both columns have the same values

A	B	$\bar{A}$	$\bar{B}$	$\bar{A}.\bar{B}$	$\overline{A + B}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Both columns have the same values

Table 15.3 Proof of De Morgan's Laws

Simplification using Boolean algebra

Example 15.1

Simplify $A + B + \bar{A} + \bar{B}$

Solution

Using the *associate laws* we have: $A + B + \bar{A} + \bar{B} \Rightarrow (A + \bar{A}) + (B + \bar{B})$

Using the *inverse laws* we have: $(A + \bar{A}) = 1$ and $(B + \bar{B}) = 1$

Therefore, we have 1 + 1, which is simply $1 \Rightarrow A + B + \bar{A} + \bar{B} = 1$

Example 15.2

Simplify $A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$

Solution

Rewrite the expression as: $A.B.C + (\bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C})$

This becomes: $(A.B.C + \bar{A}.B.C) + (A.B.C + A.\bar{B}.C) + (A.B.C + A.B.\bar{C})$

which transforms to: $B.C.(A + \bar{A}) + A.C.(B + \bar{B}) + A.B.(C + \bar{C})$

Since $A + \bar{A}$, $B + \bar{B}$ and $C + \bar{C}$ are all equal to 1

then we have: $B.C.1 + A.C.1 + A.B.1 \Rightarrow B.C + A.C + A.B$

ACTIVITY 15B

Simplify the following logic expressions showing all the stages in your simplification.

a) $A.\bar{C} + B.\bar{C}.D + A.\bar{B}.C + A.C.D$

b) $B + \bar{A}.\bar{B} + A.C.D + A.\bar{C}$

c) $\bar{A}.B.C + A.B.\bar{C} + A.B.C + \bar{A}.B.\bar{C}$

d) $\bar{A}.(A + B) + (B + A.A).(A + \bar{B})$

e) $(A + C).(A.D + A.\bar{D}) + A.C + C$

15.2.2 Further logic circuits

Half adder circuit and full adder circuit

In Chapter 3, the use of logic gates to create logic circuits to carry out specific tasks was discussed in much detail. Two important logic circuits used in computers are

- the **half adder circuit**
- the **full adder circuit**.

Half adder

One of the basic operations in any computer is binary addition. The half adder circuit is the simplest circuit. This carries binary addition on 2 bits generating two outputs

- the sum bit (S)
- the carry bit (C).

Consider 1 + 1. It will give the result 1 0 (denary value 2). The '1' is the carry and '0' is the sum. Table 15.4 shows this as a truth table.

INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 15.4

Figure 15.10 shows how this is often shown in graphic form (left) or as a logic circuit (right):

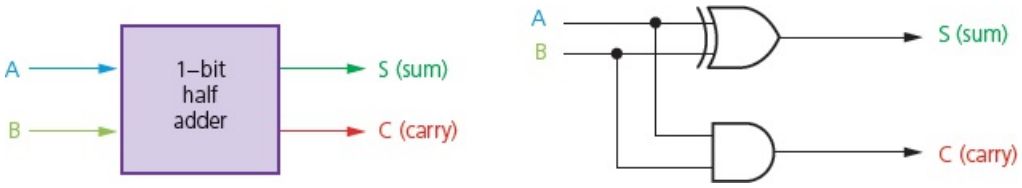


Figure 15.10

Other logic gates can be used to produce the half adder (see below).

As you have probably guessed already, the half adder is unable to deal with the addition of several binary bits (for example, an 8-bit byte). To enable this, we have to consider the full adder circuit.

Full adder

Consider the following sum using 5-bit numbers.

A	0	1	[1]	1	0	
B	0	0	[0]	1	1	
S	1	0	[0]	0	1	← this is the sum produced from the addition
C	1	1	[1]			← this is the carry from the previous bit position

Figure 15.11

The sum shows how we have to deal with CARRY from the previous column. There are three inputs to consider in this third column, for example, A = 1, B = 0 and C = 1 (S = 0).

This is why we need to join two half adders together to form a full adder:

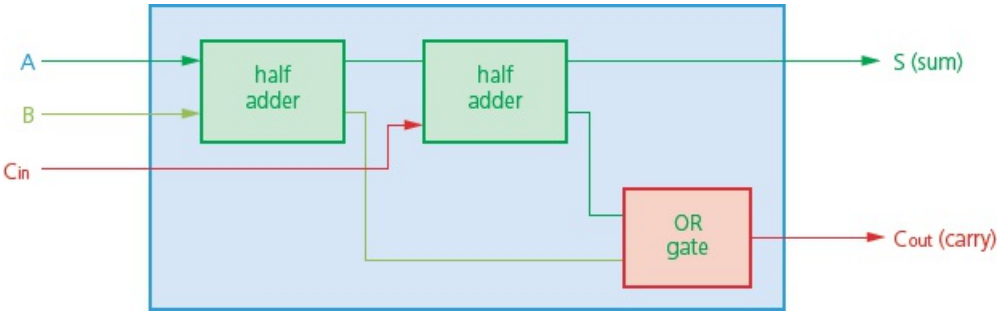


Figure 15.12

This has an equivalent logic circuit; there are a number of ways of doing this. For example, the following logic circuit uses OR, AND and XOR logic gates.

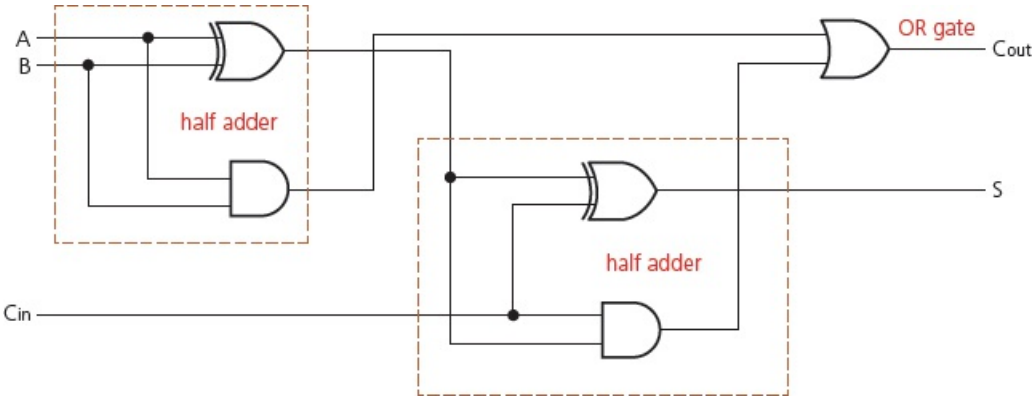


Figure 15.13

Table 15.5 is the truth table for the full adder circuit.

INPUTS			OUTPUTS	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 15.5

As with the half adder circuits, different logic gates can be used to produce the full adder circuit.

The full adder is the basic building block for multiple binary additions. For example, Figure 15.14 shows how two 4-bit numbers can be summed using four full adder circuits.

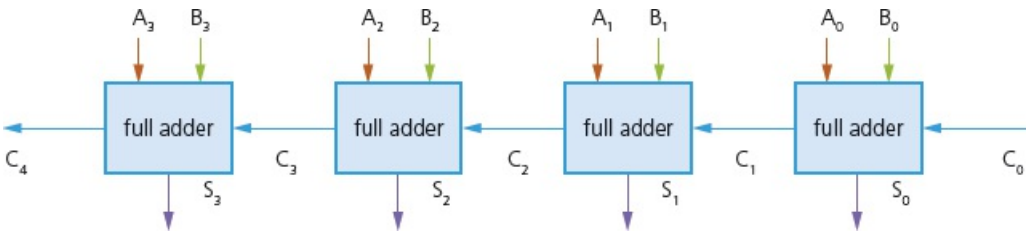


Figure 15.14

ACTIVITY 15C

1 a) Produce a half adder circuit using NAND gates only.

b) Generate a truth table for your half adder circuit in part a) and confirm it matches the one shown in Section 15.2.2.

2 a) Produce a full adder circuit using NAND gates only.

b) Generate a truth table for your full adder circuit in part a) and confirm it matches the one shown in Section 15.2.2.

EXTENSION ACTIVITY 15C

1 Find out why NAND gates are used to produce logic circuits even though they often increase the complexity and size of the overall circuit.

2 Produce half adder and full adder circuits using NOR gates only.

15.2.3 Flip-flop circuits

All of the logic circuits you have encountered up to now are **combination circuits** (the output depends entirely on the input values).

We will now consider a second type of logic circuit, known as a **sequential circuit** (the output depends on the input value produced from a previous output value).

Examples of sequential circuits include **flip-flop circuits**. This chapter will consider two types of flip-flops: SR flip-flops and JK flip-flops.

SR flip-flops

SR flip-flops consist of two **cross-coupled** NAND gates (note: they can equally well be produced from NOR gates). The two inputs are labelled 'S' and 'R', and the two outputs are labelled 'Q' and 'Q̄' (remember Q̄ is equivalent to NOT Q).

In this chapter, we will use SR flip-flop circuits constructed from NOR gates, as shown in Figure 15.15.

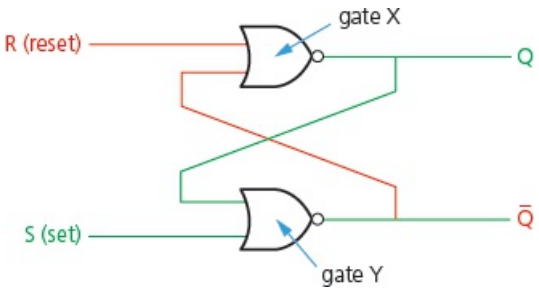
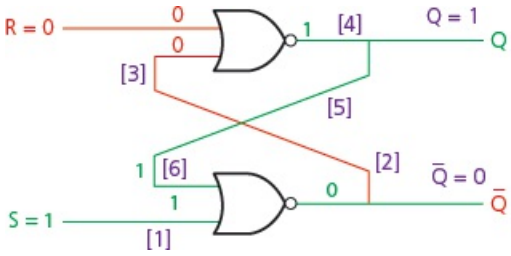


Figure 15.15 SR flip-flop circuit

The output from gate 'X' is Q and the output from gate 'Y' is Q̄. The inputs to gate 'X' are R and Q̄ (shown in red on Figure 15.15); the inputs to gate 'Y' are S and Q (shown in green on Figure 15.15). The output from each NOR gate gives a form of **positive feedback** (known as cross-coupling, as mentioned earlier).

We will now consider the truth table to match our SR flip-flop using the initial states of R = 0, S = 1 and Q = 1. The sequence of the stages in the process is shown in Figure 15.16.



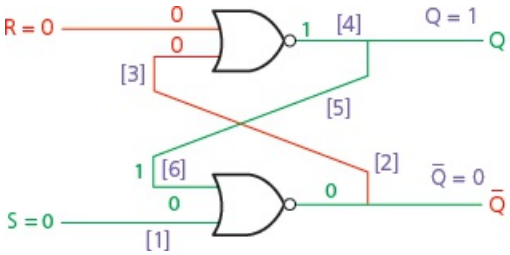
Sequence: [1] → [2] → [3] → [4] → [5] → [6]

Which gives:

S	R	Q	Q̄
1	0	1	0

Figure 15.16

Now consider what happens if we change the value of S from 1 to 0.



Sequence: [1] → [2] → [3] → [4] → [5] → [6]

Which gives:

S	R	Q	$\bar{Q}$
0	0	1	0

Figure 15.17

The reader is left to consider the other options which lead to the truth table, Table 15.6, for the flip-flop circuit.

	INPUTS		OUTPUTS		Comment
	S	R	Q	$\bar{Q}$	
(a)	1	0	1	0	
(b)	0	0	1	0	following S = 1 change
(c)	0	1	0	1	
(d)	0	0	0	1	following R = 1 change
(e)	1	1	0	0	

Table 15.6

Explanation

S = 1, R = 0, Q = 1, $\bar{Q}$ = 0 is the set state in this example

S = 0, R = 0, Q = 1, $\bar{Q}$ = 0 is the reset state in this example

S = 0, R = 1, Q = 0, $\bar{Q}$ = 1 here the value of Q in line **(b)** remembers the value of Q from line **(a)**; the value of Q in line **(d)** remembers the value of Q in line **(c)**

S = 0, R = 0, Q = 0, $\bar{Q}$ = 1 R changes from 1 to 0 and has no effect on outputs (these values are remembered from line **(c)**)

S = 1, R = 1, Q = 0, $\bar{Q}$ = 0 this is an invalid case since $\bar{Q}$ should be the complement (opposite) of Q.

The truth table shows how an input value of S = 0 and R = 0 causes no change to the two output values; S = 0 and R = 1 reverses the two output values; S = 1 and R = 0 always gives Q = 1 and $\bar{Q}$ = 0 which is the set value.

The truth table shows that SR flip-flops can be used as a storage/memory device for one bit; because a value can be remembered but can also be changed it could be used as a component in a memory device such as a RAM chip.

It is important that the fault condition in line **(e)** is considered when designing and developing storage/memory devices.

JK flip-flops

The SR flip-flop has the following problems:

- Invalid S, R conditions (leading to conflicting output values) need to be avoided.
- If inputs do not arrive at the same time, the flip-flop can become unstable.

To overcome such problems, the JK flip-flop has been developed. A clock and additional gates are added, which help to synchronise the two inputs and also prevent the illegal states shown in line **(e)** of Table 15.6. The addition of the synchronised input gives four possible input conditions to the JK flip-flop

- 1
- 0
- no change
- toggle (which takes care of the invalid S, R states).

The JK flip-flop is represented as shown in Figure 15.18.

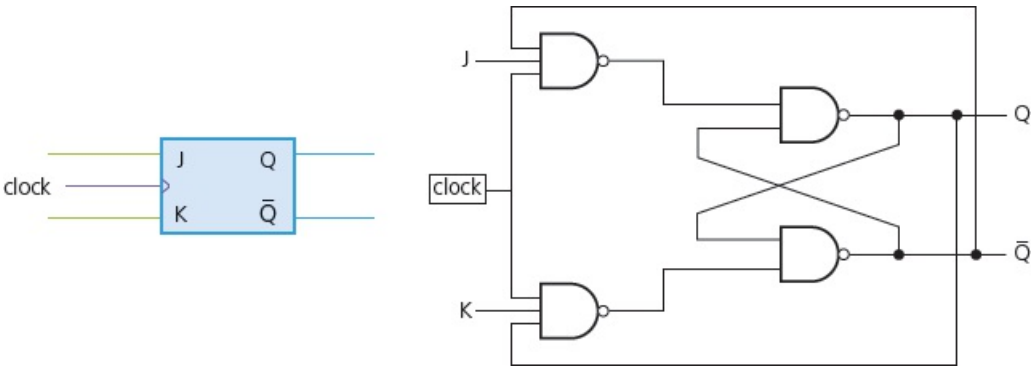


Figure 15.18 JK flip-flop symbol (left) and JK flip-flop using NAND gates only (right)

Table 15.7 is the simplified truth table for the JK flip-flop.

J	K	Value of Q before clock pulse	Value of Q after clock pulse	OUTPUT
0	0	0	0	Q is unchanged after clock pulse
0	0	1	1	
1	0	0	1	Q = 1
1	0	1	1	
0	1	0	0	Q = 0
0	1	1	0	
1	1	0	1	Q value toggles between 0 and 1
1	1	1	0	

Table 15.7

EXTENSION ACTIVITY 15D

1 Find out how JK flip-flops can be used as shift registers and binary counters in a computer.

2 Where else in computer architecture are flip-flop circuits used? Find out why they are used in each case you describe.

- When J = 0 and K = 0, there is no change to the output value of Q.
- If the values of J or K change, then the value of Q will be the same as the value of J ($\bar{Q}$ will be the value of K).
- When J = 1 and K = 1, the Q-value toggles after *each* clock pulse, thus preventing illegal states from occurring (in this case, toggle means the flip-flop will change from the 'Set' state to the 'Reset' state or the other way round).

Use of JK flip-flops

- Several JK flip-flops can be used to produce shift registers in a computer.
- A simple binary counter can be made by linking up several JK flip-flop circuits (this requires the toggle function).

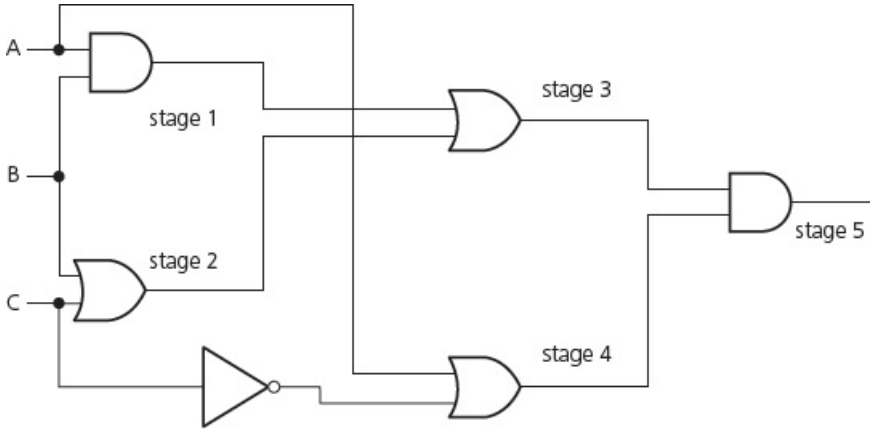
15.2.4 Boolean algebra and logic circuits

In Section 15.2.1, the concept of Boolean algebra was introduced. One of the advantages of this method is to represent logic circuits in the form of Boolean algebra.

It is possible to use the truth table and apply the **sum of products (SoP)**, or the Boolean expression can be formed directly from the logic circuit.

Example 15.3

Write down the Boolean expression to represent this logic circuit.



Solution

Stage 1: A AND B

Stage 2: B OR C

Stage 3: stage 1 OR stage 2 ⇒ (A AND B) OR (B OR C)

Stage 4: A OR (NOT C)

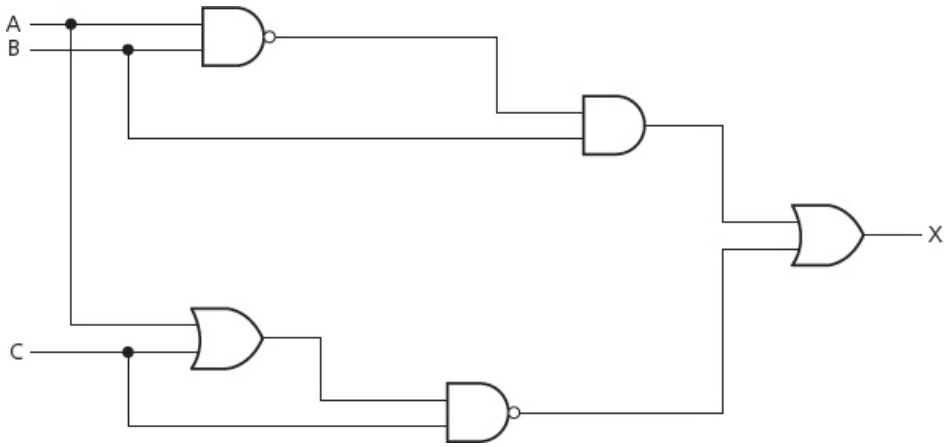
Stage 5: stage 3 AND stage 4

⇒ ((A AND B) OR (B OR C)) AND (A OR (NOT C))

Written in Boolean algebra form: $((A.B) + (B + C)).(A + \overline{C})$

Example 15.4

Write the Boolean expression which represents this logic circuit.



Solution

In this example, we will first produce the truth table and then generate the Boolean expression from the truth table, Table 15.8.

INPUTS			OUTPUT
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

To produce the Boolean expression from the truth table, we only consider those rows where the output (X) is 1:

$(\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C + \bar{A}.\bar{B}.\bar{C} + A.B.\bar{C})$

If we apply the Boolean algebra laws, we get:

$(\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C) + (\bar{A}.\bar{B}.\bar{C} + A.B.\bar{C})$

$\Rightarrow ((\bar{A}.\bar{B}.\bar{C} + \bar{A}.B.\bar{C}) + (\bar{A}.\bar{B}.\bar{C} + A.\bar{B}.\bar{C})) + (\bar{A}.B.C + A.B.\bar{C})$

$\Rightarrow \bar{A}.\bar{C}.(\bar{B} + B) + \bar{B}.\bar{C}.(\bar{A} + A) + (\bar{A}.B.C + A.B.\bar{C})$

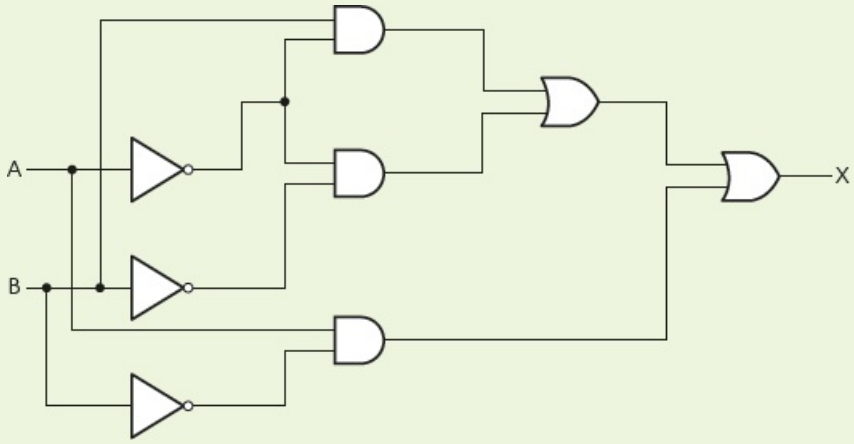
$\Rightarrow \bar{A}.\bar{C} + \bar{B}.\bar{C} + \bar{A}.B.C + A.B.\bar{C}$

Therefore, written as a Boolean expression: $\bar{A}.\bar{C} + \bar{B}.\bar{C} + \bar{A}.B.C + A.B.\bar{C}$

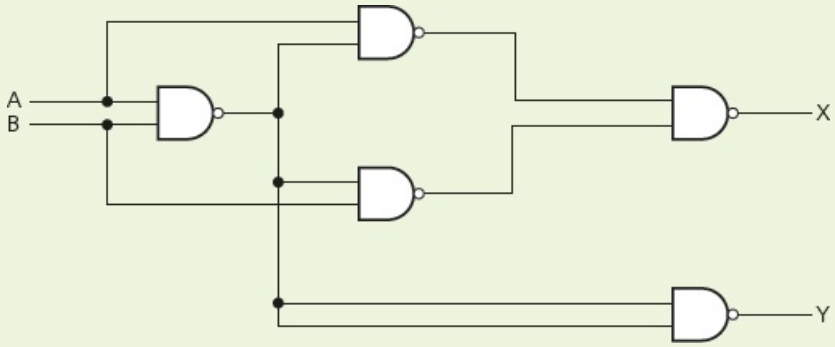
We therefore end up with a simplified Boolean expression which has the same effect as the original logic circuit. The reader is left the task of producing the truth table from the above expression to confirm they are both the same.

ACTIVITY 15D

1 Produce simplified Boolean expressions for the logic circuits in Figure 15.21 (you can do this directly from the logic circuit or produce the truth table first).



2 Produce simplified Boolean expressions for the logic circuits in Figure 15.22 (you can do this directly from the logic circuit or produce the truth table first).



15.2.5 Karnaugh maps (K-maps)

In the previous activities, it was frequently necessary to simplify Boolean expressions. Sometimes, this can be a long and complex process. **Karnaugh maps** were developed to help simplify logic expressions/circuits.

EXTENSION ACTIVITY 15E

Karnaugh maps make use of **Gray codes**. Find out the origin of Gray codes and other applications of the code.

Example 15.5

Produce a Boolean expression for the truth table for the NAND gate.

INPUTS		OUTPUT
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Solution

Using sum of products gives the following expression:

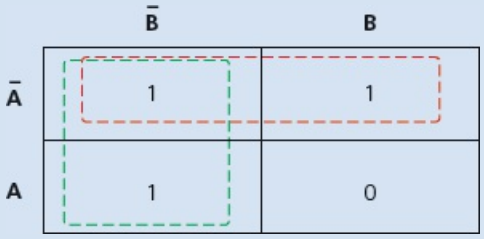
$\bar{A}.\bar{B} + \bar{A}.B + A.\bar{B}$

Boolean algebra rules produce the simplified expression:

$\bar{A} + \bar{B}$

Using Karnaugh maps is a much simpler way to do this.

Each group in the Karnaugh map in Figure 15.23 combines output values where X = 1.



Thus, $\bar{A}.\bar{B} = 1$, $\bar{A}.B = 1$ and $A.\bar{B} = 1$

The red ring shows $\bar{A}$ as

and the green ring shows $\bar{B}$ as

giving $\bar{A} + \bar{B}$.

As you might expect, there are a number of rules governing Karnaugh maps.

Karnaugh map rules

- The values along the top and the bottom follow Gray code rules.
- Only cells containing a 1 are taken account of.
- Groups can be a row, a column or a rectangle.
- Groups must contain an even number of 1s (2, 4, 6, and so on).
- Groups should be as large as possible.
- Groups may overlap within the above rules.
- Single values can be regarded as a group even if they cannot be combined with other values to form a larger group.
- The final Boolean expression can only consider those values which remain *constant* within the group (that is, remain a 1 or a 0 throughout the group).

Example 15.6

Produce a Boolean expression for the truth table.

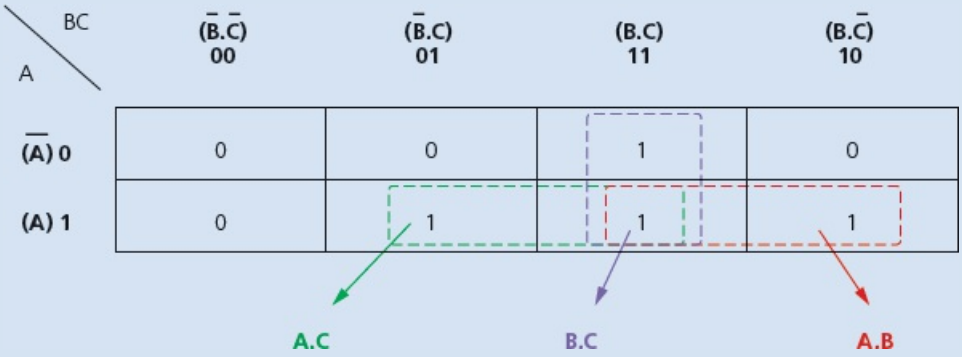
INPUTS			OUTPUT
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Solution

Sum of products gives:

$A.B.C + \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$

We can now produce the following Karnaugh map to represent this truth table (each 1 value in the K-map represents the above sum of products; so there will be four 1-values in the K-map, where A and BC intersect, where $\bar{A}$ and BC intersect, where A and $B\bar{C}$ intersect, and where A and $B\bar{C}$ intersect):



- Green ring: A remains 1, B changes from 0 to 1 and C remains 1 ⇒ A.C
- Purple ring: A changes from 0 to 1, B remains 1 and C remains 1 ⇒ B.C
- Red ring: A remains 1, B remains 1 and C changes from 1 to 0 ⇒ A.B

This gives the simplified Boolean expression: A.C + B.C + A.B

Example 15.7

Produce a Boolean expression for the truth table.

INPUTS				OUTPUT
A	B	C	D	X
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Sum of products
$\bar{A}.\bar{B}.\bar{C}.\bar{D}$
$\bar{A}.\bar{B}.\bar{C}.D$
$\bar{A}.\bar{B}.C.\bar{D}$
$\bar{A}.\bar{B}.C.D$
$\bar{A}.B.\bar{C}.D$
$A.\bar{B}.\bar{C}.D$
$A.B.\bar{C}.D$

Solution

The sum of products is shown in the right-hand column. This produces the Karnaugh map shown.

CD \ AB	$(\bar{A}.\bar{B})$ 00	$(\bar{A}.B)$ 01	$(A.B)$ 11	$(A.\bar{B})$ 10
$(\bar{C}.\bar{D})$ 00	1	0	0	0
$(\bar{C}.D)$ 01	1	1	1	1
$(C.D)$ 11	1	0	0	0
$(C.\bar{D})$ 10	1	0	0	0

This gives $\bar{A}.\bar{B} + \bar{C}.D$

Notice the following possible K-map options:

This gives the value D since the values of A and B change and the value of C changes (0 to 1); only D is constant at 1.

CD \ AB	$(\bar{A}.\bar{B})$ 00	$(\bar{A}.B)$ 01	$(A.B)$ 11	$(A.\bar{B})$ 10
$(\bar{C}.\bar{D})$ 00	1	0	0	0
$(\bar{C}.D)$ 01	1	1	1	1
$(C.D)$ 11	1	1	1	1
$(C.\bar{D})$ 10	0	0	0	0

Figure 15.19

Columns 1 and 4 can be joined to form a *vertical cylinder*. The values of both C and D change, the value of A changes, the value of B is constant at 0 giving: $\bar{B}$

CD \ AB	$(\bar{A}.\bar{B})$ 00	$(\bar{A}.B)$ 01	$(A.B)$ 11	$(A.\bar{B})$ 10
$(\bar{C}.\bar{D})$ 00	1	0	0	1
$(\bar{C}.D)$ 01	1	0	0	1
$(C.D)$ 11	1	0	0	1
$(C.\bar{D})$ 10	1	0	0	1

Figure 15.20

The two 1-values can be combined to form a *horizontal cylinder*; values of A and B are constant at 0 and 1 respectively; the value of D is constant at 0; values of C changes from 0 to 1; giving: $\bar{A}.B.\bar{D}$

CD \ AB	$(\bar{A}.\bar{B})$ 00	$(\bar{A}.B)$ 01	$(A.B)$ 11	$(A.\bar{B})$ 10
$(\bar{C}.\bar{D})$ 00	0	1	0	0
$(\bar{C}.D)$ 01	0	0	0	0
$(C.D)$ 11	0	0	0	0
$(C.\bar{D})$ 10	0	1	0	0

Figure 15.21

The four 1-values can be combined at the four corners; value B is constant at 0 and value D is also constant at 0, giving: $\bar{B}.\bar{D}$

CD \ AB	$(\bar{A}.\bar{B})$ 00	$(\bar{A}.B)$ 01	$(A.B)$ 11	$(A.\bar{B})$ 10
$(\bar{C}.\bar{D})$ 00	1	0	0	1
$(\bar{C}.D)$ 01	0	0	0	0
$(C.D)$ 11	0	0	0	0
$(C.\bar{D})$ 10	1	0	0	1

Figure 15.22

ACTIVITY 15E

1 a) Draw the truth table for the Boolean expression:
$\bar{A}.\bar{B}.C.D + \bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.C.D + A.B.\bar{C}.D + A.B.C.D$

b) Draw the Karnaugh map for the Boolean expression in part a).

c) Draw a logic circuit for the simplified Boolean expression using AND or OR gates only.

2 a) Draw the truth table for the Boolean expression:
$\bar{A}.B.C + A.B.\bar{C} + A.B.C + \bar{A}.B.\bar{C}$

b) Draw the Karnaugh map for the expression in part a) and hence write a simplified Boolean expression.

3 Four binary signals (A, B, C and D) are used to define an integer in the hexadecimal range (0 to F). The decimal digit satisfies one of the following criteria (that is, gives an output value of X = 1):
X = 1 if
A = 0
B = C, but A ≠ B and A ≠ C
B = 0, C = 0

a) Complete the truth table (with headings A, B, C, D, X) for the above criteria.

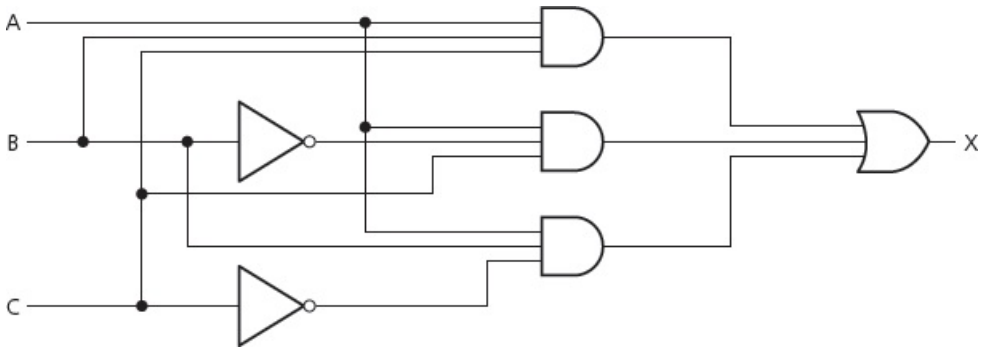
b) Construct the Karnaugh map to represent the above criteria and produce a simplified Boolean expression.

c) Hence, draw an *efficient* logic circuit using AND, OR and NOT gates only. Indicate which input value is not actually required by the logic circuit.

End of chapter questions

1 a) Write down the Boolean expression to represent the logic circuit below.

[3]



b) Produce the Karnaugh map to represent the above logic circuit and hence write down a simplified Boolean expression.

[3]

c) Draw a simplified logic circuit from your Boolean expression in part b) using AND and OR gates only.

[2]

2 a) Consider the following truth table.

INPUTS				OUTPUT
A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

i) Draw a Karnaugh map from this truth table.

[3]

ii) Use your Karnaugh map from part a) i) to produce a Boolean expression.

[4]

b) Use the laws of Boolean algebra to simplify:

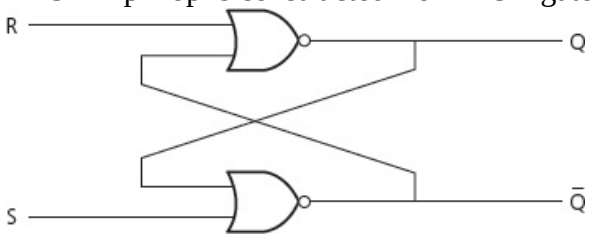
i) $(A + C).(A.D + A.\overline{D}) + A.C + C$

[2]

ii) $\bar{A}.(A + B) + (B + A.A).(A + \bar{B})$

[2]

3 a) An SR flip-flop is constructed from NOR gates:



i) Complete the truth table for the SR flip-flop.

[4]

ii) One of the S, R combinations in the truth table should not be allowed to occur. State the values of S and R that should not be allowed to occur. Explain your choice of values.

[3]

INPUTS		OUTPUTS	
S	R	Q	$\bar{Q}$
1	0	1	0
0	0		
0	1		
0	0		
1	1		

b) JK flip-flops are another type of flip-flop.

i) State the three inputs to a JK flip-flop.

[1]

ii) Give an advantage of using JK flip-flops.

[1]

iii) Describe two uses of JK flip-flops in computers.

[2]

4 a) Describe four types of processors used in parallel processing.

[4]

b) A hardware designer decided to look into the use of parallel processing. Describe three features of parallel processing she needs to consider when designing her new system.

[3]

c) A computer system uses pipelining. An assembly code program being run has eight instructions. Compare the number of clock cycles required when using pipelining compared to a sequential computer.

[3]

5 a) Four descriptions and four types of computer architecture are shown below.
Draw a line to connect each description to the appropriate type of computer architecture.

[4]

Description	Computer architecture
A computer that does not have the ability for parallel processing.	SIMD
The processor has several ALUs. Each ALU executes the same instructions but on different data.	MISD
There are several processors. Each processor executes different instructions drawn from a common pool. Each processor operates on different data drawn from a common pool.	SISD
There is only one processor executing one set of instructions on a single set of data.	MIMD

b) In a massively parallel computer explain what is meant by:

i) Massive

[1]

ii) Parallel

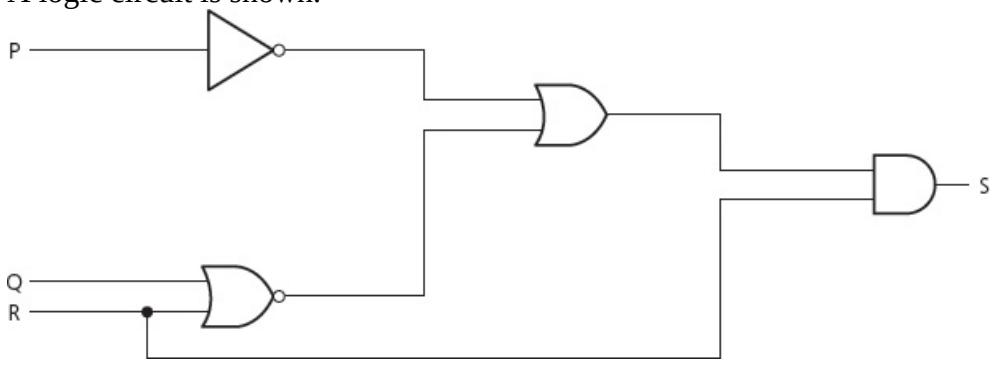
[1]

c) There are both hardware and software issues that have to be considered for parallel processing to succeed. Describe **one** hardware and **one** software issue.

[4]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q4 November 2015

6 A logic circuit is shown.



a) Write the Boolean expression corresponding to this logic circuit.

[4]

b) Copy and complete the truth table for this logic circuit.

[2]

P	Q	R	Working space	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

c) i) Copy and complete the Karnaugh map (K-map) for the truth table in part b).

[1]

		PQ			
		00	01	11	10
R	0				
	1				

The K-map can be used to simplify the function in part a).

ii) Draw loops around appropriate groups to produce an optional sum-of-products.

[1]

iii) Write a simplified sum-of-products expression, using your answer to part ii).

[1]

d) One Boolean identity is:
(A + B).C = A.C + B.C
Simplify the expression for S in part a) to the expression for S in part c) iii). You should use the given identity and De Morgan's Laws.

[3]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q3 June 2017

16 System software and virtual machines

In this chapter, you will learn about

- how an operating system (OS) can maximise the use of computing resources
- how an operating system user interface hides the complexities of hardware from the user
- processor management (including multitasking; process states of running, ready and blocked; scheduling routines such as round robin, shortest job first, first come first served and shortest remaining time first; how an OS kernel acts as an interrupt handler; how interrupt handling is used to manage low-level scheduling)
- memory management (including paging; segmentation; differences between paging and segmentation; virtual memory; how pages can be replaced; disk thrashing)
- the concept of virtual machines (including the role, benefits and limitations of virtual machines)
- how an interpreter can execute programs without producing a translated version
- various stages in the compilation of a program (including lexical analysis; syntax analysis; code generation; optimisation)
- the grammar of a language being expressed using syntax diagrams such as Backus-Naur (BNF) notation
- how Reverse Polish notation (RPN) can be used to carry out the evaluation of expressions.

16.1 Purposes of an operating system (OS)

WHAT YOU SHOULD ALREADY KNOW

Try these six questions before you read the first part of this chapter.

1 Name the key management tasks carried out by a typical operating system.

2 Explain why

a) in general, a computer needs an operating system

b) some devices using an embedded microprocessor do not always need an operating system.

3 Describe typical utility software provided with an operating system.

4 Describe the main differences between a command line user interface (CLI) and a graphical user interface (GUI).

5 Explain the need for interface software such as printer drivers or mouse drivers.

6 **a)** What is meant by a *library of files*?

b) Explain how dynamic link library (DLL) files differ from static library routines.

Key terms

Bootstrap – a small program that is used to load other programs to 'start up' a computer.

Scheduling – process manager which handles the removal of running programs from the CPU and the selection of new processes.

Direct memory access (DMA) controller – device that allows certain hardware to access RAM independently of the CPU.

Kernel – the core of an OS with control over process management, memory management, interrupt handling, device management and I/O operations.

Multitasking – function allowing a computer to process more than one task/process at a time.

Process – a program that has started to be executed.

Preemptive – type of scheduling in which a process switches from running state to steady state or from waiting state to steady state.

Quantum – a fixed time slice allocated to a process.

Non-preemptive – type of scheduling in which a process terminates or switches from a running state to a waiting state.

Burst time – the time when a process has control of the CPU.

Starve – to constantly deprive a process of the necessary resources to carry out a task/process.

Low level scheduling – method by which a system assigns a processor to a task or process based on the priority level.

Process control block (PCB) – data structure which contains all the data needed for a process to run.

Process states – running, ready and blocked; the states of a process requiring execution.

Round robin (scheduling) – scheduling algorithm that uses time slices assigned to each process in a job queue.

Context switching – procedure by which, when the next process takes control of the CPU, its previous state is reinstated or restored.

Interrupt dispatch table (IDT) – data structure used to implement an interrupt vector table.

Interrupt priority levels (IPL) – values given to interrupts based on values 0 to 31.

Optimisation (memory management) – function of memory management deciding which processes should be in main memory and where they should be stored.

Paging – form of memory management which divides up physical memory and logical memory into fixed-size memory blocks.

Physical memory – main/primary RAM memory.

Logical memory – the address space that an OS perceives to be main storage.

Frames – fixed-size physical memory blocks.

Pages – fixed-size logical memory blocks.

Page table – table that maps logical addresses to physical addresses; it contains page number, flag status, frame address and time of entry.

Dirty – term used to describe a page in memory that has been modified.

Translation lookaside buffer (TLB) – this is a memory cache which can reduce the time taken to access a user memory location; it is part of the memory management unit.

Segments memory– variable-size memory blocks into which logical memory is split up.

Segment number – index number of a segment.

Segment map table – table containing the segment number, segment size and corresponding memory location in physical memory: it maps logical memory segments to physical memory.

Virtual memory – type of paging that gives the illusion of unlimited memory being available.

Swap space – space on HDD used in virtual memory, which saves process data.

In demand paging – a form of data swapping where pages of data are not copied from HDD/SSD into RAM until they are actually required.

Disk thrashing – problem resulting from use of virtual memory. Excessive swapping in and out of virtual memory leads to a high rate of hard disk read/write head movements thus reducing processing speed.

Thrash point – point at which the execution of a process comes to a halt since the system is busier paging in/out of memory rather than actually executing them.

Page replacement – occurs when a requested page is not in memory and a free page cannot be

used to satisfy allocation.

Page fault – occurs when a new page is referred but is not yet in memory.

First in first out (FIFO) page replacement – page replacement that keeps track of all pages in memory using a queue structure. The oldest page is at the front of the queue and is the first to be removed when a new page is added.

Belady's anomaly – phenomenon which means it is possible to have more page faults when increasing the number of page frames.

Optimal page replacement – page replacement algorithm that looks forward in time to see which frame to replace in the event of a page fault.

Least recently used (LRU) page replacement – page replacement algorithm in which the page which has not been used for the longest time is replaced.

16.1.1 How an operating system can maximise the use of computer resources

When a computer is first switched on, the basic input/output system (BIOS) – which is often stored on the ROM chip – starts off a **bootstrap** program. The bootstrap program loads part of the operating system into main memory (RAM) from the hard disk/SSD and initiates the start-up procedures. This process is less obvious on tablets and mobile phones – they also use RAM, but their main internal memory is supplied by flash memory; this explains why the start-up of tablets and mobile phones is almost instantaneous.

The flash memory is split into two parts.

1. The part where the OS resides. It is read only. This is why the OS can be updated by the mobile phone/tablet manufacturers but the user cannot interfere with the software or 'steal' memory from this part of memory.
2. The part where the apps (and associated data) are stored. The user does not have direct access to this part of memory either.

The RAM is where the apps are executed and where data currently in use is stored.

One operating system task is to maximise the utilisation of computer resources. Resource management can be split into three areas

1. the CPU
2. memory
3. the input/output (I/O) system.

Resource management of the CPU involves the concept of **scheduling** to allow for better utilisation of CPU time and resources (see Sections 16.1.2 and 16.1.4). Regarding input/output operations, the operating system will need to deal with

- any I/O operation which has been initiated by the computer user
- any I/O operation which occurs while software is being run and resources, such as printers or disk drives, are requested.

Figure 16.1 shows how this links together using the internal bus structure (note that the diagram shows how it is possible to have direct data transfer between memory and I/O devices using DMA):

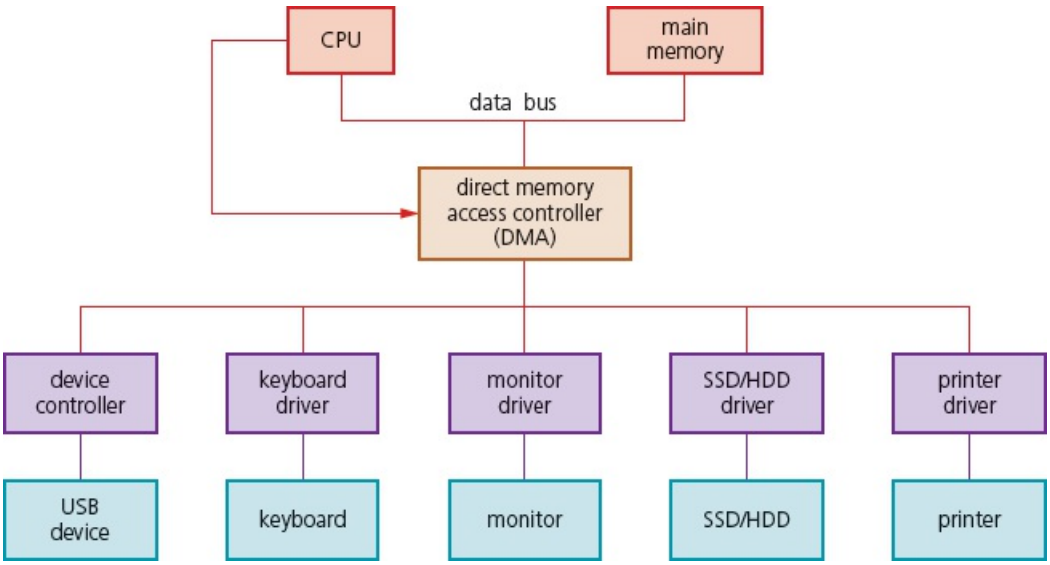


Figure 16.1

The **direct memory access (DMA) controller** is needed to allow hardware to access the main memory independently of the CPU. When the CPU is carrying out a programmed I/O operation, it is fully utilised during the entire read/write operations; the DMA frees up the CPU to allow it to carry out other tasks while the slower I/O operations are taking place.

- The DMA initiates the data transfers.
- The CPU carries out other tasks while this data transfer operation is taking place.
- Once the data transfer is complete, an interrupt signal is sent to the CPU from the DMA.

Table 16.1 shows how slow some I/O devices are when compared with a typical computer's clock speed of 2.7 GHz.

I/O device	Data transfer rate
disk	up to 100 Mbps
mouse	up to 120 bps
laser printer	up to 1 Mbps
keyboard	up to 50 bps

Table 16.1 Sample data transfer rates for some I/O devices

EXTENSION ACTIVITY 16A

An I/O device is connected to the main memory of a computer using a 16-bit data bus. The CPU is capable of executing 2×10^9 instructions per second. An instruction will use five processor cycles; three of these cycles are used by the data bus. A memory read/write operation will require one processor cycle.

Suppose the CPU is 80% utilised doing tasks that do not involve an I/O operation. Estimate the data transfer rate using the DMA.

The Kernel

The **kernel** is part of the operating system. It is the central component responsible for communication between hardware, software and memory. It is responsible for process management, device management, memory management, interrupt handling and input/output file communications, as shown in Figure 16.2.

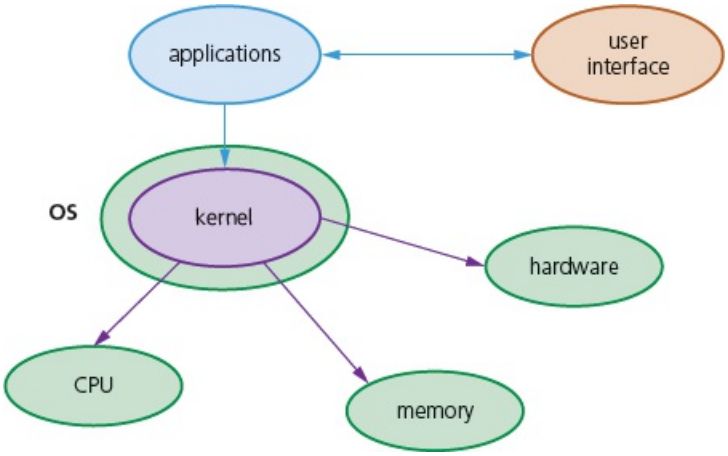


Figure 16.2

One of the most important tasks of an operating system is to hide the complexities of the hardware from the users. This can be done by

- using GUI interfaces rather than CLI (see Chapter 5)
- using device drivers (which simplifies the complexity of hardware interfaces)
- simplifying the saving and retrieving of data from memory and storage devices
- carrying out background utilities, such as virus scanning which the user can 'leave to its own devices'.

A simple example is transferring data from a hard disk to a floppy disk using a computer from the 1990s. This would require a command such as:

copy C:\windows\myfile.txt

Modern computers use a drag and drop method, which removes any of the complexities of interfacing directly with the computer. Simply dragging a file to a folder on a modern equivalent, such as a flash drive, would transfer the file; the operating system carries out all of the necessary processes – the user just moves the mouse.

16.1.2 Process management

Multitasking

Multitasking allows computers to carry out more than one task (known as a **process**) at a time. (A process is a program that has *started* to be executed.) Each of these processes will share common hardware resources. To ensure multitasking operates correctly (for example, making sure processes do not clash), scheduling is used to decide which processes should be carried out. Scheduling is considered in more depth in the next section.

Multitasking ensures the best use of computer resources by monitoring the state of each process. It should give the appearance that many processes are being carried out at the same time. In fact, the kernel overlaps the execution of each process based on scheduling algorithms. There are two types of multitasking operating systems

1 **preemptive** (processes are pre-empted after each time **quantum**)

2 **non-preemptive** (processes are pre-empted after a fixed time interval).

Table 16.2 summarises the differences between preemptive and non-preemptive.

Preemptive	Non-preemptive
resources are allocated to a process for a limited time	once the resources are allocated to a process, the process retains them until it has completed its **burst time** or the process has switched to a waiting state
the process can be interrupted while it is running	the process cannot be interrupted while running; it must first finish or switch to a waiting state
high priority processes arriving in the ready queue on a frequent basis can mean there is a risk that low priority processes may be **starved** of resources	if a process with a long burst time is running in the CPU, there is a risk that another process with a shorter burst time may be starved of resources
this is a more flexible form of scheduling	this is a more rigid form of scheduling

Table 16.2 The differences between preemptive and non-preemptive systems

Low level scheduling

Low level scheduling decides which process should next get the use of CPU time (in other words, following an OS call, which of the processes in the ready state can now be put into a running state based on their priorities). Its objectives are to maximise the system throughput, ensure response time is acceptable and ensure that the system remains stable at all times (has consistent behaviour in its delivery). For example, low level scheduling resolves situations in which there are conflicts between two processes requiring the same resource.

Suppose two apps need to use a printer; the scheduler will use interrupts, buffers and queues to

ensure only one process gets printer access – but it also ensures that the other process gets a share of the required resources.

Process scheduler

Process priority depends on

- its category (is it a batch, online or real time process?)
- whether the process is CPU-bound (for example, a large calculation such as finding 10 000! (10 000 factorial) would need long CPU cycles and short I/O cycles) or I/O bound (for example, printing a large number of documents would require short CPU cycles but very long I/O cycles)
- resource requirements (which resources does the process require, and how many?)
- the turnaround time, waiting time (see Section 16.1.3) and response time for the process
- whether the process can be interrupted during running.

Once a task/process has been given a priority, it can still be affected by

- the deadline for the completion of the process
- how much CPU time is needed when running the process
- the wait time and CPU time
- the memory requirements of the process.

16.1.3 Process states

A **process control block (PCB)** is a data structure which contains all of the data needed for a process to run; this can be created in memory when data needs to be received during execution time. The PCB will store

- current process state (ready, running or blocked)
- process privileges (such as which resources it is allowed to access)
- register values (PC, MAR, MDR and ACC)
- process priority and any scheduling information
- the amount of CPU time the process will need to complete
- a process ID which allows it to be uniquely identified.

A **process state** refers to the following three possible conditions

1 running
2 ready
3 blocked.

Figure 16.3 shows the link between these three conditions.

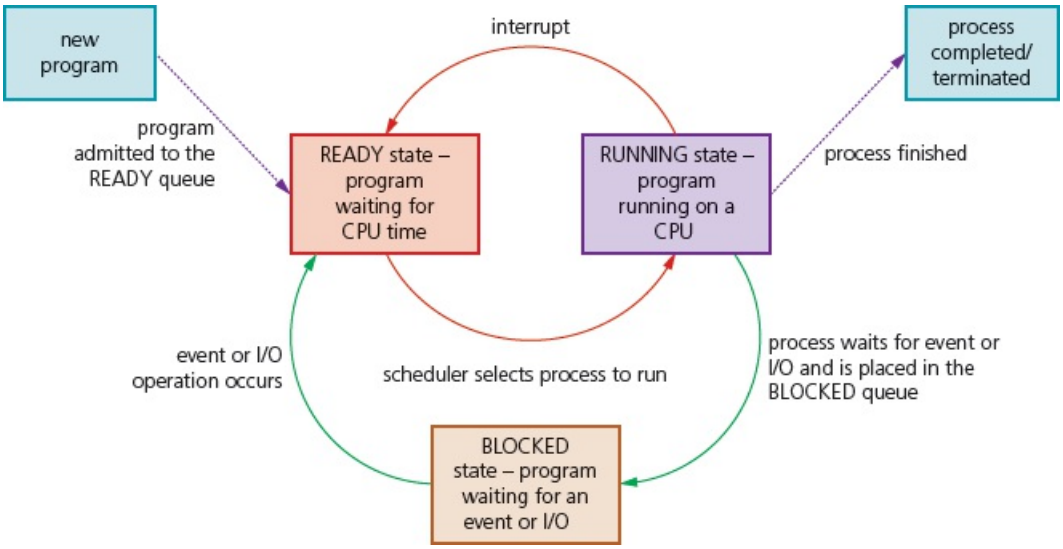


Figure 16.3

Table 16.3 summarises some of the conditions when changing from one process state to another.

Process states	Conditions
running state → ready state	a program is executed during its time slice; when the time slice is completed an interrupt occurs and the program is moved to the READY queue
ready state → running state	a process's turn to use the processor; the OS scheduler allocates CPU time to the process so that it can be executed
running state	the process needs to carry out an I/O operation; the OS scheduler places the

→ blocked state	process into the BLOCKED queue
blocked state → ready state	the process is waiting for an I/O resource; an I/O operation is ready to be completed by the process

Table 16.3

To investigate the concept of a READY QUEUE and a BLOCKED QUEUE a little further, we will consider what happens during a **round robin** process, in which each of the processes have the same priority.

Supposing two processes, P1 and P2, have completed. The current status is shown in Figure 16.4.

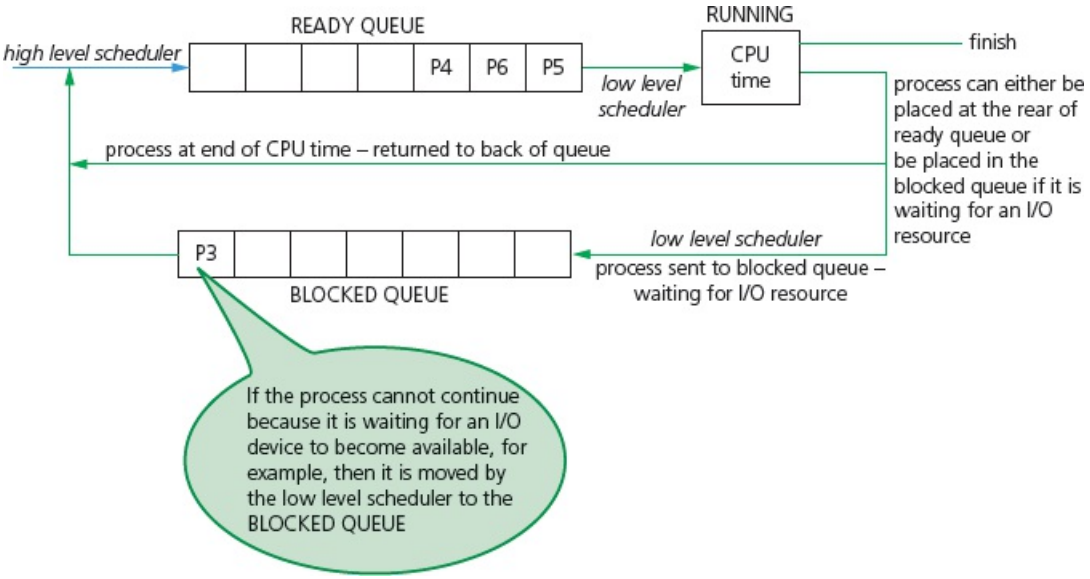


Figure 16.4

The following summarises what happens during the round robin process:

- Each process has an equal time slice (known as a quantum).
- When a time slice ends, the low level scheduler puts the process back into the READY QUEUE allowing another process to use CPU time.
- Typical time slices are about 10 to 100 ms long (a 2.7 GHz clock speed would mean that 100 ms of CPU time is equivalent to 27 million clock cycles, giving considerable amount of potential processing time to a process).
- When a time slice ends, the status of each process must be saved so that it can continue from where it left off when it is allocated its next time slice.
- The contents of the CPU registers (PC, MAR, MDR, ACC) are saved to the process control block (PCB); each process has its own control block.
- When the next process takes control of the CPU (burst time), its previous state is reinstated or restored (this is known as **context switching**).

Scheduling routine algorithms

We will consider four examples of scheduling routines

1. first come first served scheduling (FCFS)
2. shortest job first scheduling (SJF)
3. shortest remaining time first scheduling (SRTF)
4. round robin.

These are some of the most common strategies used by schedulers to ensure the whole system is running efficiently and in a stable condition at all times. It is important to consider how we manage the ready queue to minimise the waiting time for a process to be processed by the CPU.

First come first served scheduling (FCFS)

This is similar to the concept of a queue structure which uses the first in first out (FIFO) principle.

The data added to a queue first is the data that leaves the queue first.

Suppose we have four processes, P1, P2, P3 and P4, which have burst times of 23 ms, 4 ms, 9 ms and 3 ms respectively. The ready queue will be:

Figure 16.5

This will give the average waiting time for a process as:

$$\frac{(0 + 23 + 27 + 36)}{4} = 21.5\,\text{ms}$$

Shortest job first scheduling (SJF) and shortest remaining time first scheduling (SRTF)

These are the best approaches to minimise the process waiting times.

SJF is non-preemptive and SRTF is preemptive.

The burst time of a process should be known in advance; although this is not always possible.

We will consider the same four processes and burst times as the above example (P1 = 23 ms, P2 = 4 ms, P3 = 9 ms, P4 = 3 ms).

With SJF, the process requiring the least CPU time is executed first. P4 will be first, then P2, P3 and P1. The ready queue will be:

Figure 16.6

The average waiting time for a process will be: $\frac{(0 + 3 + 7 + 16)}{4} = 6.5\,\text{ms}$

With SRTF, the processes are placed in the ready queue as they arrive; but when a process with a shorter burst time arrives, the existing process is removed (pre-empted) from execution. The

shorter process is then executed first.

Let us consider the same four processes, including their arrival times.

Process	Burst time (ms)	Arrival time of process (ms)
P1	23	0
P2	4	1
P3	9	2
P4	3	3

Table 16.4

The ready queue will be:

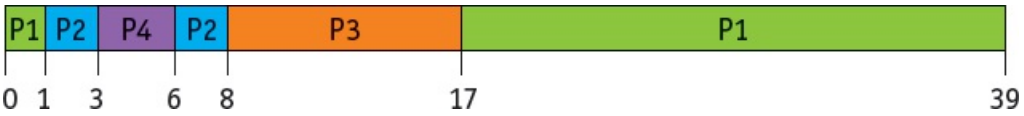


Figure 16.7

The average waiting time for a process will be:

$$\frac{((0 + (6 - 3) + (8 - 2) + (17 - 1))}{4} = 6.25\,\text{ms}$$

The following summarises what happens in more depth:

- Process P1 arrives first; but after only 1 ms of processing time, process P2 arrives with a burst time of 4 ms; this is a shorter burst time than process P1 (23 ms).
- Process P1 (remaining time for completion = 22 ms) is now removed and placed in the blocked queue; process P2 is now placed in the ready queue and processed.
- As P2 is executed, P3 arrives 1 ms later; but the burst time for process P3 (9 ms) is greater than the burst time for P2 (4 ms); therefore, P3 is put in the blocked queue for now and P2 continues.
- After another 1 ms, P4 arrives; this has a burst time of 3 ms, which is less than the burst time for P2 (4 ms); thus P2 (remaining time for completion = 2 ms) is removed and put into the blocked queue; process P4 is now put in the ready queue and is executed.
- Once P4 is completed, P2 is put back into the ready queue for 2 ms until it is completed (burst time for P2 < burst time for P3).
- P3 is now placed back in the ready queue (burst time P3 < burst time P1) and completed after 9 ms.
- Finally, process P1 is placed in the ready queue and is completed after a further 22 ms.

Round robin

A fixed time slice is given to each process; this is known as a quantum.

Once a process is executed during its time slice, it is removed and placed in the blocked queue; then another process from the ready queue is executed in its own time slice.

Context switching is used to save the state of the pre-empted processes.

The ready queue is worked out by giving each process its time slice in the correct order (if a process completes before the end of its time slice, then the next process is brought into the ready queue for its time slice).

Thus, for the same four processes, P1–P4, we get this ready queue:

Figure 16.8

The average waiting time for a process is calculated as follows:

P1: (39 – 23) = 16 ms

P2: (9 – 4) = 5 ms

P3: (26 – 9) = 17 ms

P4: (17 – 3) = 14 ms

Thus, average waiting time $\frac{(16 + 5 + 17 + 14)}{4} = 13\,\text{ms}$

So, the average waiting times for the four scheduling routines, for P1–P4, are:

FCFS	21.5 ms
SJF	6.5 ms
SRTF	6.25 ms
Round robin	13.0 ms

Table 16.5

EXTENSION ACTIVITY 16B

Five processes have the following burst times and arrival times.

Process	Burst time (ms)	Arrival time (ms)
A	45	0
B	18	8
C	5	10
D	23	14
E	11	19

a) Draw the ready queue status for the FCFS, SJF, SRTF and round robin scheduling methods.

b) Calculate the average waiting time for each process using the four scheduling routines named in part a).

Interrupt handling and OS kernels

The CPU will check for interrupt signals. The system will enter the kernel mode if any of the following type of interrupt signals are sent:

- Device interrupt (for example, printer out of paper, device not present, and so on).
- Exceptions (for example, instruction faults such as division by zero, unidentified op code, stack fault, and so on).
- Traps/software interrupt (for example, process requesting a resource such as a disk drive).

When an interrupt is received, the kernel will consult the **interrupt dispatch table (IDT)** – this table links a device description with the appropriate interrupt routine.

IDT will supply the address of the low level routine to handle the interrupt event received. The kernel will save the state of the interrupt process on the kernel stack and the process state will be restored once the interrupting task is serviced. Interrupts will be prioritised using **interrupt priority levels (IPL)** (numbered 0 to 31). A process is suspended only if its interrupt priority level is greater than that of the current task.

The process with the lower IPL is saved in the interrupt register and is handled (serviced) when the IPL value falls to a certain level. Examples of IPLs include:

31: power fail interrupt

24: clock interrupt

20-23: I/O devices

Figure 16.9 summarises the interrupt process.

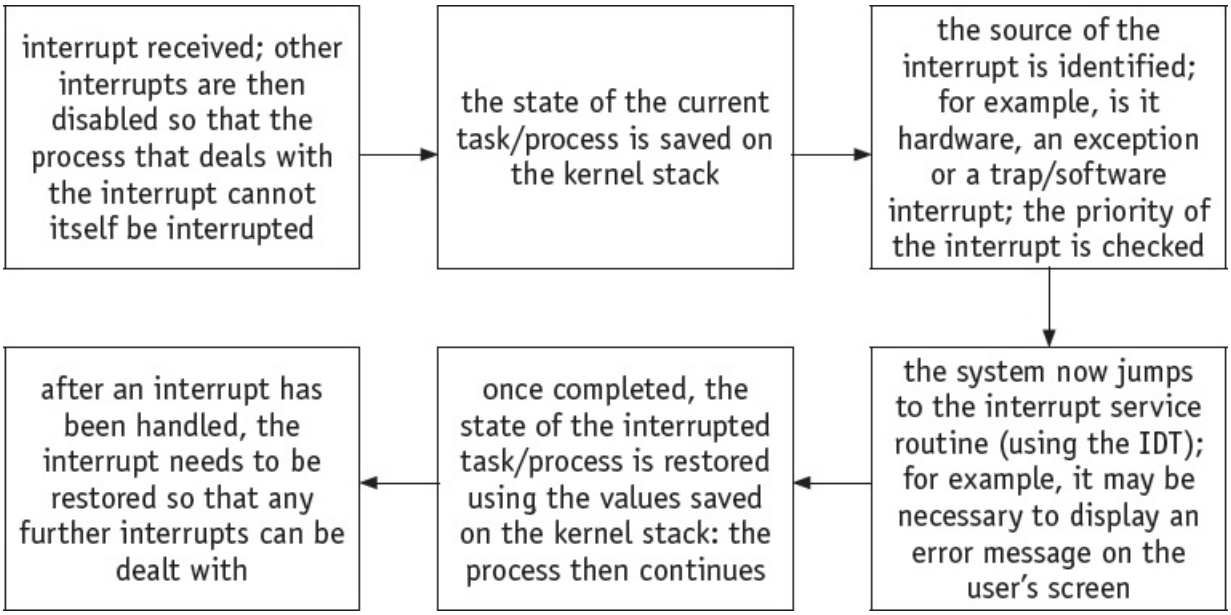


Figure 16.9

16.1.4 Memory management

As with the storage of data on a hard disk, processes carried out by the CPU may also become fragmented. To overcome this problem, memory management will determine which processes should be in main memory and where they should be stored (this is called **optimisation**); in other words, it will determine how memory is allocated when a number of processes are competing with each other. When a process starts up, it is allocated memory; when it is completed, the OS deallocates memory space.

We will now consider the methods by which memory management allocates memory to processes/programs and data.

Single (contiguous) allocation

With this method, *all* of the memory is made available to a single application. This leads to inefficient use of main memory.

Paged memory/paging

In **paging**, the memory is split up into partitions (blocks) of a fixed size. The partitions are not necessarily contiguous. The **physical memory** and **logical memory** are href u#p into the same fixed-size memory blocks. Physical memory blocks are known as **frames** and fixed-size logical memory blocks are known as **pages**. A program is allocated a number of pages that is usually just larger than what is actually needed.

When a process is executed, process pages from logical memory are loaded into frames in physical memory. A **page table** is used; it uses page number as the index. Each process has its own separate page table that maps logical addresses to physical addresses.

The page table will show page number, flag status, page frame address, and the time of entry (for example, in the form 08:25:55:08). The time of entry is important when considering page replacement algorithms. Some of the page table status flags are shown in Table 16.6 below.

Flag	Flag status	Description of status
S	S = 0	page size default value of 4 KiB
	S = 1	page size set to 4 MiB
A	A = 0	page has not yet been accessed
	A = 1	page has been accessed (read or write operation)
D	D = 0	page is unmodified (not dirty)
	D = 1	page has been written to/modified **(dirty)**
G	G = 0	address in cache memory can be updated (global)
	G = 1	when set to 1 this prevents **TLB** from updating address in cache memory

U	U = 0	only the supervisor can access the page
	U = 1	page may be accessed by all users
R	R = 0	page is in the read-only state
	R = 1	page is in the read/write state
P	P = 0	page is not yet present in the memory
	P = 1	page is present in memory

Table 16.6

The following diagram only shows page number and frame number (we will assume status flags have been set and entry time entered). Each entry in a page table points to a physical address that is then mapped to a virtual memory address – this is formed from offset in page directory + offset in page table.

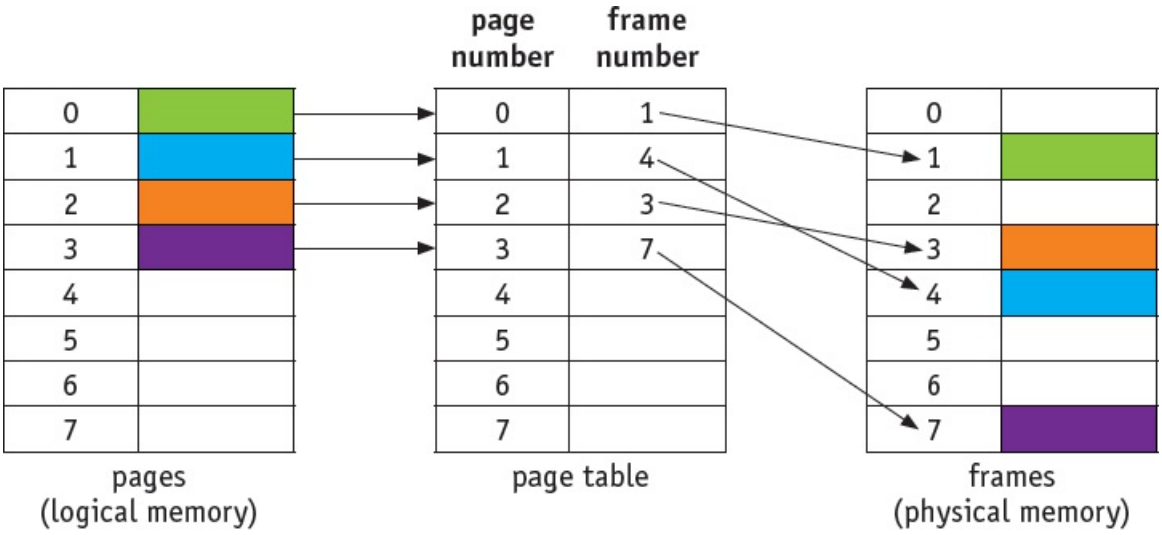


Figure 16.10

Segmentation/segmented memory

In segmented memory, logical address space is broken up into variable-size memory blocks/partitions called **segments**. Each segment has a name and size. For execution to take place, segments from logical memory are loaded into physical memory. The address is specified by the user which contains the segment name and offset value. The segments are numbered (called **segment numbers**) rather than using a name and this segment number is used as the index in a **segment map table**. The offset value decides the size of the segment:

address of segment in physical memory space = segment number + offset value

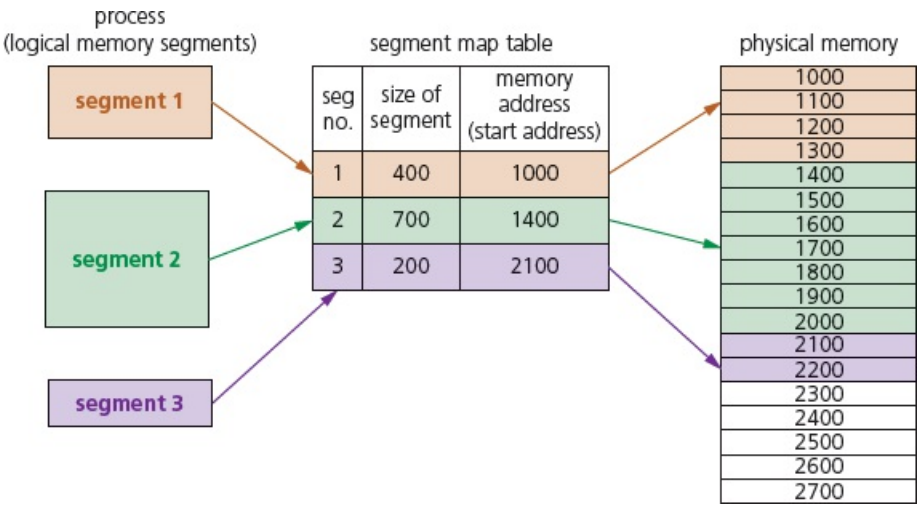


Figure 16.11

The segment map table in Figure 16.11 contains the segment number, segment size and the start address in physical memory. Note that the segments in logical memory do not need to be contiguous, but once loaded into physical memory, each segment will become a contiguous block of memory. Segmentation memory management works in a similar way to paging, but the segments are variable sized memory blocks rather than all the same fixed size.

Summary of the differences between paging and segmentation

Paging	**Segmentation**
a page is a fixed-size block of memory	a segment is a variable-size block of memory
since the block size is fixed, it is possible that all blocks may not be fully used – this can lead to internal fragmentation	because memory blocks are a variable size, this reduces risk of internal fragmentation but increases the risk of external fragmentation
the user provides a single value – this means that the hardware decides the actual page size	the user will supply the address in two values (the segment number and the segment size)
a page table maps logical addresses to physical addresses (this contains the base address of each page stored in frames in physical memory space)	segmentation uses a segment map table containing segment number + offset (segment size); it maps logical addresses to physical addresses
the process of paging is essentially invisible to the user/programmer	segmentation is essentially a visible process to a user/programmer
procedures and any associated data cannot be separated when using paging	procedures and any associated data can be separated when using segmentation
paging consists of static linking and dynamic loading	segmentation consists of dynamic linking and dynamic loading

pages are usually smaller than segments

Table 16.7

16.1.5 Virtual memory

One of the problems encountered with memory management is a situation in which processes run out of RAM main memory. If the amount of available RAM is exceeded due to multiple processes running, it is possible to corrupt the data used in some of the programs being run. This can be solved by separately mapping each program's memory space to RAM and utilising the hard disk drive (or SSD) if we need more memory. This is the basis behind **virtual memory**.

Essentially, RAM is the physical memory and virtual memory is RAM + **swap space** on the hard disk (or SSD). Virtual memory is usually implemented using **in demand paging** (segmentation can be used but is more difficult to manage). To execute a program, pages are loaded into memory from HDD (or SSD) whenever required. We can show the differences between paging without virtual memory and paging using virtual memory in two simple diagrams (Figures 16.12 and 16.13).

Without virtual management:

Figure 16.12

With virtual management:

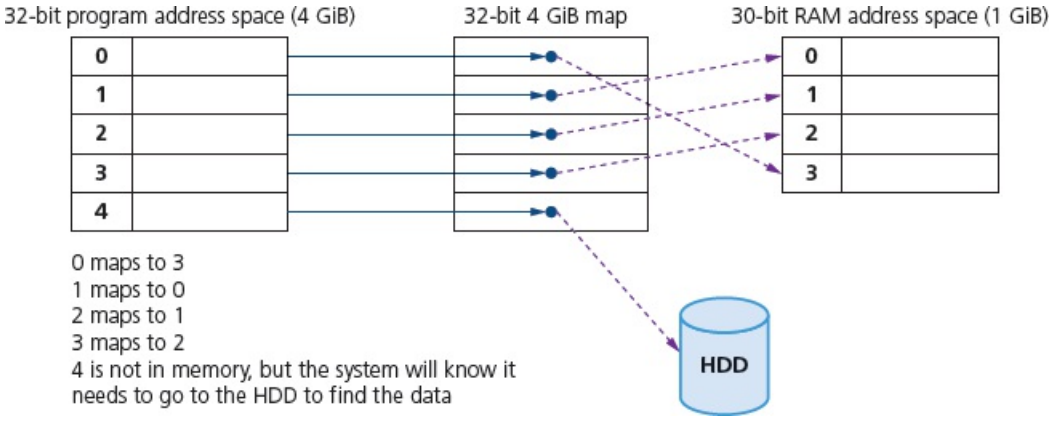


Figure 16.13

Virtual memory management now moves the oldest data to disk and the 4 GiB map is updated so that

- data 0 is now mapped to the HDD
- program/process 4 now maps to 3 in RAM.

This gives the illusion of unlimited memory being available. Even though RAM is full, data can be moved in and out of HDD to give the illusion that there is still memory available.

The main benefits of virtual memory are

- programs can be larger than physical memory and can still be executed
- it leads to more efficient multi-programming with less I/O loading and swapping programs into and out of memory
- there is no need to waste memory with data that is not being used (for example, during error handling)
- it eliminates external fragmentation/reduces internal fragmentation
- it removes the need to buy and install more expensive RAM memory.

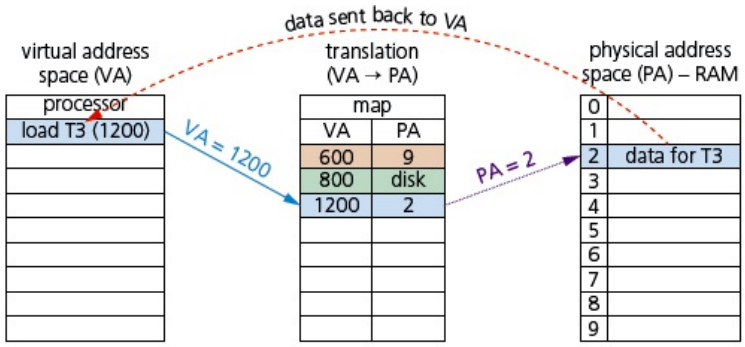
The main drawback when using HDD is that, as main memory fills, more and more data/pages need to be swapped in and out of virtual memory. This leads to a high rate of hard disk read/write head movements; this is known as **disk thrashing**. If more time is spent on moving pages in and out of memory than actually doing any processing, then the processing speed of the computer will be reduced. A point can be reached when the execution of a process comes to a halt since the system is so busy paging in and out of memory; this is known as the **thrash point**. Due to large numbers of head movements, this can also lead to premature failure of a hard disk drive. Thrashing can be reduced by installing more RAM, reducing the number of programs running at a time, or reducing the size of the swap file.

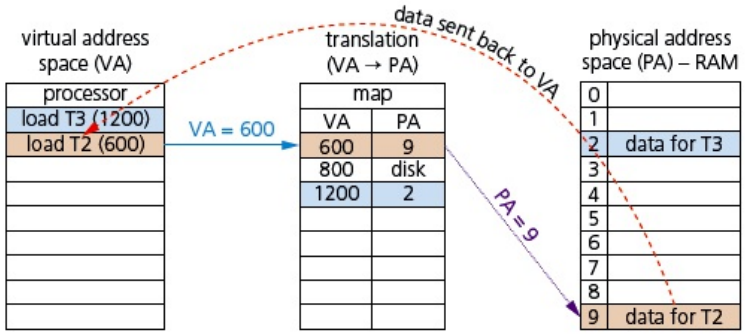
How do programs/processes access data when using virtual memory?

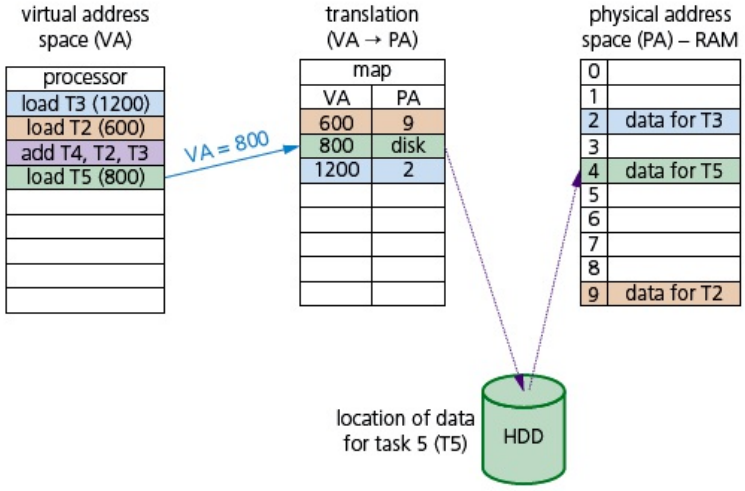
Virtual memory is used in a more general sense to manage memory using paging:

- The program executes the load process with a virtual address (VA).
- The computer translates the address to a physical address (PA) in memory.
- If PA is not in memory, the OS loads it from HDD.
- The computer then reads RAM using PA and returns the data to the program.

Figure 16.14 show this process.







The translation map is now updated and disk is replaced by PA = 4:

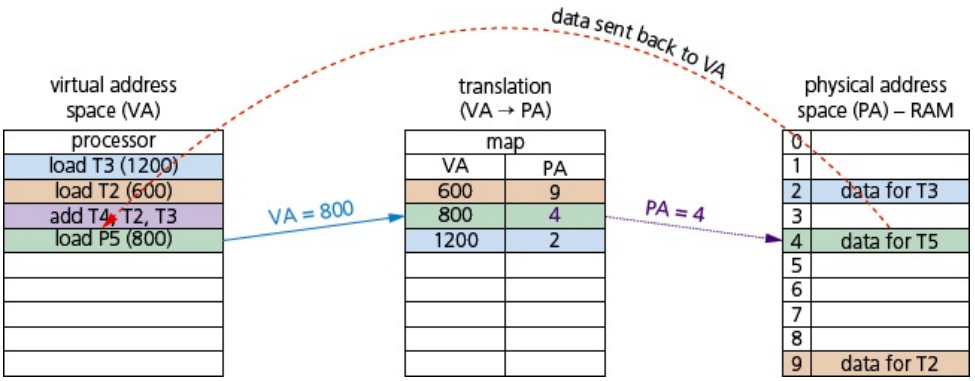


Figure 16.14

16.1.6 Page replacement

Page replacement occurs when a requested page is not in memory (P flag = 0). When paging in/out from memory, it is necessary to consider how the computer can decide which page(s) to replace to allow the requested page to be loaded. When a new page is requested but is not in memory, a **page fault** occurs and the OS replaces one of the existing pages with the new page(s). How to decide which page to replace is done in a number of different ways, but all methods have the same goal of minimising the number of page faults. A page fault is a type of interrupt (it is not an error condition) raised by hardware. When a running program accesses a page that is mapped into virtual memory address space, but not yet loaded into physical memory, then the hardware needs to raise this page fault interrupt.

Page replacement algorithms

First in first out (FIFO)

When using **first in first out (FIFO)**, the OS keeps track of all pages in memory using a queue structure. The oldest page is at the front of the queue and is the first to be removed when a new page needs to be added. FIFO algorithms do not consider page usage when replacing pages: a page may be replaced simply because it arrived earlier than another page. It suffers from what is known as **Belady's anomaly**, a situation in which it is possible to have more page faults when increasing the number of page frames – this is the reverse of the ideal situation shown by the graph in Figure 16.15.

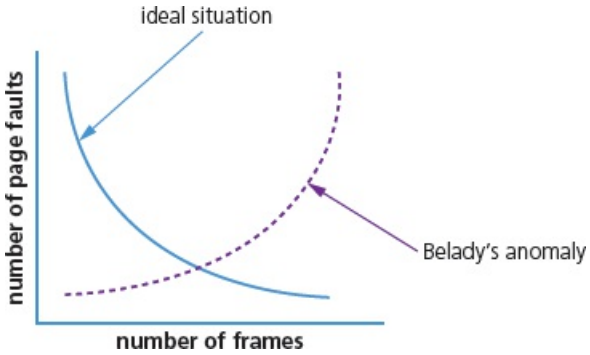


Figure 16.15

Optimal page replacement (OPR)

Optimal page replacement looks forward in time to see which frame it can replace in the event of a page fault. The algorithm is impossible to implement; at the time of a page fault, the OS has no way of knowing when each of the pages will be replaced next. It tends to get used for comparison studies – but it has the advantage that it is free of Belady's anomaly and has the fewest page faults.

Least recently used page replacement (LRU)

With **least recently used page replacement (LRU)**, the page which has not been used for the longest time is replaced. To implement this method, it is necessary to maintain a linked list of all pages in memory with the most recently used page at the front and the least recently used page at

the rear.

Clock page replacement/second-chance page replacement

Clock page replacement algorithms use a circular queue structure with a single pointer serving as both head and tail. When a page fault occurs, the page pointed to (element 3 in the diagram on the following page) is inspected. The action taken next depends on the R-flag status. If R = 0, the page is removed and a new page inserted in its place; if R = 1, the next page is looked at and this is repeated until a page where R = 0 is found.

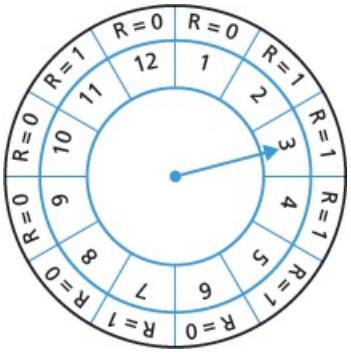


Figure 16.16

In all page replacement methods, when a page fault occurs, the OS has to decide which page to remove from memory to make room for a new page. Page replacement is done by swapping pages from back-up store to main memory (and vice versa). If the page to be removed has been modified while in memory (called dirty), it must be written back to disk. If it has not been changed, then no re-write needs to be done. As mentioned earlier page faults are not errors and are used to increase the amount of memory available to programs that utilise virtual memory.

16.1.7 Summary of the basic differences between processor management and memory management

Processor management decides which processes will be executed and in which order (to maximise resources), whereas memory management will decide where in memory data/programs will be stored and how they will be stored. Although quite different, both are essential to the efficient and stable running of any computer system.

The two OS operations can be summarised as in Figure 16.17.

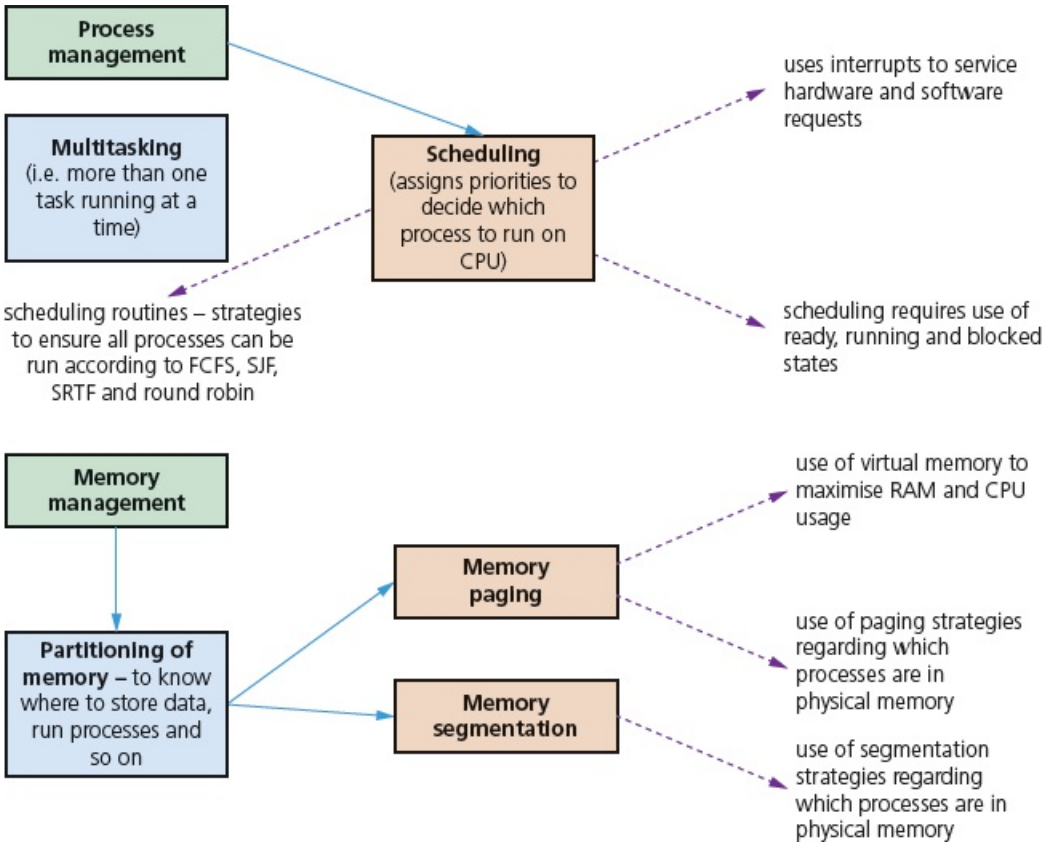


Figure 16.17

ACTIVITY 16A

For each of the following questions, choose the option which corresponds to the correct response.

1 Which of the following page replacement algorithms suffer from Belady's anomaly?

A) first in first out (FIFO) page replacement

B) clock page replacement

C) least recently used (LRU) page replacement

D) optimal page replacement

E) all the above

2 Complete this sentence: A page fault occurs when …

A) … an exception occurs.

B) … a requested page is not yet in the memory.

C) … a requested page is already in memory.

D) … the computer runs out of RAM memory.

E) … a page has become corrupted.

3 Which of the following pages will the optimal page replacement algorithm select?

A) the page that has been used the most number of times

B) the page that will not be used for the longest time in the future

C) the page that has not been used for the longest time in the past

D) the page that has been used the least number of times

E) the page that has been in memory for the shortest time

4 A virtual memory system is using the FIFO page replacement algorithm. Increasing the number of page frames in main memory will:

A) sometimes increase the number of page faults

B) always decrease the number of page faults

C) always increase the number of page faults

D) never affect the number of page faults

E) sometimes cause memory instability

5 What is the swap space on a hard disk (HDD) used for?

A) storing device drivers

B) saving temporary html pages

C) storing page faults

D) saving interrupts

E) saving process data

6 Increasing the size of RAM on a computer will usually increase its performance. Which of the following is the reason for this?

A) it will increase the size of virtual memory

B) there will be fewer segmentation faults

C) larger RAM results in fewer interrupts occurring

D) there will be fewer page faults occurring

E) larger RAMs have a faster data transfer rate

7 Which of the following is a description of virtual memory?

A) a larger secondary memory

B) a larger main memory (RAM)

C) method of reducing the number of page faults

D) system that uses host and guest operating systems

E) it gives the illusion of a larger main memory

8 Which of the following would cause disk thrashing?

A) when a page fault occurs

B) when a number of interrupts occur
C) frequent accessing of pages not in main memory
D) when the processes on a system are in the running state
E) when the processes on a system are in the blocked state

9 Which of the following is the main entry in a page table?
A) the virtual page number (VN)
B) the page frame number
C) the access rights to the page
D) the size of the page
E) the type of linking and loading used

10 Which of the following determines the minimum number of page frames that must be allocated to a running process in a virtual memory environment?
A) the instruction set architecture
B) the page size being used
C) the physical memory size
D) the virtual memory size
E) the number of processes occurring

11 Which of the following statements about virtual memories is true?
A) virtual memory allows each process to exceed the size of the main memory
B) virtual memory translates virtual addresses into physical memory addresses
C) virtual memory increases the degree of multiprogramming that can take place
D) virtual memory reduces the amount of disk thrashing that takes place
E) virtual memory reduces context switching overheads

12 Which of the following is a true statement about disk thrashing?
A) it reduces the amount of page input/output occurrences
B) it decreases the degree of multiprogramming possible
C) there will be excessive page input-output taking place
D) it generally improves system performance
E) it reduces the amount of fragmentation on the disk

13 Which of the following is a likely cause of disk thrashing?
A) the page size was too small
B) FIFO is being used as the page replacement method
C) least recently used (LRU) page replacement method is being used
D) optimal page replacement method is being used
E) too many programs/processes are being run at the same time

14 Which of the following is a description of a page fault?
A) it occurs when a program/process accesses a page not yet in memory
B) it occurs when a program/process accesses a page available in memory
C) it is an error condition when reading a page
D) it is a reference to a page belonging to a different running program

E) it is the result of disk thrashing when excessive paging is taking place

15 Which of the following statements about disk thrashing is true?

A) it always occurs on large computers

B) it is a result of using virtual memory systems

C) disk thrashing can be reduced by doing swapping

D) it is a result of internal fragmentation occurring

E) it can be caused if poor paging algorithms are being used

16.2 Virtual machines (VMs)

Key terms

Virtual machine – an emulation of an existing computer system. A computer OS running within another computer's OS.

Emulation – the use of an app/device to imitate the behaviour of another program/device; for example, running an OS on a computer which is not normally compatible.

Host OS – an OS that controls the physical hardware.

Guest OS – an OS running on a virtual machine.

Hypervisor – virtual machine software that creates and runs virtual machines.

It is important that virtual memory and **virtual machines** are not confused with each other – they are different concepts. This section explains what virtual machines are and outlines some of the benefits and limitations.

Suppose you are using a PC running in a Windows 10 environment and you have downloaded some software that will only run on an Apple computer. It is possible to **emulate** the running of the Apple software on the Windows PC (hardware) – this is known as a virtual machine. The emulation will allow the Apple software to use all of the hardware on the host PC.

Effectively, a virtual machine runs the existing OS (called the **host operating system**) and oversees the virtual hardware using a **guest operating system** – in our example above, the host OS is Windows 10 and the guest OS is Apple. The emulation engine is referred to as a **hypervisor**; this handles the virtual hardware (CPU, memory, HDD and other devices) and maps them to the physical hardware on the host computer.

First, virtual machine software is installed on the host computer. When starting up a virtual machine, the chosen guest operating system will run the emulation in a window on the host operating system. The emulation will run as an app on the host computer. The guest OS has no awareness that it is on an 'alien machine'; it believes it is running on a compatible system. It is actually possible to run more than one guest OS on a computer. This section summarises the features of host and guest operating systems, as well as the benefits and limitations of virtual machines.

16.2.1 Features of a virtual machine

Guest operating system

- This is the OS running in a virtual machine.
- It controls the virtual hardware during the emulation.
- This OS is being emulated within another OS (the host OS).
- The guest OS is running under the control of the host OS software.

Host operating system

- This is the OS that is controlling the actual physical hardware.
- It is the normal OS for the host/physical computer.
- The OS runs/monitors the virtual machine software.

Figure 16.18 shows how the hardware and operating systems are linked together to form a virtual machine.

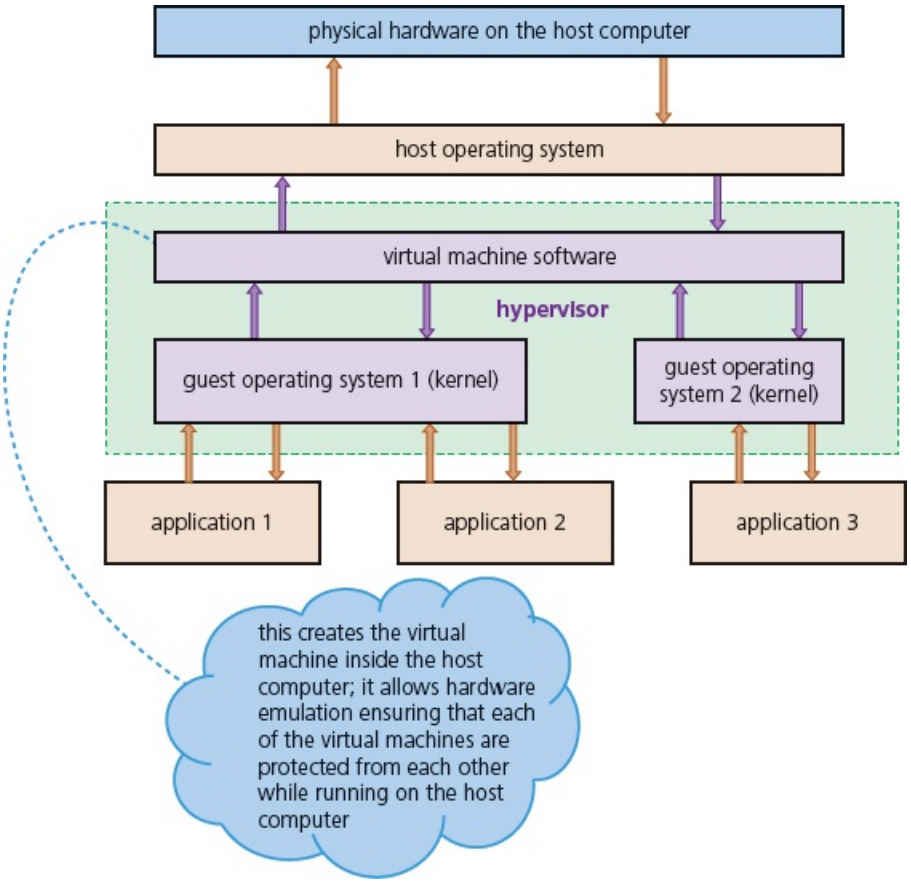


Figure 16.18

16.2.2 Benefits and limitations of virtual machines

Benefits

The guest OS hosted on a virtual machine can be used without impacting anything outside the virtual machine; any other virtual machines and host computer are protected by the virtual machine software.

It is possible to run apps which are not compatible with the host computer/OS by using a guest OS which is compatible with the app.

Virtual machines are useful if you have old/legacy software which is not compatible with a new computer system/hardware. It is possible to emulate the old software on the new system by running a compatible guest OS as a virtual machine. For example, the software controlling a nuclear power station could be transferred to new hardware in a control room – the old software would run as an emulation on the new hardware (justifying the cost and complexity issues – see Limitations).

Virtual machines are useful for testing a new OS or new app since they will not crash the host computer if something goes wrong (the host computer is protected by the virtual machine software).

Limitations

You do not get the same performance running as a guest OS as you do when running the original system.

Building an in-house virtual machine can be quite expensive for a large company. They can also be complex to manage and maintain.

16.3 Translation software

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you read the third part of this chapter.

1 Name **three** types of language translator.

2 Explain the benefits of each of the language translators named in question 1.

3 Describe the typical features of an IDE.

Key terms

Lexical analysis – the first stage in the process of compilation: removes unnecessary characters and tokenises the program.

Syntax analysis – the second stage in the process of compilation: output from the lexical analysis is checked for grammatical (syntax) errors.

Code generation – the third stage in the process of compilation: this stage produces an object program.

Optimisation (compilation) – the fourth stage in the process of compilation: the creation of an efficient object program.

Backus-Naur form (BNF) notation –
a formal method of defining the grammatical rules of a programming language.

Syntax diagram – a graphical method of defining and showing the grammatical rules of a programming language.

Reverse Polish notation (RPN) – a method of representing an arithmetical expression without the use of brackets or special punctuation.

16.3.1 How an interpreter differs from a compiler

An interpreter executes the program it is interpreting. A compiler does not execute the program it is compiling.

With a compiler the program source code is input and either the object code program or error messages are output. The object code produced can then be executed without needing recompilation.

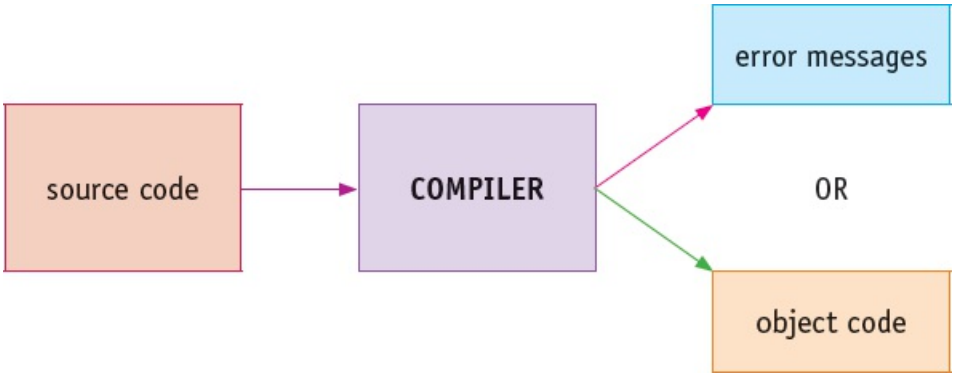


Figure 16.19

With an interpreter, the program source code is similarly input; there may also be other inputs that the program requires or to correct errors in the source program. No object code is output, but error messages from the interpreter are output, as well as any outputs produced by the program being interpreted. As there is no object code produced from the interpretation process, the interpreter will need to be used every time the program is executed.

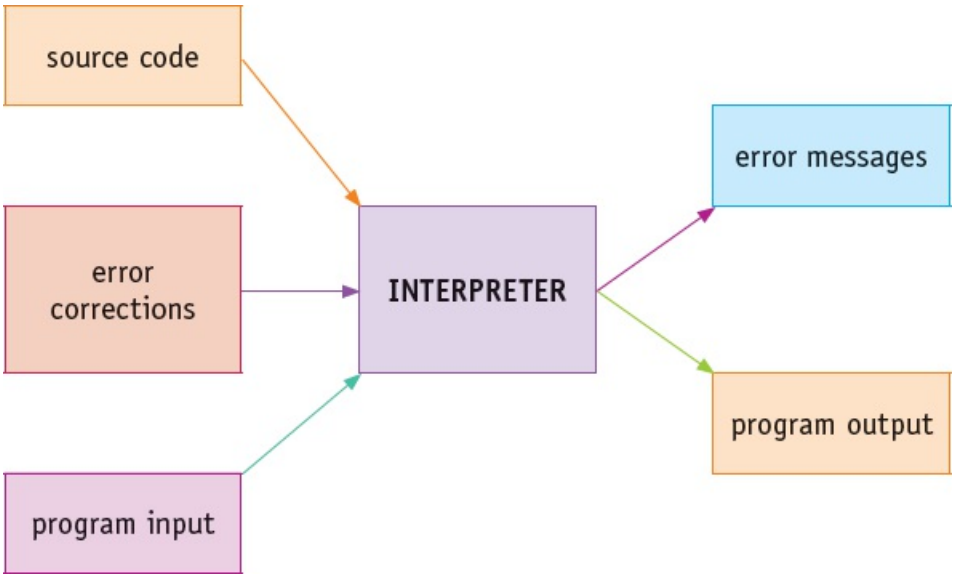


Figure 16.20

Both a compiler and an interpreter will construct a symbol table (see next section for details). An interpreter will also allocate space in memory to store any constants, variables and other data items used by the program. The interpreter checks each statement individually and reports any

errors, which can be corrected before the statement is executed. After each statement is executed, control is returned to the interpreter so the next statement can be checked before execution.

16.3.2 Stages in the compilation of a program

The process of translating a source program written in a high-level language into an object program in machine code can be divided into four stages: **lexical analysis**, **syntax analysis**, **code generation** and **optimisation**.

Lexical analysis

Lexical analysis is the first stage in the process of compilation. All unnecessary characters not required by the compiler, such as white space and comments, are removed.

The example below shows a small program both before and after unnecessary characters have been removed:

Source program

```
// My addition program Version 2
DECLARE x, y, z : INTEGER
OUTPUT "Please enter two numbers to add together"
INPUT x, y
z = x + y
OUTPUT "Answer is ", z
```

Source program after unnecessary characters have been removed

```
DECLARE x, y, z : INTEGER
OUTPUT "Please enter two numbers to add together"
INPUT x, y
z = x + y
OUTPUT "Answer is ", z
```

Before translation, the source program needs to be converted to tokens. This process is called tokenisation. In order to tokenise the program, the compiler will use a keyword table that contains all the tokens for the reserved words and symbols used in a programming language. In the example below, all the tokens are represented as two hexadecimal numbers.

A keyword table is where all the reserved words and symbols that can be used in the programming language are stored. The term fixed symbol table can also be used for the symbols stored. Every program being compiled uses the same keyword table.

For example, part of a keyword table could be as follows.

Keyword/Symbol	Token
=	01

+	02
:	03
,	04
DECLARE	31
INTEGER	32
INPUT	33
OUTPUT	34

Table 16.8

During the lexical analysis, the variables and constants and other identifiers used in a program are also added to a symbol table, produced during compilation, specifically for that program.

For example, part of a symbol table for the program above could be as follows.

	Token		
Symbol	Value	Type	Data Type
X	81	variable	integer
Y	82	variable	integer
Z	83	variable	integer
"Please enter two numbers to add together"	84	constant	integer
"Answer is "	85	constant	integer

Table 16.9

The output from the lexical analysis is a tokenised list stored in main memory.

Here is the output for the first four lines of the program:

31 81 04 82 04 83 03 34 84 33 81 04 82 83 01 81 02 82

ACTIVITY 16B

Write down the output from the lexical analysis for the last line of the program.

Syntax analysis

Syntax analysis is the next stage in the process of compilation. In this stage, output from the lexical analysis is checked for grammatical (syntax) errors.

For example, this source program statement:

z + x + y

would produce this tokenised list:

83 02 81 02 82

↑

error = (01) expected

As shown above, the complete tokenised list is checked for errors using the grammatical rules for the programming language. This is called parsing. The whole program goes through this process even if errors are found. The rules for parsing can be set out in **Backus-Naur form (BNF) notation** (see next section). If any errors are found, each statement and the associated error are output but, in the next stage of compilation, code generation will not be attempted. The compilation process will finish after this stage. If the tokenised code is error free, it will be passed to the next stage of compilation, generating the object code.

Code generation

The code generation stage produces an object program to perform the task defined in the source code. The program must be syntactically correct for an object program to be produced. The object program is in machine-readable form (binary). It is no longer in a form that is designed to be read by humans. This object program is either in machine code that can be executed by the CPU, or in an intermediate form that is converted to machine code when the program is loaded. The latter option allows greater flexibility.

For example, intermediate code can support

- the use of relocatable code so the program can be stored anywhere in main memory
- the addition of library routines to the code at this stage to minimise the size of the stored object program
- the linking of several programs to run together.

Optimisation

The optimisation stage supports the creation of an efficient object program. Optimised programs should perform the task using the minimum amount of resources. These include time, storage space, memory and CPU use. Some optimisation can take place after the syntax analysis or as part of code generation.

A simple example of code optimisation is shown here:

Original code	w = x + y	Object code	LDD x
			ADD y
			STO w
	v = x + y + z		LDD x
			ADD y
			ADD z
			STO v
Optimised code	w = x + y	Object code	LDD x
			ADD y
			STO w
	v = w + z		ADD z
			STO v

Table 16.10

The addition x + y is only performed once, and the second addition can make use of the value w stored in the accumulator. This optimisation can only work if the two lines of code are together and in this order in the program.

The task of code optimisation is one of the most challenging stages of compilation. Not every compiler is able to optimise the object code produced for every source program.

16.3.3 Syntax diagrams and Backus-Naur form

The grammatical rules or syntax for a programming language need to be set out clearly so a programmer can write code that obeys the rules, and a compiler can be built to check that a program obeys these rules. The rules can be shown graphically in a **syntax diagram** or using a meta language such as Backus-Naur form (BNF) notation, which is a formal method showing syntax as a list of replacement rules to represent the algorithm used in syntax analysis.

Syntax diagrams

Each element in the language has a diagram showing how it is built.

For example, a simple variable consisting of a letter followed by a digit would be shown as:

Figure 16.21

The alternatives for letter could be shown as:

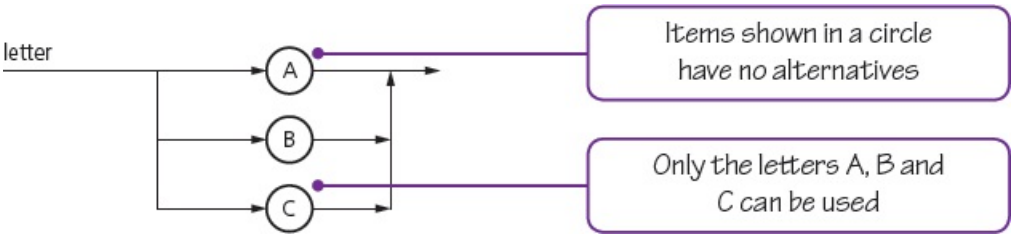


Figure 16.22

The alternatives for digit could be shown as:

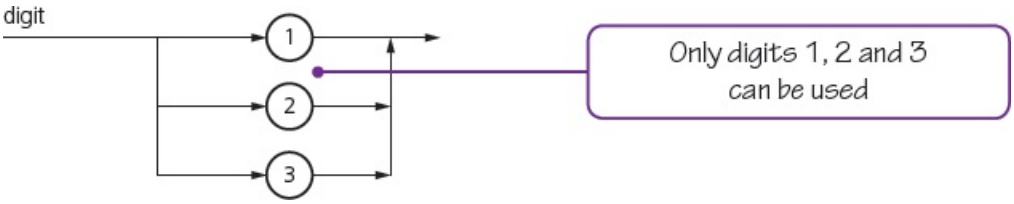


Figure 16.23

An assignment statement could be shown as:

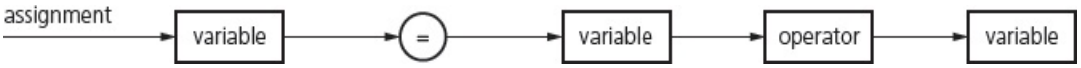


Figure 16.24

The alternatives for operator could be shown as:

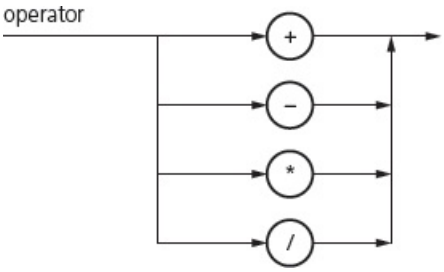


Figure 16.25

ACTIVITY 16C

1 Using the syntax diagrams shown, identify which of the following variables are invalid and explain why.

- **a)** A1
- **b)** Z2
- **c)** B5
- **d)** C3
- **e)** CC
- **f)** 1A

2 Using the syntax diagrams shown, identify which of the following assignment statements are invalid and explain why.

- **a)** A1 = B1 + C1
- **b)** A1= B1 + C1 + B2
- **c)** A1 := C1 − C2

Syntax diagrams can allow for repetitions as well as alternatives. For example, a variable that can be a letter followed by another letter or any number of digits can be shown as:

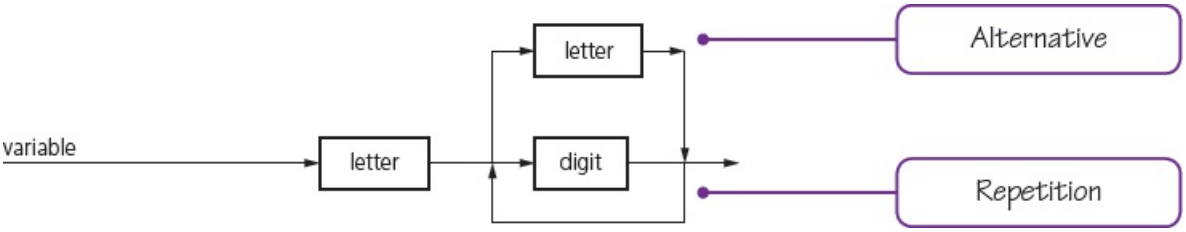


Figure 16.26

ACTIVITY 16D

Draw syntax diagrams to represent a variable consisting of one or two letters (A, B, C, D) followed by zero or one of digits (0, 1, 2, 3, 4).

Backus-Naur form (BNF)

BNF uses a set of symbols to describe the grammar rules in a programming language.

BNF notation includes:

< >	used to enclose an item
::=	separates an item from its definition
|	between items indicates a choice
;	the end of a rule

For example, a simple variable consisting of a letter (A, B or C) followed by a digit (1, 2, 3) would be shown as:

```
<variable> ::= <letter> | <digit> ;
  <letter> ::= A | B |C ;
   <digit> ::= 1 | 2 |3 ;
```

BNF notation can be used for recursive definitions where an item definition can refer to itself. For example, a variable consisting of any number of letters could be defined as:

<variable> ::= <letter> | <variable> <letter> ;

ACTIVITY 16E

Write the definition for an integer that can consist of any number of digits (0 to 9) in BNF.

EXTENSION ACTIVITY 16C

Refine the definition for an integer from Activity 16E so that it does not start with a zero.

16.3.4 Reverse Polish notation (RPN)

Reverse Polish notation (RPN) is a method of representing an arithmetical or logical expression without the use of brackets or special punctuation. RPN uses postfix notation, where an operator is placed after the variables it acts on. For example, A + B would be written as A B +.

Compilers use RPN because any expression can be processed from left to right without using any back tracking. Any expression can be systematically converted to RPN using a binary tree (see Chapter 19).

Here, we will look at how RPN can be formed by using operator precedence (brackets, multiplication and division, addition and subtraction).

In the expression	A – B * C	* has the highest precedence.
This becomes	A – B C *	B C * can now be considered as a single item.
This becomes	A B C * –	this can now be evaluated from left to right.

Table 16.11

To evaluate the expression using a stack

- the values are added to the stack in turn going from left to right
- when an operator is encountered, it is not added to the stack but used to operate on the top two values of the stack – which are popped off the stack, operated on, then the result is pushed back on the stack
- this is repeated until there is a single value on the stack and the end of the expression has been reached.

If A = 2, B = 3 and C = 4, then A B C * - is evaluated using a stack, as shown below:

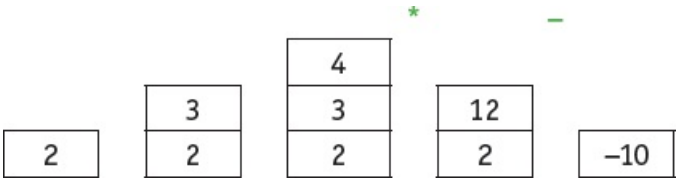


Figure 16.27 Contents of the stack at each stage of evaluation

An expression using brackets (A + B) * (C – D) becomes A B + C D - * in RPN, as brackets have the highest precedence.

If A = 2, B = 3, C = 4 and D = 5:

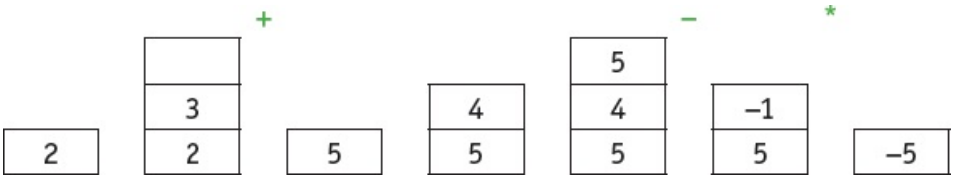


Figure 16.28 Contents of the stack at each stage of evaluation

ACTIVITY 16F

1 Identify, in order, the four stages of compilation.
Describe what happens to a program at each stage of compilation.

2 a) Draw syntax diagrams to represent a variable that must start with a letter (I, J or K) that is followed by up to three digits (0, 1, 2, 3, 4).

b) Represent the same variable in BNF.

c) These variables can be used in an assignment statement for addition or subtraction. Represent the assignment statement by a syntax diagram and in BNF.

3 a) Convert the following expressions to RPN.

i) A + B + C * D

ii) (A + B + C) * D

iii) (A + B) * D + (A – B) * C

b) Show how each of the above expressions, when converted to RPN, can be evaluated using a stack.
A = 7, B = 3, C = 5 and D = 2.

End of chapter questions

1 This table shows the burst time and arrival time for four processes:

Process	Burst time (ms)	Arrival time (ms)
P1	18	0
P2	4	1
P3	8	2
P4	3	3

This is a list of average waiting times, in ms:

6.0 6.25 6.5 8.0 8.25 10.0 11.25 14.0 14.25 14.5

Choosing from the list, select the correct average waiting time for the four processes, P1–P4, for each of these scheduling routines:

a) FCFS

[2]

b) SJF

[2]

c) SRTF

[2]

d) Round robin

[2]

2 A computer operating system (OS) uses paging for memory management.

In paging:

– main memory is divided into equal-size blocks, called page frames
– each process that is executed is divided into blocks of same size, called pages
– each process has a page table that is used to manage the pages of this process

The following table is the incomplete page table for a process, Y.

Page	Presence flag	Page frame address	Additional data
1	1	221	
2	1	222	
3	0	0	
4	0	0	
5	1	542	
6	0	0	
∫	∫	∫	∫
249	0	0	

a) State **two** facts about Page **5**.

[2]

b) Process Y executes the last instruction in Page **5**. This instruction is not a branch instruction.

i) Explain the problem that now arises in the continued execution of process Y.

[2]

ii) Explain how interrupts help to solve the problem that you explained in part b) i).

[3]

c) When the next instruction is not present in main memory, the OS must load its page into a page frame.
If all page frames are currently in use, the OS overwrites the contents of a page frame with the required page.
The page that is to be replaced is determined by a page replacement algorithm.
One possible algorithm is to replace the page which has been in memory the shortest amount of time.

i) Give the additional data that would need to be stored in the page table.

[1]

ii) Copy and complete the table entry below to show what happens when Page **6** is swapped into main memory.
Include the data you have identified in part c) i) in the final column.
Assume that Page **1** is the one to be replaced.
In the final column, give an example of the data you have identified in part c) i).

[3]

Page	Presence flag	Page frame address	Additional data
∫	∫	∫	∫
6			
∫	∫	∫	∫

Process Y contains instructions that result in the execution of a loop, a very large number of times. All instructions within the loop are in Page **1**.

The loop contains a call to a procedure whose instructions are all in Page **3**.

All page frames are currently in use.

Page **1** is the page that has been in memory for the shortest time.

iii) Explain what happens to Page **1** and Page **3**, each time the loop is executed.

[3]

iv) Name the condition described in part c) iii).

[1]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q3 November 2016

3 State the ten computer terms being described below.

a) A fixed time slice allotted to a process.

[1]

b) When a process switches from running state to steady state or from waiting state to steady state.

[1]

c) System which gives the illusion that there is unlimited memory available.

[1]

d) Algorithm which decides which process (in the ready state) should get CPU time next (running state).

[1]

e) Procedure by which, when the next process takes control of the CPU, its previous state is restored.

[1]

f) Physical memory and logical memory are split up into fixed-size memory blocks.

[1]

g) When a process terminates or switches from running state to waiting state.

[1]

h) Time when a process gets control of the CPU.

[1]

i) Logical memory is split up into variable-size memory blocks.

[1]

j) To continuously deprive a process of the necessary resources to process a task.

[1]

4 A number of processes are being executed in a computer. A process can be in one of these states: running, ready or blocked.

a) For each of the following, the process is moved from the first state to the second state. Describe the conditions that cause each of the following changes of state of a process:

i) From blocked to ready.

[2]

ii) From running to ready.

[2]

b) Explain why a process cannot move directly from the ready state to the blocked state.

[3]

c) A process in the running state can change its state to something which is neither the ready state nor the blocked state.

i) Name this state.

[1]

ii) Identify when a process would enter this state.

[1]

d) Explain the role of the low-level scheduler in a multiprogramming operating system.

[2]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q6 November 2015

5 a) Explain how programs can access data from memory when using virtual memory

[4]

b) Describe the following page replacement algorithms.

i) First in first out (FIFO)

[2]

ii) Optimal page replacement (OPR)

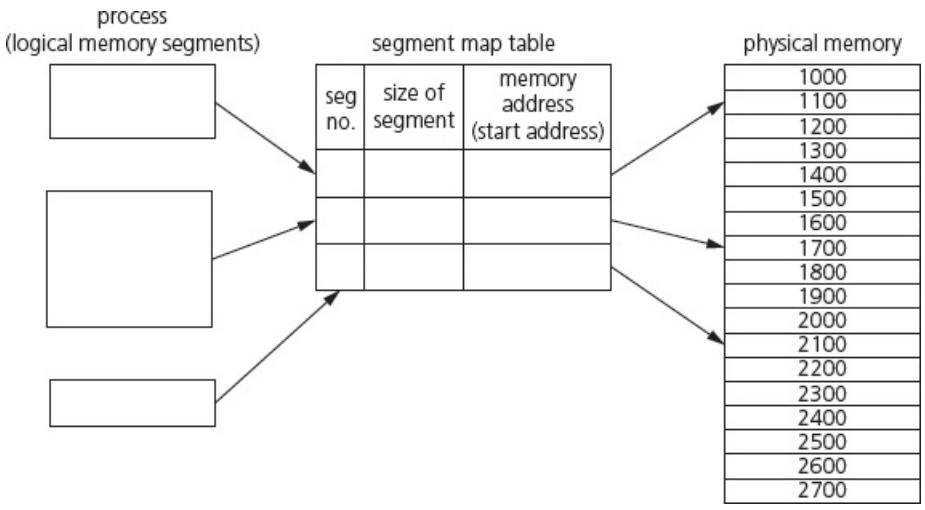
[2]

iii) Least recently used (LRU)

[2]

6 a) Three processes, A, B and C, are presently in logical memory.
Process A is 600 MiB, process B is 800 MiB and process C is 200 MiB.
The starting address in physical memory for process A is 1000.
Each of the units shown in physical memory are in MiB.
Copy and complete the following diagram to show how segmentation can be used as a type of memory management.

[6]



b) Give **three** differences between paging and segmentation.

[3]

c) Name three types of interrupt.

[3]

7 In this question, you are shown pseudocode in place of a real high-level language. A compiler uses a keyword table and a symbol table.
Part of the keyword table is shown below.

– Tokens for keywords are shown in hexadecimal.

Keyword	Token
←	01
+	02
=	03

– All the keyword tokens are in the range 00 to 5F.

IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
FOR	4E
STEP	4F
TO	50
INPUT	51
OUTPUT	52

ENDFOR	53

Entries in the symbol table are allocated tokens. These values start from 60 (hexadecimal).

Study the following piece of code:

```
Start ← 0.1
// Output values in loop
FOR Counter ← Start TO 10
    OUTPUT Counter + Start
ENDFOR
```

a) Copy and complete this symbol table to show its contents after the lexical analysis stage.

[3]

	Token	
Symbol	**Value**	**Type**
Start	60	Variable
0.1	61	Constant

b) Each cell below represents one byte of the output from the lexical analysis stage. Using the keyword table and your answer to part a), copy and complete the output from the lexical analysis.

[2]

60	01												

c) The compilation process has a number of stages. The output of the lexical analysis stage forms the input to the next stage.

i) Name this stage.

[1]

ii) State **two** tasks that occur at this stage.

[2]

d) The final stage of compilation is optimisation. There are a number of reasons for performing optimisation. One reason is to produce code that minimises the amount of memory used.

i) State another reason for the optimisation of code.

[1]

ii) What could a compiler do to optimise the following expression?

[1]

A ← B + 2 * 6

iii) These lines of code are to be compiled:

```
X ← A + B
Y ← A + B + C
```

Following the syntax analysis stage, object code is generated. The equivalent code, in assembly language, is shown below.

```
LDD 436 //loads value A
ADD 437 //adds value B
STO 612  //stores result in X
LDD 436 //loads value A
ADD 437 //adds value B
ADD 438 //adds value C
STO 613  //stores result in Y
```

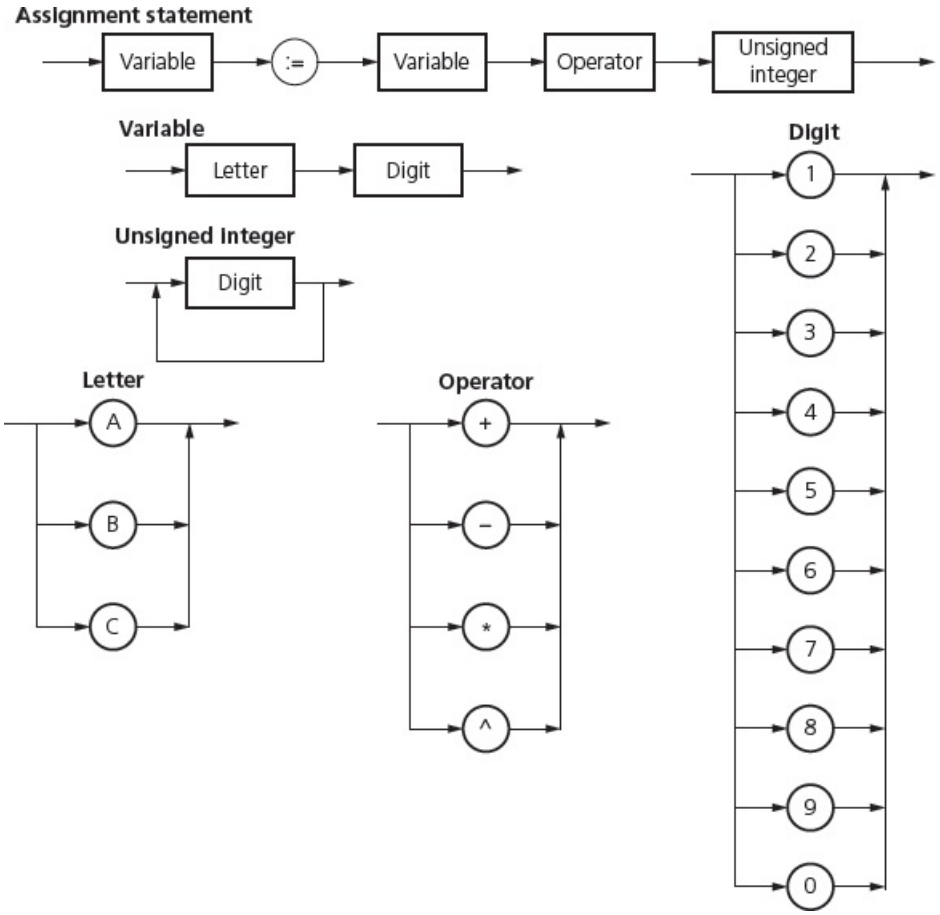
iv) Rewrite the equivalent code, given above, following optimisation.

[3]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q2 November 2015

8 The following syntax diagrams for a particular programming language show the syntax of:

- an assignment statement
- a variable
- an unsigned integer
- a letter
- an operator
- a digit.



a) The following assignment statements are invalid.
Give the reason in each case.

i) C2 = C3 + 123

[1]

ii) A3 := B1 – B2

[1]

iii) A32 := A2 * 7

[1]

b) Copy and complete the Backus-Naur Form (BNF) for the syntax diagrams shown.
<digit> has been done for you.

[6]

<assignment_statement> ::=

<variable> ::=

<unsigned_integer> ::=

<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

<letter> ::=

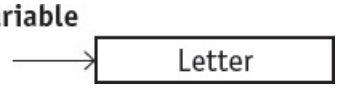
<operator> ::=

c) The definition of <variable> is changed to allow:

– one or two letters and

– zero, one or two digits.

Draw an updated version of the syntax diagram for <variable>.



d) The definition of <assignment_statement> is altered so that its syntax has <unsigned_integer> replaced by <real>.
A real is defined to be:

– at least one digit before a decimal point

– a decimal point

– at least one digit after a decimal point.

Give the BNF for the revised <assignment_statement> and <real>.

[2]

<assignment_statement> ::=

<real> ::=

Cambridge International AS & A Level Computer Science 9608
Paper 31 Q3 November 2017

9 There are four stages in the compilation of a program written in a high-level language.

a) Four statements and four compilation stages are shown below. Copy the diagram below and connect each statement to the correct compilation stage.

[4]

Statement	Compilation stage
This stage can improve the time taken to execute `x = y + 0`	Lexical analysis
This stage produces object code	Syntax analysis
This stage makes use of tree data structures	Code generation
This stage enters symbols in the symbol table	Optimisation

b) Write the Reverse Polish Notation (RPN) for the following expression.

[2]

P + Q – R / S

c) An interpreter is executing a program.

The program uses the variables a, b, c and d.

The program contains an expression written in infix form. The interpreter converts the infix expression to RPN. The RPN expression is:

b a * c d a + + -

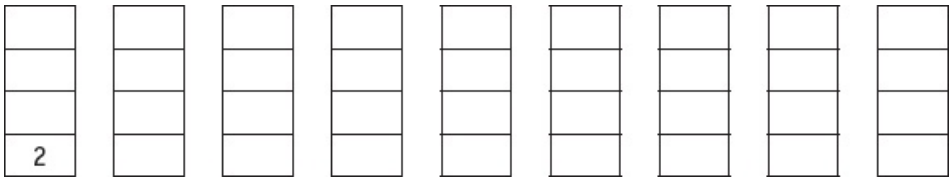
The interpreter evaluates this RPN expression using a stack.

The current values of the variables are:

a = 2 b = 2 c = 1 d = 3

i) Copy the diagram below and show the changing contents of the stack as the interpreter evaluates the expression. The first entry on the stack has been done for you.

[4]



ii) Convert back to its original infix form, the RPN expression:

[2]

b a * c d a + + -

iii) One advantage of using RPN is that the evaluation of an expression does not require rules of precedence.

Explain this statement. [2]

Cambridge International AS & A Level Computer Science 9608
Paper 32 Q2 November 2016

17 Security

In this chapter, you will learn about

- how encryption works, including the use of public keys and private keys, plaintext and ciphertext, symmetric key cryptography and asymmetric key cryptography
- how keys can be used to send verified messages
- how data is encrypted using symmetric and asymmetric cryptography
- quantum cryptography and QKD
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS)
- the use of SSL and TLS in a client/server communication
- examples of where SSL and TLS would be used
- how digital certificates are acquired
- how digital certificates are used to produce digital signatures.

WHAT YOU SHOULD ALREADY KNOW

In Chapter 6, you learnt about security. Try these four questions before you read this chapter.

1 Explain what is meant by the terms

a) data integrity

b) data privacy

c) data security.

2 Describe how it is possible to recover data after it has been lost accidentally or otherwise.

3 Describe **three** ways of protecting against data loss.

4 **a)** Explain the effect of these five security risks.

i) Hacking

ii) Malware

iii) Phishing

iv) Pharming

b) Explain how it is possible to guard against each of the five security risks named in part a).

17.1 Encryption

Key terms

Eavesdropper – a person who intercepts data being transmitted.

Plaintext – the original text/document/message before it is put through an encryption algorithm.

Ciphertext – the product when plaintext is put through an encryption algorithm.

Block cipher – the encryption of a number of contiguous bits in one go rather than one bit at a time.

Stream cipher – the encryption of bits in sequence as they arrive at the encryption algorithm.

Block chaining – form of encryption, in which the previous block of ciphertext is XORed with the block of plaintext and then encrypted thus preventing identical plaintext blocks producing identical ciphertext.

Symmetric encryption – encryption in which the same secret key is used to encrypt and decrypt messages.

Key distribution problem – security issue inherent in symmetric encryption arising from the fact that, when sending the secret key to a recipient, there is the risk that the key can be intercepted by an eavesdropper/hacker.

Asymmetric encryption – encryption that uses public keys (known to everyone) and private keys (secret keys).

Public key – encryption/decryption key known to all users.

Private key – encryption/decryption key which is known only to a single user/computer.

17.1.1 Encryption keys, plaintext and ciphertext

Why do we need encryption?

When data is transmitted over any public network (wired or wireless), there is a risk of it being intercepted by, for example, a hacker (sometimes referred to as an **eavesdropper**). Using encryption helps to minimise this risk.

Encryption alters data into a form that is unreadable by anybody for whom the data is not intended. It cannot prevent the data being intercepted, but it stops it from making any sense to the eavesdropper. This is particularly important if the data is sensitive (for example, medical or legal documents) or confidential (for example, credit card or bank details).

There are four main security concerns when data is transmitted: confidentiality, authenticity, integrity and non-repudiation.

1. **Confidentiality** is where only the intended recipient should be able to read or decipher the data.
2. **Authenticity** is the need to identify who sent the data and verify that the source is legitimate.
3. **Integrity** is that data should reach its destination without any changes.
4. **Non-repudiation** is that neither the sender nor the recipient should be able to deny that they were part of the data transmission which just took place.

Plaintext and ciphertext

The original data being sent is known as **plaintext**. Once it has gone through an encryption algorithm, it produces **ciphertext**. Figure 17.1 summarises what happens.

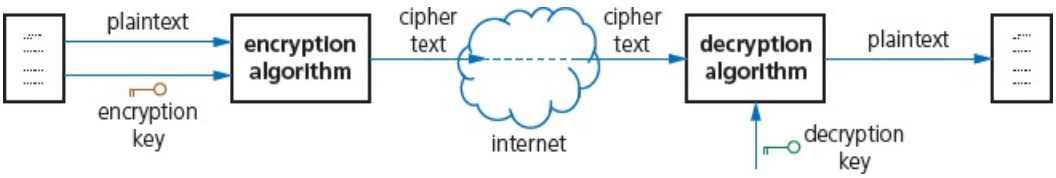


Figure 17.1

Note that, when encrypting text, **block cipher** is usually used. Here, the encryption algorithm is applied to a group of contiguous bits (for example, 128 bits) rather than one bit at a time (which is known as **stream cipher**). With block cipher, each plaintext block is XORed with the previous ciphertext block and then encrypted – this is known as **block chaining**. This prevents identical blocks of plaintext producing the same ciphertext each time they are encrypted.

Notice the use of encryption and decryption keys in Figure 17.1. These keys will be considered in the next section.

17.1.2 Symmetric encryption

Symmetric encryption uses a secret key; the same key is used to encrypt and decrypt the encoded message.

Consider a simple system which uses 10-denary-digit encryption (which gives about 10 billion possibilities). Suppose our secret key is 4 2 9 1 3 6 2 8 5 6, which means each letter in a word is shifted across the alphabet +4, +2, +9, and so on, places.

For example, here is the message, 'computer science is exciting' before and after the 10-denary-digit secret key is applied:

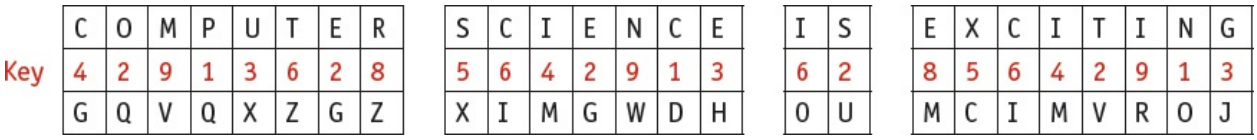


Figure 17.2

However, modern computers could 'crack' this key (and, therefore, decrypt the message) in a few seconds. To combat this, we use 256-bit encryption (in other words, a 256-bit key) which gives 2^{256} possible combinations. Even this may not be enough, as computers become more powerful.

One issue with symmetric encryption is that both sender and recipient need to use the same secret key. This is a security risk here, since the sender has to supply the key to the recipient. This key could be intercepted. This is referred to as the **key distribution problem**.

So, how can both sender and receiver have the required secret key without sending it electronically? The following routine shows one possibility.

stage	sender	recipient
1	uses an encryption algorithm and chooses a secret value, such as X = 2	uses the same algorithm and also chooses a secret value, such as Y = 4
2	this value of X is put into a simple algorithm:	the value of Y is put into the same algorithm:
	Note: MOD gives the remainder when dividing a number by 11	
	7^X (MOD 11) = 7^2 (MOD 11) = 49 (MOD 11) = 4 remainder 5 So, the value is 5	7^Y (MOD 11) = 7^4 (MOD 11) = 2401 (MOD 11) = 218 remainder 3 So, the value is 3
3	the sender sends the value just calculated (5) to the recipient	the recipient sends the value just calculated (3) to the sender
4	the new value from the recipient replaces 7 in the original algorithm:	the new value from the sender replaces 7 in the original algorithm:

	3^X (MOD 11) = 3^2 (MOD 11) = 0 remainder **9** So, the new value is **9**	5^Y (MOD 11) = 5^4 (MOD 11) = 625 (MOD 11) = 56 remainder **9** So, the new value is **9**

Table 17.1

Both sender and recipient end up with the same encryption and decryption key of **9**. This is oversimplified; in practice, computers would generate much larger keys (possibly 256 bits – equivalent to 64 denary digits if using BCD).

There are many other ways to keep the encryption key secret. But the issue of security is always the main drawback of symmetrical encryption, since a single key is required for both sender and recipient.

EXTENSION ACTIVITY 17A

Using the following sender and receiver values to check that the method described above works.

a) Sender uses the value X = 3 and the receiver uses the value Y = 5

b) Sender uses the value X = 7 and the receiver uses the value Y = 6

17.1.3 Asymmetric encryption

Asymmetric encryption uses two keys – a **public key**, available to all users, and a **private key**, known to a specific person or computer.

Suppose Tom and Meera work for the same company. Tom wishes to send a confidential document to Meera. Here's how he could do it.

Step 1: Tom and Meera both use an algorithm to generate their own matching pairs of keys (private and public) which they keep stored on their computers. The matching pairs of keys are mathematically linked but cannot be derived from each other.

Step 2:

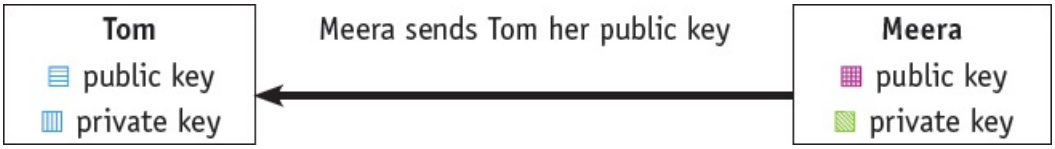


Figure 17.3

Step 3: Tom now uses Meera's public key (▦) to encrypt the document he wishes to send to her. He then sends his encrypted document (ciphertext) to Meera.

Step 4: Meera uses her matching private key (▧) to unlock Tom's document and decrypt it. This works because the public key used to encrypt the document and the private key used to decrypt it are a matching pair generated on Meera's computer.

Meera can exchange her public key with any number of people working in the company, so she is able to receive encrypted messages (which have been encrypted using her public key) and she can then decrypt them using her matching private key:

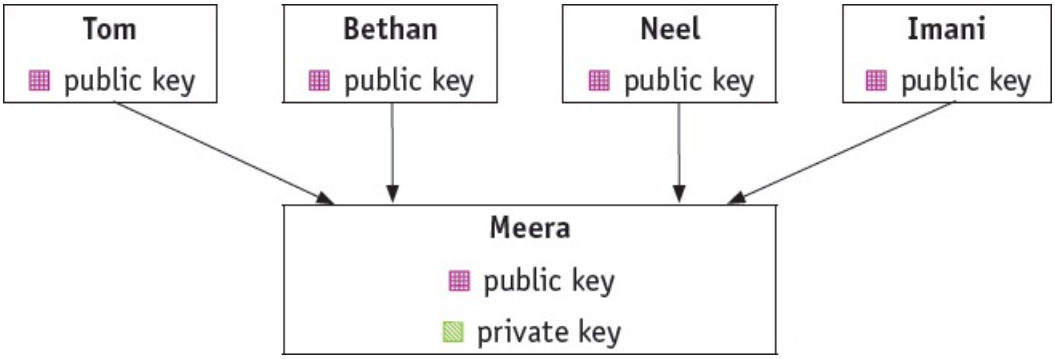


Figure 17.4

If a two-way communication is required between all five workers, then they all need to generate their own matching public and private keys. Once this is done, all users then need to swap public keys so that they can send encrypted documents, files or messages between each other. Each worker will then use their own private key to decrypt information being sent to them.

However, there are still issues. For example, how can Meera be certain that the document came from Tom, and that it has not been tampered with during transmission? Additional security is

required; this will be discussed in Section 17.4.

17.2 Quantum cryptography

Key terms

Quantum cryptography – cryptography based on the laws of quantum mechanics (the properties of photons).

Quantum key distribution (QKD) – protocol which uses quantum mechanics to securely send encryption keys over fibre optic networks.

Qubit – the basic unit of a quantum of information (**qu**antum **bit**).

Quantum cryptography utilises the physics of photons (light energy according to the formula E = hf) and their physical quantum properties to produce a virtually unbreakable encryption system. This helps protect the security of data being transmitted over fibre optic cables.

Photons oscillate in various directions and produce a sequence of random bits (0s and 1s) across the optical network.

Sending encryption keys across a network uses quantum cryptography – a **quantum key distribution (QKD)** protocol (one of the most common is BB84).

QKD uses quantum mechanics to facilitate the secure transmission of encryption keys. Quantum mechanics use a **qubit** (**qu**antum **bit**) as the basic unit of quantum data. Unlike normal binary (which uses discrete 0s and 1s), the state of a qubit can be 0 or 1, but it can also be **both** 0 and 1 simultaneously. Figure 17.5 shows a representation of a photon and how a photon can be affected by one of four types of polarising filter.

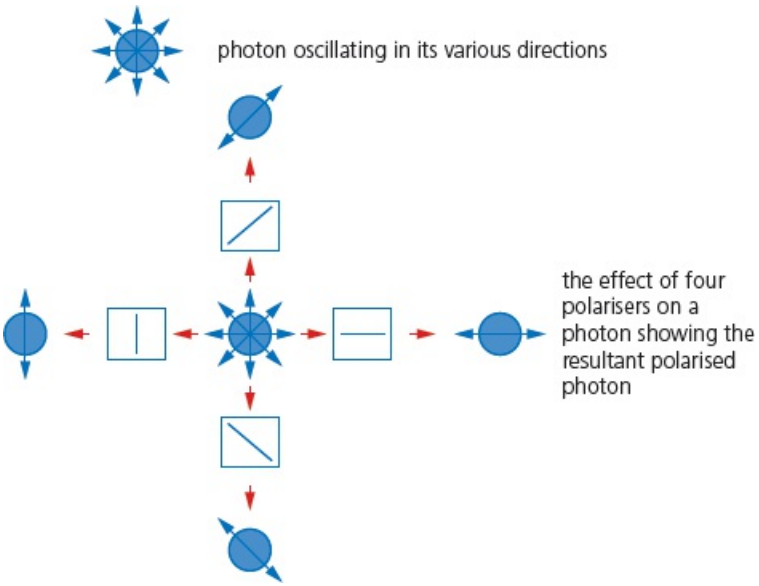


Figure 17.5

So, how do we use quantum cryptography to send an encryption key from ‘A’ to ‘B’ using the QKD protocol?

Stage 1: The sender uses a light source to generate photons.

Stage 2: The photons are sent through four random polarisers (see Figure 17.2) which randomly give one of four possible polarisations and bit values:

vertical polarisation	≡ 1 bit	
horizontal polarisation	≡ 0 bit	
45° right polarisation	≡ 1 bit	*diagonal polarisation shows 0 bit and 1 bit simultaneously*
45° left polarisation	≡ 0 bit	

Stage 3: The polarised photon travels along a fibre optic cable to its destination.

Stage 4: At the destination, there are two beam splitters:

diagonal splitter	⊠ 'X'
vertical/horizontal splitter	⊞ 'Y'

and two photon detectors.

Stage 5: One of the two beam splitters is chosen at random and the photon detectors are read.

Stage 6: The whole process is repeated until the whole of the encryption key has been transmitted from 'A' to 'B'.

Stage 7: The recipient sends back the sequence of beam splitters that were used (for example, XXXYYXXYYXXYYYY) to the sender.

Stage 8: The sender now compares this sequence to the polarisation sequence used at the sending station.

Stage 9: The sender now informs the recipient where in the sequence the correct beam splitters were used.

Stage 10: This now ensures that the sender and recipient are fully synchronised.

Stage 11: The encryption key can again be sent and received safely; even if intercepted, the eavesdropper would find it almost impossible to read the encryption key making the whole process extremely secure. Encrypted messages can now be sent along the fibre optic cable with the decryption key being used to decode all messages.

Despite the advantages of quantum cryptography, there are some potential drawbacks:

- It requires a dedicated line and specialist hardware, which can be expensive to implement initially.
- It still has a limited range (at the time of writing the limit is about 250 km).
- It is possible for the polarisation of the light to be altered (due to various conditions) while travelling down fibre optic cables.
- Due to the inherent security system generated by quantuin cryptography, terrorists and other criminals can use the technology to hide their activities from government law enforcers.

17.3 Protocols

Key terms

Secure Sockets Layer (SSL) – security protocol used when sending data over the internet.

Transport Layer Security (TLS) – a more up-to-date version of SSL.

Handshake – the process of initiating communication between two devices. This is initiated by one device sending a message to another device requesting the exchange of data.

Session caching – function in TLS that allows a previous computer session to be 'remembered', therefore preventing the need to establish a new link each time a new session is attempted.

Certificate authority (CA) – commercial organisation used to generate a digital certificate requested by website owners or individuals.

Public key infrastructure (PKI) – a set of protocols, standards and services that allow users to authenticate each other using digital certificates issued by a CA.

The two main protocols used to ensure security when using the internet are **Secure Sockets Layer (SSL)** and **Transport Layer Security (TLS)**; these are both part of the transport layer discussed in Chapter 14.

TLS is the more modern; it is based on SSL. The primary use of SSL and TLS is in the client/server application (see Chapter 2). They both use the standard cryptographic protocols to ensure there is a secure and authenticated communication between client and server. However, normally only the server is authenticated with the client remaining unauthenticated. Once a secure link between server and client is established, SSL or TLS protocols ensure no third party can eavesdrop.

17.3.1 Secure Sockets Layer (SSL)

When a user logs onto a website, SSL encrypts the data – only the client's computer and the web server are able to make sense of what is being transmitted. Two other functions of SSL are data compression (reducing the amount of data being transmitted), and data integrity checks. A user will know if SSL is being applied when they see the https protocol and/or the small green closed padlock.

The browser address display is different when the http or https protocol is used:

Figure 17.6

Similar banners will be seen when using TLS.

As mentioned in Chapter 14, TCP is used to establish a connection between the client and the server. A **handshake** takes place, thus enabling communication to begin between the client and server. One part of the SSL protocol is to agree which encryption algorithms are to be used; this is essential to ensure a secure, encrypted communication takes place. To be able to create an SSL connection, a web server requires an SSL digital certificate; the website owner needs to obtain this certificate to allow SSL protocols to be used (see Section 17.4).

Examples of where and when SSL (and TLS) would be used include

- online banking and all online financial transactions
- online shopping/commerce
- sending software to a restricted list of users
- sending and receiving emails
- using cloud storage facilities
- intranets and extranets (as well as the internet)
- using virtual private networks (VPNs)
- using Voice over Internet Protocols (VoIP) for video and/or audio chatting over the internet
- using instant messaging
- making use of a social networking site.

17.3.2 Transport Layer Security (TLS)

Transport Layer Security (TLS) is a modern, more secure version of SSL – it provides encryption, authentication and data integrity in a more effective way. It ensures the security and privacy of data between devices and users when communicating over a network (such as the internet). When a website and client communicate over the internet, TLS prevents third party eavesdropping.

TLS is formed of two main layers:

1. **Record protocol** can be used with or without encryption (it contains the data being transmitted over the network/internet).
2. **Handshake protocol** permits the web server and client to authenticate each other and to make use of encryption algorithms (a secure session between client and server is then established).

Only the most recent web browsers support both SSL and TLS, which is why the older, less secure, SSL is still used in many cases (although soon SSL will not be supported and users will have to adopt the newer TLS protocol if they wish to access the internet using a browser).

- It is possible to extend TLS by adding new authentication methods (unlike SSL).
- TLS can make use of **session caching** which improves the overall performance of the communication when compared to SSL (see below).
- TLS separates the handshaking process from the record protocol (layer) where all the data is held.

Session caching

When opening a TLS session, it requires considerable computer time (due mainly to complex cryptographic processes taking place). The use of session caching can avoid the need to utilise as much computer time for each connection. TLS can either establish a new session or attempt to resume an existing session; using the latter can considerably boost the system performance.

Summary

As already indicated, two of the main functions of SSL/TLS are

- the encryption of data
- the identification of client and server to ensure each knows who they are communicating with.

Stage 1: Once the client types in the URL into the browser and hits the <enter> key, several steps will occur before any actual encrypted data is sent; this is known as the handshaking stage.

Stage 2: The client's browser now requests secure pages (https) from the web server.

Stage 3: The web server sends back the SSL digital certificate (which also contains the public key) – the certificate is digitally signed by a third party called the **certificate authority (CA)** (see Section 17.4.2).

Stage 4: Once the client's browser receives the digital certificate, it checks

- the digital signature of the CA (is it one of those in the browser's trusted store – a list of trusted CAs is part of the browser which the client downloads to their computer)

- if the start and end dates shown on the certificate are still valid
- if the domain listed in the certificate is an exact match with the domain requested by the client in the first place.

Stage 5: Once the browser trusts the digital certificate, the public key (which forms part of the digital certificate) is used by the browser to generate a temporary session key with the web server; this session key is then sent back to the web server.

Stage 6: The web server uses its private key to decrypt the session key and then sends back an acknowledgement that is encrypted using the same session key.

Stage 7: The browser and web server can now encrypt all the data/traffic sent over the connection using this session key; a secure communication can now take place.

The **public key infrastructure (PKI)** is a set of protocols, standards and services that allow clients and servers to authenticate each other using digital certificates issued by the CA (for example, X509, PKI X.509); digital signatures also follow the same protocol. PKI requires the provider to use an encryption algorithm to generate public and private keys.

17.4 Digital signatures and digital certificates

Key terms

Digital signature – electronic way of validating the authenticity of digital documents (that is, making sure they have not been tampered with during transmission) and also proof that a document was sent by a known user.

Digest –a fixed-size numeric representation of the contents of a message produced from a hashing algorithm. This can be encrypted to form a digital signature.

Hashing algorithm (cryptography) – a function which converts a data string into a numeric string which is used in cryptography.

Digital certificate – an electronic document used to prove the identity of a website or individual. It contains a public key and information identifying the website owner or individual, issued by a CA.

17.4.1 Digital signatures

Digital signatures are a way of validating the authenticity of digital documents and identifying the sender (signing with a digital signature indicates that the original message, document or file is safe and has not been tampered with). As mentioned earlier on, there are four main purposes of digital signatures: authentication, non-repudiation, data integrity and confidentiality. A digital signature is a digital code which is often derived from the digital certificate (described below), although other methods of generating digital signatures will be described throughout this section.

The example used in Section 17.1 required Meera to send her public key to each of the workers, and she used her private key to decrypt their messages. However, the two keys can be reversed – the other workers can encrypt messages using their own private keys and then send these encrypted messages to other workers in the company, who use their matching public key to decrypt the messages. While this would be a bad idea if the messages were confidential, it could be used as a way of identifying or verifying who the sender of the message was (in other words, the private key would act like a digital signature, identifying the sender, since the private keys will be unique to the sender).

This also needs a lot of processing time to encrypt everything in the message. The following method, which is used to identify the sender and ensure the message was not tampered with, does not encrypt the messages but uses a generated numerical value known as a **digest**.

With this method, to actually identify the sender, it is not necessary to encrypt the whole message. The plaintext message is put through a **hashing algorithm** which produces the digest.

For example, if the first page of this chapter was going to be sent, we could put it through a hashing algorithm (such as MD4) and it would produce a digest, for example, it might produce the following digest:

873add9ed804fc5ce0338d2e9f7e0962

The sender's private key and digest are then put through an encryption algorithm to produce a digital signature.

Therefore, the plaintext and digital signature are sent to the recipient as two separate files. The recipient puts the digital signature through a decryption algorithm (using the sender's public key) to produce a digest. The recipient then puts the plaintext through the same hashing algorithm and also produces a digest.

If these two digests are the same, then the document has been sent correctly (and has not been tampered with). Since this process does not encrypt the document, if it needed to be kept confidential then it would be necessary to put the document through the asymmetric encryption process, as described earlier, before sending.

Note: a digest is a fixed-size numerical value which represents the content of a message. It is generated by putting the message through a hashing algorithm. The digest can be encrypted to produce a digital signature.

Figure 17.7 outlines the process.

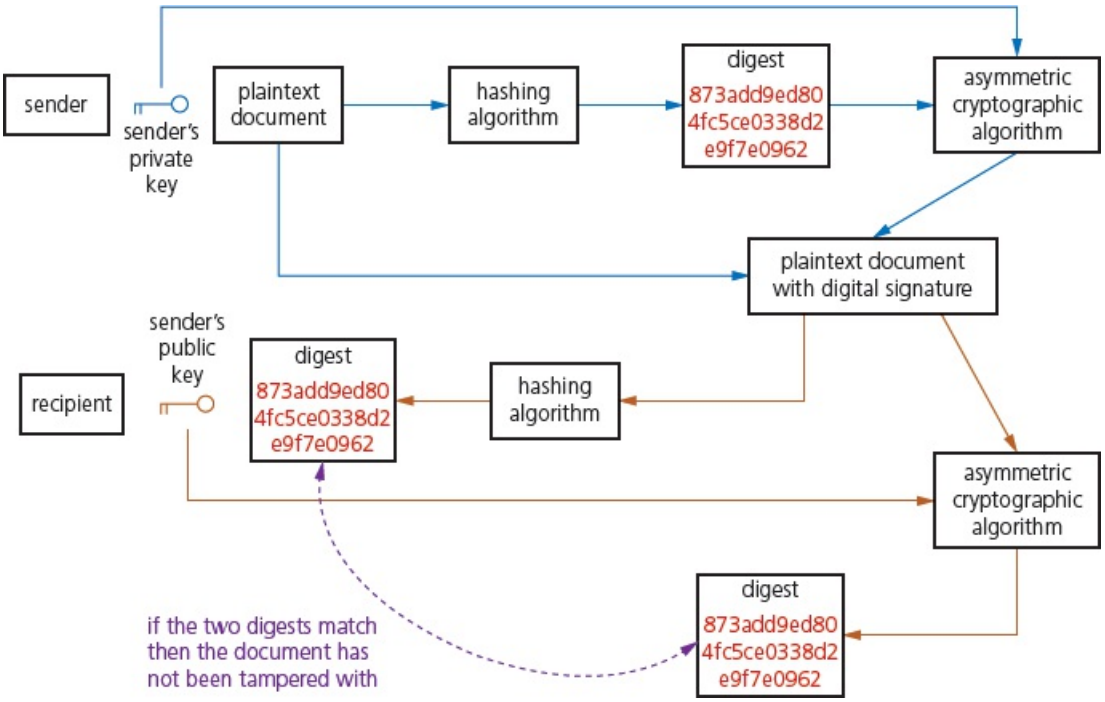


Figure 17.7

However, this method still is not safe enough, since the public key could be forged by a third party, which means the recipient still cannot be certain that the message came from a legitimate source. Therefore, an even more robust system is needed to give confidence that the sender is really who they claim to be.

17.4.2 Digital certificates

A **digital certificate** is an electronic 'document' used to prove the online identity of a website or an individual. The certificate contains a public key and other information identifying the owner of the certificate. A digital certificate is issued by the certificate authority (CA) – they independently validate the identity of the certificate owner.

This is a list of the items commonly found on a digital certificate

- version number
- serial number of certificate
- algorithm identification
- name of certificate issuer
- validity (start date and expiry date of certificate)
- company details
- public key
- issuer's identifier
- company's identifier
- signature algorithm used
- digital signature.

The digital signature is created by condensing all of the certificate details and then putting it through a hashing algorithm (such as MD4/5). The number generated is then put through an encryption algorithm, together with the CA's private key, thus producing a digital signature.

Figure 17.8 shows how a user can apply for a digital certificate. Figure 17.9 shows what a typical SSL digital certificate looks like.

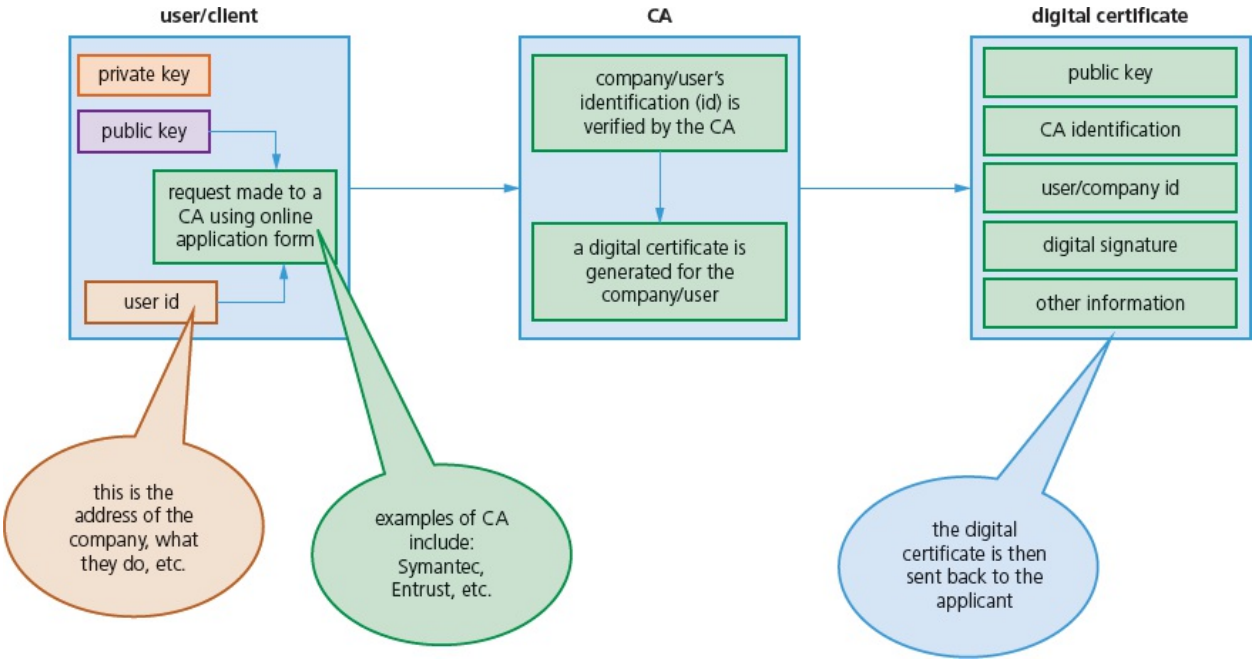


Figure 17.8

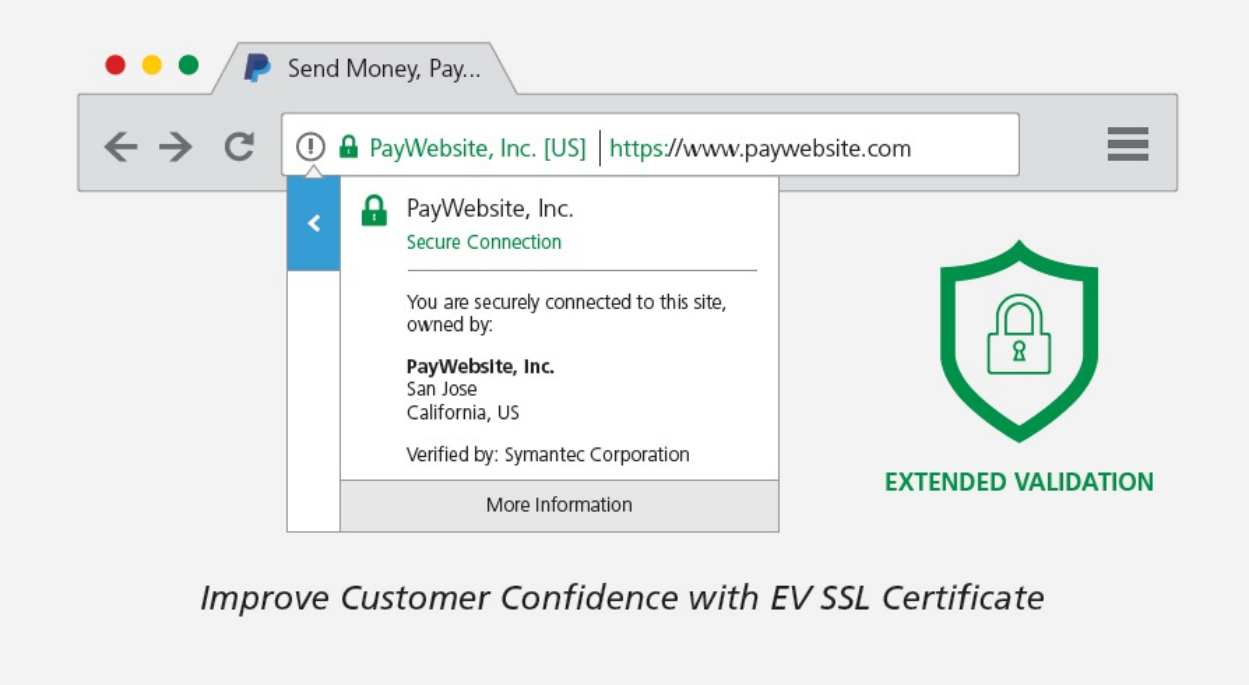


Figure 17.9

It is possible for a user to produce a self-signed digital certificate rather than use a commercial CA (for example, if an individual builds their own website, called my-site.com, and wants to make this generally available on the internet, they could produce their own digital certificate).

However, if a user attempts to log onto my-site.com they might see an error screen, like this:

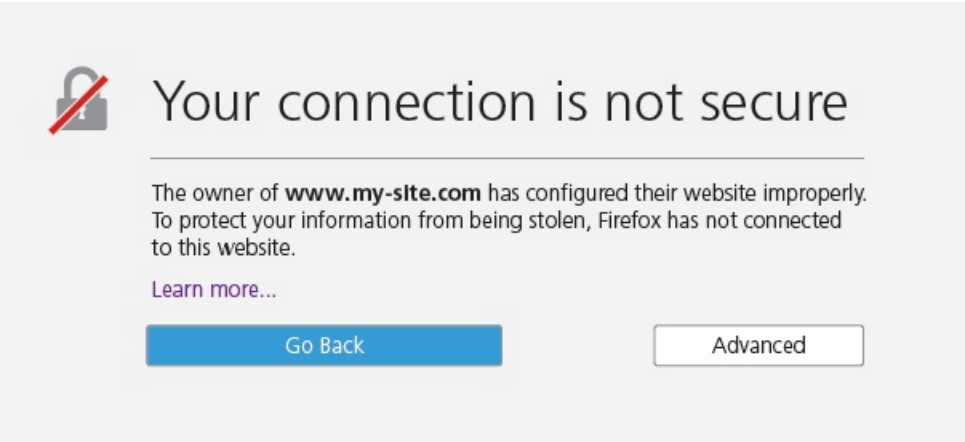


Figure 17.10

ACTIVITY 17A

For each of the following questions, choose the option which corresponds to the correct response.

1 What is meant by the term *cipher* when used in cryptography?

A an encryption or decryption algorithm
B an encrypted message
C a type of session key
D a digital signature
E text following an encryption algorithm

2 When carrying out asymmetric encryption, which of the following users would keep the private key?
A the sender
B the receiver
C both sender and receiver
D all recipients of the message
E none of the above

3 In cryptography, which of the following is the term used to describe the message before it is encrypted?
A simpletext
B plaintext
C notext
D ciphertext
E firsttext

4 Which of the following is the biggest disadvantage of using symmetric encryption?
A it is very complex and time consuming
B it is rarely used any more
C the value of the key reads the same in both directions
D it only works on computers with older operating systems
E there is a security problem when transmitting the secret key

5 Which of the following is the correct name for a form of encryption in which both the sender and the recipient use the same key to encrypt/decrypt?
A symmetric key encryption
B asymmetric key encryption
C public key encryption
D same key encryption
E block cipher encryption

6 Which of the following is involved in temporary key generation?
A session keys
B private key and certificate
C public key and certificate
D master keys
E public keys

7 Which of the following is a correct statement about PKIs?
A they use private and public keys but not digital certificates

B they use digital signatures and public keys

C they are a combination of digital certificates, public key cryptography and CAs

D they use asymmetric keys, hashing algorithms and certificate authorities

E they are a combination of digests, hashing algorithms and asymmetric cryptographic algorithms

8 SSL provides which of the following?

A message integrity only

B confidentiality only

C compression and authentication

D message integrity, confidentiality and compression

E authentication, encryption and digital signatures

9 Which of the following indicates a secure website?

A http and closed padlock

B http and open padlock

C https and closed padlock

D https and open padlock

E green closed padlock only

10 Which of the following is not part of security?

A non-repudiation

B bit streaming

C data integrity

D data privacy

E user authentication

End of chapter questions

1 a) Explain what is meant by QKD.

[2]

b) The following eleven statements refer to the transmission of an encryption key using quantum key distribution protocols.
Put each statement into its correct sequence, 1-11. The first one has been numbered for you.

[10]

sequence	statement
	the sender and receiver are now fully synchronised
	the photons are sent through four random polarisers which give one of four possible polarisations and bit values
	the process is repeated until the whole of the encryption key has been transmitted

1	the sender uses a light source to create the photons
	one of the two beam splitters is chosen at random and the photon detectors are read
	the sender now informs the recipient where, in the sequence, the correct beam splitters had been used
	the polarised photons travel along the fibre optic cable to the destination
	the encryption key can now be sent and received safely since eavesdroppers would find it impossible to crack the key code
	the sender now compares this sequence to the polarisation sequence used by the sending station
	at the destination, there are two beam splitters (diagonal and vertical/ horizontal) and two photon detectors
	the recipient sends back the sequence of beam splitters to the sender

2 a) Explain the terms *SSL* and *TLS*.

[3]

b) Explain the following terms used in TLS.

i) Record protocol

ii) Handshake protocol

iii) Session caching

[5]

c) Give **two** differences between SSL and TLS.

[2]

3 A user keys a URL into their browser and hits the <enter> key.
Re-order the following stages, 1-6, to show how an SSL digital certificate is used to set up a secure connection between client (user) and website.

[6]

order	**stage**
	browser and web server now encrypt all data/traffic sent over the connection using the session key and a secure communication can now take place
	client's browser requests secure pages (https) from the web server
	once trusted, the browser uses public key to agree temporary session key with web server; session key is sent back to web server
	the web server uses its private key to decrypt the session key and then sends back an acknowledgement that is encrypted using the session key
	once the client's browser gets the SSL digital certificate it checks the digital

	signature, validity of start and end dates and whether the domain listed in the certificate matches the domain requested by the user
	the web server sends back the SSL digital certificate containing the public key; this is digitally signed by a third party called the Certificate Authority (CA)

b) List **four** items found on a digital certificate.

[4]

c) Explain how a digital signature can be formed from a digital certificate.

[2]

18 Artificial intelligence (AI)

In this chapter, you will learn about

- how to use A* and Dijkstra's algorithms to find the shortest path in a graph
- how artificial neural networks have helped with machine learning
- deep learning, machine learning and reinforcement learning and the reasons for using these methods in AI
- supervised learning, active learning and unsupervised learning when applied to AI
- back propagation of errors and regression methods in machine learning.

WHAT YOU SHOULD ALREADY KNOW

Try these three questions before you start this chapter.

1 What is meant by the term *artificial intelligence (AI)*?

2 Describe some of the pros and cons of using AI in everyday life.

3 Give **four** examples of the use of AI and briefly describe each.

18.1 Shortest path algorithms

Key terms

Dijkstra's algorithm – an algorithm that finds the shortest path between two nodes or vertices in a graph/network.

Node or **vertex** – fundamental unit from which graphs are formed (nodes and vertices are the points where edges converge).

A* algorithm – an algorithm that finds the shortest route between nodes or vertices but uses an additional heuristic approach to achieve better performance than Dijkstra's algorithm.

Heuristic – method that employs a practical solution (rather than a theoretical one) to a problem; when applied to algorithms this includes running tests and obtaining results by trial and error.

18.1.1 Dijkstra's algorithm

Dijkstra's algorithm (pronounced *dyke – strah*) is a method of finding the shortest path between two points on a graph. Each point on the graph is called a **node** or a **vertex (for more information on the features and uses of graphs, see Chapter 19)**. It is the basis of technology such as GPS tracking and, therefore, is an important part of AI.

This set of instructions briefly describes how it works.

1. Give the start vertex a final value of 0.
2. Give each vertex connected to the vertex we have just given a final value to (in the first instance, this is the start vertex) a working (temporary) value. If a vertex already has a working value, only replace it with another working value if it is a lower value.
3. Check the working value of any vertex that has not yet been assigned a final value. Assign the smallest value to this vertex; this is now its final value.
4. Repeat steps 2 and 3 until the end vertex is reached, and all vertices have been assigned a final value.
5. Trace the route back from the end vertex to the start vertex to find the shortest path between these two vertices.

Here is a step-by-step example.

Suppose we have the following graph (route map) with seven vertices labelled A to G. We want to find the shortest path between A and G. The numbers show the distance between each pair of vertices.

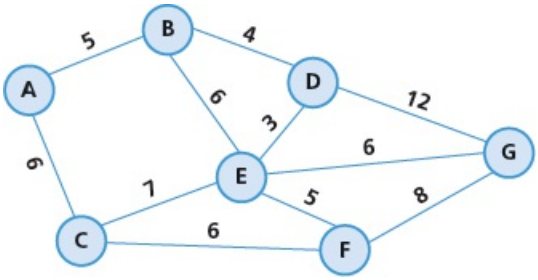


Figure 18.1

First, redraw the diagram, replacing the circled letters as per the key:

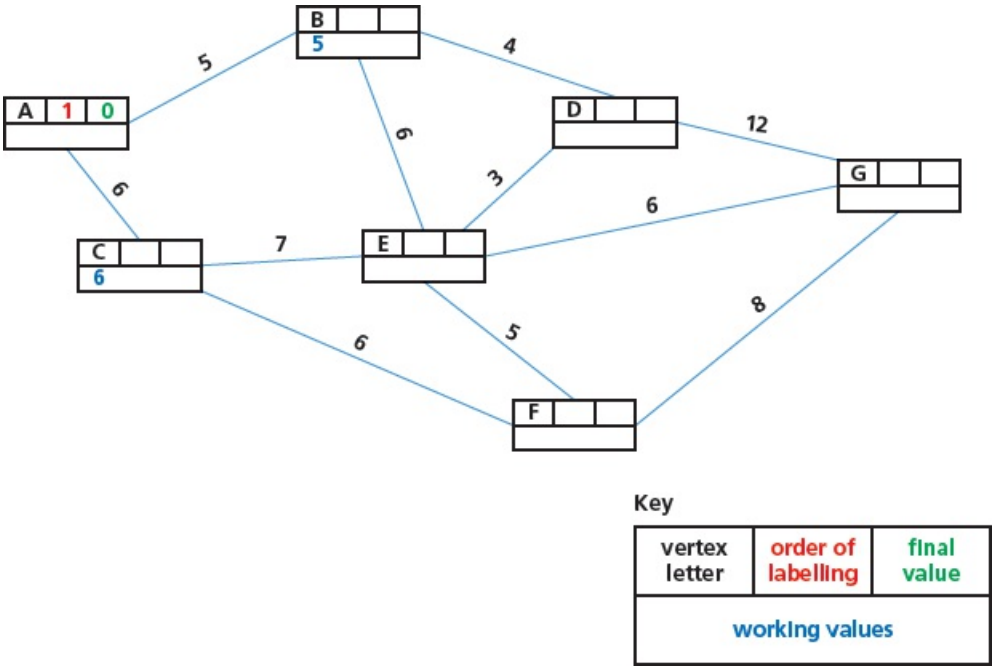


Figure 18.2

Set the final value of the start vertex (vertex A) to 0 (as per step 1 above).

The two vertices connected to A (B and C) are given the working values 5 and 6 respectively.

Make the smallest working value (vertex B) a final value. Then give the vertices connected to B (D and E) working values based on the original distances. The working value for E is the final value of B plus the value of B to E (5 + 6 = 11). The working value for D is the final value of B plus the value of B to D (5 + 4 = 9).

The diagram now looks like this:

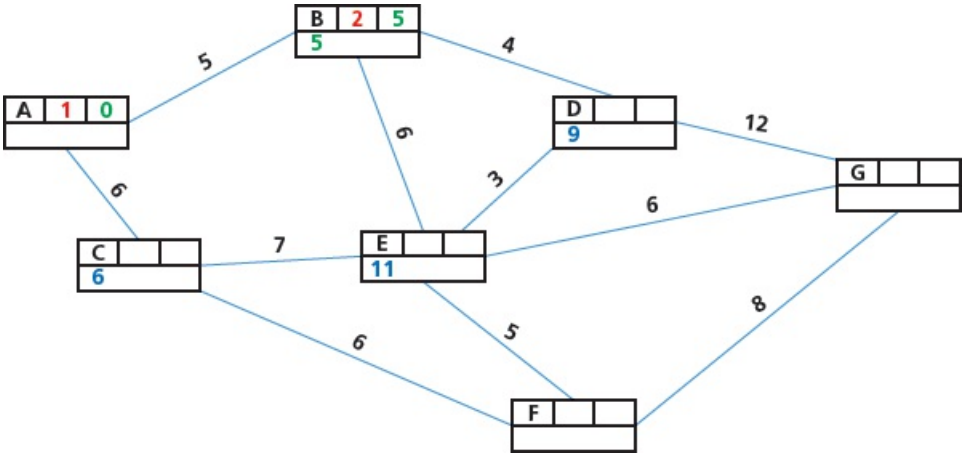


Figure 18.3

Make the smallest working value a final value: vertex C becomes 6.

Now give working values to all vertices connected to C. Note that the working value for E remains 11 since the final value of C plus the value of C to E is 13, which is greater than 11.

Vertex D retains its working value since it is not connected to C and is not affected.

Vertex F takes the working value of C plus the value of C to F (6 + 6 = 12).

The diagram now looks like this:

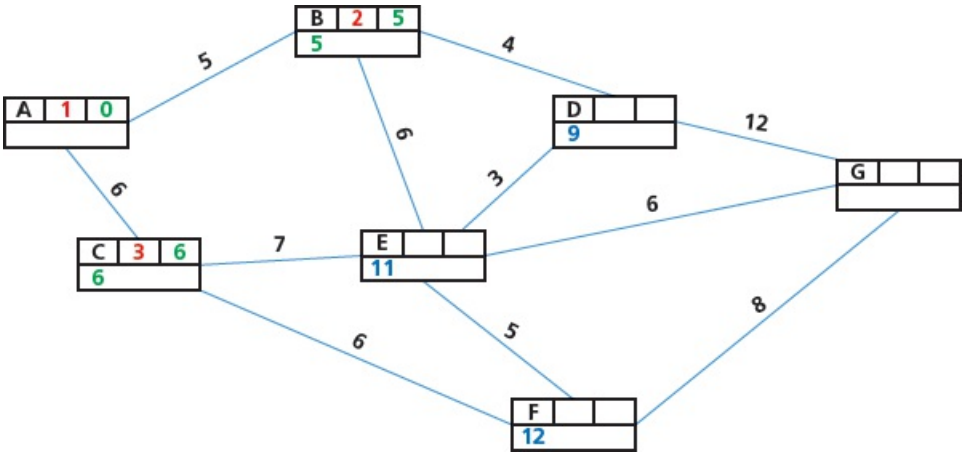


Figure 18.4

Vertex D now has the smallest working value (9), so this becomes a final value.

Vertices E and G are connected to D, so these are now assigned working values. Note that G has the working value 21 since it is the final value of D plus the value of D to G (9 + 12 = 21); E keeps the value of 11 since the final value of D plus the value of D to E is greater than 11 (9 + 11 = 20).

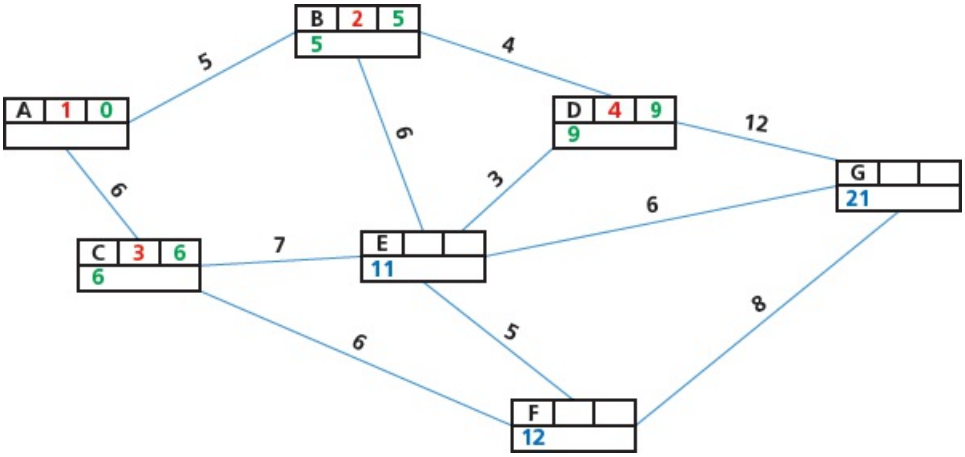


Figure 18.5

Vertex E now has the smallest working value (11), so this becomes a final value.

Vertices D, F and G are all connected to E.

D already has a final value so it can be ignored.

F retains its value of 12 since E + E to F = 16 (> 12).

G changes since E + E to G = 17 (< 21).

The diagram now looks like this:

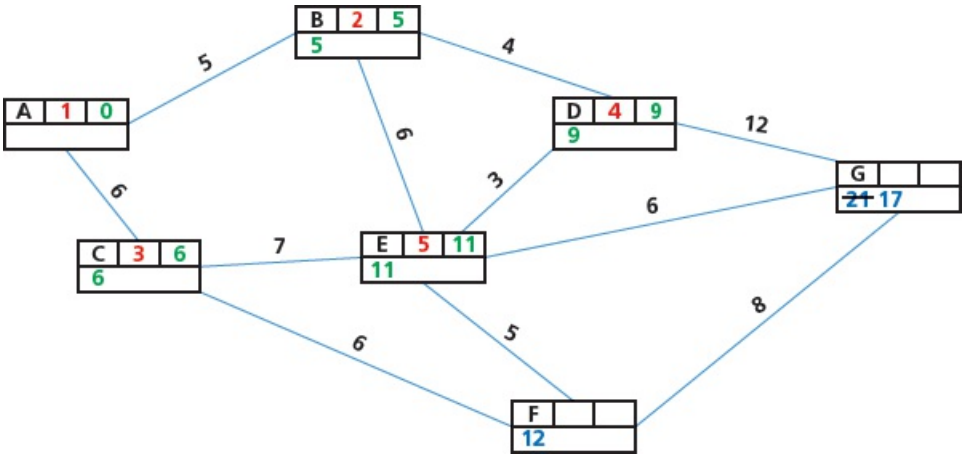


Figure 18.6

Vertex F now has the smallest working value (12), so this becomes a final value.

G retains its value of 17 since F + F to G = 20 (> 17).

The final diagram now looks like this:

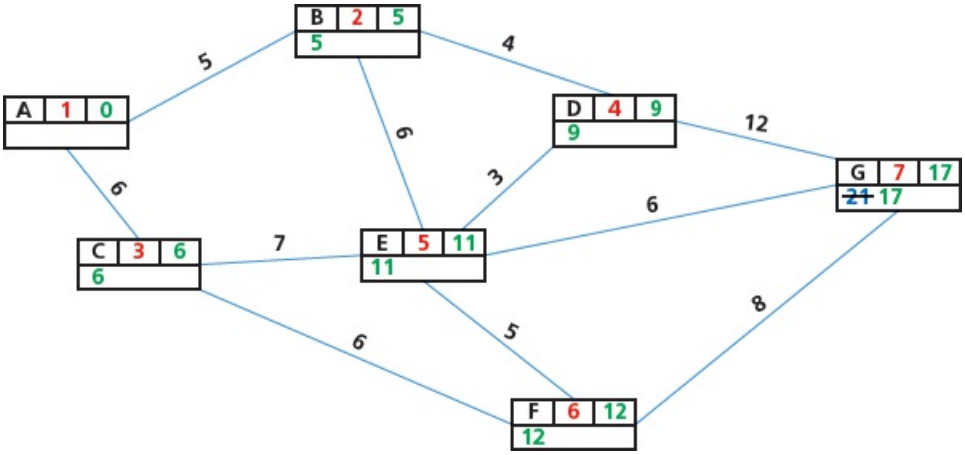


Figure 18.7

The final step is to work back from G to A.

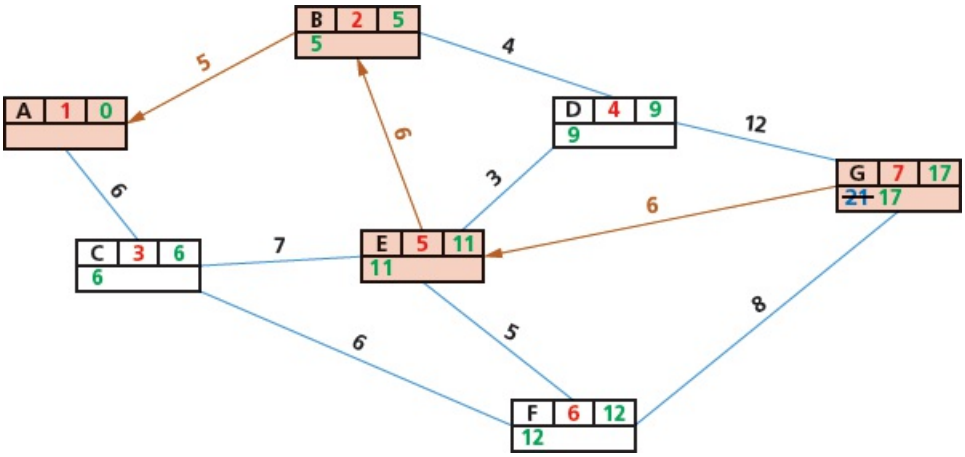


Figure 18.8

Thus, the shortest path is: A → B → E → G

The reasoning is as follows:

- The difference between the final values E and G is 6, which is the same as the path length E to G.
- The difference between the final values of B and E is 6, which is the same as the path length B to E.
- The difference between the final value of B and A is 5, which is the same as the path length A to B.

You will know if the shortest route is correct by applying this rule:

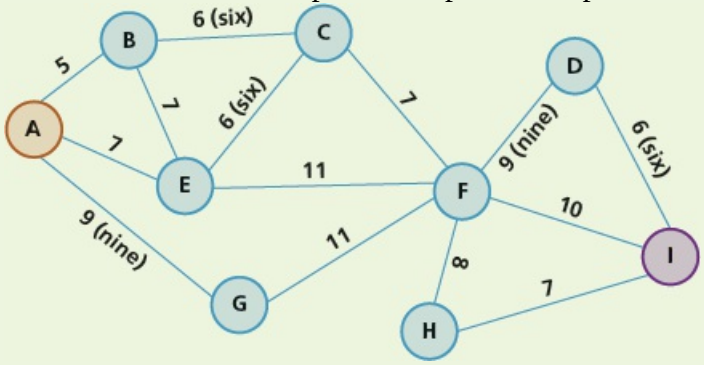
Path length is the same as the difference between final values at either end of the path.

If the path length between two points is not the same as the difference of the final values between the same two points, then this is not a route that can be taken.

ACTIVITY 18A

The following graph shows the routes through a large park.

Use Dijkstra's algorithm to find the shortest path from point A to point I.



18.1.2 A* algorithm

Dijkstra's algorithm simply checks each vertex looking for the shortest path, even if that takes you away from your destination – it pays no attention to direction. With larger, more complex problems, that can be a time-consuming drawback.

A* algorithm is based on Dijkstra, but adds an extra **heuristic (h)** value – an 'intelligent guess' on how far we have to go to reach the destination most efficiently.

The A* algorithm can also find shortest path lengths for graphs similar to the type used in Section 18.1.1.

Suppose we have the following graph made up of an 8 × 6 matrix. White squares show permitted moves, and grey squares show blocked moves.

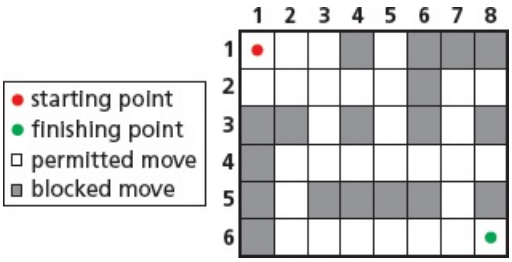


Figure 18.9

Each of the parts of the graph are called nodes (or vertices). Each node has four values

- h (the heuristic value)
- g (movement cost)
- f (sum of g and h values)
- n (previous node in the path).

Note that the weight of a node usually represents movement cost, which is the distance between the two nodes.

First, find the heuristic values (distances) using the Manhattan method (named after the criss-cross street arrangement in New York). The distance from the starting square (1, 1) to the end square (8, 6) is 12 squares (follow the purple line in the diagram below). Similarly, the distance from square (2, 4) to (8, 6) is eight squares (follow the orange line in the diagram below).

Note: you can ignore the blocked moves when calculating heuristic distance from each node to the final node.

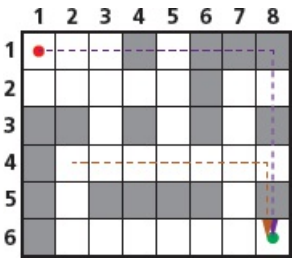


Figure 18.10

Use this method to find the heuristic distances for all permitted nodes:

h-values:

	1	2	3	4	5	6	7	8
1	12	11	10		8			
2	11	10	9	8	7		5	4
3			8		6		4	
4		8	7	6	5	4	3	2
5		7					2	
6		6	5	4	3	2	1	–

Figure 18.11

Now, find the g-values (the movement costs). Since we can either move up/down, left/right or diagonally, we can choose our g-values based on a right-angled triangle. To make the maths easy we will use a triangle with sides 10, 10 and 14:

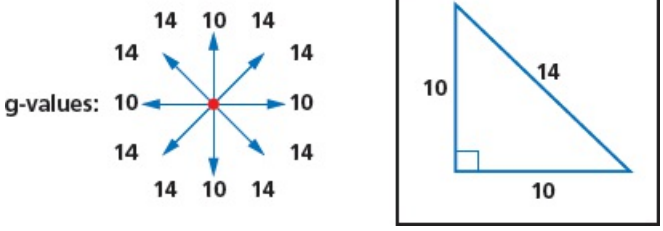


Figure 18.12

Find the f values using f(n) = g(n) + h(n).

Starting with square (1, 1), look at the surrounding squares:

g-values

•	10	-
10	14	-
		-

h-values

•	11	10
11	10	9
		8

Figure 18.13

- square (1, 2): f = 10 + 11 = 21
- square (2, 1): f = 10 + 11 = 21
- square (2, 2): f = 14 + 10 = 24

Since 21 < 24, (1, 2) or (2, 1) are the possible directions.

We will choose (2, 1) as the next step:

g-values

-	•	10
14	10	14
		-

h-values

-	•	10
11	10	9
		8

Figure 18.14

- square (3, 1): f = 10 + 10 = 20
- square (3, 2): f = 14 + 9 = 23

- square (1, 2): f = 14 + 11 = 25
- square (2, 2): f = 10 + 10 = 20

Since 20 is the smallest value, the next stage can be (3, 1) or (2, 2).

We will choose (3, 1) as the next step:

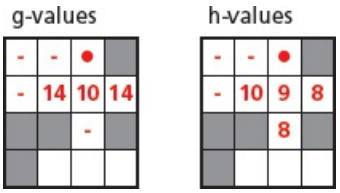


Figure 18.15

- square (2, 2): f = 14 + 10 = 24
- square (3, 2): f = 10 + 9 = 19
- square (4, 2): f = 14 + 8 = 22

Since square (3, 2) is the smallest value, this must be the next step.

Now look at the possible routes already found to decide where to move next:

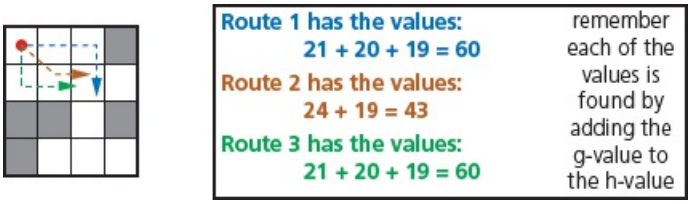


Figure 18.16

It seems route 2 is the shortest route: (1, 1) → (2, 2) → (3, 2).

When considering the next squares (3, 3) or (4, 2), and applying the above rules, it becomes clear that the next stage in the route is:

(1, 1) → (2, 2) → (3, 3)

Continue throughout the matrix and produce the following shortest route from (1, 1) to (8, 6):

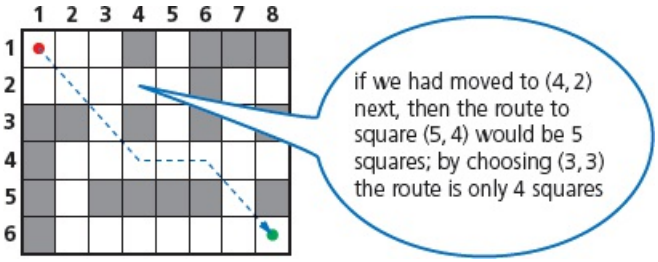


Figure 18.17

The shortest path is:

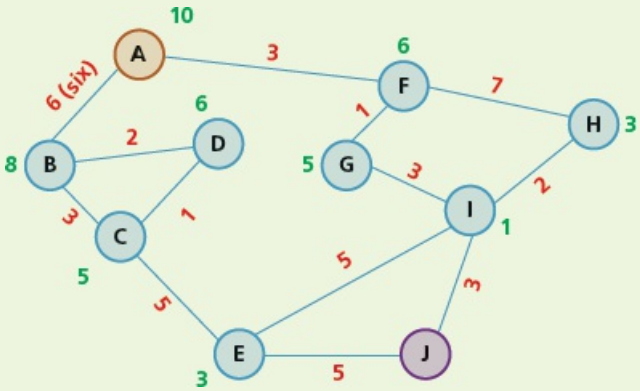
(1, 1) → (2, 2) → (3, 3) → (4, 4) → (5, 4) → (6, 4) → (7, 5) → (8, 6)

Examples of applications of shortest path algorithms include

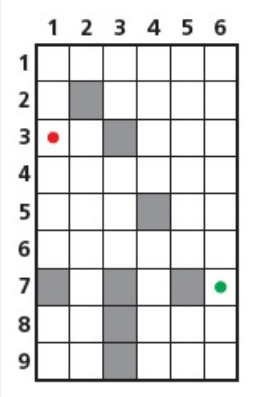
- global positioning satellites (GPS)
- Google Maps
- modelling the spread of infectious diseases
- IP routing.

ACTIVITY 18B

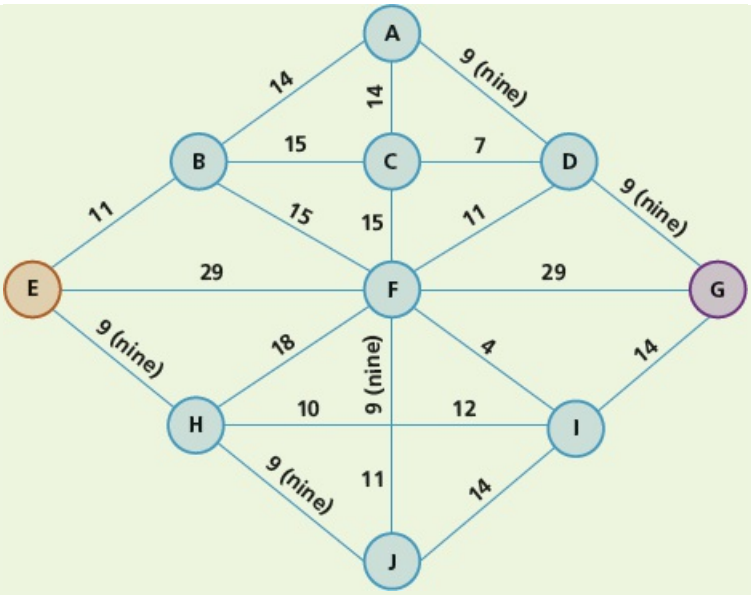
1 Find the shortest route from node A to node J using the A* algorithm. Note that you are given the heuristic values (h) in green, and the movement cost values (g), in red.



2 Use the A* algorithm to find the shortest route from the starting point (1, 3) to the end point (6, 7). Note that you will need to calculate your own h-values and g-values in this question using the method shown earlier.



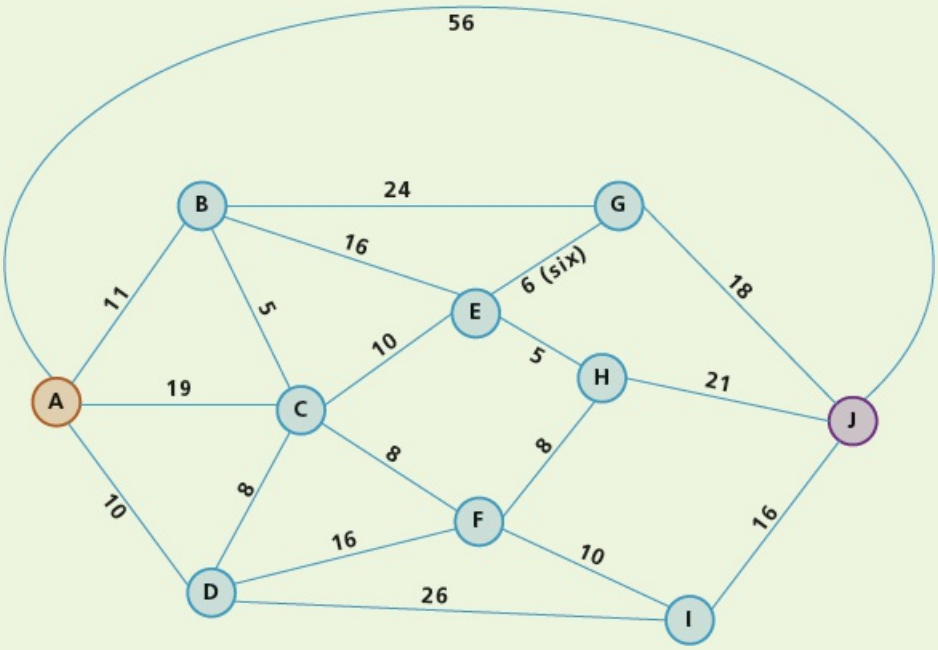
3 The following network shows 11 places of interest in a city. The times (in minutes) to walk between pairs of places of interest are shown on the vertices.



a) Using Dijkstra's algorithm, find the shortest time to walk from E to G.

b) Write down the corresponding shortest route.
Total walking time is 288 minutes.

4 The road network below connects a number of towns. The distances shown are in kilometres (km) between roads connecting the towns.



a) **i)** Use Dijkstra's algorithm on the road network to find the minimum distance between towns A and J.

ii) Write down the corresponding shortest route.

b) The road AJ is a dual carriageway where the speed limit is 95 kph.
The speed limit on all other roads is 80 kph.
Assuming Karl drives at the maximum speed limit on each road, calculate the minimum

time to drive from town A to town J.

18.2 Artificial intelligence, machine learning and deep learning

Key terms

Machine learning – systems that learn without being programmed to learn.

Deep learning – machines that think in a way similar to the human brain. They handle huge amounts of data using artificial neural networks.

Labelled data – data where we know the target answer and the data object is fully recognised.

Unlabelled data – data where objects are undefined and need to be manually recognised.

Supervised learning – system which is able to predict future outcomes based on past data. It requires both input and output values to be used in the training process.

Unsupervised learning – system which is able to identify hidden patterns from input data – the system is not trained on the 'right' answer.

Reinforcement learning – system which is given no training – learns on basis of 'reward and punishment'.

Semi-supervised (active) learning – system that interactively queries source data to reach the desired result. It uses both labelled and unlabelled data, but mainly unlabelled data on cost grounds.

Reward and punishment – improvements to a model based on whether feedback is positive or negative; actions are optimised to receive an increase in positive feedback.

Web crawler – internet bot that systematically browses the world wide web to update its web page content.

Artificial neural networks – networks of interconnected nodes based on the interconnections between neurons in the human brain. The system is able to think like a human using these neural networks, and its performance improves with more data.

Back propagation – method used in artificial neural networks to calculate error gradients so that actual node/neuron weightings can be adjusted to improve the performance of the model.

Chatbot – computer program set up to simulate conversational interaction between humans and a website.

Regression – statistical measure used to make predictions from data by finding learning relationships between the inputs and outputs.

18.2.1 Artificial Intelligence (AI)

Figure 18.18 shows the link between artificial intelligence (AI), **machine learning** and **deep learning**. Deep learning is a subset of machine learning, which is itself a subset of AI.

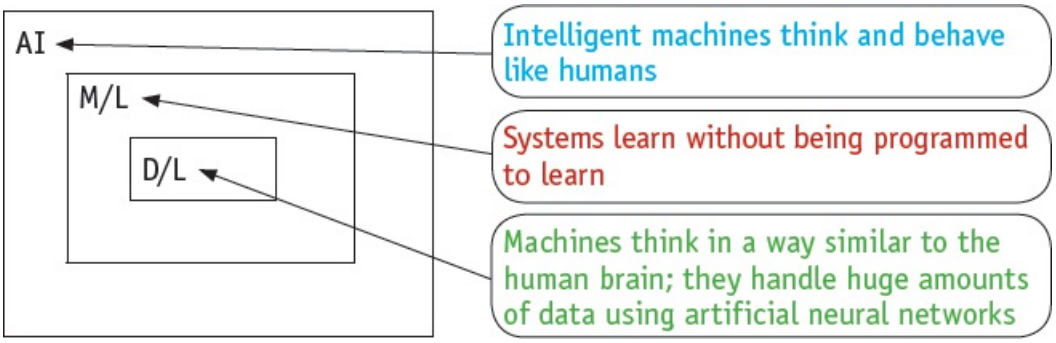


Figure 18.18

AI can be thought of as a machine with cognitive abilities such as problem solving and learning from examples. All of these can be measured against human benchmarks such as reasoning, speech and sight. AI is often split into three categories.

1. **Narrow AI** is when a machine has superior performance to a human when doing one specific task.
2. **General AI** is when a machine is similar in its performance to a human in any intellectual task.
3. **Strong AI** is when a machine has superior performance to a human in many tasks.

EXTENSION ACTIVITY 18A

Carry out some research into the following AI concepts.

- Knowledge representation
- Automated reasoning
- Computer vision
- Robotics

Examples of AI include

- news generation based on live news feeds (this will involve some form of human interaction)
- smart home devices (such as Amazon Alexa, Google Now, Apple Siri and Microsoft Cortana); again these all involve some form of human interaction (see Figure 18.19).

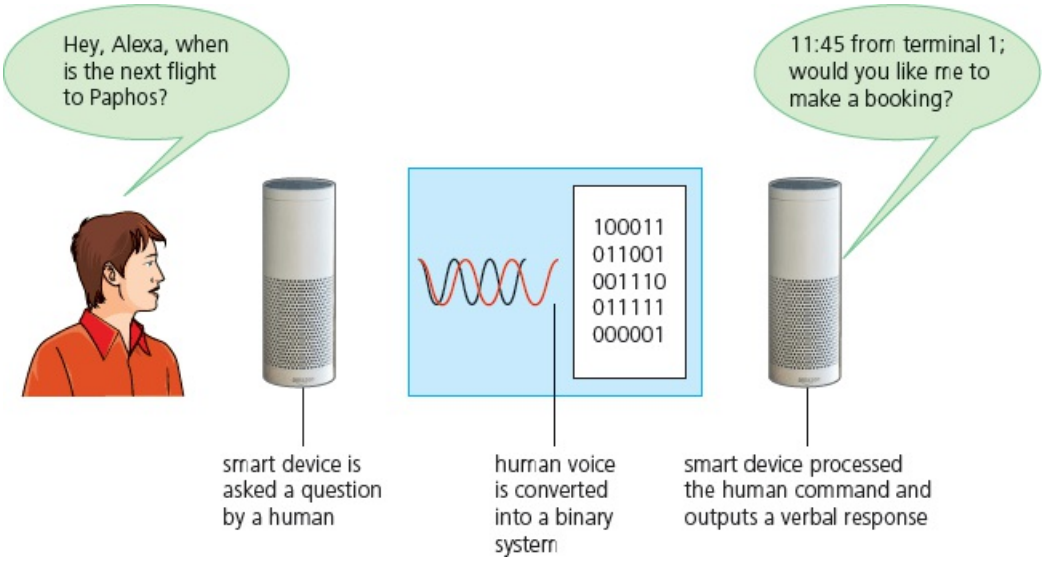


Figure 18.19

In this example, the AI device interacts with a human by recognising their verbal commands. The device will learn from its environment and the data it receives, becoming increasingly sophisticated in its responses, showing the ability to use automated repetitive learning via artificial neural networks.

18.2.2 Machine learning

Machine learning is a subset of AI, in which the algorithms are 'trained' and learn from their past experiences and examples. It is possible for the system to make predictions or even take decisions based on previous scenarios. They can offer fast and accurate outcomes due to very powerful processing capability. One of the key factors is the ability to manage and analyse considerable volumes of complex data – some of the tasks would take humans years, if they were to analyse the data using traditional computing processing methods. A good example is a search engine:

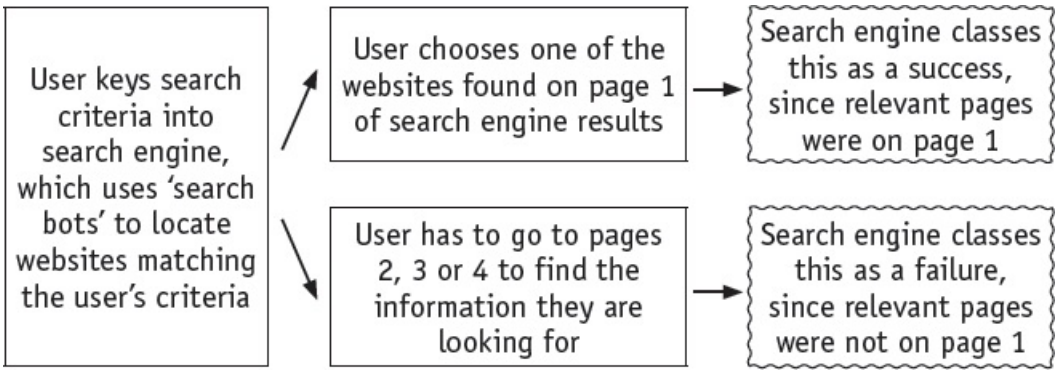


Figure 18.20

The search engine will learn from its past performance, meaning its ability to carry out searches becomes more sophisticated and accurate.

Machine learning is a key part of AI and the various types of machine learning will be covered later in this chapter.

Labelled and unlabelled data

Let us consider what is meant by **labelled** and **unlabelled data**:

Suppose a garage selling vehicles obtains them from three sources.

Vehicles from source 1 are always cars and always come fully serviced.

Vehicles from source 2 are vans and are usually unserviced.

Vehicles from source 3 are motorbikes and are usually serviced.

Vehicles less than three years old also come with a warranty. Thus, the garage has in stock

- vehicle 1 – car, serviced, warranty
- vehicle 2 – van, no service, no warranty
- vehicle 3 – car, no service, warranty
- vehicle 4 – motorbike, serviced, warranty.

Coming into stock in the next few days are

- vehicle 5 – from source 1, two years old
- vehicle 6 – from source 3, four years old

- vehicle 7 – from source 2, one year old.

Vehicles 1, 2, 3 and 4 are all labelled data since we know

- what type of vehicle they are
- whether they have been serviced
- whether they have a warranty.

They are fully defined and recognisable.

However, vehicles 5, 6 and 7 are unlabelled data since we do not know what type of vehicle they are and we only know their source and age.

Unlabelled data is data which is unidentified and needs to be recognised. Some processing would need to be done before they could be recognised as a car, van or motorbike.

Now, suppose you want to automatically count the types of birds seen in a bird sanctuary. The proposed system will consider bird features such as shape of beak, colour of feathers, body size, and so on. This requires the use of labelled data to allow the birds to be recognised by the system (see Figure 18.21).

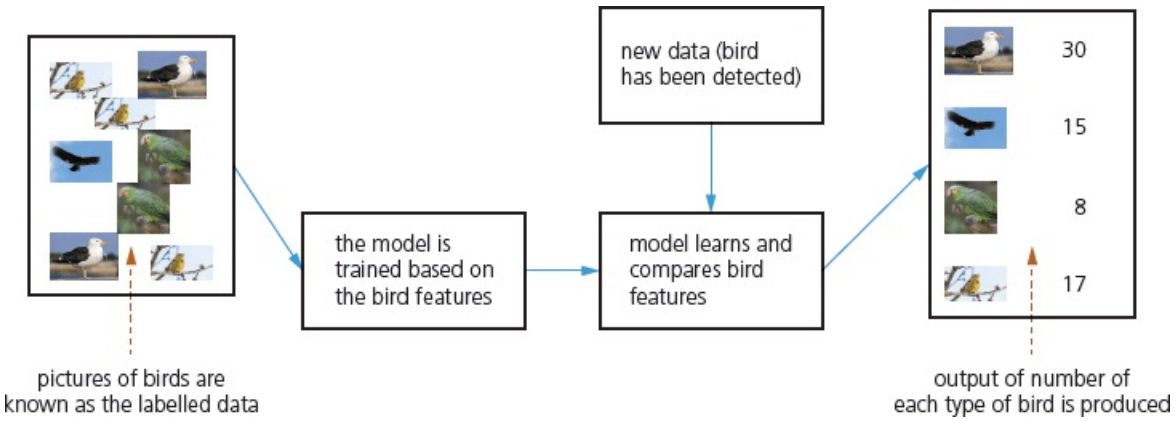


Figure 18.21

Machine learning recognises the birds as labelled data, allowing it to be trained. Once trained, it is able to recognise each type of bird from the original data set. Algorithms are used to analyse the incoming data (by comparing it to bird features already recognised by the model) and to learn from this data. Informed decisions are then made based on what it has learned. Thus, in this example, it is able to recognise new data and produce an output automatically showing how many of each type of bird was detected.

Examples of machine learning include

- spam detection (the system learns to recognise spam emails without the need of any human interactions)
- search engines refining searches based on earlier searches carried out (the system learns from its mistakes).

Types of machine learning

There are a number of different types of machine learning, including **supervised**, **unsupervised**

learning, **reinforcement** and **semi-supervised (active)**.

Supervised learning

Supervised learning makes use of regression analysis and classification analysis. It is used to predict future outcomes based on past data:

- The system requires both an input and an output to be given to the model so it can be trained.
- The model uses labelled data, so the desired output for a given input is known.
- Algorithms receive a set of inputs and the correct outputs to permit the learning process.
- Once trained, the model is run using labelled data.
- The results are compared with the expected output; if there are any errors, the model needs further refinement.
- The model is run with unlabelled data to predict the outcome.

An example of supervised learning is categorising emails as relevant or spam/junk without human intervention.

Unsupervised learning

Systems are able to identify hidden patterns from the input data provided; they are not trained using the 'right' answer.

By making data more readable and more organised, patterns, similarities and anomalies will become evident (unsupervised learning makes use of density estimation and k-mean clustering; in other words, it classifies unlabelled real data).

Algorithms evaluate the data to find any hidden patterns or structures within the data set.

An example is used in product marketing: a group of individuals with similar purchasing behaviour are regarded as a single unit for promotions.

Reinforcement learning

The system is not trained. It learns on the basis of '**reward and punishment**' when carrying out an action (in other words, it uses trial and error in algorithms to determine which action gives the highest/optimal outcome).

This type of learning helps to increase the efficiency of the system by making use of optimisation techniques.

Examples include search engines, online games and robotics.

Semi-supervised (active) learning

Semi-supervised learning makes use of labelled and unlabelled data to train algorithms that can interactively query the source data and produce a desired output.

It makes as much use of unlabelled data as possible (this is for cost reasons, since unlabelled data is less expensive than labelled data when carrying out data categorisation).

A small amount of labelled data is used combined with large amounts of unlabelled data.

Examples of uses include the classification of web pages into sport, science, leisure, finance, and so on. A **web crawler** is used to look at large amounts of unlabelled web pages, which is much

cheaper than going through millions of web pages and manually annotating (labelling) them.

18.2.3 Deep learning

Deep learning structures algorithms in layers (input layer, output layer and hidden layer(s)) to create an artificial neural network that can learn and make intelligent decisions on its own.

Its **artificial neural networks** are based on the interconnections between neurons in the human brain. The system is able to think like a human using these neural networks, and its performance improves with more data.

The hidden layers are where data from the input layer is processed into something which can be sent to the output layer. Artificial neural networks are excellent at identifying patterns which would be too complex or time consuming for humans to carry out.

For example, they can be used in face recognition. The face in Figure 18.22 shows several of the positions used by the face recognition software. Each position is checked when the software tries to compare two facial images. A face is identified using data such as

- distance between the eyes
- width of the nose
- shape of the cheek bones
- length of the jaw line
- shape of the eyebrows.

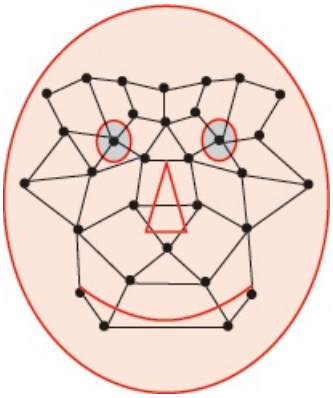


Figure 18.22 Face recognition

Figure 18.23 shows an artificial neural network (with two hidden layers).

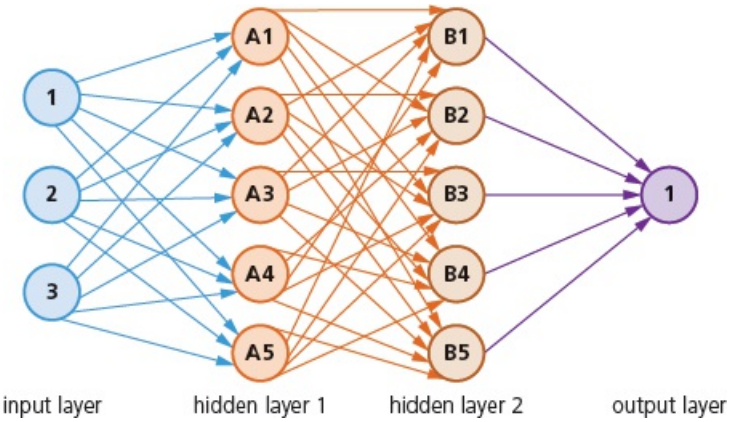


Figure 18.23 An artificial neural network

These systems are able to recognise objects, such as birds, by their shape and colour. With machine learning, the objects form labelled data which can be used in the training process.

But how is it possible to recognise a bird if the data is unlabelled? By analysing pixel densities of objects, for example, it is possible for a deep learning system to take unlabelled objects and then recognise them through a sophisticated set of algorithms.

Deep learning using artificial neural networks can be used to recognise objects by looking at the binary codes of each pixel, thus building up a picture of the object. For example, Figure 18.24 shows a close up of a face where each pixel can be assigned its binary value and, therefore, the image could be recognised by deep learning algorithms as a person's face.

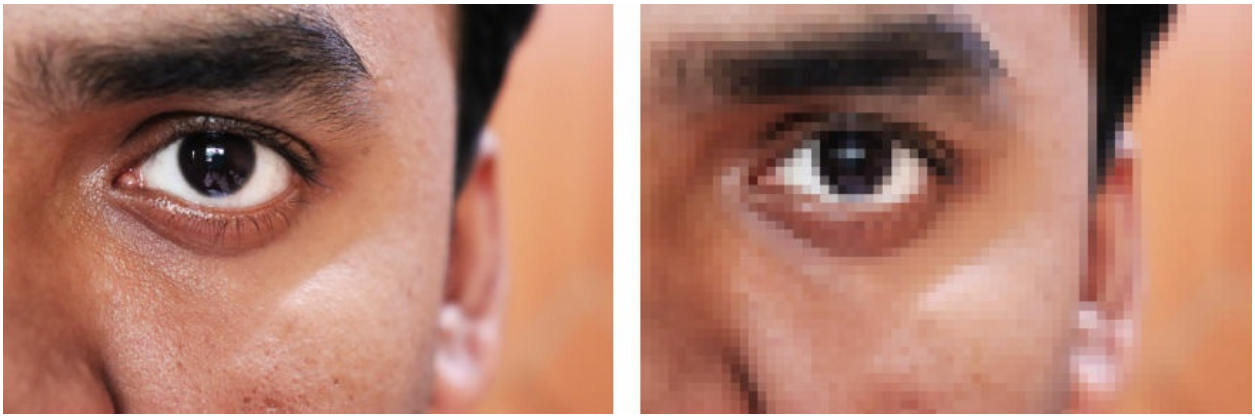


Figure 18.24 Deep learning algorithms can recognise this as a person's face

This summarises how deep learning works:

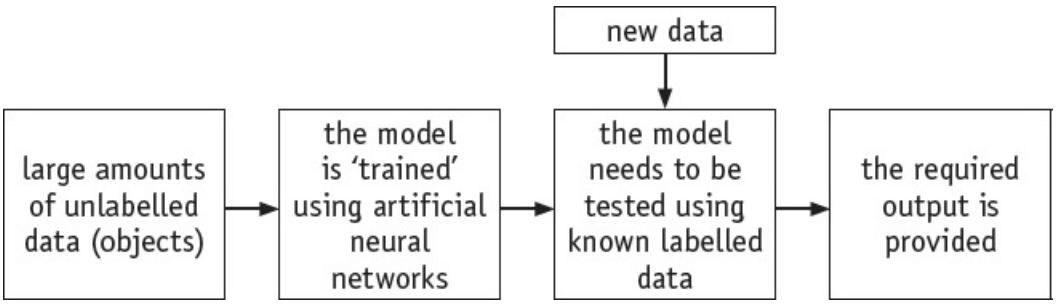


Figure 18.25

Large amounts of unlabelled data (data which is undefined and needs to be recognised) is input into the model. One of the methods of object recognition, using pixel densities, was described above. Using artificial neural networks, each of the objects is identified by the system. Labelled data (data which has already been defined and is, therefore, recognised) is then entered into the model to make sure it gives the correct responses. If the output is not sufficiently accurate, the model is refined until it gives satisfactory results (known as **back propagation** – see Section 18.2.6). The refinement process may take several adjustments until it provides reliable and consistent outputs.

Text mining

Suppose a warehouse contains hundreds of books. A system is being developed to translate the text from each book and determine which genre the book belongs to, such as a car engine repair manual. Each book could then be identified by the system and placed in its correct category. How could this be done using deep learning and machine learning techniques?

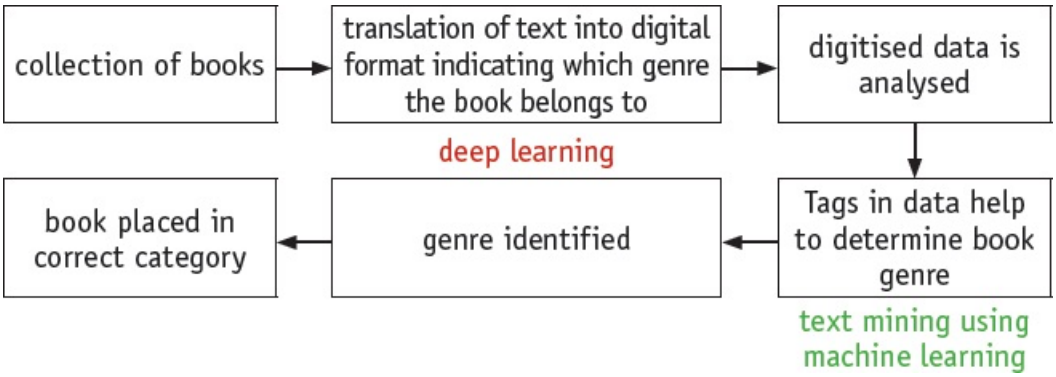


Figure 18.26

Computer-assisted translation (CAT)

Existing online language translators have a limited use: they often have difficulty translating words or phrases with double meanings, or idioms specific to a language. Computer-assisted translation (CAT) goes some way to overcome these issues. CAT uses deep learning algorithms to help in the translation process.

In particular, CAT uses two tools:

- Terminology databases (linguistic databases that grow and 'learn' from translations being carried out).
- Translation memories (these automatically insert known translations for certain words, phrases or sentences).

Photograph enhancement

Some of the latest smartphones cameras use deep learning to give DSLR quality to the photographs. The technology was developed by taking the same photos using a smartphone and then using a DSLR camera. The deep learning system was then trained by comparing the two photographs. A large number of photographs already taken by a DSLR camera (but not by the

smartphone) were then used to test the model. The results can be seen in Figure 18.27.

Figure 18.27 Original photo taken by smartphone (left); enhanced photo using deep learning model (right)

Turning monochrome photos into colour

Deep learning can be used to turn monochrome (black and white) photographs into coloured photographs. This is a sophisticated system which produces a better photograph than simply turning grey-scale values into an approximated colour. In Figure 18.28, the original monochrome image has been processed to give a very accurate coloured image.

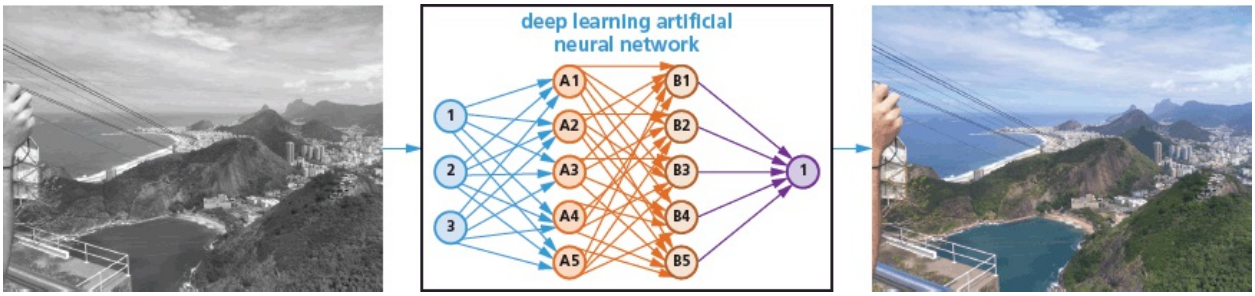


Figure 18.28 Deep learning can change black and white photographs to colour

The deep learning system is trained by searching websites for data which allows it to recognise features and then map a particular colour to a photograph/object thus producing an accurate coloured image.

Chatbots

Chatbots interact through instant messaging, artificially replicating patterns of human interactions using machine learning. Typed messages or voice recordings make use of predefined scripts and machine learning: when a question is asked, a chatbot responds using the information known at the time.

Hello! I am the booksite.com chatbot. How can I help you?
I ordered the new A Level Computer Science textbook – when will it be delivered?
Can you give me your order number please and I will check for you
CSH123456. Thank you!

Figure 18.29

18.2.4 Comparison between machine learning and deep learning

machine learning	deep learning
enables machines to make decisions on their own based on past data	enables machines to make decisions using an artificial neural network
needs only a small amount of data to carry out the training	the system needs large amounts of data during the training stages
most of the features in the data used need to be identified in advance and then manually coded into the system	deep learning machine learns the features of the data from the data itself and it does not need to be identified in advance
a modular approach is taken to solve a given problem/task; each module is then combined to produce the final model	the problem is solved from beginning to end as a single entity
testing of the system takes a long time to carry out	testing of the system takes much less time to carry out
there are clear rules which explain why each stage in the model was made	since the system makes decisions based on its own logic, the reasoning behind those decisions may be very difficult to understand (they are often referred to as a **black box**)

Table 18.1 Comparison between machine learning and deep learning

18.2.5 What will happen in the future?

Table 18.2 lists some artificial intelligence, machine learning and deep learning developments expected in the not too distant future.

AI	detection of crimes before they happen by looking at existing patterns
	development of humanoid AI machines which carry out human tasks (androids)
Machine learning	increased efficiency in health care and diagnostics (for example, early detection of cancers, eye defects, and so on)
	better marketing techniques based on buying behaviours of target groups
Deep learning	increased personalisation in various areas (such as individual cancer care which personalises treatment based on many factors)
	hyper intelligent personal assistants

Table 18.2 Developments in AI, machine learning and deep learning

EXTENSION ACTIVITY 18B

Carry out some research into present day and future developments in AI, machine learning and deep learning (these will change every year and it is necessary to update yourself with all the latest developments).

18.2.6 Back propagation and regression methods

Back propagation

When designing neural networks, it is necessary to give random weightings to each of the neural connections. However, the system designer will not initially know which weight factors to use to produce the best results. It is necessary to train the neural networks during the development stage:

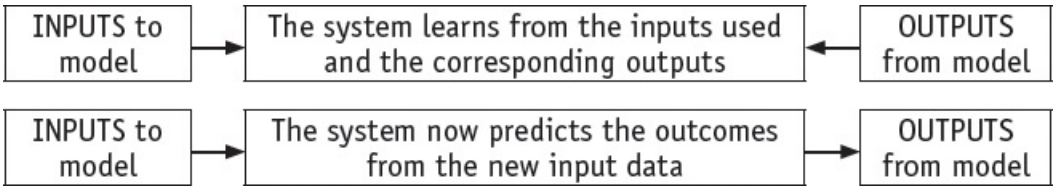


Figure 18.30

The training program is iterative; the outputs produced from the system are compared to the expected results and any differences in the two values/results are calculated. These errors are propagated back into the neural network in order to update the initial network weightings.

This process (training) is repeated until the desired outputs are eventually obtained, or the errors in the outputs are within acceptable limits.

Here is a summary of the back propagation process:

- The initial outputs from the system are compared to the expected outputs and the system weightings are adjusted to minimise the difference between actual and expected results.
- Calculus is used to find the error gradient in the obtained outputs: the results are fed back into the neural networks and the weightings on each neuron are adjusted (note: this can be used in both supervised and unsupervised networks).
- Once the errors in the output have been eliminated (or reduced to acceptable limits) the neural network is functioning correctly and the model has been successfully set up.
- If the errors are still too large, the weightings are altered – the process continues until satisfactory outputs are produced.

Figure 18.31 shows the ultimate goal of the back propagation process.

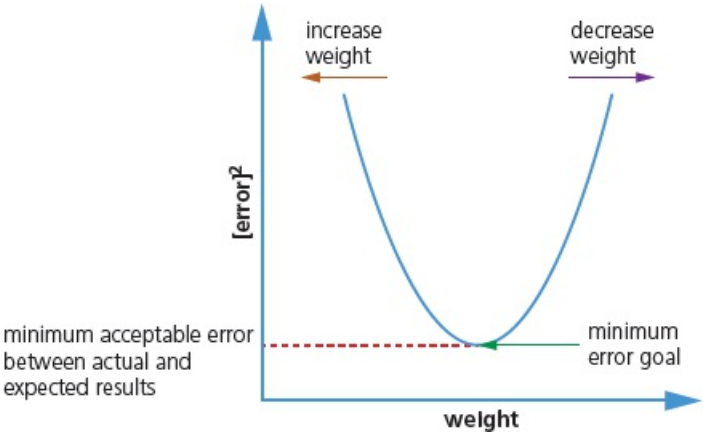


Figure 18.31

There are two types of back propagation: static and recurrent:

- Static maps static inputs to a static output.
- Mapping is instantaneous in static, but this is not the case with recurrent.
- Training a network/model is more difficult with recurrent than with static.
- With recurrent, activation is fed forward until a fixed value is achieved.

Regression

Machine learning builds heavily on statistics; for example, **regression** is one way of analysing data before it is input into a system or model. Regression is used to make predictions from given data by learning some relationship between the input and the output. It helps in the understanding of how the value of a dependent variable changes when the values of independent variables are also changed. This makes it a valuable tool in prediction applications, such as weather forecasting.

In machine learning, this is used to predict the outcome of an event based on any relationship between variables obtained from input data and the hidden parameters.

ACTIVITY 18C

1 a) Explain the difference(s) between *narrow AI*, *general AI* and *strong AI*.

b) In machine learning, what is meant by *reward and punishment*? Give an example of its use.

c) Explain the term *artificial neural networks*. Use diagrams to help in your explanation.

2 a) Explain the difference between *supervised learning*, *unsupervised learning*, *reinforcement learning* and *active learning*.

b) i) Describe how back propagation (of errors) is used to train an AI model.

ii) Name **two** types of back propagation.

3 a) Give **one** use of each of the following.

i) supervised learning

ii) unsupervised learning

iii) reinforcement learning

iv) semi-supervised (active) learning

b) Name the ten terms, i)–x), being described.

i) Intelligent machines that think and behave like human beings.

ii) System that learns without being programmed to learn.

iii) Machines that process information in a similar way to the human brain; they handle large amounts of data using artificial neural networks.

iv) Data where objects are undefined and need to be manually recognised.

v) An internet bot that systematically browses the world wide web to update its web content.

vi) A computer program which is set up to automatically simulate a conversational

interaction between a human and a website.

vii) A statistical measure used in artificial neural networks to calculate error gradients so that actual neuron weightings can be adjusted to improve the performance of the model.

viii) A statistical measure used to make predictions from data by finding learning relationships between input and output values.

ix) Data where we know the target answer and data objects are fully recognised and identified.

x) Improvements made to a model based on negative and positive feedback: actions are optimised to increase the amount of positive feedback.

End of chapter questions

1 a) Answer these multiple choice questions.

i) Identify the statement that best describes *artificial intelligence*.

[1]

A putting human intelligence into a computer system
B programming a computer using a user's intelligence
C making a machine intelligent
D playing a game on a computer
E adding more memory units to a computer

ii) Identify the programming language that is **not** used when programming AI systems.

[1]

A Java
B JavaScript
C Lisp
D Perl
E Prolog

iii) Identify the correct description of a heuristic approach.

[1]

A trying to improve algorithms using back propagation
B searching and measuring how far away a node is from its destination
C comparison of two nodes in a graph to see which is nearer to the destination node
D an informed 'guess' about which node will lead to the required goal
E all the above

b) Copy the diagram below and connect each description to either machine learning or deep learning.

[8]

Learning type	Description
	needs only a small amount of training data
	problems are solved in an end to end manner
Deep learning	enables machines to make decisions with the help of artificial neural networks
	has clear rules to explain how decisions were made
Machine learning	makes use of modular approach at design and training stages
	needs vast amounts of data during training and development
	enables machines to make decisions on their own based on past data
	makes decisions based on its own logic so the reasoning may be difficult to interpret

2 a) Describe **three** features you would associate with:

i) reinforcement learning

[3]

ii) supervised learning.

[3]

b) Explain why these applications are regarded as artificial intelligence.

i) Chat bots

[2]

ii) Search engines

[2]

iii) Photographic enhancement applications

[2]

3 Copy and complete the text, using words from the box. Words may be used once, more than once, or not at all.

[10]

actual output	machine learning	reinforcement learning
back propagation	minimised	removed
deep learning	random weighting	static
error gradients	recurrent	supervised learning
expected results	regression	testing

When designing artificial neural networks, each neuron is given a .....

The ..... is compared to the ..... as part of the training.

..... are used to update the neural weightings.

Weightings are updated until the errors are ..... or are .....

This process is known as .....

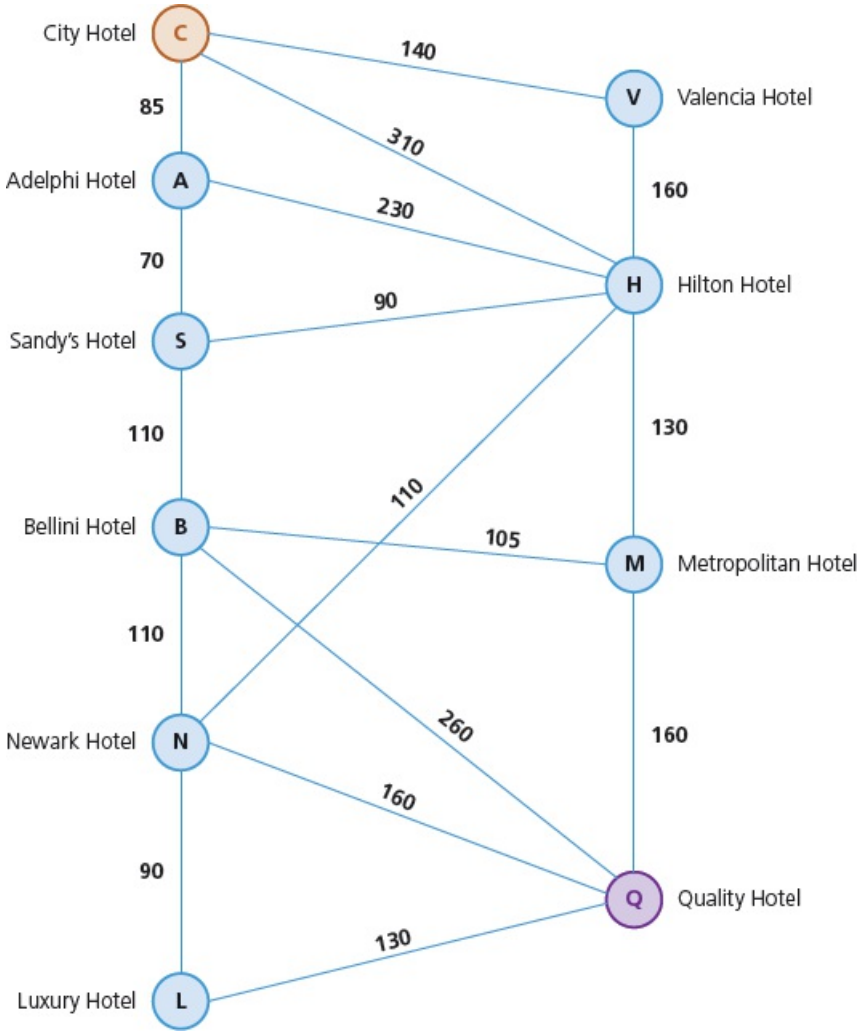
There are two types: ..... and .....

Machine learning uses ..... to make predictions from data by looking at learning relationships.

4 a) Explain the difference between the A* algorithm and Dijkstra's algorithm.

[2]

b) The following graph (network) shows how long it takes (in seconds) to walk between ten hotels in a city.



i) Using Dijkstra's algorithm, show the shortest time to walk from the City Hotel (C) to the Quality Hotel (Q).

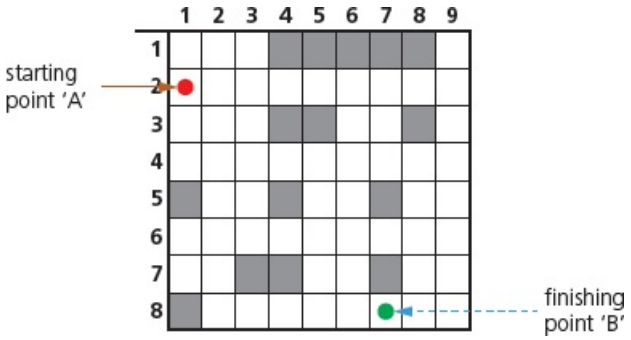
[8]

ii) Give the route corresponding to your answer in part b)i).

[1]

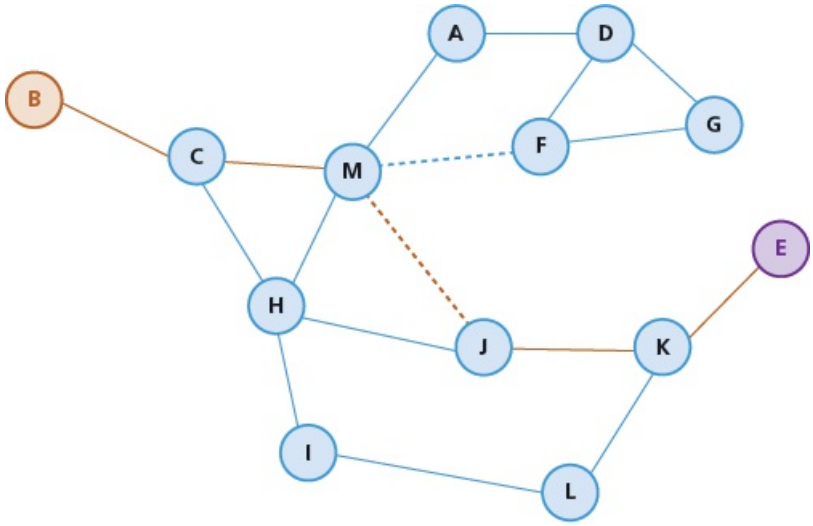
5 The following graph is made up of a (9 × 8) matrix.
Use the A* algorithm to show the shortest route from A to B.

[8]

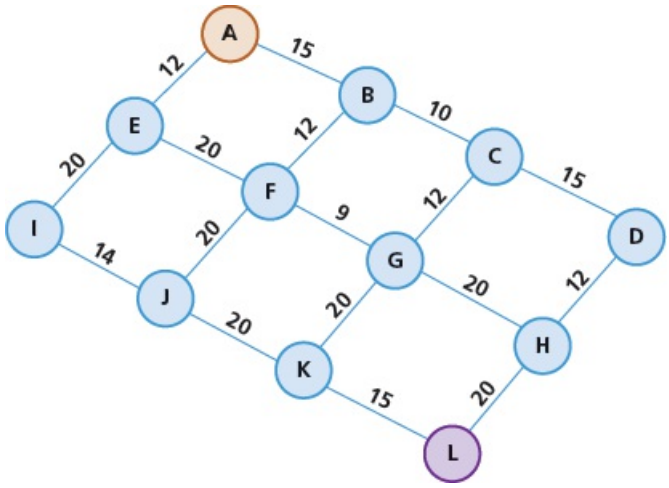


6 Tom is using a GPS device to navigate from point B to point E.
Tom's GPS uses the A* algorithm to find the shortest route:
B → C → M → J → K → E
This route is shown in orange on the diagram.
However, due to some major flooding, routes M to J and M to F have been closed, making the original path no longer possible.
Describe how the GPS system will use the A* algorithm to find an alternative route from B to E.

[4]



7 The following graph shows the routes connecting buildings on a university campus. The numbers represent the time taken (in minutes) to cycle from one building to another.



a) i) Use Dijkstra's algorithm to find the minimum time to cycle from building A to building L.

[8]

ii) Write down the corresponding shortest route.

[1]

b) It has been decided to construct a new cycle path, either from A directly to D (cycle time

30 minutes) or from A directly to I (cycle time 20 minutes).
Identify the option that would reduce the cycle time from building A to building L by the greatest amount.

[4]

19 Computational thinking and problem solving

In this chapter, you will learn about

- how to write algorithms to implement linear and binary searches
- the conditions necessary for the use of a binary search
- how the performance of a binary search varies according to the number of data items
- how to write algorithms to implement insertion and bubble sorts
- the use of abstract data types (ADTs) including finding, inserting and deleting items from linked lists, binary trees, stacks and queues
- the key features of a graph and why graphs are used
- how to implement ADTs from other ADTs
- the comparison of algorithms including the use of Big O notation
- how recursion is expressed in a programming language
- when to use recursion
- writing recursive algorithms
- how a compiler implements recursion in a programming language.

19.1 Algorithms

WHAT YOU SHOULD ALREADY KNOW

In Chapter 10 you learnt about Abstract Data Types (ADTs) and searching and sorting arrays. You should also have been writing programs in your chosen programming language (Python, VB or Java). Try the following five questions to refresh your memory before you start to read this chapter.

1 Explain what is meant by the terms

a) stack

b) queue

c) linked list.

2 **a)** Describe how it is possible to implement a stack using an array.

```
// Java program for Hello World
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

```
# Python program for Hello World
print ("Hello World")
```

b) Describe how it is possible to implement a queue using an array.

c) Describe how it is possible to implement a linked list using an array.

3 Write pseudocode to search an array of twenty numbers using a linear search.

4 Write pseudocode to sort an array that contains twenty numbers using a bubble sort.

5 Make sure that you can write, compile and run a short program in your chosen programming language. The programs below show the code you need to write the same program in each of three languages.

```
'VB program for Hello World
Module Module1
   Sub Main()
      Console.WriteLine("Hello World")
      Console.ReadKey()
   End Sub
End Module
```

Key terms

Binary search – a method of searching an ordered list by testing the value of the middle item in the list and rejecting the half of the list that does not contain the required value.

Insertion sort – a method of sorting data in an array into alphabetical or numerical order by placing each item in turn in the correct position in the sorted list.

Binary tree – a hierarchical data structure in which each parent node can have a maximum of two child nodes.

Graph – a non-linear data structure consisting of nodes and edges.

Dictionary – an abstract data type that consists of pairs, a key and a value, in which the key is used to find the value.

Big O notation – a mathematical notation used to describe the performance or complexity of an algorithm.

19.1.1 Understanding linear and binary searching methods

Linear search

In Chapter 10, we looked at the linear search method of searching a list. In this method, each element of an array is checked in order, from the lower bound to the upper bound, until the item is found, or the upper bound is reached.

The pseudocode linear search algorithm and identifier table to find if an item is in the populated 1D array myList from Chapter 10 is repeated here.

```
DECLARE myList : ARRAY[0:9] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE item : INTEGER
DECLARE found : BOOLEAN
upperBound ← 9
lowerBound ← 0
OUTPUT "Please enter item to be found"
INPUT item
found ← FALSE
index ← lowerBound
REPEAT
    IF item = myList[index]
      THEN
        found ← TRUE
    ENDIF
    index ← index + 1
UNTIL (found = TRUE) OR (index > upperBound)
IF found
  THEN
    OUTPUT "Item found"
  ELSE
    OUTPUT "Item not found"
ENDIF
```

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
item	Item to be found

found	Flag to show when item has been found

Table 19.1

This method works for a list in which the items can be stored in any order, but as the size of the list increases, the average time taken to retrieve an item increases correspondingly.

The Cambridge International AS & A Level Computer Science syllabus requires you to be able to write code in one of the following programming languages: Python, VB and Java. It is very important to practice writing different routines in the programming language of your choice; the more routines you write, the easier it is to write programming code that works.

Here is a simple linear search program written in Python, VB and Java using a FOR loop.

Python

```
#Python program for Linear Search
#create array to store all the numbers
myList = [4, 2, 8, 17, 9, 3, 7, 12, 34, 21]
#enter item to search for
item = int(input("Please enter item to be found "))
found = False
for index in range(len(myList)):
  if(myList[index] == item):
      found = True
if(found):
  print("Item found")
else:
  print("Item not found")
```

VB

```
'VB program for Linear Search
 Module Module1
    Public Sub Main()
        Dim index As Integer
        Dim item As Integer
        Dim found As Boolean
        'Create array to store all the numbers
        Dim myList() As Integer = New Integer() {4, 2, 8, 17, 9, 3, 7, 12, 34, 21}
        'enter item to search for
        Console.Write("Please enter item to be found ")
        item = Integer.Parse(Console.ReadLine())
        For index = 0 To myList.Length - 1
            If (item = myList(index)) Then
                found = True
            End If
        Next
        If (found) Then
            Console.WriteLine("Item found")
        Else : Console.WriteLine("Item not found")
        End If
        Console.ReadKey()
    End Sub
End Module
```

Java

```
//Java program Linear Search
import java.util.Scanner;
public class LinearSearch
{
   public static void main(String args[])
   {
       Scanner myObj = new Scanner(System.in);
   //Create array to store the all the numbers
       int myList[] = new int[] {4, 2, 8, 17, 9, 3, 7, 12, 34, 21};
       int item, index;
       boolean found = false;
       // enter item to search for
   System.out.println("Please enter item to be found ");
   item = myObj.nextInt();
   for (index = 0; index < myList.length - 1; index++)
   {
     if (myList[index] == item)
     {
     found = true;
     }
   }
   if (found)
        {
   System.out.println("Item found");
   }
        else
          {
              System.out.println("Item not found");
            }
        }
}
```

ACTIVITY 19A

Write the linear search in the programming language you are using, then change the code to use a similar type of loop that you used in the pseudocode at the beginning of Section 19.1.1, *Linear search.*

Binary search

A **binary search** is more efficient if a list is already sorted. The value of the middle item in the list is first tested to see if it matches the required item, and the half of the list that does *not* contain the required item is discarded. Then, the next item of the list to be tested is the middle item of the half of the list that was kept. This is repeated until the required item is found or there is nothing left to test.

For example, consider a list of the letters of the alphabet.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

To find the letter W using a linear search there would be 23 comparisons.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23			

Figure 19.1 Linear search showing all the comparisons

To find the letter W using a binary search there could be just three comparisons.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
												-							-			-			
												W							W			W			
												1							2			3			

Figure 19.2 Binary search showing all the comparisons

ACTIVITY 19B

Check how many comparisons for each type of search it takes to find the letter D. Find any letters where the linear search would take less comparisons than the binary search.

A binary search usually takes far fewer comparisons than a linear search to find an item in a list. For example, if a list had 1024 elements, the maximum number of comparisons for a binary search would be 16, whereas a linear search could take up to 1024 comparisons.

Here is the pseudocode for the binary search algorithm to find if an item is in the populated 1D array myList. The identifier table is the same as the one used for the linear search.

```
DECLARE myList : ARRAY[0:9] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE item : INTEGER
DECLARE found : BOOLEAN
upperBound ← 9
lowerBound ← 0
OUTPUT "Please enter item to be found"
INPUT item
found ← FALSE
REPEAT
    index ← INT ( (upperBound + lowerBound) / 2 )
    IF item = myList[index]
      THEN
        found ← TRUE
    ENDIF
    IF item > myList[index]
      THEN
        lowerBound ← index + 1
    ENDIF
    IF item < myList[index]
      THEN
        upperBound ← index - 1
    ENDIF
UNTIL (found = TRUE) OR (lowerBound = upperBound)
IF found
  THEN
    OUTPUT "Item found"
  ELSE
    OUTPUT "Item not found"
ENDIF
```

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array

index	Pointer to current array element
item	Item to be found
found	Flag to show when item has been found

Table 19.2

The code structure for a binary search is very similar to the linear search program shown for each of the programming languages. You will need to populate myList before searching for an item, as well as the variables found, lowerBound and upperBound.

You will need to use a conditional loop like those shown in the table below.

Loop	Language
`while (not found) and (lowerBound != lowerBound):`	Python uses a condition to repeat the loop at the start of the loop

`Do`
`:`
`:`
`Loop Until (found) Or (lowerBound = upperBound)`
	VB uses a condition to stop the loop at the end of the loop

`Do {`
`:`
`:`
`}`
`while ((!found) && (upperBound != lowerBound));`
	Java uses a condition to repeat the loop at the end of the loop

Table 19.3

You will need to use If statements like those shown in the table below to test if the item is found, or to decide which part of myList to use next, and to update the upperBound or lowerBound accordingly.

If	Language

`index = (upperBound + lowerBound)//2)`
`if(myList[index] == item):`
`found = True`
`if item > myList[index]:`
`lowerBound = index + 1`
`if item < myList[index]:`
`upperBound = index - 1`
	Python using integer division

	VB using integer division

index = (upperBound + lowerBound)\2
If (item = myList(index)) Then
found = True
End If
If item > myList(index) Then
lowerBound = index + 1
End if
If item < myList(index) Then
upperBound = index -1
End if	

index = (upperBound + lowerBound) / 2;
if (myList[index] == item)
{
found = true;
}
if (item > myList[index])
{
lowerBound = index + 1;
}
if (item < myList[index])
{
upperBound = index - 1;
}
	Java automatic integer division

Table 19.4

ACTIVITY 19C

In your chosen programming language, write a short program to complete the binary search.

Use this sample data:

16, 19, 21, 27, 36, 42, 55, 67, 76, 89

Search for the values 19 and 77 to test your program.

19.1.2 Understanding insertion and bubble sorting methods

Bubble sort

In Chapter 10, we looked at the bubble sort method of sorting a list. This is a method of sorting data in an array into alphabetical or numerical order by comparing adjacent items and swapping them if they are in the wrong order.

The bubble sort algorithm and identifier table to sort the populated 1D array myList from Chapter 10 is repeated here.

```
DECLARE myList : ARRAY[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE swap : BOOLEAN
DECLARE temp : INTEGER
DECLARE top : INTEGER
upperBound ← 8
lowerBound ← 0
top ← upperBound
REPEAT
   FOR index = lowerBound TO top - 1
         Swap ← FALSE
         IF myList[index] > myList[index + 1]
           THEN
           temp ← myList[index]
           myList[index] ← myList[index + 1]
           myList[index + 1] ← temp
           swap ← TRUE
         ENDIF
    NEXT
    top ← top -1
UNTIL (NOT swap) OR (top = 0)
```

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
swap	Flag to show when swaps have been made
top	Index of last element to compare
temp	Temporary storage location during swap

Table 19.5

Here is a simple bubble sort program written in Python, VB and Java, using a pre-condition loop and a FOR loop in Python and post-condition loops and FOR loops in VB and Java.

Python

Pre-condition loop

```
#Python program for Bubble Sort
myList = [70,46,43,27,57,41,45,21,14]
top = len(myList)
swap = True
while (swap) or (top > 0):
        swap = False
        for index in range(top - 1):
                if myList[index] > myList[index + 1]:
                        temp = myList[index]
                        myList[index] = myList[index + 1]
                        myList[index + 1] = temp
                        swap = True
        top = top - 1
#output the sorted array
print(myList)
```

VB

```
'VB program for bubble sort
 Module Module1
    Sub Main()
        Dim myList() As Integer = New Integer() {70, 46, 43, 27, 57, 41, 45, 21, 14}
        Dim index, top, temp As Integer
        Dim swap As Boolean
        top = myList.Length - 1
        Do
            swap = False
            For index = 0 To top - 1 Step 1
                If myList(index) > myList(index + 1) Then
                    temp = myList(index)
                    myList(index) = myList(index + 1)
                    myList(index + 1) = temp
                    swap = True
                End If
            Next
            top = top - 1
        Loop Until (Not swap) Or (top = 0)
        'output the sorted array
        For index = 0 To myList.Length - 1
            Console.Write(myList(index) & " ")
        Next
        Console.ReadKey() 'wait for keypress
    End Sub
End Module
```

Post-condition loop

Java

```
// Java program for Bubble Sort
class BubbleSort
{
 public static void main(String args[])
  {
      int myList[] = {70, 46, 43, 27, 57, 41, 45, 21, 14};
      int index, top, temp;
      boolean swap;
        top = myList.length;
        do {
              swap = false;
              for (index = 0; index < top - 1; index++)
              {
                if (myList[index] > myList[index + 1])
                {
                   temp = myList[index];
                   myList[index] = myList[index + 1];
                   myList[index + 1] = temp;
                     swap = true;
                     }
              }
              top = top - 1;
      }
      while ((swap) || (top > 0));
      // output the sorted array
        for (index = 0; index < myList.length; index++)
              System.out.print(myList[index] + " ");
        System.out.println();
  }
}
```

Post-condition loop

Insertion sort

The bubble sort works well for short lists and partially sorted lists. An insertion sort will also work well for these types of list. An **insertion sort** sorts data in a list into alphabetical or numerical order by placing each item in turn in the correct position in a sorted list. An insertion sort works well for incremental sorting, where elements are added to a list one at a time over an extended period while keeping the list sorted.

Here is the pseudocode and the identifier table for the insertion sort algorithm sorting the populated 1D array myList.

```
DECLARE myList : ARRAY[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE key : BOOLEAN
DECLARE place : INTEGER
upperBound ← 8
lowerBound ← 0
FOR index ← lowerBound + 1 TO upperBound
    key ← myList[index]
    place ← index - 1
    IF myList[place] > key
      THEN
        WHILE place >= lowerBound AND myList[place] > key
            temp ← myList[place + 1]
            myList[place + 1] ← myList[place]
            myList[place] ← temp
            place  ← place - 1
        ENDWHILE
        myList[place + 1] ← key
    ENDIF
NEXT index
```

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
key	Element being placed
place	Position in array of element being moved

Table 19.6

Figure 19.3 shows the changes to the 1D array myList as the insertion sort is completed.

myList	Index of element being checked													
	1		2	3	4		5	6		7		8		
[0]	27	19	19	19	19	16	16	16	16	16	16	16	16	
[1]	19	27	27	27	27	19	19	19	19	19	16	16	16	
[2]	36	36	36	36	36	27	27	27	21	21	19	19	19	
[3]	42	42	42	42	42	36	36	36	27	27	21	21	21	
[4]	16	16	16	16	16	42	42	42	36	36	27	27	27	
[5]	89	89	89	89	89	89	89	89	42	42	36	36	36	
[6]	21	21	21	21	21	21	21	21	89	89	42	42	42	
[7]	16	16	16	16	16	16	16	16	16	16	89	89	55	
[8]	55	55	55	55	55	55	55	55	55	55	55	55	89	

Figure 19.3

The element shaded blue is being checked and placed in the correct position. The elements shaded yellow are the other elements that also need to be moved if the element being checked is out of position. When sorting the same array, myList, the insert sort made 21 swaps and the bubble sort shown in Chapter 10 made 38 swaps. The insertion sort performs better on partially sorted lists because, when each element is found to be in the wrong order in the list, it is moved to approximately the right place in the list. The bubble sort will only swap the element in the wrong order with its neighbour.

As the number of elements in a list increases, the time taken to sort the list increases. It has been shown that, as the number of elements increases, the performance of the bubble sort deteriorates faster than the insertion sort.

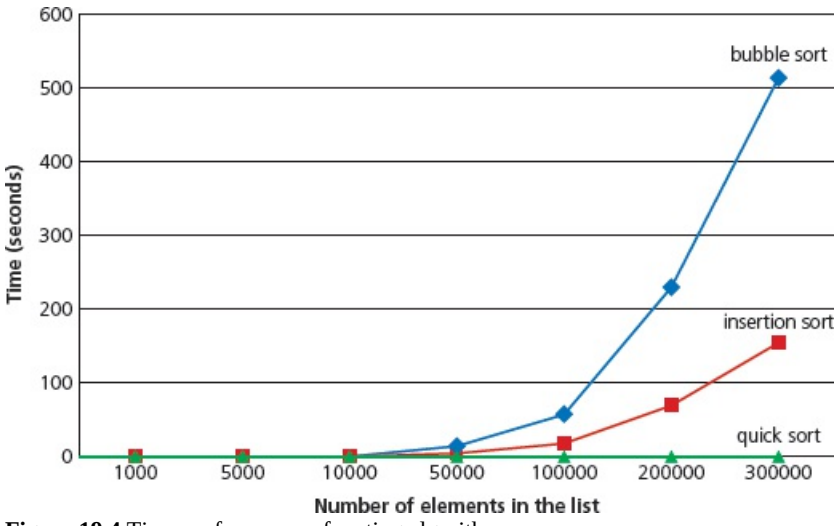


Figure 19.4 Time performance of sorting algorithms

The code structure for an insertion sort in each of the programming languages is very similar to the bubble sort program. You will need to assign values to lowerBound and upperBound and use a nested loop like those shown in the table below.

Nested loop	Language

for index in range(lowerBound + 1, upperBound):
    key = myList[index]
    place = index -1
    if myList[place] > key:
        while place >= lowerBound and myList[place] > key:
            temp = myList[place + 1]
            myList[place + 1] = myList [place]
            myList[place] = temp
            place = place -1
        myList[place + 1] = key
	Python

For index = lowerBound + 1 To upperBound
    myKey = myList(index)
    place = index - 1
    If myList(place) > myKey Then
        While (place >= lowerBound) And (myList(place) > myKey)
            temp = myList(place + 1)
            myList(place + 1) = myList(place)
            myList(place) = temp
            place = place - 1
        End While
        myList(place + 1) = myKey
    End If
Next
	VB cannot use key as a variable

for (index = lowerBound + 1; index < upperBound; index++)
    {
        key = myList[index];
        place = index - 1;
        if (myList[place] > key)
        {
        do
        {
        temp = myList[place + 1];
        myList[place + 1] = myList[place];
        myList[place] = temp;
        place = place - 1
        }
        while ((place >= lowerBound) || (myList[place + 1] > key));
        myList[place + 1] = key;
        }
    }
	Java

Table 19.7

ACTIVITY 19D

In your chosen programming language write a short program to complete the insertion sort.

EXTENSION ACTIVITY 19A

There are many other more efficient sorting algorithms. In small groups, investigate different sorting algorithms, finding out how the method works and the efficiency of that method. Share your results.

19.1.3 Understanding and using abstract data types (ADTs)

Abstract data types (ADTs) were introduced in Chapter 10. Remember that an ADT is a collection of data and a set of operations on that data. There are several operations that are essential when using an ADT

- finding an item already stored
- adding a new item
- deleting an item.

We started considering the ADTs stacks, queues and linked lists in Chapter 10. If you have not already done so, read Section 10.4 to ensure that you are ready to work with these data structures. Ensure that you can write algorithms to set up then add and remove items from stacks and queues.

Stacks

In Chapter 10, we looked at the data and the operations for a stack using pseudocode. You will need to be able to write a program to implement a stack. The data structures and operations required to implement a similar stack using a fixed length integer array and separate sub routines for the push and pop operations are set out below in each of the three prescribed programming languages. If you are unsure how the operations work, look back at Chapter 10.

Stack data structure	Languag

`stack = [None for index in range(0,10)]`
`basePointer = 0`
`topPointer = -1`
`stackFull = 10`
`item = None`
	Python
	empty stack wit no elements

`Public Dim stack() As Integer = {Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}`
`Public Dim basePointer As Integer = 0`
`Public Dim topPointer As Integer = -1`
`Public Const stackFull As Integer = 10`
`Public Dim item As Integer`
	VB
	empty stack wit no elements and variables set to public fo subroutin access

	Java

`public static int stack[] = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};`
`public static int basePointer = 0;`
`public static int topPointer = -1;`
`public static final int stackFull = 10;`
`public static int item;`
	empty stack wit no elements and variables set to public fo subroutin access

Table 19.8

Stack pop operation	Language

`def pop():`
`  global topPointer, basePointer, item`
`  if topPointer == basePointer -1:`
`  print("Stack is empty,cannot pop")`
`  else:`
`  item = stack[topPointer]`
`  topPointer = topPointer -1`
	Python
	global used within subroutine to access variables
	topPointer points to the top of the stack

`Sub pop()`
`  If topPointer = basePointer - 1 Then`
`  Console.WriteLine("Stack is empty, cannot pop")`
`  Else`
`  item = stack(topPointer)`
`  topPointer = topPointer - 1`
`  End If`
`End Sub`
	VB
	topPointer points to the top of the stack

	Java topPointer points to the top of the stack

```
static void pop()
{
      if (topPointer == basePointer - 1)
                  System.out.println("Stack is empty,cannot pop");
      else
      {
            item = stack[topPointer - 1];
              topPointer = topPointer - 1;
      }
}
```

Table 19.9

Stack push operation	Language

`def push(item):`
`  global topPointer`
`  if topPointer < stackFull - 1:`
`  topPointer = topPointer + 1`
`  stack[topPointer] = item`
`  else:`
`  print("Stack is full, cannot push")`
	Python

`Sub push(ByVal item)`
`  If topPointer < stackFull - 1 Then`
`  topPointer = topPointer + 1`
`  stack(topPointer) = item`
`  Else`
`  Console.WriteLine("Stack is full, cannot push")`
`  End if`
`End Sub`
	VB

`static void push(int item)`
`{`
`  if (topPointer < stackFull - 1)`
`  {`
`  topPointer = topPointer + 1;`
`  stack[topPointer] = item;`
`  }`
`  else`
`  System.out.println("Stack is full, cannot push");`
`}`
	Java

Table 19.10

ACTIVITY 19E

In your chosen programming language, write a program using subroutines to implement a stack with 10 elements. Test your program by pushing two integers 7 and 32 onto the stack, popping these integers off the stack, then trying to remove a third integer, and by pushing the integers 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 onto the stack, then trying to push 11 on to the stack.

Queues

In Chapter 10, we looked at the data and the operations for a circular queue using pseudocode. You will need to be able to write a program to implement a queue. The data structures and operations required to implement a similar queue using a fixed length integer array and separate sub routines for the enqueue and dequeue operations are set out below in each of the three programing languages. If you are unsure how the operations work, look back at Chapter 10.

Queue data structure	Languag

`queue = [None for index in range(0,10)]`
`frontPointer = 0`
`rearPointer = -1`
`queueFull = 10`
`queueLength = 0`
	Python
	empty queue with no items

`Public Dim queue() As Integer = {Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}`
`Public Dim frontPointer As Integer = 0`
`Public Dim rearPointer As Integer = -1`
`Public Const queueFull As Integer = 10`
`Public Dim queueLength As Integer = 0`
`Public Dim item As Integer`
	VB
	empty queue with no items and variables, set to public for subroutin access

`public static int queue[] = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};`
`public static int frontPointer = 0;`
`public static int rearPointer = -1;`
`public static final int queueFull = 10;`
`public static int queueLength = 0;`
`public static int item;`
	Java
	empty queue with no elements and variables, set to public for subroutin access

Table 19.11

Queue enqueue (add item to queue) operation	Language

def enQueue(item):
  global queueLength, rearPointer
  if queueLength < queueFull:
  if rearPointer < len(queue) - 1:
  rearPointer = rearPointer + 1
  else:
  rearPointer = 0
  queueLength = queueLength + 1
  queue[rearPointer] = item
  else:
  print("Queue is full, cannot enqueue")
	Python
	global used within subroutine to access variables
	If the rearPointer is pointing to the last element of the array and the queue is not full, the item is stored in the first element of the array

Sub enQueue(ByVal item)
  If queueLength < queueFull Then
  If rearPointer < queue.length - 1 Then
  rearPointer = rearPointer + 1
  Else
  rearPointer = 0
  End If
  queueLength = queueLength + 1
  queue(rearPointer) = item
  Else
  Console.WriteLine("Queue is full, cannot enqueue")
  End If
End Sub
	VB
	If the rearPointer is pointing to the last element of the array and the queue is not full, the item is stored in the first element of the array

	Java If the rearPointer is pointing to the last element of the array and the queue is not full, the item is stored in the first

static void enQueue(int item)
{
  if (queueLength < queueFull)
  {
  if (rearPointer < queue.length - 1)
  rearPointer = rearPointer + 1;
  else
  rearPointer = 0;
  queueLength = queueLength + 1;
  queue[rearPointer] = item;
  }
  else
  System.out.println("Queue is full, cannot enqueue");
};
	element of the array

Table 19.12

Queue dequeue (remove item from queue) operation	Language

def deQueue():
  global queueLength, frontPointer, item
  if queueLength == 0:
  print("Queue is empty,cannot dequeue")
  else:
  item = queue[frontPointer]
  if frontPointer == len(queue) - 1:
  frontPointer = 0
  else:
  frontPointer = frontPointer + 1
  queueLength = queueLength -1
	Python
	If the frontPointer points to the last element in the array and the queue is not empty, the pointer is updated to point at the first item in the array rather than the next item in the array

	VB If the frontPointer points to the last element in the array and the queue is not

Sub deQueue()
  If queueLength = 0 Then
  Console.WriteLine("Queue is empty, cannot dequeue")
  Else
  item = queue(frontPointer)
  If frontPointer = queue.length - 1 Then
  frontPointer = 0
  Else
  frontPointer = frontPointer + 1
  End if
  queueLength = queueLength - 1
  End If
End Sub
	empty, the pointer is updated to point at the first item in the array rather than the next item in the array

static void deQueue()
{
  if (queueLength == 0)
  System.out.println("Queue is empty,cannot dequeue");
  else
  {
  item = queue[frontPointer];
  if (frontPointer == queue.length - 1)
  frontPointer = 0;
  else
  frontPointer = frontPointer + 1;
  queueLength = queueLength - 1;
  }
}
	Java
	If the frontPointer points to the last element in the array and the queue is not empty, the pointer is updated to point at the first item in the array rather than the next item in the array

Table 19.13

ACTIVITY 19F

In your chosen programming language, write a program using subroutines to implement a queue with 10 elements. Test your program by adding two integers 7 and 32 to the queue, removing these integers from the queue, then trying to remove a third integer, and by adding the integers 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 to the queue then trying to add 11 to the queue.

Linked lists

Finding an item in a linked list

In Chapter 10, we looked at defining a linked list as an ADT; now we need to consider writing

algorithms using a linked list. Here is the declaration algorithm and the identifier table from Chapter 10.

```
DECLARE mylinkedList ARRAY[0:11] OF INTEGER
DECLARE myLinkedListPointers ARRAY[0:11] OF INTEGER
DECLARE startPointer : INTEGER
DECLARE heapStartPointer : INTEGER
DECLARE index : INTEGER
heapStartPointer ← 0
startPointer ← -1 // list empty
FOR index ← 0 TO 11
    myLinkedListPointers[index] ← index + 1
NEXT index
// the linked list heap is a linked list of all the
spaces in the linked list, this is set up when the
linked list is initialised
myLinkedListPointers[11] ← -1
// the final heap pointer is set to -1 to show no
further links
```

The above code sets up a linked list ready for use. The identifier table is below.

Identifier	Description
myLinkedList	Linked list to be searched
myLinkedListPointers	Pointers for linked list
startPointer	Start of the linked list
heapStartPointer	Start of the heap
index	Pointer to current element in the linked list

Table 19.14

Figure 19.5 below shows an empty linked list and its corresponding pointers.

		myLinkedList	myLinkedListPointers
heapStartPointer →	[0]		1
	[1]		2
	[2]		3
	[3]		4
startPointer = -1	[4]		5
	[5]		6
	[6]		7
	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		–1

Figure 19.5

Figure 19.6 below shows a populated linked list and its corresponding pointers.

		myLinkedList	myLinkedListPointers
	[0]	27	–1
	[1]	19	0
	[2]	36	1
	[3]	42	2
startPointer →	[4]	16	3
heapStartPointer →	[5]		6
	[6]		7
	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		–1

Figure 19.6

The algorithm to find if an item is in the linked list myLinkedList and return the pointer to the item if found or a null pointer if not found, could be written as a function in pseudocode as shown below.

```
DECLARE itemSearch : INTEGER
DECLARE itemPointer : INTEGER
CONSTANT nullPointer = -1
FUNCTION find(itemSearch) RETURNS INTEGER
DECLARE found : BOOLEAN
itemPointer ← startPointer
found ← FALSE
    WHILE (itemPointer <> nullPointer) AND NOT found DO
        IF myLinkedList[itemPointer] = itemSearch
          THEN
            found ← TRUE
          ELSE
            itemPointer ← myLinkedListPointers[itemPointer]
        ENDIF
    ENDWHILE
RETURN itemPointer
// this function returns the item pointer of the value found or -1 if the
item is not found
```

The following programs use a function to search for an item in a populated linked list.

Python

```
#Python program for finding an item in a linked list
myLinkedList = [27, 19, 36, 42, 16, None, None, None, None, None, None, None]
myLinkedListPointers = [-1, 0, 1, 2, 3 ,6 ,7 ,8 ,9 ,10 ,11, -1]
startPointer = 4
nullPointer = -1

def find(itemSearch):
    found = False
    itemPointer = startPointer
    while itemPointer != nullPointer and not found:
        if myLinkedList[itemPointer] == itemSearch:
            found = True
        else:
            itemPointer = myLinkedListPointers[itemPointer]
    return itemPointer

#enter item to search for
item = int(input("Please enter item to be found "))
result = find(item)
if result != -1:
    print("Item found")
else:
    print("Item not found")
```

Populating the linked list

Defining the find function

Calling the find function

VB

```
'VB program for finding an item in a linked list
Module Module1
    Public Dim startPointer As Integer = 4
    Public Const nullPointer As Integer = -1
      Public Dim item As Integer
      Public Dim itemPointer As Integer
      Public Dim result As Integer
      Public Dim myLinkedList() As Integer = {27, 19, 36, 42, 16,
        Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}
      Public Dim myLinkedListPointers() As Integer = {-1, 0, 1, 2,
        3, 6, 7, 8, 9, 10, 11, -1}
         Public Sub Main()
         'enter item to search for
            Console.Write("Please enter item to be found ")
            item = Integer.Parse(Console.ReadLine())
            result = find(item)
            If result <> -1 Then
               Console.WriteLine("Item found")
            Else
               Console.WriteLine("Item not found")
            End If
            Console.ReadKey()
         End Sub

    Function find(ByVal itemSearch As Integer) As Integer
         Dim found As Boolean = False
         itemPointer = startPointer
         While (itemPointer <> nullPointer) And Not found
            If itemSearch = myLinkedList(itemPointer) Then
            found = True
          Else
            itemPointer = myLinkedListPointers(itemPointer)
          End If
         End While
         Return itemPointer
    End Function
End Module
```

Populating the linked list

Calling the find function

Defining the find function

Java

```
//Java program for finding an item in a linked list
import java.util.Scanner;
class LinkedListAll
{
  public static int myLinkedList[] = new int[] {27, 19, 36, 42, 16, 0,
    0, 0, 0, 0, 0, 0};
  public static int myLinkedListPointers[] = new int[] {-1, 0, 1, 2,
    3, 6, 7, 8, 9, 10, 11, -1};
  public static int startPointer = 4;
  public static final int nullPointer = -1;
  static int find(int itemSearch)
  {
  boolean found = false;
     int itemPointer = startPointer;
     do
     {
          if (itemSearch == myLinkedList[itemPointer])
          {
             found = true;
          }
          else
          {
              itemPointer = myLinkedListPointers[itemPointer];
          }
     }
     while ((itemPointer != nullPointer) && !found);
     return itemPointer;

  }
   public static void main(String args[])
  {
     Scanner input = new Scanner(System.in);
   System.out.println("Please enter item to be found ");
   int item = input.nextInt();
   int result = find(item);
   if (result != -1 )
     {
   System.out.println("Item found");
  }
    else
     {
          System.out.println("Item not found");
        }
     }
  }
```

Populating the linked list

Defining the find function

Calling the find function

The trace table below shows the algorithm being used to search for 42 in myLinkedList.

startPointer	itemPointer	searchItem
Already set to 4	4	42
	3	

Table 19.15 Trace table

ACTIVITY 19G

In the programming language of your choice, use the code given to write a program to set up the populated linked list and find an item stored in it.

Inserting items into a linked list

The algorithm to insert an item in the linked list myLinkedList could be written as a procedure in pseudocode as shown below.

```
DECLARE itemAdd : INTEGER
DECLARE startPointer : INTEGER
DECLARE heapstartPointer : INTEGER
DECLARE tempPointer : INTEGER
CONSTANT nullPointer = -1
PROCEDURE linkedListAdd(itemAdd)
  // check for list full
  IF heapStartPointer = nullPointer
    THEN
      OUTPUT "Linked list full"
    ELSE
      // get next place in list from the heap
      tempPointer ← startPointer // keep old start pointer
      startPointer ← heapStartPointer  // set start pointer to next position in heap
      heapStartPointer ← myLinkedListPointers[heapStartPointer] // reset heap start pointer
      myLinkedList[startPointer] ← itemAdd // put item in list
      myLinkedListPointers[startPointer] ← tempPointer  // update linked list pointer
  ENDIF
ENDPROCEDURE
```

Here is the identifier table.

Identifier	Description
startPointer	Start of the linked list

heapStartPointer	Start of the heap
nullPointer	Null pointer set to -1
itemAdd	Item to add to the list
tempPointer	Temporary pointer

Table 19.16

Figure 19.7 below shows the populated linked list and its corresponding pointers again.

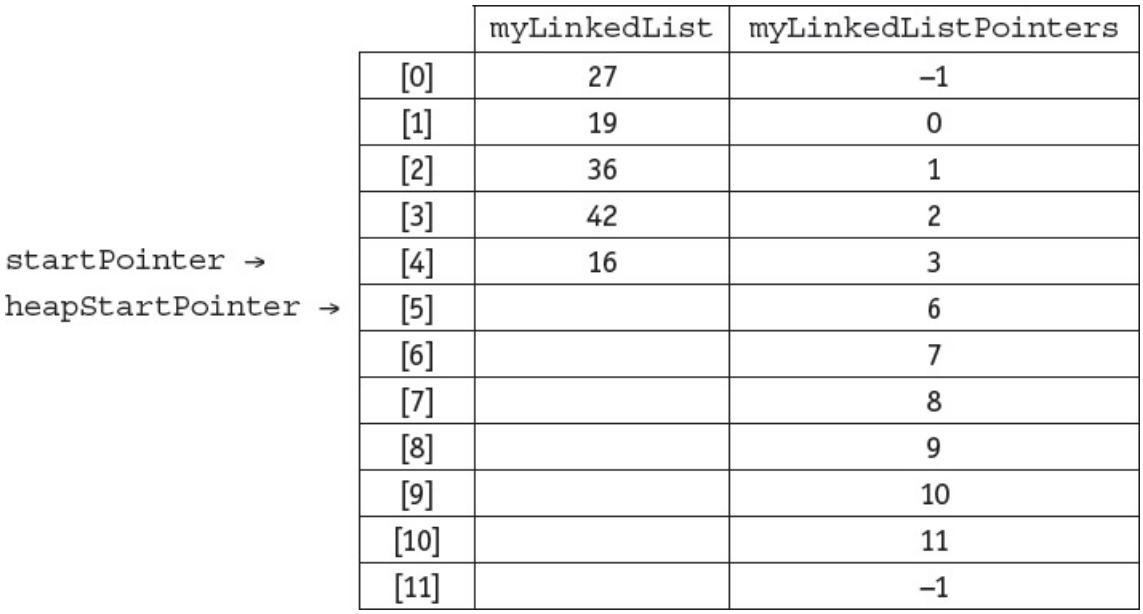


Figure 19.7

The trace table below shows the algorithm being used to add 18 to myLinkedList.

startPointer	heapStartPointer	itemAdd	tempPointer
Already set to 4	Already set to 5	18	
5	6		4

Table 19.17 Trace table

The linked list, myLinkedList, will now be as shown below.

		myLinkedList	myLinkedListPointers
	[0]	27	–1
	[1]	19	0
	[2]	36	1
	[3]	42	2
	[4]	16	3
startPointer →	[5]	18	4
heapStartPointer →	[6]		7
	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		–1

Figure 19.8

The following procedure adds an item to a linked list.

Python

```
def insert(itemAdd):
    global startPointer
    if heapStartPointer == nullPointer:
       print("Linked List full")
    else:
       tempPointer = startPointer
       startPointer = heapStartPointer
heapStartPointer = myLinkedListPointers[heapStartPointer]
       myLinkedList[startPointer] = itemAdd
       myLinkedListPointers[startPointer] = tempPointer
```

Adjusting the pointers and adding the item

VB

```
Sub insert (ByVal itemAdd)
    Dim tempPointer As Integer
    If heapStartPointer = nullPointer Then
       Console.WriteLine("Linked List full")
    Else
       tempPointer = startPointer
       startPointer = heapStartPointer
       heapStartPointer = myLinkedListPointers(heapStartPointer)
       myLinkedList(startPointer) = itemAdd
       myLinkedListPointers(startPointer) = tempPointer
    End if
End Sub
```

Adjusting the pointers and adding the item

Java

```
static void insert(int itemAdd)
{
    if (heapStartPointer == nullPointer)
       System.out.println("Linked List is full");
    else
    {
int tempPointer = startPointer;
       startPointer = heapStartPointer;
       heapStartPointer = myLinkedListPointers[heapStartPointer];
       myLinkedList[startPointer] = itemAdd;
       myLinkedListPointers[startPointer] = tempPointer;      }
}
```

ACTIVITY 19H

Use the algorithm to add 25 to myLinkedList. Show this in a trace table and show myLinkedList once 25 has been added. Add the insert procedure to your program, add code to input an item, add this item to the linked list then print out the list and the pointers before and after the item was added.

Deleting items from a linked list

The algorithm to delete an item from the linked list myLinkedList could be written as a procedure in pseudocode as shown below.

```
DECLARE itemDelete : INTEGER
DECLARE oldIndex : INTEGER
DECLARE index : INTEGER
DECLARE startPointer : INTEGER
DECLARE heapStartPointer : INTEGER
DECLARE tempPointer : INTEGER
CONSTANT nullPointer = -1
PROCEDURE linkedListDelete(itemDelete)
  // check for list empty
  IF startPointer = nullPointer
    THEN
      OUTPUT "Linked list empty"
    ELSE
      // find item to delete in linked list
      index ← startPointer
      WHILE myLinkedList[index] <> itemDelete AND
        (index <> nullPointer) DO
          oldIndex ← index
          index ← myLinkedListPointers[index]
      ENDWHILE
      IF index = nullPointer
        THEN
          OUTPUT "Item ", itemDelete, " not found"
        ELSE
        // delete the pointer and the item
         tempPointer ← myLinkedListPointers[index]
         myLinkedListPointers[index] ← heapStartPointer
         heapStartPointer ← index
         myLinkedListPointers[oldIndex] ← tempPointer
     ENDIF
  ENDIF
ENDPROCEDURE
```

Here is the identifier table.

Identifier	Description
startPointer	Start of the linked list

heapStartPointer	Start of the heap
nullPointer	Null pointer set to −1
index	Pointer to current list element
oldIndex	Pointer to previous list element
itemDelete	Item to delete from the list
tempPointer	Temporary pointer

Figure 19.18

The trace table below shows the algorithm being used to delete 36 from myLinkedList.

startPointer	heapStartPointer	itemDelete	index	oldIndex	tempPointer
Already set to 4	Already set to 5	36	4	4	
			3	3	
			2		
	2				1

Table 19.19 Trace table

The linked list, myLinkedList, will now be as follows.

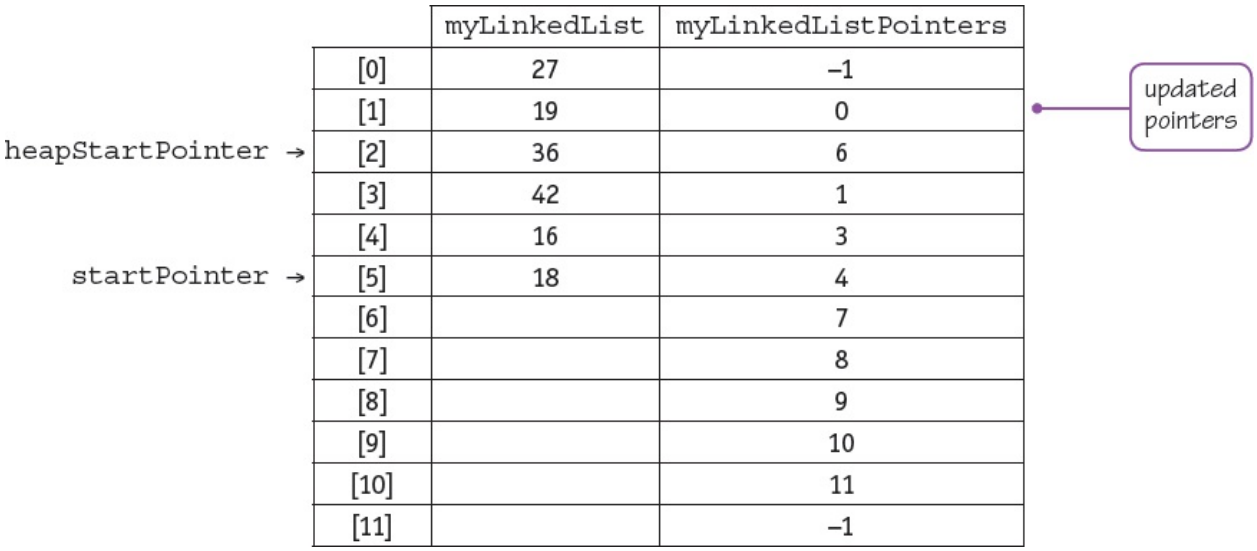


Figure 19.9

The following procedure deletes an item from a linked list.

Python

```
def delete(itemDelete):
    global startPointer, heapStartPointer
    if startPointer == nullPointer:
        print("Linked List empty")
    else:
        index = startPointer
        while myLinkedList[index] != itemDelete and index != nullPointer:
            oldindex = index
            index = myLinkedListPointers[index]
        if index == nullPointer:
            print("Item ", itemDelete, " not found")
        else:
            myLinkedList[index] = None
            tempPointer = myLinkedListPointers[index]
            myLinkedListPointers[index] = heapStartPointer
            heapStartPointer = index
            myLinkedListPointers[oldindex] = tempPointer
```

VB

```
Sub delete (ByVal itemDelete)
    Dim tempPointer, index, oldIndex  As Integer
    If startPointer = nullPointer Then
        Console.WriteLine("Linked List empty")
    Else
        index = startPointer
        While myLinkedList(index) <> itemDelete And index <> nullPointer
            Console.WriteLine( myLinkedList(index) & " " & index)
            Console.ReadKey()
            oldIndex = index
            index = myLinkedListPointers(index)
        End While
        if index = nullPointer Then
            Console.WriteLine("Item " & itemDelete & " not found")
        Else
            myLinkedList(index) = nothing
            tempPointer = myLinkedListPointers(index)
            myLinkedListPointers(index) = heapStartPointer
            heapStartPointer = index
            myLinkedListPointers(oldIndex) = tempPointer
        End If
    End If
End Sub
```

Java

```
static void delete(int itemDelete)
{
    int oldIndex = -1;
    if (startPointer == nullPointer)
        System.out.println("Linked List is empty");
    else
    {
        int index = startPointer;
        while (myLinkedList[index] != itemDelete && index != nullPointer)
        {
            oldIndex = index;
            index = myLinkedListPointers[index];
        }
        if (index == nullPointer)
            System.out.println("Item " + itemDelete + " not found");
        else
        {
            myLinkedList[index] = 0;
            int tempPointer = myLinkedListPointers[index];
            myLinkedListPointers[index] = heapStartPointer;
            heapStartPointer = index;
            myLinkedListPointers[oldIndex] = tempPointer;
        }
    }
}
```

ACTIVITY 19I

Use the algorithm to remove 16 from myLinkedList. Show this in a trace table and show myLinkedList once 16 has been removed. Add the delete procedure to your program, add code to input an item, delete this item to the linked list, then print out the list and the pointers before and after the item was deleted.

Binary trees

A **binary tree** is another frequently used ADT. It is a hierarchical data structure in which each parent node can have a maximum of two child nodes. There are many uses for binary trees; for example, they are used in syntax analysis, compression algorithms and 3D video games.

Figure 19.10 shows the binary tree for the data stored in myList sorted in ascending order. Each item is stored at a node and each node can have up to two branches with the rule if the value to be added is less than the current node branch left, if the value to be added is greater than or equal

to the current node branch right.

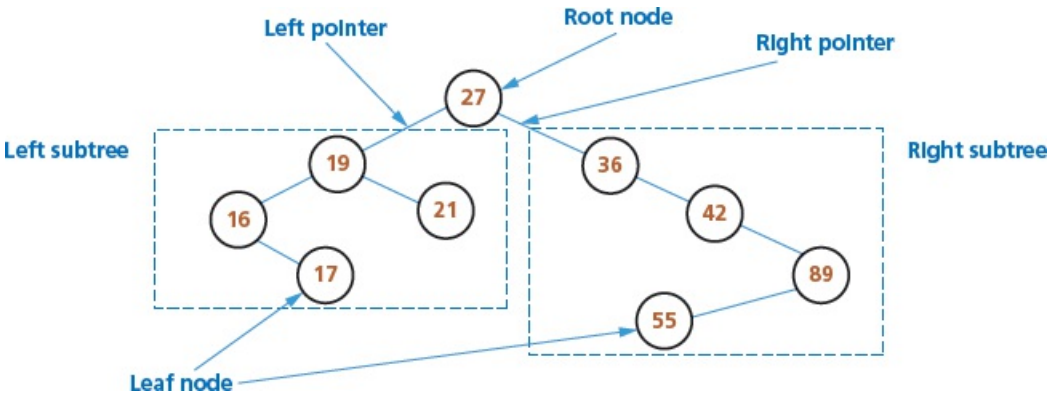


Figure 19.10 Example of an ordered binary tree

A binary tree can also be used to represent an arithmetic expression. Consider (a + b) * (c – a)

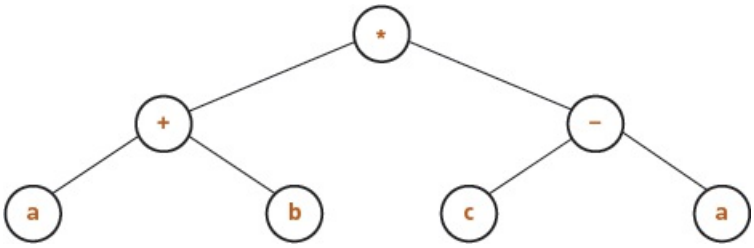


Figure 19.11 Example of an expression as a binary tree

ACTIVITY 19J

Draw the binary tree for the expression (x – y) / (x * y + z).

EXTENSION ACTIVITY 19B

Find out about different tree traversals and how they are used to convert an expression into reverse Polish.

The data structure for an ordered binary tree can be created in pseudocode as follows:

```
TYPE node
    DECLARE item : INTEGER
    DECLARE leftPointer : INTEGER
    DECLARE rightPointer : INTEGER
ENDTYPE
DECLARE myTree[0 : 8] OF node
DECLARE rootPointer : INTEGER
DECLARE nextFreePointer : INTEGER
```

ACTIVITY 19K

Create the data structure in pseudocode for a binary tree to store a list of names. Your list must be able to store at least 50 names.

The populated contents of the data structure myTree is shown below.

Root pointer

myTree	item	leftPointer	rightPointer
[0]	27	1	2
[1]	19	4	6
[2]	36	–1	3
[3]	42	–1	5
[4]	16	–1	7
[5]	89	8	–1
[6]	21	–1	–1
[7]	17	–1	–1
[8]	55	–1	–1

Pointers to items in the tree. –1 is used as a null pointer

Figure 19.12

The root pointer points to the first node in a binary tree. A null pointer is a value stored in the left or right pointer in a binary tree to indicate that there are no nodes below this node on the left or right.

Finding an item in a binary tree

The algorithm to find if an item is in the binary tree myTree and return the pointer to its node if found or a null pointer if not found, could be written as a function in pseudocode, as shown.

```
DECLARE rootPointer : INTEGER
DECLARE itemPointer : INTEGER
DECLARE itemSearch : INTEGER
CONSTANT nullPointer = -1
rootPointer ← 0
FUNCTION find(itemSearch) RETURNS INTEGER
itemPointer ← rootPointer
WHILE myTree[itemPointer].item <> itemSearch AND
  (itemPointer <> nullPointer) DO
    IF myTree[itemPointer].item > itemSearch
      THEN
        itemPointer ← myTree[itemPointer].leftPointer
      ELSE
        itemPointer ← myTree[itemPointer].rightPointer
    ENDIF
ENDWHILE
RETURN itemPointer
```

Here is the identifier table for the binary tree search algorithm shown above.

Identifier	**Description**
myTree	Tree to be searched
node	ADT for tree
rootPointer	Pointer to the start of the tree
leftPointer	Pointer to the left branch
rightPointer	Pointer to the right branch
nullPointer	Null pointer set to −1
itemPointer	Pointer to current item
itemSearch	Item being searched for

Table 19.20

The trace table below shows the algorithm being used to search for 42 in myTree.

rootPointer	itemPointer	itemSearch

0	0	42
	2	
	3	

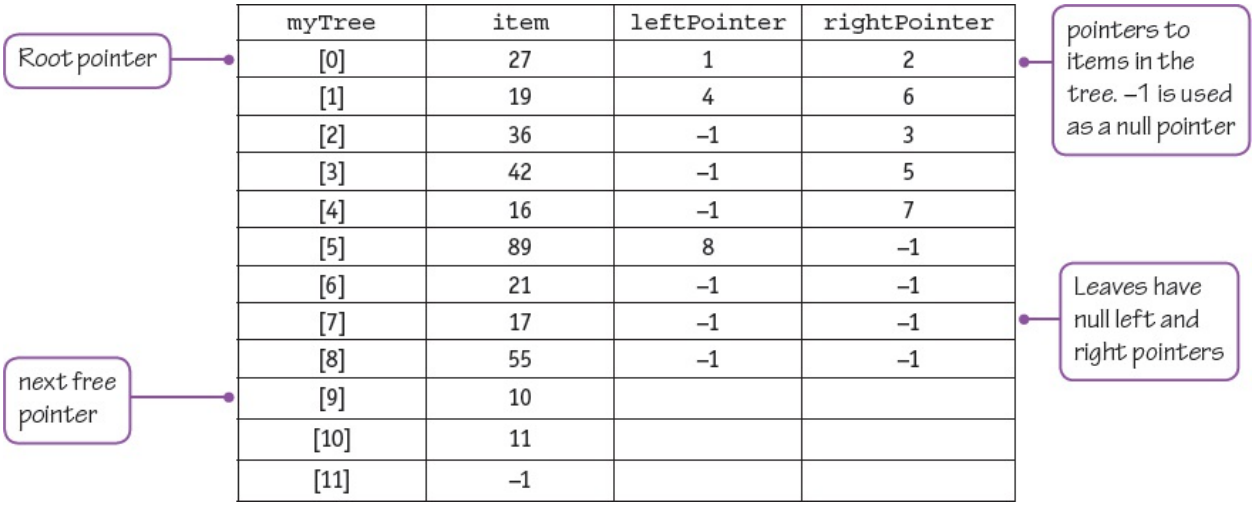
Table 19.21 Trace table

ACTIVITY 19L

Use the algorithm to search for 55 and 75 in myTree. Show the results of each search in a trace table.

Inserting items into a binary tree

The binary tree needs free nodes to add new items. For example, myTree, shown in Figure 19.13 below, now has room for 12 items. The last three nodes have not been filled yet, there is a pointer to the next free node and the free nodes are set up like a heap in a linked list, using the left pointer.



myTree	item	leftPointer	rightPointer
[0]	27	1	2
[1]	19	4	6
[2]	36	–1	3
[3]	42	–1	5
[4]	16	–1	7
[5]	89	8	–1
[6]	21	–1	–1
[7]	17	–1	–1
[8]	55	–1	–1
[9]	10		
[10]	11		
[11]	–1		

Figure 19.13

The algorithm to insert an item at a new node in the binary tree myTree could be written as a procedure in pseudocode as shown below.

```
TYPE node
    DECLARE item : INTEGER
    DECLARE leftPointer : INTEGER
    DECLARE rightPointer : INTEGER
    DECLARE oldPointer : INTEGER
    DECLARE leftBranch : BOOLEAN
ENDTYPE
DECLARE myTree[0 : 11] OF node
// binary tree now has extra spaces
DECLARE rootPointer : INTEGER
DECLARE nextFreePointer : INTEGER
DECLARE itemPointer : INTEGER
DECLARE itemAdd : INTEGER
DECLARE itemAddPointer : Integer
CONSTANT nullPointer = -1
// needed to use the binary tree
PROCEDURE nodeAdd(itemAdd)
    // check for full tree
    IF nextFreePointer = nullPointer
      THEN
        OUTPUT "No nodes free"
    ELSE
      //use next free node
      itemAddPointer ← nextFreePointer
      nextFreePointer ← myTree[nextFreePointer].leftPointer
      itemPointer ← rootPointer
      // check for empty tree
      IF itemPointer = nullPointer
        THEN
          rootPointer ← itemAddPointer
        ELSE
          // find where to insert a new leaf
          WHILE (itemPointer <> nullPointer) DO
              oldPointer ← itemPointer
              IF myTree[itemPointer].item > itemAdd
                THEN  // choose left branch
                  leftBranch ← TRUE
                  itemPointer ← myTree[itemPointer].leftPointer
                ELSE  // choose right branch
                  leftBranch ← FALSE
                  itemPointer ← myTree[itemPointer].rightPointer
              ENDIF
          ENDWHILE
          IF leftBranch  //use left or right branch
            THEN
              myTree[oldPointer].leftPointer ← itemAddPointer
            ELSE
              myTree[oldPointer].rightPointer ← itemAddPointer
          ENDIF
      ENDIF
      // store item to be added in the new node
      myTree[itemAddPointer].leftPointer ← nullPointer
      myTree[itemAddPointer].rightPointer ← nullPointer
      myTree[itemAddPointer].item ← itemAdd
    ENDIF
ENDPROCEDURE
```

Here is the identifier table.

Identifier	Description
myTree	Tree to be searched
node	ADT for tree
rootPointer	Pointer to the start of the tree
leftPointer	Pointer to the left branch
rightPointer	Pointer to the right branch
nullPointer	Null pointer set to -1
itemPointer	Pointer to current item in tree
itemAdd	Item to add to tree
nextFreePointer	Pointer to next free node
itemAddPointer	Pointer to position in tree to store item to be added
oldPointer	Pointer to leaf node that is going to point to item added
leftBranch	Flag to identify whether to go down the left branch or the right branch

Table 19.22

The trace table below shows the algorithm being used to add 18 to myTree.

leftBranch	nextFreePointer	itemAddPointer	rootPointer	itemAdd	itemPointer	oldPointer
	Already set to 9	9	Already set to 0	18		
	10				0	0
TRUE					1	1
TRUE					4	4
FALSE					7	7
					–1	

Table 19.23

The tree, myTree will now be as shown below.

myTree	item	leftPointer	rightPointer
[0]	27	1	2
[1]	19	4	6
[2]	36	–1	3
[3]	42	–1	5
[4]	16	–1	7
[5]	89	8	–1
[6]	21	–1	–1
[7]	17	–1	9
[8]	55	–1	–1
[9]	18	–1	–1
[10]	11		
[11]	–1		

pointer to new node in correct position

new leaf node

next free pointer now 10

Figure 19.14

ACTIVITY 19M

Use the algorithm to add 25 to myTree. Show this in a trace table and show myTree once 25 has been added.

Implementing binary trees in Python, VB.NET or Java requires the use of objects and recursion. An example will be given in Chapter 20.

Graphs

A **graph** is a non-linear data structure consisting of nodes and edges. This is an ADT used to implement directed and undirected graphs. A graph consists of a set of nodes and edges that join a pair of nodes. If the edges have a direction from one node to the other it is a directed graph.

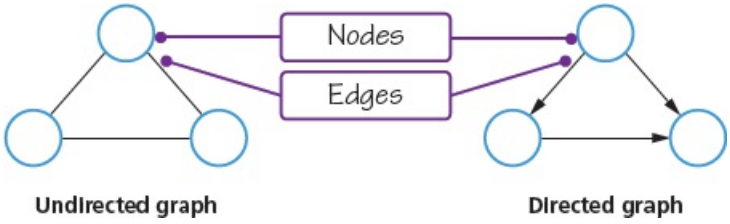


Figure 19.15

As we saw in Chapter 18, graphs are used to represent real life networks, such as

- bus routes, where the nodes are bus stops and the edges connect two stops next to each other
- websites, where each web page is a node and the edges show the links between each web page
- social media networks, where each node contains information about a person and the edges connect people who are friends.

Each edge may have a weight; for example, in the bus route, the weight could be the distance between bus stops or the cost of the bus fare.

A path is the list of nodes connected by edges between two given nodes and a cycle is a list of

nodes that return to the same node.

For example, a graph of the bus routes in a town could be as follows. The distance between each bus stop in kilometres is shown on the graph.

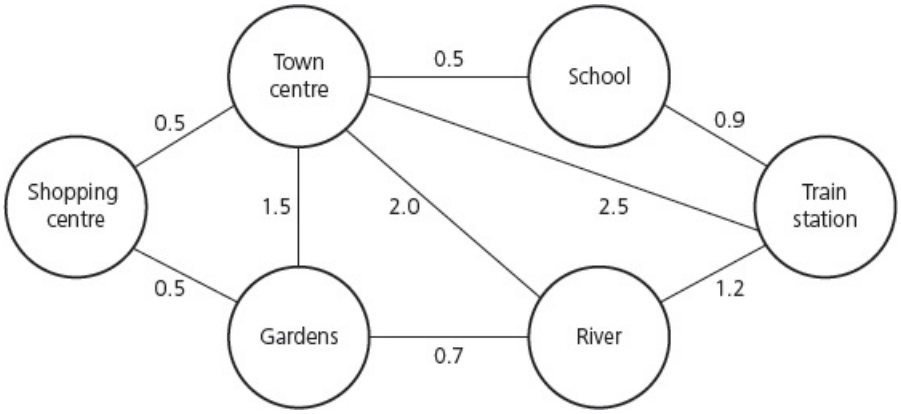


Figure 19.16

ACTIVITY 19N

Find another path from School to Gardens. Find the shortest path from Town centre to Train station. Find the shortest cycle from the Town centre.

A path from School to Gardens could be Path = (School, Train station, River, Gardens).

19.1.4 Implementing one ADT from another ADT

Every ADT is a collection of data and the methods used on the data. When an ADT is defined, the definition can refer to other data types. For example, myLinkedList refers to the data type INTEGER in its data definition.

A linked list type could be defined as follows.

```
TYPE linkedList
  DECLARE item : INTEGER
  DECLARE Pointer : INTEGER
ENDTYPE
// a linked list to store integers
```

And then used as follows.

```
DECLARE myLinkedList : ARRAY [0:11] OF linkedList
DECLARE heapStartPointer : INTEGER
DECLARE startPointer : INTEGER
DECLARE index : INTEGER
```

ACTIVITY 19O

Write pseudocode to declare a linked list to store names. Use this to write pseudocode to set up a linked list that will store 30 names. Write a program to store and display names in this linked list.

The data types for a stack, queue and a binary tree have been defined using existing data types.

Another data type is a **dictionary**, which is an ADT that consists of pairs consisting of a key and a value, where the key is used to find the value. Each key can only appear once. Keys in a dictionary are unordered. A value is retrieved from a dictionary by specifying its corresponding key. The same value may appear more than once. A dictionary differs from a set because the values can be duplicated. As a dictionary is not an ordered list, it can be declared using a linked list as part of the definition.

A dictionary type could be defined in pseudocode as follows.

```
TYPE linkedList
  DECLARE item : STRING
  DECLARE pointer : INTEGER
ENDTYPE
TYPE dictionary
  DECLARE key : myLinkedList : ARRAY [0:19] OF
  linkedList
  DECLARE value : ARRAY [0:19] OF STRING
ENDTYPE
```

And then used as follows.

```
DECLARE myDictionary : linkedList
DECLARE heapStartPointer : INTEGER
DECLARE startPointer : INTEGER
DECLARE index : INTEGER
```

Each of the programming languages used in Cambridge International A Level Computer Science provide a dictionary data type, as shown in the table below.

Dictionary data type example	Language

studentdict = {
"Leon": 27,
"Ahmad": 78,
"Susie": 64
}
	Python

Dim studentdict As New Dictionary(Of String, Integer) studentdict.Add("Leon", 27) studentdict.Add("Ahmad", 78) studentdict.Add("Susie", 64)	VB

studentdict = dict([("Leon", 27), ("Ahmad", 78), ("Susie", 64)])
Or
Dictionary<Integer, String> studentdict = new Hashtable<Integer, String>();
studentdict.put(27,"Leon");
studentdict.put(78,"Ahmad");
studentdict.put(64,"Susie");
	Java Dictionary is no longer used in Java but can be implemented using a hash table

Table 19.24

ACTIVITY 19P

In the programming language of your choice, write a program to use a dictionary to store the names of students as their keys and their examination scores as their values. Then find a student's examination score, add a student and score and delete a student and score.

19.1.5 Comparing algorithms

Big O notation is a mathematical notation used to describe the performance or complexity of an algorithm in relation to the time taken or the memory used for the task. It is used to describe the worst-case scenario; for example, how the maximum number of comparisons required to find a value in a list using a particular search algorithm increases with the number of values in the list.

Big O order of time complexity

	Description	Example
O(1)	describes an algorithm that always takes the same time to perform the task	deciding if a number is even or odd
O(N)	describes an algorithm where the time to perform the task will grow linearly in direct proportion to N, the number of items of data the algorithm is using	a linear search
$O(N^2)$	describes an algorithm where the time to perform the task will grow linearly in direct proportion to the square of N, the number of items of data the algorithm is using	bubble sort, insertion sort
$O(2^N)$	describes an algorithm where the time to perform the task doubles every time the algorithm uses an extra item of data	calculation of Fibonacci numbers using recursion (see Section 19.2)
O(Log N)	describes an algorithm where the time to perform the task goes up linearly as the number of items goes up exponentially	binary search

Table 19.25 Big O order of time complexity

Big O order of space complexity

	Description	Example
O(1)	describes an algorithm that always uses the same space to perform the task	any algorithm that just uses variables, for example d = a + b + c
O(N)	describes an algorithm where the space to perform the task will grow linearly in direct proportion to N, the number of items of data the algorithm is using	any algorithm that uses arrays, for example a loop to calculate a running total of values input to an array of N elements

Table 19.26 Big O order of space complexity

ACTIVITY 19Q

1 Using diagrams, describe the structure of
   a) a binary tree
   b) a linked list.
2 a) Explain what is meant by a *dictionary data type*.
   b) Show how a dictionary data type can be constructed from a linked list.
3 Compare the performance of a linear search and a binary search using Big O notation.

19.2 Recursion

WHAT YOU SHOULD ALREADY KNOW

Remind yourself of the definitions of the following mathematical functions, which many of you will be familiar with, and see how they are constructed.

- Factorials
- Arithmetic sequences
- Fibonacci numbers
- Compound interest

Key terms

Recursion – a process using a function or procedure that is defined in terms of itself and calls itself.

Base case – a terminating solution to a process that is not recursive.

General case – a solution to a process that is recursively defined.

Winding – process which occurs when a recursive function or procedure is called until the base case is found.

Unwinding – process which occurs when a recursive function finds the base case and the function returns the values.

19.2.1 Understanding recursion

Recursion is a process using a function or procedure that is defined in terms of itself and calls itself. The process is defined using a **base case**, a terminating solution to a process that is not recursive, and a **general case**, a solution to a process that is recursively defined.

For example, a function to calculate a factorial for any positive whole number n! is recursive. The definition for the function uses:

a base case of 0! = 1

a general case of n! = n * (n–1)!

This can be written in pseudocode as a recursive function.

```
FUNCTION factorial (number : INTEGER) RETURNS INTEGER
    IF number = 0
      THEN
        answer ← 1 // base case
      ELSE
        answer ← number * factorial (number - 1)
        // recursive call with general case
    ENDIF
    RETURN answer
ENDFUNCTION
```

With recursive functions, the statements after the recursive function call are not executed until the base case is reached; this is called **winding**. After the base case is reached and can be used in the recursive process, the function is **unwinding**.

In order to understand how the winding and unwinding processes in recursion work, we can use a trace table for a specific example: 3!

Call number	Function call	number	answer	RETURN
1	Factorial (3)	3	3 * factorial (2)	
2	Factorial (2)	2	2 * factorial (1)	
3	Factorial (1)	1	1 * factorial (0)	
4	Factorial (0)	0	1	1
3 continued	Factorial (1)	1	1 * 1	1
2 continued	Factorial (2)	2	2 * 1	2
1 continued	Factorial (3)	3	3 * 2	6

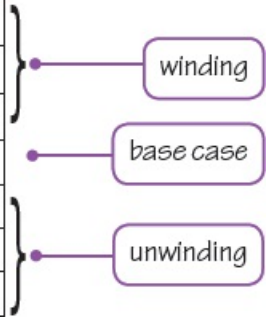


Table 19.27

Here is a simple recursive factorial program written in Python, VB and Java using a function.

Python

```
#Python program recursive factorial function
def factorial(number):
    if number == 0:
        answer = 1
    else:
        answer = number * factorial(number - 1)
    return answer
print(factorial(0))
print(factorial(5))
```

VB

```
'VB program recursive factorial function
Module Module1
    Sub Main()
        Console.WriteLine(factorial(0))
        Console.Writeline(factorial(5))
        Console.ReadKey()
    End Sub
    Function factorial(ByVal number As Integer) As Integer
           Dim answer As Integer
        If number = 0 Then
           answer = 1
       Else
            answer = number * factorial(number - 1)
        End If
                return answer
    End Function
End Module
```

Java

```
// Java program recursive factorial function
public class Factorial {
    public static void main(String[] args) {
        System.out.println(factorial(0));
        System.out.println(factorial(5));
    }
    public static int factorial(int number)
    {
        int answer;
            if (number == 0)
                    answer = 1;
        else
                    answer = number * factorial(number - 1);
        return answer;
    }
}
```

ACTIVITY 19R

Write the recursive factorial function in the programming language of your choice. Test your program with 0! and 5!

Complete trace tables for 0! and 5! using the recursive factorial function written in pseudocode and compare the results from your program with the trace tables.

Compound interest can be calculated using a recursive function. Where the principal is the amount of money invested, rate is the rate of interest and years is the number of years the money has been invested.

The base case is	total0 = principal where years = 0
The general case is	totaln = totaln-1 * rate

Table 19.28

```
DEFINE FUNCTION compoundInt(principal, rate, years : REAL) RETURNS REAL
    IF years = 0
      THEN
        total ← principal
      ELSE
        total ← compoundInt(principal * rate, rate, years - 1)
    ENDIF
    RETURN total
ENDFUNCTION
```

This function can be traced for a principal of 100 over three years at 1.05 (5% interest).

Call number	Function call	years	total	RETURN
1	compoundInt(100, 1.05, 3)	3	compoundInt(105, 1.05, 2)	
2	compoundInt(105, 1.05, 2)	2	compoundInt(105, 1.05, 1)	
3	compoundInt(105, 1.05, 1)	1	compoundInt(105, 1.05, 0)	
4	compoundInt(105, 1.05, 0)	0	100	100
3 cont	compoundInt(105, 1.05, 1)	1	105	105
2 cont	compoundInt(105, 1.05, 2)	2	110.25	110.25
1 cont	compoundInt(105, 1.05, 3)	3	115.76	115.76

Table 19.29

ACTIVITY 19S

The Fibonacci series is defined as a sequence of numbers in which the first two numbers are 0 and 1, depending on the selected beginning point of the sequence, and each subsequent number is the sum of the previous two.

Identify the base case and the general case for this series. Write a pseudocode algorithm to find and output the nth term. Test your algorithm by drawing a trace table for the fourth term.

EXTENSION ACTIVITY 19C

Write your function from Activity 19S in the high-level programming language of your choice. Test this with the 5th and 27th terms.

Benefits of recursion

Recursive solutions can contain fewer programming statements than an iterative solution. The solutions can solve complex problems in a simpler way than an iterative solution. However, if recursive calls to procedures and functions are very repetitive, there is a very heavy use of the stack, which can lead to stack overflow. For example, factorial(100) would require 100 function calls to be placed on the stack before the function unwinds.

19.2.2 How a compiler implements recursion

Recursive code needs to make use of the stack; therefore, in order to implement recursive procedures and functions in a high-level programming language, a compiler must produce object code that pushes return addresses and values of local variables onto the stack with each recursive call, winding. The object code then pops the return addresses and values of local variables off the stack, unwinding.

ACTIVITY 19T

1 Explain what is meant by *recursion* and give the benefits of using recursion in programming.

2 Explain why a compiler needs to produce object code that uses the stack for a recursive procedure.

End of chapter questions

1 Data is stored in the array NameList[1:10]. This data is to be sorted.

a) i) Copy and complete this pseudocode algorithm for an insertion sort.

[7]

```
FOR ThisPointer ← 2 TO ..........................................
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1
    WHILE (NameList[Pointer] > Temp) AND ....................
        // move list item to next location
        NameList[.....................] ← NameList[.....................]
        Pointer ← ..............................................
    ENDWHILE
    // insert value of Temp in correct location
    NameList[.................................] ← ........................
ENDFOR
```

ii) A special case is when NameList is already in order. The algorithm in part a) i) is applied to this special case.
Explain how many iterations are carried out for each of the loops.

[3]

b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 9
        IF NameList[Pointer] > NameList[Pointer + 1]
          THEN
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
ENDFOR
```

i) As in part a) ii), a special case is when NameList is already in order. The algorithm in part b) is applied to this special case.
Explain how many iterations are carried out for each of the loops.

[2]

ii) Rewrite the algorithm in part b), using **pseudocode**, to reduce the number of unnecessary comparisons.
Use the same variable names where appropriate.

[5]

Cambridge International AS & A Level Computer Science 9608
Paper 41 Q5 June 2015

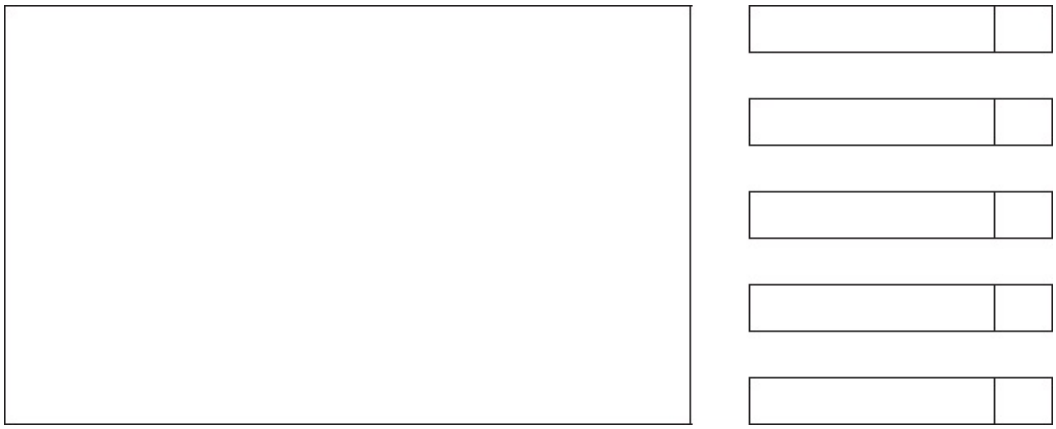
2 A Queue Abstract Data type (ADT) has these associated operations:
– create queue
– add item to queue
– remove item from queue
The queue ADT is to be implemented as a linked list of nodes.
Each node consists of data and a pointer to the next node.

a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Copy the diagram and add appropriate labels to show the final state of the queue. Use the space on the left as a workspace.
Show your final answer in the node shapes on the right.

[3]



b) Using pseudocode, a record type, Node, is declared as follows:

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

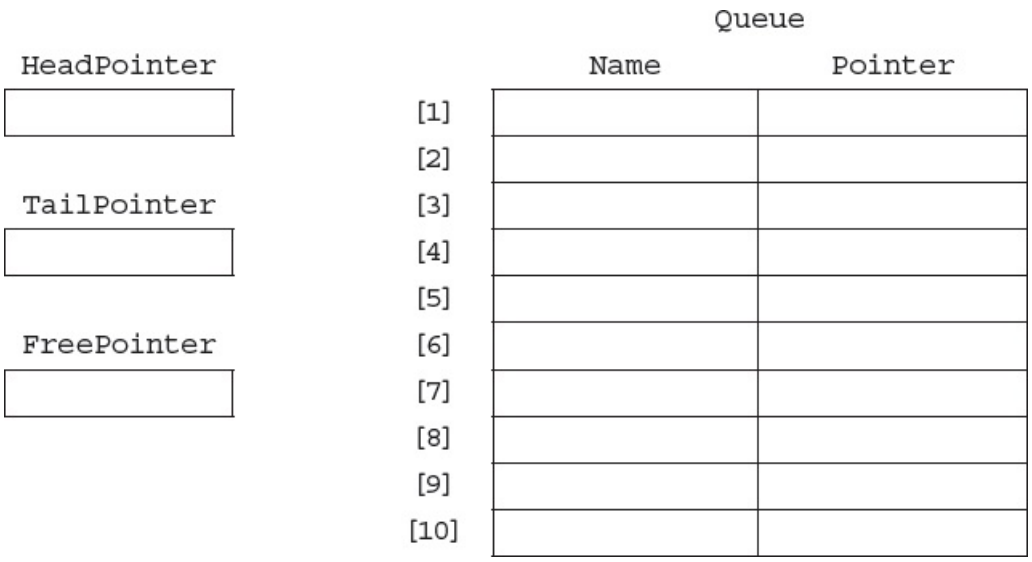
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array Queue.

i) The CreateQueue operation links all nodes and initialises the three pointers that need to be used: HeadPointer, TailPointer and FreePointer.
Copy and complete the diagram to show the value of all pointers after CreateQueue has been executed.

[4]



ii) The algorithm for adding a name to the queue is written, using pseudocode, as a procedure with the header:

```
PROCEDURE AddName(NewName)
```

where NewName is the new name to be added to the queue.
The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
Queue	Array[1:10] OF Node	Array to store node data
NewName	STRING	Name to be added
FreePointer	INTEGER	Pointer to next free node in array
HeadPointer	INTEGER	Pointer to first node in queue
TailPointer	INTEGER	Pointer to last node in queue
CurrentPointer	INTEGER	Pointer to current node

```
PROCEDURE AddName(BYVALUE NewName : STRING)
   // Report error if no free nodes remaining
   IF FreePointer = 0
     THEN
       Report Error
     ELSE
       // new name placed in node at head of
       free list
        CurrentPointer ← FreePointer
        Queue[CurrentPointer].Name ← NewName
       // adjust free pointer
       FreePointer ← Queue[CurrentPointer].
       Pointer
        // if first name in queue then adjust
        head pointer
        IF HeadPointer = 0
         THEN
           HeadPointer ← CurrentPointer
      ENDIF
     // current node is new end of queue
     Queue[CurrentPointer].Pointer ← 0
     TailPointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

Copy and complete the **pseudocode** for the procedure RemoveName. Use the variables listed in the identifier table.

[6]

```
    PROCEDURE RemoveName()
     // Report error if Queue is empty
.....
.....
.....
.....
     OUTPUT Queue[.....].Name
     // current node  is head of queue
.....
     // update head pointer
.....
     // if only one element in queue then update tail
        pointer
.....
.....
.....
.....
     // link released node to free list
.....
.....
.....
    ENDPROCEDURE
```

Cambridge International AS & A Level Computer Science 9608
Paper 41 Q6 June 2015

20 Further programming

In this chapter, you will learn about

- the characteristics of a number of programming paradigms, including low-level programming, imperative (procedural) programming, object-oriented programming and declarative programming
- how to write code to perform file-processing operations on serial, sequential and random files
- exceptions and the importance of exception handling.

20.1 Programming paradigms

WHAT YOU SHOULD ALREADY KNOW

In Chapter 4, Section 4.2, you learnt about assembly language, and in Chapter 11, Section 11.3, you learnt about structured programming. Review these sections then try these three questions before you read the first part of this chapter.

1 Describe **four** modes of addressing in assembly language.

2 Write an assembly language program to add the numbers 7 and 5 together and store the result in the accumulator.

3 **a)** Explain the difference between a procedure and a function.

b) Describe how to pass parameters.

c) Describe the difference between a procedure definition and a procedure call.

4 Write a short program that uses a procedure.

Throughout this section, you will be prompted to refer to previous chapters to review related content.

Key terms

Programming paradigm – a set of programming concepts.

Low-level programming – programming instructions that use the computer's basic instruction set.

Imperative programming – programming paradigm in which the steps required to execute a program are set out in the order they need to be carried out.

Object-oriented programming (OOP) – a programming methodology that uses self-contained objects, which contain programming statements (methods) and data, and which communicate with each other.

Class – a template defining the methods and data of a certain type of object.

Attributes (class) – the data items in a class.

Method – a programmed procedure that is defined as part of a class.

Encapsulation – process of putting data and methods together as a single unit, a class.

Object – an instance of a class that is self-contained and includes data and methods.

Property – data and methods within an object that perform a named action.

Instance – An occurrence of an object during the execution of a program.

Data hiding – technique which protects the integrity of an object by restricting access to the data and methods within that object.

Inheritance – process in which the methods and data from one class, a superclass or base

class, are copied to another class, a derived class.

Polymorphism – feature of object-oriented programming that allows methods to be redefined for derived classes.

Overloading – feature of object-oriented programming that allows a method to be defined more than once in a class, so it can be used in different situations.

Containment (aggregation) – process by which one class can contain other classes.

Getter – a method that gets the value of a property.

Setter – a method used to control changes to a variable.

Constructor – a method used to initialise a new object.

Destructor – a method that is automatically invoked when an object is destroyed.

Declarative programming – statements of facts and rules together with a mechanism for setting goals in the form of a query.

Fact – a 'thing' that is known.

Rules – relationships between facts.

A **programming paradigm** is a set of programming concepts. We have already considered two different programming paradigms: low-level and imperative (procedural) programming.

The style and capability of any programming language is defined by its paradigm. Some programming languages, for example JavaScript, only follow one paradigm; others, for example Python, support multiple paradigms. Most programming languages are multi-paradigm. In this section of the chapter, we will consider four programming paradigms: low-level, imperative, object-oriented and declarative.

20.1.1 Low-level programming

Low-level programming uses instructions from the computer's basic instruction set. Assembly language and machine code both use low-level instructions. This type of programming is used when the program needs to make use of specific addresses and registers in a computer, for example when writing a printer driver.

In Chapter 4, Section 4.2.4, we looked at addressing modes. These are also covered by the Cambridge International A Level syllabus. Review Section 4.2.4 before completing Activity 20A.

ACTIVITY 20A

A section of memory in a computer contains these denary values:

Address	Denary value
230	231
231	5
232	7
233	9
234	11
235	0

Give the value stored in the accumulator (ACC) and the index register (IX) after each of these instructions have been executed and state the mode of addressing used.

Address	Opcode	Operand
500	LDM	#230
501	LDD	230
502	LDI	230
503	LDR	#1
504	LDX	230
505	CMP	#0
506	JPE	509
507	INC	IX
508	JMP	504

509	JMP	509

// this stops the program, it executes the same instruction until the computer is turned off!

20.1.2 Imperative programming

In **imperative programming**, the steps required to execute a program are set out in the order they need to be carried out. This programming paradigm is often used in the early stages of teaching programming. Imperative programming is often developed into structured programming, which has a more logical structure and makes use of procedures and functions, together with local and global variables. Imperative programming is also known as procedural programming.

Programs written using the imperative paradigm may be smaller and take less time to execute than programs written using the object-oriented or declarative paradigms. This is because there are fewer instructions and less data storage is required for the compiled object code. Imperative programming works well for small, simple programs. Programs written using this methodology can be easier for others to read and understand.

In Chapter 11, Section 11.3, we looked at structured programming. This is also covered by the Cambridge International A Level syllabus. Review Section 11.3 then complete Activity 20B.

ACTIVITY 20B

Write a pseudocode algorithm to calculate the areas of five different shapes (square, rectangle, triangle, parallelogram and circle) using the basic imperative programming paradigm (no procedures or functions, and using only global variables).

Rewrite the pseudocode algorithm in a more structured way using the procedural programming paradigm (make sure you use procedures, functions, and local and global variables).

Write and test both algorithms using the programming language of your choice.

20.1.3 Object-oriented programming (OOP)

Object-oriented programming (OOP) is a programming methodology that uses self-contained objects, which contain programming statements (methods) and data, and which communicate with each other. This programming paradigm is often used to solve more complex problems as it enables programmers to work with real life things. Many procedural programming languages have been developed to support OOP. For example, Java, Python and Visual Basic all allow programmers to use either procedural programming or OOP.

Object-oriented programming uses its own terminology, which we will explore here.

Class

A **class** is a template defining the methods and data of a certain type of object. The **attributes** are the data items in a class. A **method** is a programmed procedure that is defined as part of a class. Putting the data and methods together as a single unit, a class, is called **encapsulation**. To ensure that only the methods declared can be used to access the data within a class, attributes need to be declared as private and the methods need to be declared as public.

For example, a shape can have `name`, `area` and `perimeter` as attributes and the methods `set shape`, `calculate area`, `calculate perimeter`. This information can be shown in a class diagram (Figure 20.1).

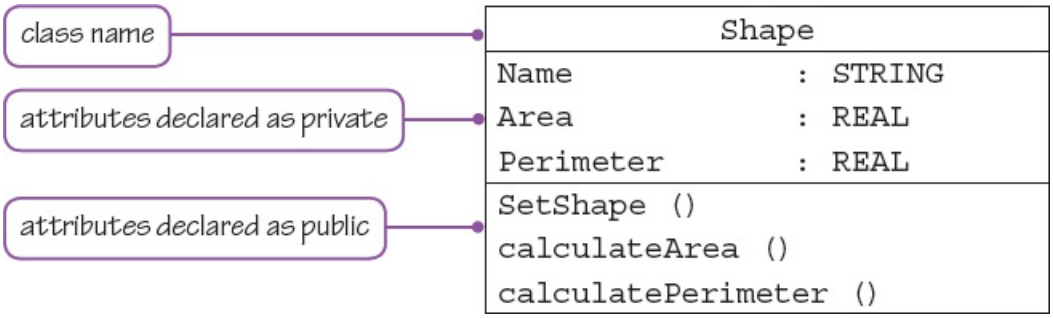


Figure 20.1 Shape class diagram

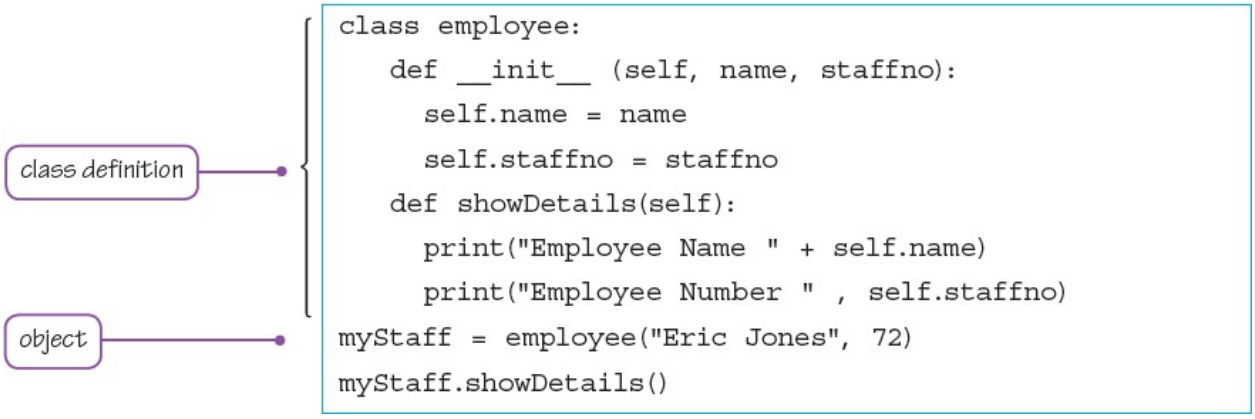
Object

When writing a program, an **object** needs to be declared using a class type that has already been defined. An object is an instance of a class that is self-contained and includes data and methods. **Properties** of an object are the data and methods within an object that perform named actions. An occurrence of an object during the execution of a program is called an **instance**.

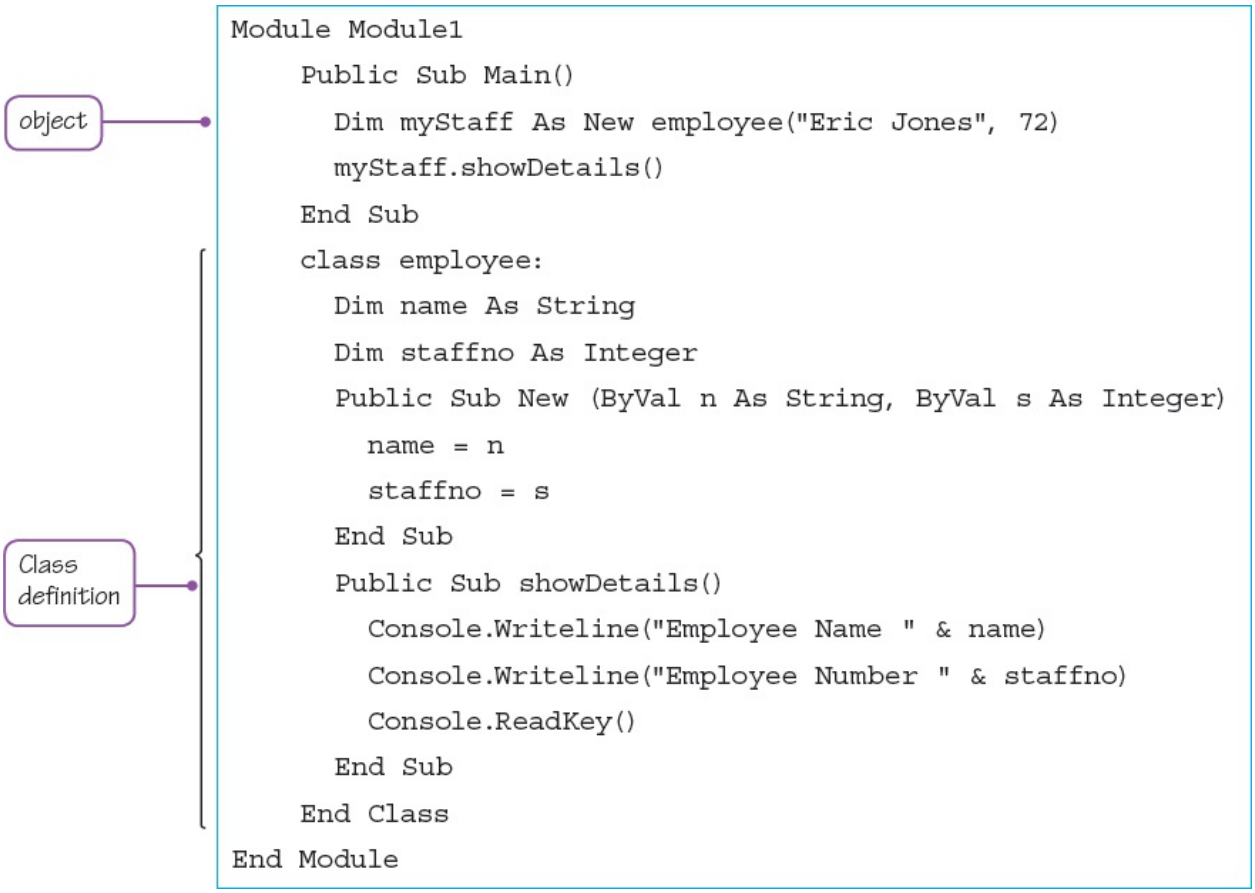
For example, a class `employee` is defined and the object `myStaff` is instanced in these programs using Python, VB and Java.

Python

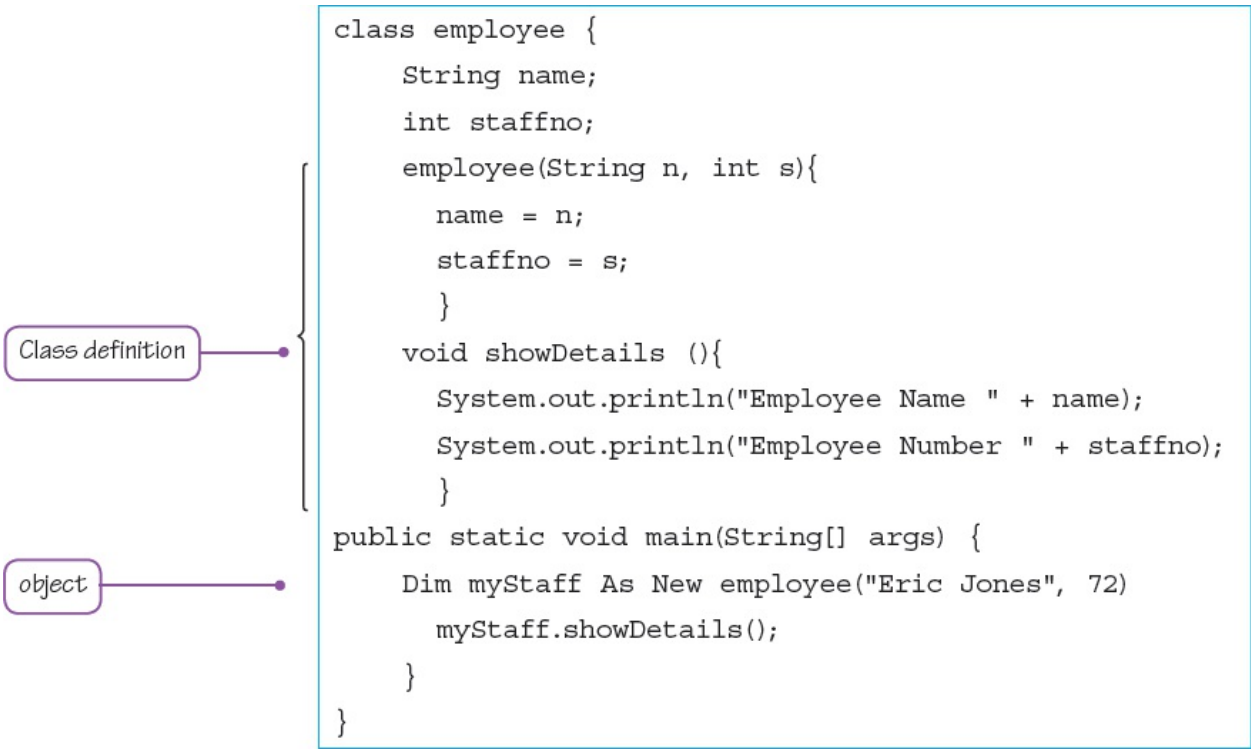
Python



VB



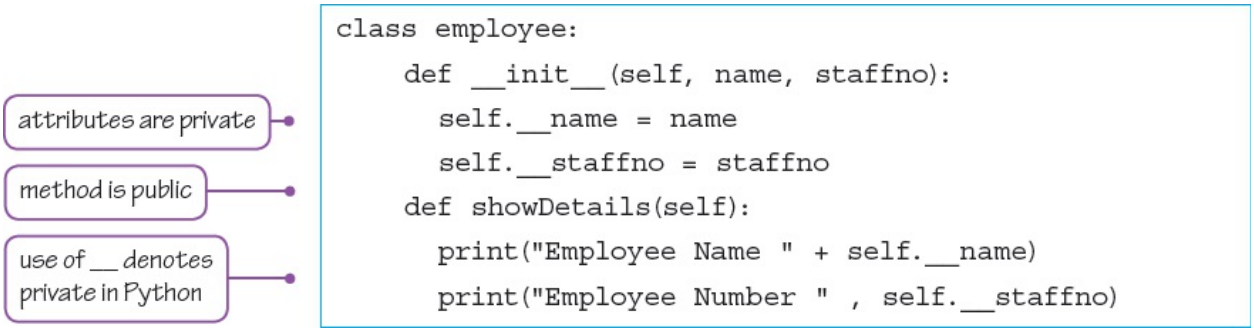
Java



Data hiding protects the integrity of an object by restricting access to the data and methods within that object. One way of achieving data hiding in OOP is to use encapsulation. Data hiding reduces the complexity of programming and increases data protection and the security of data.

Here is an example of a definition of a class with private attributes in Python, VB and Java.

Python



VB

Attributes are private

Methods are public

```
class employee:
    Private name As String
    Private staffno As Integer
    Public Sub New (ByVal n As String, ByVal s As Integer)
      name = n
      staffno = s
    End Sub
    Public Sub showDetails()
      Console.Writeline("Employee Name "  & name)
      Console.Writeline("Employee Number " & staffno)
      Console.ReadKey()
    End Sub
End Class
```

Constructor to set attributes

Java

Attributes are private

Methods are public

```
// Java employee OOP program
class employee {
    private String name;
    private int staffno;
    employee(String n, int s){
      name = n;
      staffno = s;
      }
    public void showDetails (){
      System.out.println("Employee Name " + name);
      System.out.println("Employee Number " + staffno);
      }
}
 public class MainObject{
    public static void main(String[] args) {
      employee myStaff = new employee("Eric Jones", 72);
        myStaff.showDetails();
    }
}
```

Constructor to set attributes

ACTIVITY 20C

Write a short program to declare a class, `student`, with the private attributes `name`, `dateOfBirth` and `examMark`, and the public method `displayExamMark`. Declare an object `myStudent`, with a name and exam mark of your choice, and use your method to display the exam mark.

Inheritance

Inheritance is the process by which the methods and data from one class, a superclass or base class, are copied to another class, a derived class.

Figure 20.2 shows single inheritance, in which a derived class inherits from a single superclass.

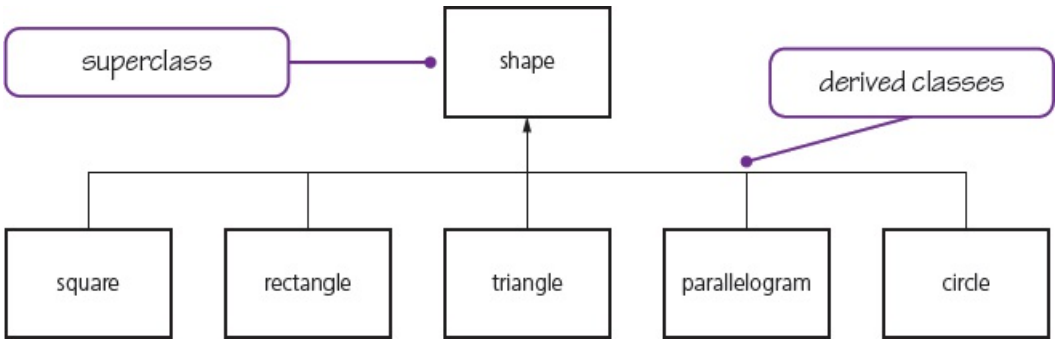


Figure 20.2 Inheritance diagram – single inheritance

Multiple inheritance is where a derived class inherits from more than one superclass (Figure 20.3).

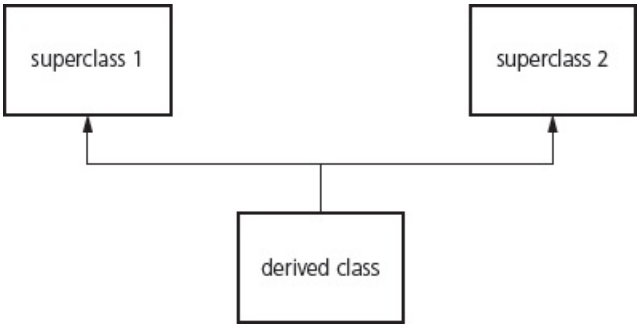


Figure 20.3 Inheritance diagram – multiple inheritance

EXTENSION ACTIVITY 20A

Not all programming languages support multiple inheritance. Check if the language you are using does.

Here is an example that shows the use of inheritance.

A base class `employee` and the derived classes `partTime` and `fullTime` are defined. The objects `permanentStaff` and `temporaryStaff` are instanced in these examples and use the method `showDetails`.

Python

base class employee

derived class partTime

derived class fullTime

```
class employee:
    def __init__ (self, name, staffno):
      self.__name = name
      self.__staffno = staffno
      self.__fullTimeStaff = True
    def showDetails(self):
      print("Employee Name " + self.__name)
      print("Employee Number " , self.__staffno)
class partTime(employee):
    def __init__(self, name, staffno):
      employee.__init__(self, name, staffno)
      self.__fullTimeStaff = False
      self.__hoursWorked = 0
    def getHoursWorked (self):
      return(self.__hoursWorked)
class fullTime(employee):
    def __init__(self, name, staffno):
      employee.__init__(self, name, staffno)
      self.__fullTimeStaff = True
      self.__yearlySalary = 0
    def getYearlySalary (self):
      return(self.__yearlySalary)
permanentStaff = fullTime("Eric Jones", 72)
permanentStaff.showDetails()
temporaryStaff = partTime ("Alice Hue", 1017)
temporaryStaff.showDetails ()
```

VB

```
'VB Employee OOP program with inheritance
Module Module1
    Public Sub Main()
      Dim permanentStaff As New fullTime("Eric Jones", 72, 50000.00)
      permanentStaff.showDetails()
      Dim temporaryStaff As New partTime("Alice Hu", 1017, 45)
      temporaryStaff.showDetails()
    End Sub
```

```
    class employee
      Protected name As String
      Protected staffno As Integer
      Private fullTimeStaff As Boolean
      Public Sub New (ByVal n As String, ByVal s As Integer)
        name = n
        staffno = s
      End Sub
      Public Sub showDetails()
        Console.Writeline("Employee Name "  & name)
        Console.Writeline("Employee Number " & staffno)
        Console.ReadKey()
      End Sub
    End Class
    class partTime : inherits employee
      Private ReadOnly fullTimeStaff = false
      Private hoursWorked As Integer
      Public Sub New (ByVal n As String, ByVal s As Integer, ByVal h As Integer)
        MyBase.new (n, s)
        hoursWorked = h
      End Sub
      Public Function getHoursWorked () As Integer
        Return (hoursWorked)
      End Function
    End Class
    class fullTime : inherits employee
      Private ReadOnly fullTimeStaff = true
      Private yearlySalary As Decimal
      Public Sub New (ByVal n As String, ByVal s As Integer, ByVal y As Decimal)
        MyBase.new (n, s)
        yearlySalary = y
      End Sub
     Public Function getYearlySalary () As Decimal
        Return (yearlySalary)
      End Function
    End Class
End Module
```

base class employee

derived class partTime

derived class fullTime

Java

Java

```
// Java employee OOP program with inheritance
class employee {
    private String name;
    private int staffno;
    private boolean fullTimeStaff;
    employee(String n, int s){
      name = n;
      staffno = s;
      }
    public void showDetails (){
      System.out.println("Employee Name " + name);
      System.out.println("Employee Number " + staffno);
      }
}
class partTime extends employee {
    private boolean fullTimeStaff = false;
    private int hoursWorked;
    partTime (String n, int s, int h){
      super (n, s);
      hoursWorked = h;
    }
    public int getHoursWorked () {
    return hoursWorked;
    }
}
 class fullTime extends employee {
    private boolean fullTimeStaff = true;
    private double yearlySalary;
    fullTime (String n, int s, double y){
      super (n, s);
     yearlySalary = y;
    }
    public double getYearlySalary () {
    return yearlySalary;
    }
}
```

base class employee

derived class partTime

derived class fullTime

```
public class MainInherit{
    public static void main(String[] args) {
      fullTime permanentStaff = new fullTime("Eric Jones", 72, 50000.00);
      permanentStaff.showDetails();
      partTime temporaryStaff = new partTime("Alice Hu", 1017, 45);
      temporaryStaff.showDetails();
    }
}
```

Figure 20.4 shows the inheritance diagram for the base class `employee` and the derived classes `partTime` and `fullTime`.

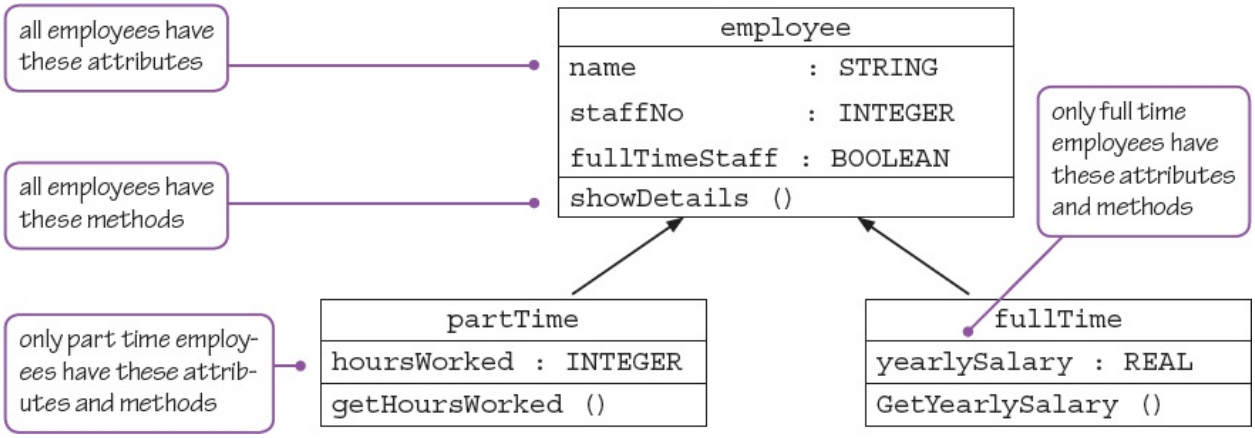


Figure 20.4 Inheritance diagram for `employee`, `partTime` and `fullTime`

ACTIVITY 20D

Write a short program to declare a class, `student`, with the private attributes `name`, `dateOfBirth` and `examMark`, and the public method `displayExamMark`.

Declare the derived classes `fullTimeStudent` and `partTimeStudent`.

Declare objects for each derived class, with a name and exam mark of your choice, and use your method to display the exam marks for these students.

Polymorphism and overloading

Polymorphism is when methods are redefined for derived classes. **Overloading** is when a method is defined more than once in a class so it can be used in different situations.

Example of polymorphism

A base class `shape` is defined, and the derived classes `rectangle` and `circle` are defined. The method `area` is redefined for both the `rectangle` class and the `circle` class. The objects `myRectangle` and `myCircle` are instanced in these programs.

Python

```
class shape:
  def __init__(self):
    self.__areaValue = 0
    self.__perimeterValue = 0
  def area(self):
    print("Area ", self.__areaValue)
  def perimeter(self):
    print("Perimeter ", self.__areaValue)
class rectangle(shape):
    def __init__(self, length, breadth):
      shape.__init__(self)
      self.__length = length
      self.__breadth = breadth
    def area (self):
      self.__areaValue = self.__length * self.__breadth
      print("Area ", self.__areaValue)
class circle(shape):
    def __init__(self, radius):
      shape.__init__(self)
      self.__radius = radius
    def area (self):
      self.__areaValue = self.__radius * self.__radius * 3.142
      print("Area ", self.__areaValue)
myCircle = circle(20)
myCircle.area()
myRectangle = rectangle (10,17)
myRectangle.area()
```

original method in shape class

redefined method in rectangle class

redefined method in circle class

VB

```
'VB shape OOP program with polymorphism
Module Module1
    Public Sub Main()
      Dim myCircle As New circle(20)
      myCircle.area()
      Dim myRectangle As New rectangle(10,17)
```

```
        myRectangle.area()
        Console.ReadKey()
    End Sub
        class shape
            Protected areaValue As Decimal
            Protected perimeterValue As Decimal
            Overridable  Sub area()
                Console.Writeline("Area "  & areaValue)
            End Sub
            Overridable Sub perimeter()
                Console.Writeline("Perimeter "  & perimeterValue)
            End Sub
        End Class
        class rectangle :    inherits shape
            Private length As Decimal
            Private breadth As Decimal
            Public Sub New (ByVal l As Decimal, ByVal b As Decimal)
                length = l
                breadth = b
            End Sub
            Overrides Sub Area ()
                areaValue = length * breadth
                Console.Writeline("Area "  & areaValue)
            End Sub
        End Class
        class circle :  inherits shape
            Private radius As Decimal
            Public Sub New (ByVal r As Decimal)
                radius = r
            End Sub
            Overrides Sub Area ()
                areaValue = radius * radius * 3.142
                Console.Writeline("Area "  & areaValue)
            End Sub
        End Class
End Module
```

original method in shape class

redefined method in rectangle class

redefined method in circle class

Java

```
// Java shape OOP program with polymorphism
class shape {                                   original method in shape class
    protected double areaValue;
    protected double perimeterValue;
    public void area (){
      System.out.println("Area " + areaValue);
      }
}
class rectangle extends shape {
    private double length;
    private double breadth;
    rectangle(double l, double b){
      length = l;
      breadth = b;
    }
    public void area (){                        redefined method in rectangle class
      areaValue = length * breadth;
      System.out.println("Area " + areaValue);
      }
}
class circle extends shape {
    private double radius;
    circle (double r){
      radius = r;
    }
    public void area (){                        redefined method in circle class
      areaValue = radius * radius * 3.142;
      System.out.println("Area " + areaValue);
      }
}
public class MainShape{
    public static void main(String[] args) {
      circle myCircle = new circle(20);
    myCircle.area();
    rectangle myRectagle = new rectangle(10, 17);
    myRectagle.area();
    }
}
```

ACTIVITY 20E

Write a short program to declare the class `shape` with the public method `area`.

Declare the derived classes `circle`, `rectangle` and `square`.

Use polymorphism to redefine the method area for these derived classes.

Declare objects for each derived class and instance them with suitable data.

Use your methods to display the areas for these shapes.

Example of overloading

One way of overloading a method is to use the method with a different number of parameters. For example, a class `greeting` is defined with the method `hello`. The object `myGreeting` is instanced and uses this method with no parameters or one parameter in this Python program. This is how Python, VB and Java manage overloading.

Python

```
class greeting:
    def hello(self, name = None):
        if name is not None:
            print ("Hello " + name)
        else:
            print ("Hello")
myGreeting = greeting()
myGreeting.hello()
myGreeting.hello("Christopher")
```

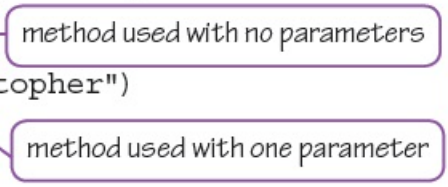
method used with no parameters

method used with one parameter

VB

```
Module Module1
    Public Sub Main()
        Dim myGreeting As New greeting
        myGreeting.hello()
        myGreeting.hello("Christopher")
        Console.ReadKey()
    End Sub
    Class greeting
      Public Overloads Sub hello()
        Console.WriteLine("Hello")
      End Sub
      Public Overloads Sub hello(ByVal name As String)
        Console.WriteLine("Hello " & name)

      End Sub
    End Class
End Module
```



Java

```
class greeting{
    public void hello(){
      System.out.println("Hello");
    }
    public void hello(String name){
      System.out.println("Hello " + name);
    }
}
class mainOverload{
    public static void main(String args[]){
        greeting myGreeting = new greeting();
        myGreeting.hello();
        myGreeting.hello("Christopher");
    }
}.
```

method used with no parameters

method used with one parameter

ACTIVITY 20F

Write a short program to declare the class `greeting`, with the public method `hello`, which can be used without a name, with one name or with a first name and last name.

Declare an object and use the method to display each type of greeting.

Containment

Containment, or **aggregation**, is the process by which one class can contain other classes. This can be presented in a class diagram.

When the class 'aeroplane' is defined, and the definition contains references to the classes – seat, fuselage, wing, cockpit – this is an example of containment.

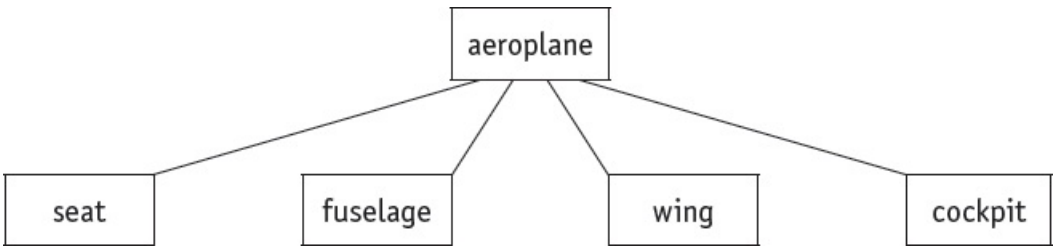


Figure 20.5

When deciding whether to use inheritance or containment, it is useful to think about how the classes used would be related in the real world.

For example

- when looking at shapes, a circle is a shape – so inheritance would be used
- when looking at the aeroplane, an aeroplane contains wings – so containment would be used.

Consider the people on board an aeroplane for a flight. The containment diagram could look like this if there can be up to 10 crew and 350 passengers on board:

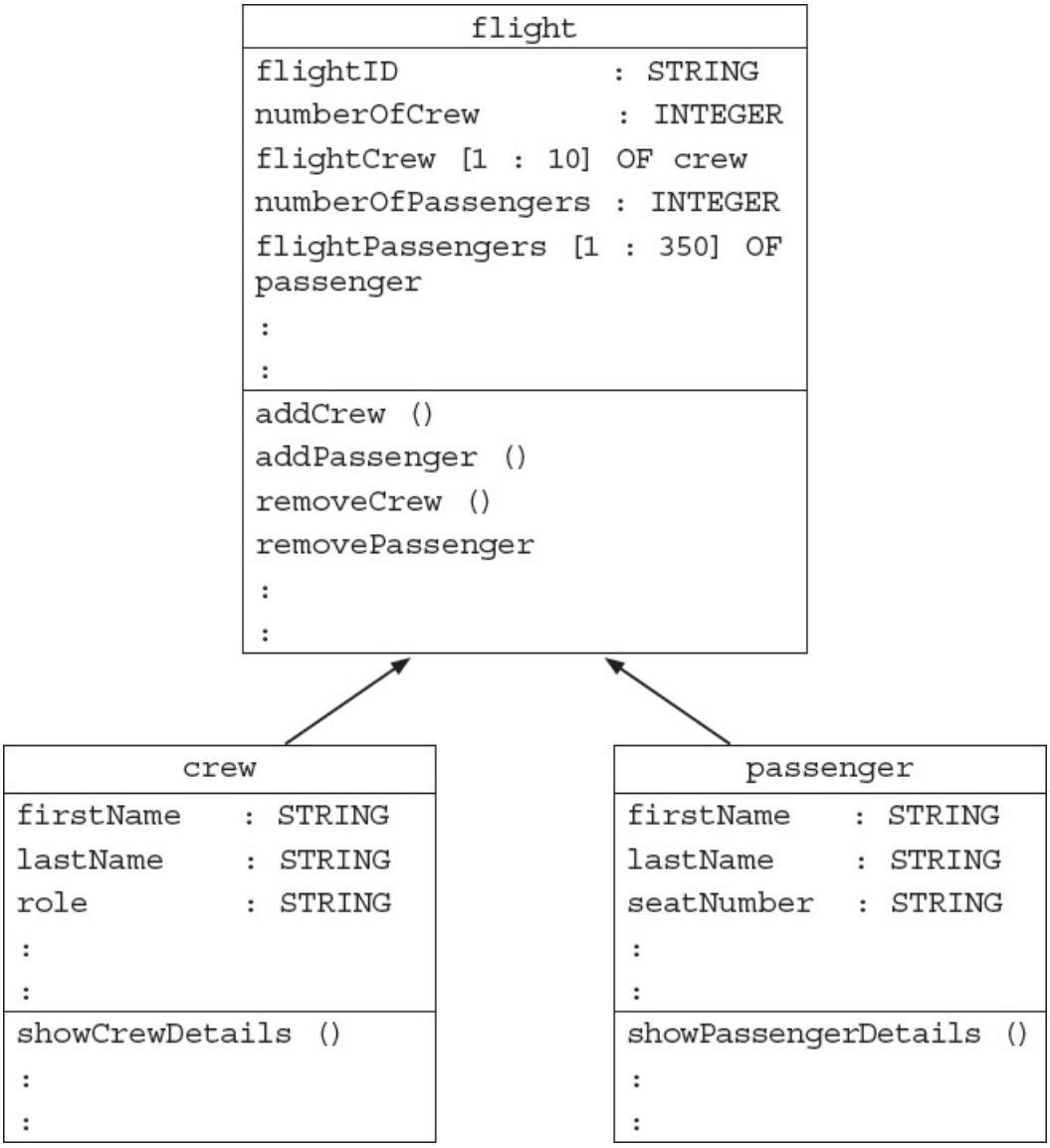


Figure 20.6

ACTIVITY 20G

Draw a containment diagram for a course at university where there are up to 50 lectures, three examinations and the final mark is the average of the marks for the three examinations.

Object methods

In OOP, the basic methods used during the life of an object can be divided into these types: **constructors**, **setters**, **getters**, and **destructors**.

A constructor is the method used to initialise a new object. Each object is initialised when a new instance is declared. When an object is initialised, memory is allocated.

For example, in the first program in Chapter 20, this is the method used to construct a new employee object.

Constructor	Language

`def __init__(self, name, staffno):`
`    self.__name = name`
`    self.__staffno = staffno`
	Python

`Public Sub New (ByVal n As String, ByVal s As Integer)`
`      name = n`
`      staffno = s`
`End Sub`
	VB

Table 20.1

Constructing an object	Language
`myStaff = employee("Eric Jones",72)`	Python
`Dim myStaff As New employee("Eric Jones", 72)`	VB
`employee myStaff = new employee("Eric Jones", 72);`	Java

Table 20.2

A setter is a method used to control changes to any variable that is declared within an object. When a variable is declared as private, only the setters declared within the object's class can be used to make changes to the variable within the object, thus making the program more robust.

For example, in the `employee` base class, this code is a setter:

Setter	Language
`def setName(self, n):` `    self.__name = n`	Python

`Public Sub setName (ByVal n As String)`
`      name = n`
`End Sub`
	VB

`public void setName(String n){`
`      this.name = n;`
`}`
	Java

Table 20.3

A getter is a method that gets the value of a property of an object.

For example, in the `partTimeStaff` derived class, this method is a getter:

Getter	Language
`def getHoursWorked (self):` `  return(self.__hoursWorked)`	Python
`Public Function getHoursWorked () As Integer` `  Return (hoursWorked)`	VB
`public int getHoursWorked () {` `  return hoursWorked;}`	Java

Table 20.4

A destructor is a method that is invoked to destroy an object. When an object is destroyed the memory is released so that it can be reused. Both Java and VB use garbage collection to automatically destroy objects that are no longer used so that the memory used can be released. In VB garbage collection can be invoked as a method if required but it is not usually needed.

For example, in any of the Python programs above, this could be used as a destructor:

```
def __del__(self)
```

Here is an example of a destructor being used in a Python program:

```
class shape:
    def __init__(self):
      self.__areaValue = 0
      self.__perimeterValue = 0
    def __del__(self):
      print("Shape deleted")
    def area(self):
      print("Area ", self.__areaValue)
  def perimeter(self):
    print("Perimeter ", self.__areaValue)
:
:
del myCircle
```

destructor

object destroyed

Here are examples of destructors in Python and VB.

Destructor	Language
	Python

`def __del __(self):` `  print ("Object deleted")`	

`Protected Overrides Sub Finalize()`
`  Console.WriteLine("Object deleted")`
`  Console.ReadKey()`
	VB – only if required, automatically called at end of program

	Java – not used

Table 20.5

Writing a program for a binary tree

In Chapter 19, we looked at the data structure and some of the operations for a binary tree using fixed length arrays in pseudocode. You will need to be able to write a program to implement a binary tree, search for an item in the binary tree and store an item in a binary tree. Binary trees are best implemented using objects, constructors, containment, functions, procedures and recursion.

- **Objects –** tree and node
- **Constructor –** adding a new node to the tree
- **Containment –** the tree contains nodes
- **Function –** search the binary tree for an item
- **Procedure –** insert a new item in the binary tree

The data structures and operations to implement a binary tree for integer values in ascending order are set out in Tables 20.6–9 below. If you are unsure how the binary tree works, review Chapter 19.

Binary tree data structure – Class node	**Language**

`class Node:`
`  def __init __(self, item):`
`  self.left = None`
`  self.right = None`
`  self.item = item`
	Python – the values for new nodes are set here. Python uses None for null pointers

	VB with a recursive definition of node to allow for a tree of any size

Public Class Node
  Public item As Integer
  Public left As Node
  Public right As Node
  Public Function GetNodeItem()
  Return item
  End Function
End Class	

class Node
{
  int item;
  Node left;
  Node right;
  GetNodeItem(int item)
  {
  this.item = item;
  }
}
	Java with a recursive definition of node to allow for a tree of any size

Table 20.6

Binary tree data structure – Class tree	**Language**
tree = Node(27)	Python – the root of the tree is set as an instance of Node

Public Class BinaryTree
  Public root As Node
  Public Sub New()
  root = Nothing
  End Sub
End Class
	VB uses Nothing for null pointers

	Java uses null for null pointers

```
class BinaryTree
{
    Node root;
    BinaryTree(int item)
    {
      this.item = item;
    }
}
```

Table 20.7

Add integer to binary tree	Language

`def insert(self, item):`
`  if self.item:`
`  if item < self.item:`
`  if self.left is None:`
`  self.left = Node(item)`
`  else:`
`  self.left.insert(item)`
`  elif item > self.item:`
`  if self.right is None:`
`  self.right = Node(item)`
`  else:`
`  self.right.insert(item)`
`  else:`
`  self.item = item`
	Python showing a recursive procedure to insert a new node and the pointers to it

	VB showing a recursive procedure to insert a new node

```
Public Sub insert(ByVal item As Integer)
    Dim newNode As New Node()
    if root Is Nothing Then
        root = newNode
    Else
        Dim CurrentNode As Node = root
        If item < current.item Then
            If current.left Is Nothing Then
                current.left = Node(item)
            Else
                current.left.insert(item)
            End If
        Else If
        If item > current.item Then
            If current.right Is Nothing Then
                current.right = Node(item)
            Else
                current.right.insert(item)
            End If
        Else If
            current.item = item
        End If
    End If
End Sub
```

Java showing a recursive procedure to insert a new node

```
void insert(tree node, int item)
{
    if (item < node.item)
    {
      if (node.left != null)
        insert(node.left, item);
      else
        node.left = new tree(item);
    }
    else if (item > node.item)
    {
      if (node.right != null)
        insert(node.right, item);
      else
        node.right = new tree(item);
    }
}
```

Table 20.8

Search for integer in binary tree	Language

`def search(self, item):`
`  while self.item != item:`
`  if item < self.item:`
`  self.item = self.left`
`  else:`
`  self.item = self.right`
`  if self.item is None:`
`  return None`
`  return self.item`
	Python – the function returns the value searched for if it is found, otherwise it returns `None`

	VB – the function returns the value searched for if it is found, otherwise it returns `Nothing`

`Public Function search(ByVal item As Integer) As Integer`
`  Dim current As Node = root`
`  While current.item <> item`
`  If item < current.item Then`
`  current = current.left`
`  Else`
`  current = current.right`
`  End If`
`  If current Is Nothing Then`
`  Return Nothing`
`  End If`
`  End While`
`  Return current.item`
`End Function`	

`tree search(int item, tree node)`
`{`
`  while (item <> node.item)`
`  {`
`  if(item < node.item)`
`  node = node.left;`
`  else`
`  node = node.right;`
`  if (node = null)`
`  return null;`
`  }`
`  return node;`
`}`
	Java – the function returns the value searched for if it is found, otherwise it returns `null`

Table 20.9

ACTIVITY 20H

In your chosen programming language, write a program using objects and recursion to implement a binary tree. Test your program by setting the root of the tree to 27, then adding the integers 19, 36, 42 and 16 in that order.

EXTENSION ACTIVITY 20B

Complete a pre-order and post-order traverse of your binary tree and print the results.

20.1.4 Declarative programming

Declarative programming is used to extract knowledge by the use of queries from a situation with known facts and rules. In Chapter 8, Section 8.3 we looked at the use of SQL scripts to query relational databases. It can be argued that SQL uses declarative programming. Review Section 8.3 to remind yourself how SQL performs queries.

Here is an example of an SQL query from Chapter 8:

```
SELECT FirstName, SecondName
FROM Student
WHERE ClassID = '7A'
ORDER BY SecondName
```

Declarative programming uses statements of facts and rules together with a mechanism for setting goals in the form of a query. A **fact** is a 'thing' that is known, and **rules** are relationships between facts. Writing declarative programs is very different to writing imperative programs. In imperative programming, the programmer writes a list of statements in the order that they will be performed. But in declarative programming, the programmer can state the facts and rules in any order before writing the query.

Prolog is a declarative programming language that uses predicate logic to write facts and rules. For example, the fact that France is a country would be written in predicate logic as:

```
country(france).
```

Note that all facts in Prolog use lower-case letters and end with a full stop.

Another fact about France – the language spoken in France is French – could be written as:

```
language(france,french).
```

A set of facts could look like this:

```
country(france).
country(germany).
country(japan).
country(newZealand).
country(england).
country(switzerland).
language(france,french).
language(germany,german).
language(japan,japanese).
language(newZealand,english).
language(england,english).
language(switzerland,french).
language(switzerland,german).
language(switzerland,italian).
```

These facts are used to set up a knowledge base. This knowledge base can be consulted using queries.

For example, a query about countries that speak a certain language, English, could look like this:

```
language(Country,english)
```

Note that a variable in Prolog – `Country`, in this example – begins with an uppercase-letter.

This would give the following results:

```
newZealand ;
england.
```

The results are usually shown in the order the facts are stored in the knowledge base.

A query about the languages spoken in a country, Switzerland, could look like this:

```
language(switzerland,Language).
```

And these are the results:

```
french, german, italian.
```

When a query is written as a statement, this statement is called a goal and the result is found when the goal is satisfied using the facts and rules available.

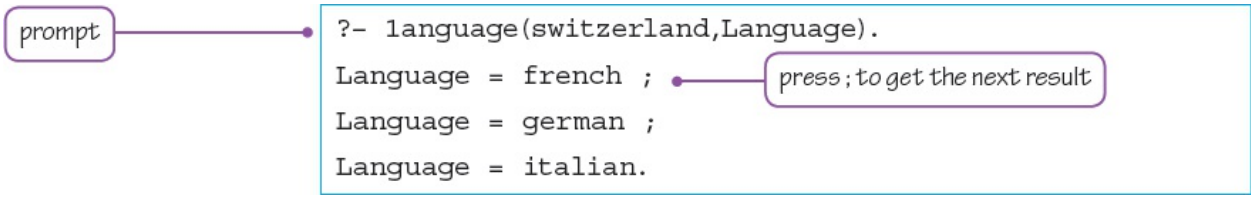
ACTIVITY 20I

Use the facts above to write queries to find out which language is spoken in England and which country speaks Japanese. Take care with the use of capital letters.

EXTENSION ACTIVITY 20C

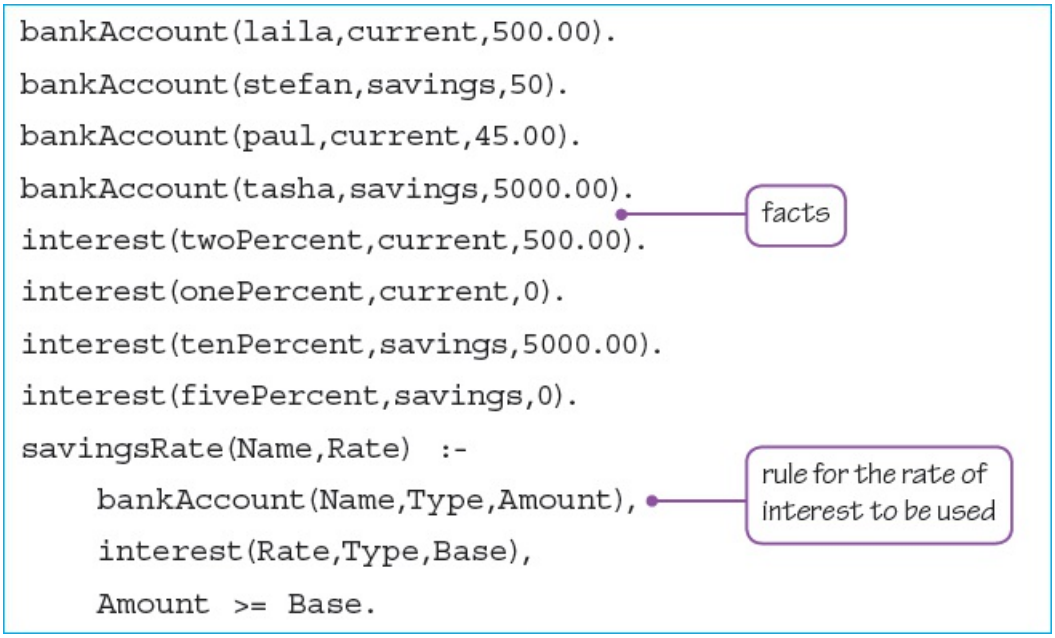
Download SWI-Prolog and write a short program to provide facts about other countries and languages and save the file. Then consult the file to find out which languages are spoken in some of the countries. Note that SWI-prolog is available as a free download.

The results for the country Switzerland query would look like this in SWI-Prolog:



Most knowledge bases also make use of rules, which are also written using predicate logic.

Here is a knowledge base for the interest paid on bank accounts. The facts about each account include the name of the account holder, the type of account (current or savings), and the amount of money in the account. The facts about the interest rates are the percentage rate, the type of account and the amount of money needed in the account for that interest rate to be paid.



Here is an example of a query using the above rule:

```
savingsRate(stefan,X).
```

And here is the result:

```
fivePercent
```

Here are examples of queries to find bank account details:

```
bankAccount(laila,X,Y).      bankAccount(victor,X,Y)
```

And here are the results:

```
current,  500.0      false
```

ACTIVITY 20J

Carry out the following activities using the information above.

1 Write a query to find out the interest rate for Laila's bank account.

2 Write a query to find who has savings accounts.

3 **a)** Set up a savings account for Robert with 300.00.

b) Set up a new fact about savings accounts allowing for an interest rate of 7% if there is 2000.00 or more in a savings account.

EXTENSION ACTIVITY 20D

Use SWI-Prolog to check your answers to the previous activity.

ACTIVITY 20K

1 Explain the difference between the four modes of addressing in a low-level programming language. Illustrate your answer with assembly language code for each mode of addressing.

2 Compare and contrast the use of imperative (procedural) programming with OOP. Use the shape programs you developed in Activities 20B and 20E to illustrate your answer with examples to show the difference in the paradigms.

3 Use the knowledge base below to answer the following questions:

```
language(fortran,highLevel).
language(cobol,highLevel).
language(visualBasic,highLevel).
language(visualBasic,oop).
language(python,highLevel).
language(python,oop).
language(assembly,lowLevel).
language(masm,lowLevel).

translator(assembler,lowLevel).
translator(compiler,highLevel).
```

```
teaching(X):-
    language(X,oop),
    language(X,highLevel).
```

a) Write **two** new facts about Java, showing that it is a high-level language and uses OOP.

b) Show the results from these queries

- **i)** teaching(X).
- **ii)** teaching(masm).

c) Write a query to show all programming languages translated by an assembler.

20.2 File processing and exception handling

WHAT YOU SHOULD ALREADY KNOW

In Chapter 10, Section 10.3, you learnt about text files, and in Chapter 13, Section 13.2, you learnt about file organisation and access. Review these sections, then try these three questions before you read the second part of this chapter.

1 a) Write a program to set up a text file to store records like this, with one record on every line.

```
TYPE
TstudentRecord
    DECLARE name : STRING
    DECLARE address : STRING
    DECLARE className : STRING
ENDTYPE
```

b) Write a procedure to append a record.

c) Write a procedure to find and delete a record.

d) Write a procedure to output all the records.

2 Describe **three** types of file organisation

3 Describe **two** types of file access and explain which type of files each one is used for.

Key terms

Read – file access mode in which data can be read from a file.

Write – file access mode in which data can be written to a file; any existing data stored in the file will be overwritten.

Append – file access mode in which data can be added to the end of a file.

Open – file-processing operation; opens a file ready to be used in a program.

Close – file-processing operation; closes a file so it can no longer be used by a program.

Exception – an unexpected event that disrupts the execution of a program.

Exception handling – the process of responding to an exception within the program so that the program does not halt unexpectedly.

20.2.1 File processing operations

Files are frequently used to store records that include data types other than string. Also, many programs need to handle random access files so that a record can be found quickly without reading through all the preceding records.

A typical record to be stored in a file could be declared like this in pseudocode:

```
TYPE
TstudentRecord
    DECLARE name : STRING
    DECLARE registerNumber : INTEGER
    DECLARE dateOfBirth : DATE
    DECLARE fullTime : BOOLEAN
ENDTYPE
```

Storing records in a serial or sequential file

The algorithm to store records sequentially in a serial (unordered) or sequential (ordered on a key field) file is very similar to the algorithm for storing lines of text in a text file. The algorithm written in pseudocode below stores the student records sequentially in a serial file as they are input.

Note that `PUTRECORD` is the pseudocode to **write** a record to a data file and `GETRECORD` is the pseudocode to **read** a record from a data file.

```
DECLARE studentRecord : ARRAY[1:50] OF TstudentRecord
DECLARE studentFile : STRING
DECLARE counter : INTEGER
counter ← 1
studentFile ← "studentFile.dat"
OPEN studentFile FOR WRITE
REPEAT
    OUTPUT "Please enter student details"
    OUTPUT "Please enter student name"
    INPUT studentRecord.name[counter]
    IF studentRecord.name <> ""
      THEN
        OUTPUT "Please enter student's register number"
        INPUT studentRecord.registerNumber[counter]
        OUTPUT "Please enter student's date of birth"
        INPUT studentRecord.dateOfBirth[counter]
        OUTPUT "Please enter True for fulltime or
        False for part-time"
        INPUT studentRecord.fullTime[counter]
        PUTRECORD, studentRecord[counter]
        counter ← counter + 1

      ELSE
        CLOSEFILE(studentFile)
    ENDIF
UNTIL studentRecord.name = ""
OUTPUT "The file contains these records: "
OPEN studentFile FOR READ
counter ← 1
REPEAT
    GETRECORD, studentRecord[counter]
    OUTPUT studentRecord[counter]
    counter ← counter + 1
UNTIL EOF(studentFile)
CLOSEFILE(studentFile)
```

Identifier name	Description
studentRecord	Array of records to be written to the file
studentFile	File name
counter	Counter for records

Table 20.10

If a sequential file was required, then the student records would need to be input into an array of records first, then sorted on the key field `registerNumber`, before the array of records was written to the file.

Here are programs in Python, VB and Java to write a single record to a file.

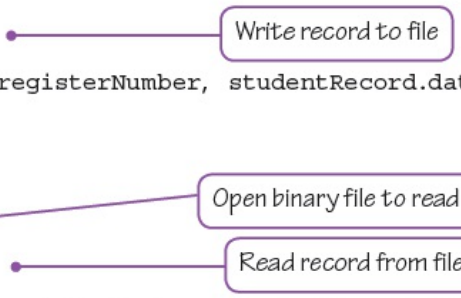
Python

```
import pickle
class student:
    def __init__(self):
      self.name = ""
      self.registerNumber = 0
      self.dateOfBirth = datetime.datetime.now()
      self.fullTime = True
studentRecord = student()
studentFile = open('students.DAT','w+b')
print("Please enter student details")
studentRecord.name = input("Please enter student name  ")
studentRecord.registerNumber = int(input("Please enter student's register number  "))
year = int(input("Please enter student's year of birth YYYY "))
month = int(input("Please enter student's month of birth MM "))
day = int(input("Please enter student's day of birth DD "))
```

Library to use binary files

Create a binary file to store the data

```
studentRecord.dateOfBirth = datetime.datetime(year, month, day)
studentRecord.fullTime = bool(input("Please enter True for full-time or False for
part-time  "))
pickle.dump (studentRecord, studentFile)
print(studentRecord.name, studentRecord.registerNumber, studentRecord.dateOfBirth,
  studentRecord.fullTime)
studentFile.close()
studentFile = open('students.DAT','rb')
studentRecord = pickle.load(studentFile)
print(studentRecord.name, studentRecord.registerNumber, studentRecord.dateOfBirth,
  studentRecord.fullTime)
studentFile.close()
```



VB

```
Option Explicit On
Imports System.IO
Module Module1
    Public Sub Main()
      Dim studentFileWriter As BinaryWriter
      Dim studentFileReader As BinaryReader
      Dim studentFile As FileStream
      Dim year, month, day As Integer
      Dim studentRecord As New student()
      studentFile = New FileStream("studentFile.DAT", FileMode.Create)
      studentFileWriter = New BinaryWriter(studentFile)
    Console.Write("Please enter student name ")
      studentRecord.name = Console.ReadLine()
      Console.Write("Please enter student's register number ")
      studentRecord.registerNumber = Integer.Parse(Console.ReadLine())
      Console.Write("Please enter student's year of birth YYYY ")
      year =Integer.Parse(Console.ReadLine())
      Console.Write("Please enter student's month of birth MM ")
      month =Integer.Parse(Console.ReadLine())
      Console.Write("Please enter student's day of birth DD ")
      day =Integer.Parse(Console.ReadLine())
      studentRecord.dateOfBirth = DateSerial(year, month, day)
      Console.Write("Please enter True for full-time or False for part-time ")
      studentRecord.fullTime = Boolean.Parse(Console.ReadLine())
```

Library to use Input and Output

Create a file to store the data

```
        studentFileWriter.Write(studentRecord.name)
        studentFileWriter.Write(studentRecord.registerNumber)
        studentFileWriter.Write(studentRecord.dateOfBirth)
        studentFileWriter.Write(studentRecord.fullTime)
        studentFileWriter.Close()
        studentFile.Close()
        studentFile = New FileStream("studentFile.DAT", FileMode.Open)
        studentFileReader = New BinaryReader(studentFile)
        studentRecord.name = studentFileReader.ReadString()
        studentRecord.registerNumber = studentFileReader.ReadInt32()
        studentRecord.dateOfBirth = studentFileReader.ReadString()
        studentRecord.fullTime = studentFileReader.ReadBoolean()
        studentFileReader.Close()
        studentFile.Close()
        Console.WriteLine (studentRecord.name & " " & studentRecord.registerNumber
         & " " & studentRecord.dateOfBirth & " " & studentRecord.fullTime)
        Console.ReadKey ()
    End Sub
    class student:
        Public name As String
        Public registerNumber As Integer
        Public dateOfBirth As Date
        Public fullTime As Boolean
    End Class
End Module
```

Write record to file

Open file to read from

Read record from file

Java

(Java programs using files need to include exception handling – see Section 20.2.2 later in this chapter.)

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;
import java.util.Date;
import java.text.SimpleDateFormat;
class Student {
    private String name;
    private int registerNumber;
    private Date dateOfBirth;
```

```
    private boolean fullTime;
    Student(String name, int registerNumber, Date dateOfBirth, boolean fullTime) {
        this.name = name;
        this.registerNumber = registerNumber;
        this.dateOfBirth = dateOfBirth;
        this.fullTime = fullTime;
    }
    public String toString() {
        return name + " " + registerNumber + " " + dateOfBirth + " " + fullTime;
    }
}
public class StudentRecordFile {
    public static void main(String[] args) throws Exception {
        Scanner input = new Scanner(System.in);
        System.out.println("Please Student details");
        System.out.println("Please enter Student name ");
        String nameIn = input.next();
        System.out.println("Please enter Student's register number ");
        int registerNumberIn = input.nextInt();
        System.out.println("Please enter Student's date of birth as YYYY-MM-DD ");
        String DOBIn = input.next();
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
        Date dateOfBirthIn = format.parse(DOBIn);
        System.out.println("Please enter true for full-time or false for part-time ");
        boolean fullTimeIn = input.nextBoolean();
        Student studentRecord = new Student(nameIn, registerNumberIn, dateOfBirthIn,
fullTimeIn);
        System.out.println(studentRecord.toString());
        // This is the file that we are going to write to and then read from
        File studentFile = new File("Student.txt");
        // Write the record to the student file
        // Note - this try-with-resources syntax only works with Java 7 and later
        try (FileWriter studentFileWriter = new FileWriter(studentFile)) {
            studentFileWriter.write(studentRecord.toString());
        }
        // Print all the lines of text in the student file
        try (Scanner studentReader = new Scanner(studentFile)) {
```

```
            while (studentReader.hasNextLine()) {
                String data = studentReader.nextLine();
                System.out.println(data);
            }
        }
    }
}
```

ACTIVITY 20L

In the programming language of your choice, write a program to

- input a student record and save it to a new serial file
- read a student record from that file
- extend your program to work for more than one record.

EXTENSION ACTIVITY 20E

In the programming language of your choice, extend your program to sort the records on `registerNumber` before storing in the file.

Adding a record to a sequential file

Records can be appended to the end of a serial file by opening the file in append mode. If records need to be added to a sequential file, then the whole file needs to be recreated and the record stored in the correct place.

The algorithm written in pseudocode below inserts a student record into the correct place in a sequential file.

```
DECLARE studentRecord : TstudentRecord
DECLARE newStudentRecord : TstudentRecord
DECLARE studentFile : STRING
DECLARE newStudentFile : STRING
DECLARE recordAddedFlag : BOOLEAN
recordAddedFlag ← FALSE
studentFile ← "studentFile.dat"
newStudentFile ← "newStudentFile.dat"
CREATE newStudentFile  // creates a new file to write to
OPEN newStudentFile FOR WRITE
OPEN studentFile FOR READ
OUTPUT "Please enter student details"
OUTPUT "Please enter student name"
INPUT newStudentRecord.name
OUTPUT "Please enter student's register number"
```

```
INPUT newStudentRecord.registerNumber
OUTPUT "Please enter student's date of birth"
INPUT newStudentRecord.dateOfBirth
OUTPUT "Please enter True for full-time or False for part-time"
INPUT newStudentRecord.fullTime
REPEAT
  WHILE NOT recordAddedFlag OR EOF(studentFile)
      GETRECORD, studentRecord  // gets record from existing file
      IF newStudentRecord.registerNumber > studentRecord.registerNumber
        THEN
          PUTRECORD studentRecord
          // writes record from existing file to new file
        ELSE
          PUTRECORD newStudentRecord
          // or writes new record to new file in the correct place
          recordAddedFlag ← TRUE
      ENDIF
  ENDWHILE
IF EOF (studentFile)
THEN
  PUTRECORD newStudentRecord
  // add new record at end of the new file
ELSE
  REPEAT
  GETRECORD, studentRecord
  PUTRECORD studentRecord
  //transfers all remaining records to the new file
  ENDIF UNTIL EOF(studentRecord)
CLOSEFILE(studentFile)
CLOSEFILE(newStudentFile)
DELETE(studentFile)
// deletes old file of student records
RENAME newStudentfile, studentfile
// renames new file to be the student record file
```

Identifier name	Description

studentRecord	record from student file
newStudentRecord	new record to be written to the file
studentFile	student file name
newStudentFile	temporary file name

Table 20.11

Note that you can directly **append** a record to the end of a file in a programming language by opening the file in append mode, as shown in the table below.

Opening a file in append mode	Language
`myFile = open("fileName", "a")`	**Opens** the file with the name fileName in append mode in Python
`myFile = New FileStream("fileName", FileMode.Append)`	Opens the file with the name fileName in append mode in VB.NET
`FileWriter myFile = new FileWriter("fileName", true);`	Opens the file with the name fileName in append mode in Java

Table 20.12

ACTIVITY 20M

In the programming language of your choice, write a program to

- put a student record and append it to the end of a sequential file
- find and output a student record from a sequential file using the key field to identify the record
- extend your program to check for record not found (if required).

EXTENSION ACTIVITY 20F

Extend your program to input a student record and save it to in the correct place in the sequential file created in Extension Activity 20E.

Adding a record to a random file

Records can be added to a random file by using a hashing function on the key field of the record to be added. The hashing function returns a pointer to the address where the record is to be added.

In pseudocode, the address in the file can be found using the command:

```
SEEK <filename>,<address>
```

The record can be stored in the file using the command:

```
PUTRECORD <filename>,<recordname>
```

Or it can be retrieved using:

```
GETRECORD <filename>,<recordname>
```

The file needs to be opened as random:

```
OPEN studentFile FOR RANDOM
```

The algorithm written in pseudocode below inserts a student record into a random file.

```
DECLARE studentRecord : TstudentRecord
DECLARE studentFile : STRING
DECLARE Address : INTEGER
studentFile ← "studentFile.dat"
OPEN studentFile FOR RANDOM
// opens file for random access both read and write
OUTPUT "Please enter student details"
OUTPUT "Please enter student name"
INPUT StudentRecord.name
OUTPUT "Please enter student's register number"
INPUT studentRecord.registerNumber
OUTPUT "Please enter student's date of birth"
INPUT studentRecord.dateOfBirth
OUTPUT "Please enter True for full-time or False for
part-time"
INPUT studentRecord.fullTime
address ← hash(studentRecord,registerNumber)
// uses function hash to find pointer to address
SEEK studentFile,address
// finds address in file
PUTRECORD studentFile,studentRecord
//writes record to the file
CLOSEFILE(studentFile)
```

EXTENSION ACTIVITY 20G

In the programming language of your choice, write a program to input a student record and save it to a random file.

Finding a record in a random file

Records can be found in a random file by using a hashing function on the key field of the record to be found. The hashing function returns a pointer to the address where the record is to be found, as shown in the example pseudocode below.

```
DECLARE studentRecord : TstudentRecord
DECLARE studentFile : STRING
DECLARE Address : INTEGER
studentFile ← "studentFile.dat"
OPEN studentFile FOR RANDOM
// opens file for random access both read and write
OUTPUT "Please enter student's register number"
INPUT studentRecord.registerNumber
address ← hash(studentRecord.registerNumber)
// uses function hash to find pointer to address
SEEK studentFile,address
// finds address in file
GETRECORD studentFile,studentRecord
//reads record from the file
OUTPUT studentRecord
CLOSEFILE(studentFile)
```

EXTENSION ACTIVITY 20H

In the programming language of your choice, write a program to find and output a student record from a random file using the key field to identify the record.

20.2.2 Exception handling

An **exception** is an unexpected event that disrupts the execution of a program. **Exception handling** is the process of responding to an exception within the program so that the program does not halt unexpectedly. Exception handling makes a program more robust as the exception routine traps the error then outputs an error message, which is followed by either an orderly shutdown of the program or recovery if possible.

An exception may occur in many different ways, for example

- dividing by zero during a calculation
- reaching the end of a file unexpectedly when trying to read a record from a file
- trying to open a file that has not been created
- losing a connection to another device, such as a printer.

Exceptions can be caused by

- programming errors
- user errors
- hardware failure.

Error handling is one of the most important aspects of writing robust programs that are to be used every day, as users frequently make errors without realising, and hardware can fail at any time. Frequently, error handling routines can take a programmer as long, or even longer, to write and test as the program to perform the task itself.

The structure for error handling can be shown in pseudocode as:

```
TRY
    <statements>
EXCEPT
    <statements>
ENDTRY
```

Here are programs in Python, VB and Java to catch an integer division by zero exception.

Python

```
def division(firstNumber, secondNumber):
    try:
      myAnswer = firstNumber // secondNumber
      print('Answer ', myAnswer)
    except:
      print('Divide by zero')
division(12, 3)
division(10, 0)
```

integer division //

VB

```
Module Module1
Public Sub Main()
    division(12, 3)
    division(10, 0)
    Console.ReadKey()
End Sub
Sub division(ByVal firstNumber As Integer, ByVal secondNumber As Integer)
    Dim myAnswer As Integer
    Try
      myAnswer = firstNumber \ secondNumber
      Console.WriteLine("Answer " & myAnswer)
    Catch e As DivideByZeroException
      Console.WriteLine("Divide by zero")
    End Try
End Sub
End Module
```

integer division \

Java

```
public class Division {
    public static void main(String[] args) {
    division(12, 3);
    division(10, 0);
    }
    public static void division(int firstNumber, int secondNumber){
      int myAnswer;
      try {
        myAnswer = firstNumber / secondNumber;
        System.out.println("Answer " +  myAnswer);
      }
      catch (ArithmeticException e){
        System.out.println("Divide by zero");
      }
    }
}
```

Automatic Integer division because there are integers on both sides of the division operator

ACTIVITY 20N

In the programming language of your choice, write a program to check that a value input is an integer.

ACTIVITY 20O

In the programming language of your choice, extend the file handling programs you wrote in Section 20.2.1 to use exception handling to ensure that the files used exist and allow for the condition unexpected end of file.

End of chapter questions

1 A declarative programming language is used to represent the following facts and rules:

```
01 male(ahmed).
02 male(raul).
03 male(ali).
04 male(philippe).
05 female(aisha).
06 female(gina).
07 female(meena).
08 parent(ahmed, raul).
09 parent(aisha, raul).
10 parent(ahmed, philippe).
11 parent(aisha, philippe).
12 parent(ahmed, gina).
13 parent(aisha, gina).
14 mother(A, B) IF female(A) AND parent(A, B).
```

These clauses have the following meaning:

Clause	Explanation
01	Ahmed is male
05	Aisha is female
08	Ahmed is a parent of Raul
14	A is the mother of B if A is female and A is a parent of B

a) More facts are to be included.
Ali and Meena are the parents of Ahmed.
Write the additional clauses to record this.

[2]

```
15 .....
16 .....
```

b) Using the variable C, the goal

```
parent(ahmed, C)
```

returns

```
C = raul, philippe, gina
```

Write the result returned by the goal

[2]

```
parent(P, gina)
P = ……
```

c) Use the variable M to write the goal to find the mother of Gina.

[1]

d) Write the rule to show that F is the father of C.

[2]

```
father(F, C)
IF……
```

e) Write the rule to show that X is a brother of Y.

[4]

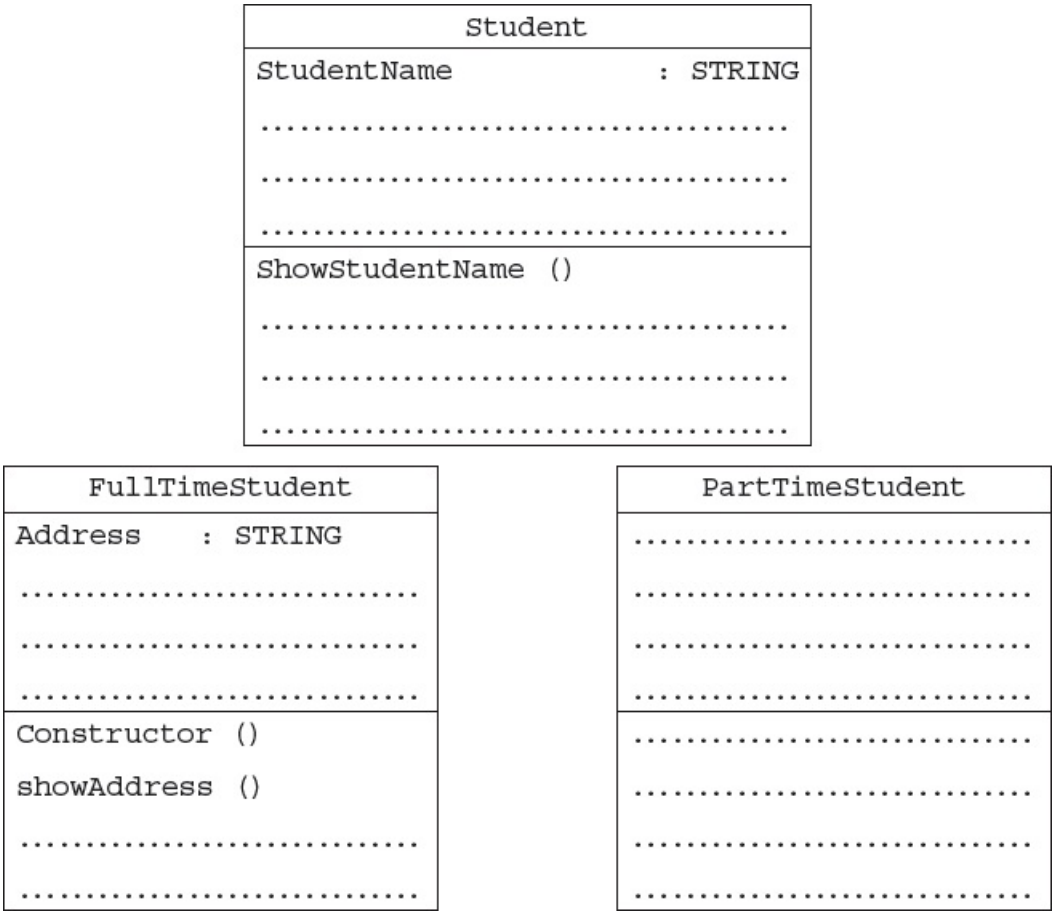
```
brother(X, Y)
IF……
```

Cambridge International AS & A Level Computer Science 9608
Paper 42 Q2 November 2015

2 A college has two types of student: full-time and part-time.
All students have their name and date of birth recorded.
A full-time student has their address and telephone number recorded.
A part-time student attends one or more courses. A fee is charged for each course. The number of courses a part-time student attends is recorded, along with the total fee and whether or not the fee has been paid.
The college needs a program to process data about its students. The program will use an object-oriented programming language.

a) Copy and complete the class diagram showing the appropriate properties and methods.

[7]



b) Write **program code**:

i) for the class definition for the superclass `Student`.

[2]

ii) for the class definition for the subclass `FullTimeStudent`.

[3]

iii) to create a new instance of `FullTimeStudent` with:
– identifier: `NewStudent`
– name: A. Nyone
– date of birth: 12/11/1990
– telephone number: 099111

[3]

Cambridge International AS & A Level Computer Science 9608
Paper 42 Q3 November 2015

3 a) When designing and writing program code, explain what is meant by:
– an exception
– exception handling.

[3]

b) A program is to be written to read a list of exam marks from an existing text file into a 1D array.

Each line of the file stores the mark for one student.
State **three** exceptions that a programmer should anticipate for this program.

[3]

c) The following pseudocode is to read two numbers.

```
01 DECLARE Num1 : INTEGER
02 DECLARE Num2 : INTEGER
03 DECLARE Answer : INTEGER
04 TRY
05 OUTPUT "First number..."
06 INPUT Num1
07 OUTPUT "Second number..."
08 INPUT Num2
09 Answer ← Num1 / (Num2 - 6)
10 OUTPUT Answer
11 EXCEPT ThisException : EXCEPTION
12 OUTPUT ThisException.Message
13 FINALLY
14 // remainder of the program follows
                    ∫

29
30 ENDTRY
```

The programmer writes the corresponding program code.
A user inputs the number 53 followed by 6. The following output is produced:

```
First number...53
Second number...6
Arithmetic operation resulted in an overflow
```

i) State the pseudocode line number which causes the exception to be raised.

[1]

ii) Explain the purpose of the pseudocode on lines 11 and 12.

[3]

Cambridge International AS & A Level Computer Science 9608
Paper 42 Q5(b)–(d) June 2016

Glossary

A* algorithm – an algorithm that finds the shortest route between nodes or vertices but uses an additional heuristic approach to achieve better performance than Dijkstra's algorithm.

Abnormal test data – test data that should be rejected by a program.

Absolute addressing – mode of addressing in which the contents of the memory location in the operand are used.

Abstract data type (ADT) – a collection of data and a set of operations on that data.

Abstraction – the process of extracting information that is essential, while ignoring what is not relevant, for the provision of a solution.

Acceptance testing – the testing of a completed program to prove to the customer that it works as required.

Access rights (data security) – use of access levels to ensure only authorised users can gain access to certain data.

Access rights (database) – the permissions given to database users to access, modify or delete data.

Accumulator – temporary general purpose register which stores numerical values at any part of a given operation.

Acknowledgement – message sent to a receiver to indicate that data has been received without error.

ACM – Association for Computing Machinery.

Adaptive maintenance – the alteration of a program to perform new tasks.

Address bus – carries the addresses throughout the computer system.

Addressing modes – different methods of using the operand part of a machine code instruction as a memory address.

Algorithm – an ordered set of steps to be followed in the completion of a task.

Alpha testing – the testing of a completed or nearly completed program in-house by the development team.

Analogue to digital converter (ADC) – needed to convert analogue data (read from sensors, for example) into a form understood by a computer.

Analysis – part of the program development lifecycle; a process of investigation, leading to the specification of what a program is required to do.

Anti-spyware software – software that detects and removes spyware programs installed illegally on a user's computer system.

Antivirus software – software that quarantines and deletes files or programs infected by a virus (or other malware); it can be run in the background or initiated by the user.

Append – file access mode in which data can be added to the end of a file.

Argument – the value passed to a procedure or function.

Arithmetic shift – the sign of the number is preserved.

Arithmetic-logic unit (ALU) – component in the processor which carries out all arithmetic and

logical operations.

ARPAnet – Advanced Research Projects Agency Network.

Array – a data structure containing several elements of the same data type.

Artificial intelligence (AI) – machine or application which carries out a task that requires some degree of intelligence when carried out by a human counterpart.

Artificial neural networks – networks of interconnected nodes based on the interconnections between neurons in the human brain; the system is able to think like a human using these neural networks, and its performance improves with more data.

ASCII code – coding system for all the characters on a keyboard and control codes.

Assembler – a computer program that translates programming code written in assembly language into machine code; assemblers can be one pass or two pass.

Assembly language – a low-level chip/machine specific programming language that uses mnemonics.

Asymmetric encryption – encryption that uses public keys (known to everyone) and private keys (secret keys).

Asynchronous serial data transmission – serial refers to a single wire being used to transmit bits of data one after the other; asynchronous refers to a sender using its own clock/timer device rather sharing the same clock/timer with the recipient device.

Attribute (database) – an individual data item stored for an entity; for example, for a person, attributes could include name, address, date of birth.

Attributes (class) – the data items in a class.

Audio compression – method used to reduce the size of a sound file using perceptual music shaping.

Authentication – a way of proving somebody or something is who or what they claim to be.

Automatic repeat request (ARQ) – a type of verification check.

Back propagation – method used in artificial neural networks to calculate error gradients so that actual node/neuron weightings can be adjusted to improve the performance of the model.

Back-up utility – software that makes copies of files on another portable storage device.

Backus-Naur form (BNF) notation – a formal method of defining the grammatical rules of a programming language.

Bad sector – a faulty sector on an HDD which can be soft or hard.

Base case – a terminating solution to a process that is not recursive.

BCS – British Computer Society.

Belady's anomaly – phenomenon which means it is possible to have more page faults when increasing the number of page frames.

Beta testing – the testing of a completed program by a small group of users before it is released.

Bidirectional – used to describe a bus in which bits can travel in both directions.

Big O notation – a mathematical notation used to describe the performance or complexity of an algorithm.

Binary – base two number system based on the values 0 and 1 only.

Binary coded decimal (BCD) – number system that uses 4 bits to represent each denary digit.

Binary file – a file that does not contain text only; the file is machine-readable but not human-

readable.

Binary floating-point number – a binary number written in the form $M \times 2^E$ (where M is the mantissa and E is the exponent).

Binary search – a method of searching an ordered list by testing the value of the middle item in the list and rejecting the half of the list that does not contain the required value.

Binary tree – a hierarchical data structure in which each parent node can have a maximum of two child nodes.

Binder 3D printing – 3D printing method that uses a two-stage pass; the first stage uses dry powder and the second stage uses a binding agent.

Biometrics – use of unique human characteristics to identify a user (such as fingerprints or face recognition).

BIOS – basic input/output system.

Birefringence – a reading problem with DVDs caused by refraction of laser light into two beams.

Bit – abbreviation for binary digit.

Bit depth – number of bits used to represent the smallest unit in, for example, a sound or image file; the larger the bit depth, the better the quality of the sound or colour image.

Bit rate – number of bits per second that can be transmitted over a network; it is a measure of the data transfer rate over a digital telecoms network.

Bit streaming – contiguous sequence of digital bits sent over a network/internet.

Bit-map image – system that uses pixels to make up an image.

BitTorrent – protocol used in peer-to-peer networks when sharing files between peers.

Black-box testing – a method of testing a program that tests a module's inputs and outputs.

Block chaining – form of encryption, in which the previous block of ciphertext is XORed with the block of plaintext and then encrypted thus preventing identical plaintext blocks producing identical ciphertext.

Block cipher – the encryption of a number of contiguous bits in one go rather than one bit at a time.

Bluetooth – wireless connectivity that uses radio waves in the 2.45 GHz frequency band.

Boolean algebra – a form of algebra linked to logic circuits and based on TRUE and FALSE.

Bootstrap – a small program that is used to load other programs to 'start up' a computer.

Boundary test data – test data that is on the limit of that accepted by a program or data that is just outside the limit of that rejected by a program.

Breakpoint – a deliberate pause in the execution of a program during testing so that the contents of variables, registers, and so on can be inspected to aid debugging.

Bridge – device that connects LANs which use the same protocols.

Broadcast – communication where pieces of data are sent from sender to receiver.

Bubble sort – a method of sorting data in an array into alphabetical or numerical order by comparing adjacent items and swapping them if they are in the wrong order.

Buffering – store which holds data temporarily.

Burst time – the time when a process has control of the CPU.

Bus network topology – network using single central cable in which all devices are connected

to this cable; data can only travel in one direction and only one device is allowed to transmit at a time.

By reference – a method of passing a parameter to a procedure in which the value of the variable can be changed by the procedure.

By value – a method of passing a parameter to a procedure in which the value of the variable cannot be changed by the procedure.

Cache memory – a high speed auxiliary memory which permits high speed data transfer and retrieval.

Candidate key – an attribute or smallest set of attributes in a table where no tuple has the same value.

Capacitive – type of touch screen technology based on glass layers forming a capacitor; fingers touching the screen cause a change in the electric field.

Certificate authority (CA) – commercial organisation used to generate a digital certificate requested by website owners or individuals.

Character set – a list of characters that have been defined by computer hardware and software; it is necessary to have a method of coding, so that the computer can understand human characters.

Chatbot – computer program set up to simulate conversational interaction between humans and a website.

Check digit – additional digit appended to a number to check if entered data is error-free.

Checksum – verification method used to check if data transferred has been altered or corrupted; calculated from the block of data to be sent.

Ciphertext – the product when plaintext is put through an encryption algorithm.

Circuit switching – method of transmission in which a dedicated circuit/channel lasts throughout the duration of the communication.

CISC – complex instruction set computer.

Class – a template defining the methods and data of a certain type of object.

Classless inter-domain routing (CIDR) – increases IPv4 flexibility by adding a suffix to the IP address, such as 200.21.100.6/18.

CLI – command line interface.

Client-server – network that uses separate dedicated servers and specific client work stations; all client computers are connected to the dedicated servers.

Clock cycle – clock speeds are measured in terms of GHz; this is the vibrational frequency of the clock which sends out pulses along the control bus; a 3.5 GHZ clock cycle means 3.5 billion clock cycles a second.

Close – file-processing operation; closes a file so it can no longer be used by a program.

Cloud storage – method of data storage where data is stored on off-site servers.

Cluster – a number of computers (containing SIMD processors) networked together.

CMOS – complementary metal-oxide semiconductor.

Coaxial cable – cable made up of central copper core, insulation, copper mesh and outer insulation.

Code generation – the third stage in the process of compilation; this stage produces an object

program.

Coding – part of the program development lifecycle; the writing of the program or suite of programs.

Collision – situation in which two messages/data from different sources are trying to transmit along the same data channel.

Colour depth – number of bits used to represent the colours in a pixel, e.g. 8 bit colour depth can represent $2^8 = 256$ Colours.

Combination circuit – circuit in which the output depends entirely on the input values.

Compiler – a computer program that translates a source program written in a high-level language to machine code or p-code, object code.

Composite data type – a data type constructed using several of the basic data types available in a particular programming language.

Composite key – a set of attributes that form a primary key to provide a unique identifier for a table.

Conflict – situation in which two devices have the same IP address.

Constant – a named value that cannot change during the execution of a program.

Constructor – a method used to initialise a new object.

Containment (aggregation) – process by which one class can contain other classes.

Context switching – procedure by which, when the next process takes control of the CPU, its previous state is reinstated or restored.

Contiguous – items next to each other.

Control – to automatically take readings from a device, then use the data from those readings to adjust the device.

Control bus – carries signals from control unit to all other computer components.

Control unit – ensures synchronisation of data flow and programs throughout the computer by sending out control signals along the control bus.

Core – a unit made up of ALU, control unit and registers which is part of a CPU; a CPU may contain a number of cores.

Corrective maintenance – the correction of any errors that appear during use.

Cross-coupling – interconnection between two logic gates which make up a flip-flop.

CSMA/CD – carrier sense multiple access with collision detection; a method used to detect collisions and resolve the issue.

Culture – the attitudes, values and practices shared by a group of people/society.

Current instruction register (CIR) – this is a register used to contain the instruction which is currently being executed or decoded.

Cyclic shift – no bits are lost; bits shifted out of one end of the register are introduced at the other end of the register.

Data bus – allows data to be carried from processor to memory (and vice versa) or to and from input/output devices.

Data definition language (DDL) – a language used to create, modify and remove the data structures that form a database.

Data dictionary – a set of data that contains metadata (data about other data) for a database.

Data hiding – technique which protects the integrity of an object by restricting access to the data and methods within that object.

Data integrity – the accuracy, completeness and consistency of data.

Data management – the organisation and maintenance of data in a database to provide the information required.

Data manipulation language (DML) – a language used to add, modify, delete and retrieve the data stored in a relational database.

Data modelling – the analysis and definition of the data structures required in a database and to produce a data model.

Data privacy – the privacy of personal information, or other information stored on a computer, that should not be accessed by unauthorised parties.

Data protection laws – laws which govern how data should be kept private and secure.

Data redundancy – situation in which the same data is stored on several servers in case of maintenance or repair.

Data security – methods taken to prevent unauthorised access to data and to recover data if lost or corrupted.

Data type – a classification attributed to an item of data, which determines the types of value it can take and how it can be used.

Database – a structured collection of items of data that can be accessed by different applications programs.

Database management system (DBMS) – systems software for the definition, creation and manipulation of a database.

Debugging – the process of finding logic errors in a computer program by running or tracing the program.

Declarative programming – statements of facts and rules together with a mechanism for setting goals in the form of a query.

Decomposition – the process of breaking a complex problem into smaller parts.

Deep learning – machines that think in a way similar to the human brain; they handle huge amounts of data using artificial neural networks.

Design – part of the program development lifecycle; it uses the program specification from the analysis stage to show how the program should be developed.

Destructor – a method that is automatically invoked when an object is destroyed.

Developer interface – feature of a DBMS that provides developers with the commands required for definition, creation and manipulation of a database.

Device driver – software that communicates with the operating system and translates data into a format understood by the device.

Dictionary – an abstract data type that consists of pairs, a key and a value, in which the key is used to find the value.

Digest –a fixed-size numeric representation of the contents of a message produced from a hashing algorithm; this can be encrypted to form a digital signature.

Digital certificate – an electronic document used to prove the identity of a website or individual; it contains a public key and information identifying the website owner or individual; issued by a CA.

Digital rights management (DRM) – used to control the access to copyrighted material.

Digital signature – electronic way of validating the authenticity of digital documents (that is, making sure they have not been tampered with during transmission) and also proof that a document was sent by a known user.

Digital to analogue converter (DAC) – needed to convert digital data into electric currents that can drive motors, actuators and relays, for example.

Dijkstra's algorithm – an algorithm that finds the shortest path between two nodes or vertices in a graph/network.

Direct 3D printing – 3D printing technique where print head moves in the *x*, *y* and *z* directions. Layers of melted material are built up using nozzles like an inkjet printer.

Direct access – a method of file access in which a record can be physically found in a file without physically reading other records.

Direct addressing – mode of addressing in which the contents of the memory location in the operand are used; same as absolute addressing.

Direct memory access (DMA) controller – device that allows certain hardware to access RAM independently of the CPU.

Dirty – term used to describe a page in memory that has been modified.

Disk compression – software that compresses data before storage on an HDD.

Disk content analysis software – utility that checks disk drives for empty space and disk usage by reviewing files and folders.

Disk defragmenter – utility that reorganises the sectors on a hard disk so that files can be stored in contiguous data blocks.

Disk formatter – utility that prepares a disk to allow data/files to be stored and retrieved.

Disk thrashing – problem resulting from use of virtual memory; excessive swapping in and out of virtual memory leads to a high rate of hard disk read/write head movements thus reducing processing speed.

DNS cache poisoning – altering IP addresses on a DNS server by a 'pharmer' or hacker with the intention of redirecting a user to their fake website.

Domain name service (DNS) – (also known as domain name system) gives domain names for internet hosts and is a system for finding IP addresses of a domain name.

Dry run – a method of testing a program that involves working through a program or module from a program manually.

Dual core – a CPU containing two cores.

Dual layering – used in DVDs; uses two recording layers.

Dynamic link file (DLL) – a library routine that can be linked to another program only at the run time stage.

Dynamic RAM (DRAM) – type of RAM chip that needs to be constantly refreshed.

Eavesdropper – a person who intercepts data being transmitted.

Electronically erasable programmable read-only memory (EEPROM) – a read-only (ROM) chip that can be modified by the user which can then be erased and written to repeatedly using pulsed voltages.

Emulation – the use of an app/device to imitate the behaviour of another program/device; for

example, running an OS on a computer which is not normally compatible.

Encapsulation – process of putting data and methods together as a single unit, a class.

Encryption – the use of encryption keys to make data meaningless without the correct decryption key.

Entity – anything that can have data stored about it; for example, a person, place, event, thing.

Entity-relationship (E-R) model or E-R diagram – a graphical representation of a database and the relationships between the entities.

Enumerated data type – a non-composite data type defined by a given list of all possible values that has an implied order.

Erasable PROM (EPROM) – type of ROM that can be programmed more than once using ultraviolet (UV) light.

Ethernet – protocol IEEE 802.3 used by many wired LANs.

Ethical hacking – hacking used to test the security and vulnerability of a computer system; the hacking is carried out with the permission of the computer system owner, for example, to help a company identify risks associated with malicious hacking of their computer systems.

Ethics – moral principles governing an individual's or organisation's behaviour, such as a code of conduct.

Even parity – binary number with an even number of 1-bits.

Exception – an unexpected event that disrupts the execution of a program.

Exception handling – the process of responding to an exception within the program so that the program does not halt unexpectedly.

Exponent – the power of 2 that the mantissa (fractional part) is raised to in a floating-point number.

Extreme test data – test data that is on the limit of that accepted by a program.

Fact – a 'thing' that is known.

False positive – a file or program identified by a virus checker as being infected but the user knows this cannot be correct.

Fetch-execute cycle – a cycle in which instructions and data are fetched from memory and then decoded and finally executed.

Fibre optic cable – cable made up of glass fibre wires which use pulses of light (rather than electricity) to transmit data.

Field – a column in a table in a database.

File – a collection of data stored by a computer program to be used again.

File access – the method used to physically find a record in the file.

File organisation – the way that records of data are physically stored in a file, including the structure and ordering of the records.

File server – a server on a network where central files and other data are stored; they can be accessed by a user logged onto the network.

Finite state machine (FSM) – a mathematical model of a machine that can be in one state of a fixed set of possible states; one state is changed to another by an external input; this is known as a transition.

Firewall – software or hardware that sits between a computer and external network which

monitors and filters all incoming and outgoing activities.

First in first out (FIFO) page replacement – page replacement that keeps track of all pages in memory using a queue structure; the oldest page is at the front of the queue and is the first to be removed when a new page is added.

First normal form (1NF) – the status of a relational database in which entities do not contain repeated groups of attributes.

Flag – indicates the status of a bit in the status register; for example, N = 1 indicates the result of an addition gives a negative value.

Flash memory – a type of EEPROM, particularly suited to use in drives such as SSDs, memory cards and memory sticks.

Flip-flop circuits – electronic circuits with two stable conditions using sequential circuits.

Flowchart – a diagrammatic representation of an algorithm.

Foreign key – a set of attributes in one table that refer to the primary key in another table.

Fragmented – storage of data in non-consecutive sectors; for example, due to editing and deletion of old data.

Frame rate – number of video frames that make up a video per second.

Frames – fixed-size physical memory blocks.

Free Software Foundation – organisation promoting the free distribution of software, giving users the freedom to run, copy, change or adapt the coding as needed.

Freeware – software that can be downloaded free of charge; however, it is covered by the usual copyright laws and cannot be modified; nor can the code be used for another purpose.

FTP – file transfer protocol.

Full adder circuit – two half adders combined to allow the sum of several binary bits.

Function – a set of statements that can be grouped together and easily called in a program whenever required, rather than repeating all of the statements each time. Unlike a procedure, a function always returns a value.

Gateway – device that connects LANs which use different protocols.

General case – a solution to a process that is recursively defined.

Getter – a method that gets the value of a property.

Graph – a non-linear data structure consisting of nodes and edges.

Gray codes – ordering of binary numbers such that successive numbers differ by one bit value only, for example, 00 01 11 10.

Guest OS – an OS running on a virtual machine.

GUI – graphical user interface.

Hacking – illegal access to a computer system without the owner's permission.

Half adder circuit – carries out binary addition on two bits giving sum and carry.

Handshake – the process of initiating communication between two devices; this is initiated by one device sending a message to another device requesting the exchange of data.

Hard disk drive (HDD) – type of magnetic storage device that uses spinning disks.

Hardware management – part of the operating system that controls all input/output devices connected to a computer (made up of sub-management systems such as printer management, secondary storage management, and so on).

Hashing algorithm (cryptography) – a function which converts a data string into a numeric string which is used in cryptography.

Hashing algorithm (file access) – a mathematical formula used to perform a calculation on the key field of the record; the result of the calculation gives the address where the record should be found.

HCI – human–computer interface.

Header (procedure or function) – the first statement in the definition of a procedure or function, which contains its name, any parameters passed to it, and, for a function, the type of the return value.

Header (data packet) – part of a data packet containing key data such as destination IP address, sequence number, and so on.

Heuristic – method that employs a practical solution (rather than a theoretical one) to a problem; when applied to algorithms this includes running tests and obtaining results by trial and error.

Heuristic checking – checking of software for behaviour that could indicate a possible virus.

Hexadecimal – a number system based on the value 16 (uses the denary digits 0 to 9 and the letters A to F).

High-bandwidth digital copy protection (HDCP) – part of HDMI technology which reduces risk of piracy of software and multimedia.

High-definition multimedia interface (HDMI) – type of port connecting devices to a computer.

Hop number/hopping – number in the packet header used to stop packets which never reach their destination from 'clogging up' routes.

Host – a computer or device that can communicate with other computers or devices on a network.

Host OS – an OS that controls the physical hardware.

Host-to-host – a protocol used by TCP when communicating between two devices.

HTTP – hypertext transfer protocol.

Hub – hardware used to connect together a number of devices to form a LAN; directs incoming data packets to all devices on the network (LAN).

Hybrid network – network made up of a combination of other network topologies.

HyperText Mark-up Language (HTML) – used to design web pages and to write http(s) protocols, for example.

Hypervisor – virtual machine software that creates and runs virtual machines.

Icon – small picture or symbol used to represent, for example, an application on a screen.

Identifier – a unique name applied to an item of data.

IEEE – Institute of Electrical and Electronics Engineers.

Image resolution – number of pixels that make up an image; for example, an image could contain 4096 × 3192 pixels (13 074 432 pixels in total).

IMAP – internet message access protocol.

Immediate access store (IAS) – holds all data and programs needed to be accessed by the control unit.

Immediate addressing – mode of addressing in which the value of the operand only is used.

Imperative programming – programming paradigm in which the steps required to execute a program are set out in the order they need to be carried out.

In demand paging – a form of data swapping where pages of data are not copied from HDD/SSD into RAM until they are actually required.

Index (database) – a data structure built from one or more columns in a database table to speed up searching for data.

Index (array) – a numerical indicator of an item of data's position in an array.

Indexed addressing – mode of addressing in which the contents of the memory location found by adding the contents of the index register (IR) to the address of the memory location in the operand are used.

Indirect addressing – mode of addressing in which the contents of the contents of the memory location in the operand are used.

Inheritance – process in which the methods and data from one class, a superclass or base class, are copied to another class, a derived class.

Insertion sort – a method of sorting data in an array into alphabetical or numerical order by placing each item in turn in the correct position in the sorted list.

Instance – An occurrence of an object during the execution of a program.

Instruction – a single operation performed by a CPU.

Instruction set – the complete set of machine code instructions used by a CPU.

Integrated development environment (IDE) – a suite of programs used to write and test a computer program written in a high-level programming language.

Integration testing – a method of testing a program that tests combinations of program modules that work together.

Intellectual property rights – rules governing an individual's ownership of their own creations or ideas, prohibiting the copying of, for example, software without the owner's permission.

Internet – massive network of networks, made up of computers and other electronic devices; uses TCP/IP communication protocols.

Internet protocol (IP) – uses IPv4 or IPv6 to give addresses to devices connected to the internet.

Internet service provider (ISP) – company which allows a user to connect to the internet; they will usually charge a monthly fee for the service they provide.

Interpreter – a computer program that analyses and executes a program written in a high-level language line by line.

Interrupt – signal sent from a device or software to a processor requesting its attention; the processor suspends all operations until the interrupt has been serviced.

Interrupt dispatch table (IDT) – data structure used to implement an interrupt vector table.

Interrupt priority – all interrupts are given a priority so that the processor knows which need to be serviced first and which interrupts are to be dealt with quickly.

Interrupt priority levels (IPL) – values given to interrupts based on values 0 to 31.

Interrupt service routine (ISR) or interrupt handler – software which handles interrupt requests (such as 'printer out of paper') and sends the request to the CPU for processing.

IPv4 – IP address format which uses 32 bits, such as 200.21.100.6.

IPv6 – newer IP address format which uses 128 bits, such as A8F0:7FFF:F0F1:F000:3DD0:256A:22FF:AA00.

Iterative model – a type of program development cycle in which a simple subset of the requirements is developed, then expanded or enhanced, with the development cycle being repeated until the full system has been developed.

JavaScript – object-orientated (or scripting) programming language used mainly on the web; used to enhance HTML pages.

JPEG – Joint Photographic Expert Group; a form of lossy file compression based on the inability of the eye to spot certain colour changes and hues.

Karnaugh maps (K-maps) – a method used to simplify logic statements and logic circuits; uses Gray codes.

Kernel – the core of an OS with control over process management, memory management, interrupt handling, device management and I/O operations.

Key distribution problem – security issue inherent in symmetric encryption arising from the fact that, when sending the secret key to a recipient, there is the risk that the key can be intercepted by an eavesdropper/hacker.

Labelled data – data where we know the target answer and the data object is fully recognised.

LAN – local area network (network covering a small area such as a single building).

Latency – the lag in a system; for example, the time to find a track on a hard disk, which depends on the time taken for the disk to rotate around to its read-write head.

Least recently used (LRU) page replacement – page replacement algorithm in which the page which has not been used for the longest time is replaced.

Leech – a peer with negative feedback from swarm members.

Left shift – bits are shifted to the left.

Legal – relating to, or permissible by, law.

Lexical analysis – the first stage in the process of compilation; removes unnecessary characters and tokenises the program.

Library program – a program stored in a library for future use by other programmers.

Library routine – a tested and ready-to-use routine available in the development system of a programming language that can be incorporated into a program.

Linear search – a method of searching in which each element of an array is checked in order.

Linked list – a list containing several items in which each item in the list points to the next item in the list.

Logic circuit – formed from a combination of logic gates and designed to carry out a particular task; the output from a logic circuit will be 0 or 1.

Logic error – an error in the logic of a program.

Logic gates – electronic circuits which rely on ‘on/off’ logic; the most common ones are NOT, AND, OR, NAND, NOR and XOR.

Logical memory – the address space that an OS perceives to be main storage.

Logical schema – a data model for a specific database that is independent of the DBMS used to build that database.

Logical shift – bits shifted out of the register are replaced with zeros.

Lossless file compression – file compression method where the original file can be restored following decompression.

Lossy file compression – file compression method where parts of the original file cannot be recovered during decompression; some of the original detail is lost.

Low level scheduling – method by which a system assigns a processor to a task or process based on the priority level.

Lower bound – the index of the first element in an array, usually 0 or 1.

Low-level programming – programming instructions that use the computer's basic instruction set.

Lurker – user/client that downloads files but does not supply any new content to the community.

Machine code – the programming language that the CPU uses.

Machine learning – systems that learn without being programmed to learn.

Maintenance – part of the program development lifecycle; the process of making sure that the program continues to work during use.

Malicious hacking – hacking done with the sole intent of causing harm to a computer system or user (for example, deletion of files or use of private data to the hacker's advantage).

Malware – malicious software that seeks to damage or gain unauthorised access to a computer system.

MAN – metropolitan area network (network which is larger than a LAN but smaller than a WAN; can cover several buildings in a single city, such as a university campus).

Mantissa – the fractional part of a floating point number.

Mask – a number that is used with the logical operators AND, OR or XOR to identify, remove or set a single bit or group of bits in an address or register.

Massively parallel computers – the linking together of several computers effectively forming one machine with thousands of processors.

Memory cache – high speed memory external to processor which stores data which the processor will need again.

Memory dump – contents of a computer memory output to screen or printer.

Memory management – part of the operating system that controls the main memory.

Memory optimisation – function of memory management that determines how memory is allocated and deallocated.

Memory organisation – function of memory management that determines how much memory is allocated to an application.

Memory protection – function of memory management that ensures two competing applications cannot use same memory locations at the same time.

Mesh network topology – interlinked computers/devices, which use routing logic so data packets are sent from sending stations to receiving stations only by the shortest route.

Metadata – a set of data that describes and gives information about other data.

Method – a programmed procedure that is defined as part of a class.

MIMD – multiple instruction multiple data, computer architecture which uses many processors, each of which can use a separate data source.

MIME – multi-purpose internet mail extension; a protocol that allows email attachments

containing media files as well as text to be sent.

MISD – multiple instruction single data, computer architecture which uses many processors but the same shared data source.

Modem – modulator demodulator; device which converts digital data to analogue data (to be sent down a telephone wire); conversely it also converts analogue data to digital data (which a computer can process).

Modulo-11 – method used to calculate a check digit based on modulus division by 11.

Monitor – to automatically take readings from a device.

Morality – an understanding of the difference between right and wrong, often founded in personal beliefs.

MP3/MP4 files – file compression method used for music and multimedia files.

Multitasking – function allowing a computer to process more than one task/process at a time.

NIC – network interface card; these cards allow devices to connect to a network/internet (usually associated with a MAC address set at the factory).

Node – device connected to a network (it can be a computer, storage device or peripheral device).

Node or vertex – fundamental unit from which graphs are formed (nodes and vertices are the points where edges converge).

Non-composite data type – a data type that does not reference any other data types.

Non-preemptive – type of scheduling in which a process terminates or switches from a running state to a waiting state.

Normal test data – test data that should be accepted by a program.

Normalisation (database) – the process of organising data to be stored in a database into two or more tables and relationships between the tables, so that data redundancy is minimised.

Normalisation (floating-point) – a method to improve the precision of binary floating-point numbers; positive numbers should be in the format 0.1 and negative numbers in the format 1.0.

Object – an instance of a class that is self-contained and includes data and methods.

Object code – a computer program after translation into machine code.

Object-oriented programming (OOP) – a programming methodology that uses self-contained objects, which contain programming statements (methods) and data, and which communicate with each other.

Odd parity – binary number with an odd number of 1-bits.

On demand (bit streaming) – system that allows users to stream video or music files from a central server as and when required without having to save the files on their own computer/tablet/phone.

One's complement – each binary digit in a number is reversed to allow both negative and positive numbers to be represented.

Opcode – short for operation code, the part of a machine code instruction that identifies the action the CPU will perform.

Open – file-processing operation; opens a file ready to be used in a program.

Open Source Initiative – organisation offering the same freedoms as the Free Software

Foundation, but with more of a focus on the practical consequences of the four shared rules, such as more collaborative software development.

Operand – the part of a machine code instruction that identifies the data to be used by the CPU.

Operating system – software that provides an environment in which applications can run and provides an interface between hardware and human operators.

Optical storage – CDs, DVDs and Blu-ray® discs that use laser light to read and write data.

Optimal page replacement – page replacement algorithm that looks forward in time to see which frame to replace in the event of a page fault.

Optimisation (compilation) – the fourth stage in the process of compilation; the creation of an efficient object program.

Optimisation (memory management) – function of memory management deciding which processes should be in main memory and where they should be stored.

Organic LED (OLED) – uses movement of electrons between cathode and anode to produce an on-screen image; generates its own light so no back lighting required.

Overclocking – changing the clock speed of a system clock to a value higher than the factory/recommended setting.

Overflow – the result of carrying out a calculation which produces a value too large for the computer's allocated word size.

Overloading – feature of object-oriented programming that allows a method to be defined more than once in a class, so it can be used in different situations.

Packet – a message/data is split up into smaller groups of bits for transmission over a network.

Packet switching – method of transmission where a message is broken into packets which can be sent along paths independently from each other.

Page fault – occurs when a new page is referred but is not yet in memory.

Page replacement – occurs when a requested page is not in memory and a free page cannot be used to satisfy allocation.

Page table – table that maps logical addresses to physical addresses; it contains page number, flag status, frame address and time of entry.

Pages – fixed-size logical memory blocks.

Paging – form of memory management which divides up physical memory and logical memory into fixed-size memory blocks.

PAN – network that is centred around a person or their workspace.

Parallel processing – operation which allows a process to be split up and for each part to be executed by a different processor at the same time.

Parameter – a variable applied to a procedure or function that allows one to pass in a value for the procedure to use.

Parity bit – an extra bit found at the end of a byte that is set to 1 if the parity of the byte needs to change to agree with sender/receiver parity protocol.

Parity block – horizontal and vertical parity check on a block of data being transferred.

Parity byte – additional byte sent with transmitted data to enable vertical parity checking (as well as horizontal parity checking) to be carried out.

Parity check – method used to check if data has been transferred correctly; uses even or odd

parity.

Pattern recognition – the identification of parts of a problem that are similar and could use the same solution.

Peer – a client who is part of a peer-to-peer network/file sharing community.

Peer-to-peer – network in which each node can share its files with all the other nodes; each node has its own data; there is no central server.

Perceptual music shaping – method where sounds outside the normal range of hearing of humans, for example, are eliminated from the music file during compression.

Perfective maintenance – the process of making improvements to the performance of a program.

Pharming – redirecting a user to a fake website in order to illegally obtain personal data about the user.

Phishing – legitimate-looking emails designed to trick a recipient into giving their personal data to the sender of the email.

PHP – hypertext processor; an HTML-embedded scripting language used to write web pages.

Physical memory – main/primary RAM memory.

Pieces – splitting up of a file when using peer-to-peer file sharing.

Pinching and rotating – actions by fingers on a touch screen to carry out tasks such as move, enlarge, reduce, and so on.

Pipelining – allows several instructions to be processed simultaneously without having to wait for previous instructions to finish.

Piracy – the practice of using or making illegal copies of, for example, software.

Pixel – smallest picture element that makes up an image.

Pixel density – number of pixels per square centimetre.

Plagiarism – the act of taking another person's work and claiming it as one's own.

Plaintext – the original text/document/message before it is put through an encryption algorithm.

Pointer data type – a non-composite data type that uses the memory address of where the data is stored.

Polymorphism – feature of object-oriented programming that allows methods to be redefined for derived classes.

POP – post office protocol.

Port – external connection to a computer which allows it to communicate with various peripheral devices; a number of different port technologies exist.

Positive feedback – the output from a process which influences the next input value to the process.

Post-WIMP – interfaces that go beyond WIMP and use touch screen technology rather than a pointing device.

Preemptive – type of scheduling in which a process switches from running state to steady state or from waiting state to steady state.

Prettyprinting – the practice of displaying or printing well set out and formatted source code, making it easier to read and understand.

Primary key – a unique identifier for a table, it is a special case of a candidate key.

Privacy – the right to keep personal information and data secret, and for it to not be unwillingly accessed or shared through, for example, hacking.

Private IP address – an IP address reserved for internal network use behind a router.

Private key – encryption/decryption key which is known only to a single user/computer.

Procedure – a set of statements that can be grouped together and easily called in a program whenever required, rather than repeating all of the statements each time.

Process – a program that has started to be executed.

Process control block (PCB) – data structure which contains all the data needed for a process to run.

Process management – part of the operating system that involves allocation of resources and permits the sharing and exchange of data.

Process states – running, ready and blocked; the states of a process requiring execution.

Product key – security method used in software to protect against illegal copies or use.

Program counter (PC) – a register used in a computer to store the address of the instruction which is currently being executed.

Program development lifecycle – the process of developing a program set out in five stages: analysis, design, coding, testing and maintenance.

Program library – a library on a computer where programs and routines are stored which can be freely accessed by other software developers for use in their own programs.

Programmable ROM (PROM) – type of ROM chip that can be programmed once.

Programming paradigm – a set of programming concepts.

Property – data and methods within an object that perform a named action.

Protocol – a set of rules governing communication across a network; the rules are agreed by both sender and recipient.

Pseudocode – a method of showing the detailed logical steps in an algorithm, using keywords, identifiers with meaningful names, and mathematical operators.

Public IP address – an IP address allocated by the user's ISP to identify the location of their device on the internet.

Public key – encryption/decryption key known to all users.

Public key infrastructure (PKI) – a set of protocols, standards and services that allow users to authenticate each other using digital certificates issued by a CA.

Public switched telephone network (PSTN) – network used by traditional telephones when making calls or when sending faxes.

Pull protocol – protocol used when receiving emails, in which the client periodically connects to a server, checks for and downloads new emails from a server and then closes the connection.

Push protocol – protocol used when sending emails, in which the client opens the connection to the server and keeps the connection active all the time, then uploads new emails to the server.

Quad core – a CPU containing four cores.

Quantum – a fixed time slice allocated to a process.

Quantum cryptography – cryptography based on the laws of quantum mechanics (the properties of photons).

Quantum key distribution (QKD) – protocol which uses quantum mechanics to securely send

encryption keys over fibre optic networks.

Quarantine – file or program identified as being infected by a virus which has been isolated by antivirus software before it is deleted at a later stage.

Qubit – the basic unit of a quantum of information (quantum bit).

Query processor – feature of a DBMS that processes and executes queries written in structured query language (SQL).

Queue – a list containing several items operating on the first in, first out (FIFO) principle.

Random access memory (RAM) – primary memory unit that can be written to and read from.

Random file organisation – a method of file organisation in which records of data are physically stored in a file in any available position; the location of any record in the file is found by using a hashing algorithm on the key field of a record.

Rapid application development (RAD) – a type of program development cycle in which different parts of the requirements are developed in parallel, using prototyping to provide early user involvement in testing.

Read – file access mode in which data can be read from a file.

Read-only memory (ROM) – primary memory unit that can only be read from.

Real-time (bit streaming) – system in which an event is captured by camera (and microphone) connected to a computer and sent to a server where the data is encoded; the user can access the data 'as it happens' live.

Record (database) – a row in a table in a database.

Record (data type) – a composite data type comprising several related items that may be of different data types.

Recursion – a process using a function or procedure that is defined in terms of itself and calls itself.

Referential integrity – property of a database that does not contain any values of a foreign key that are not matched to the corresponding primary key.

Refreshed – requirement to charge a component to retain its electronic state.

Register – temporary component in the processor which can be general or specific in its use; holds data or instructions as part of the fetch-execute cycle.

Register Transfer Notation (RTN) – short hand notation to show movement of data and instructions in a processor, can be used to represent the operation of the fetch-execute cycle.

Regression – statistical measure used to make predictions from data by finding learning relationships between the inputs and outputs.

Reinforcement learning – system which is given no training; learns on basis of 'reward and punishment'.

Relational database – a database where the data items are linked by internal pointers.

Relationship – situation in which one table in a database has a foreign key that refers to a primary key in another table in the database.

Relative addressing – mode of addressing in which the memory address used is the current memory address added to the operand.

Removable hard disk drive – portable hard disk drive that is external to the computer; it can be connected via a USB port when required; often used as a device to back up files and data.

Repeater – device used to boost a signal on both wired and wireless networks.

Repeating hubs – network devices which are a hybrid of hub and repeater unit.

Report window – a separate window in the runtime environment of the IDE that shows the contents of variables during the execution of a program.

Resistive – type of touch screen technology; when a finger touches the screen, the glass layer touches the plastic layer, completing the circuit and causing a current to flow at that point.

Resolution – number of pixels per column and per row on a monitor or television screen.

Reverse Polish notation (RPN) – a method of representing an arithmetical expression without the use of brackets or special punctuation.

Reward and punishment – improvements to a model based on whether feedback is positive or negative; actions are optimised to receive an increase in positive feedback.

Right shift – bits are shifted to the right.

RISC – reduced instruction set computer.

Round robin (scheduling) – scheduling algorithm that uses time slices assigned to each process in a job queue.

Router – device which enables data packets to be routed between different networks (for example, can join LANs to form a WAN).

Routing table – a data table that contains the information necessary to forward a package along the shortest or best route to allow it to reach its destination.

Rules – relationships between facts.

Run length encoding (RLE) – a lossless file compression technique used to reduce text and photo files in particular.

Run-time error – an error found in a program when it is executed; the program may halt unexpectedly.

Sampling rate – number of sound samples taken per second.

Sampling resolution/bit depth – number of bits used to represent sound amplitude.

Scheduling – process manager which handles the removal of running programs from the CPU and the selection of new processes.

Screen resolution – number of horizontal and vertical pixels that make up a screen display; if the screen resolution is smaller than the image resolution, the whole image cannot be shown on the screen, or the original image will become lower quality.

Second normal form (2NF) – the status of a relational database in which entities are in 1NF and any non-key attributes depend upon the primary key.

Secondary key – a candidate key that is an alternative to the primary key.

Secure Sockets Layer (SSL) – security protocol used when sending data over the internet.

Security management – part of the operating system that ensures the integrity, confidentiality and availability of data.

Seed – a peer that has downloaded a file (or pieces of a file) and has then made it available to other peers in the swarm.

Segment (transport layer) – this is a unit of data (packet) associated with the transport layer protocols.

Segment map table – table containing the segment number, segment size and corresponding

memory location in physical memory; it maps logical memory segments to physical memory.

Segment number – index number of a segment.

Segments (memory) – variable-size memory blocks into which logical memory is split up.

Semi-supervised (active) learning – system that interactively queries source data to reach the desired result; it uses both labelled and unlabelled data, but mainly unlabelled data on cost grounds.

Sensor – input device that reads physical data from its surroundings.

Sequential access – a method of file access in which records are searched one after another from the physical start of the file until the required record is found.

Sequential circuit – circuit in which the output depends on input values produced from previous output values.

Sequential file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in a given order.

Serial access – a method of file access in which records are searched one after another from the physical start of the file until the required record is found.

Serial file organisation – a method of file organisation in which records of data are physically stored in a file, one after another, in the order they were added to the file.

Session caching – function in TLS that allows a previous computer session to be 'remembered', therefore preventing the need to establish a new link each time a new session is attempted.

Set – a given list of unordered elements that can use set theory operations such as intersection and union.

Setter – a method used to control changes to a variable.

Shareware – software that is free of charge initially (free trial period); the full version of the software can only be downloaded once the full fee for the software has been paid.

Shift – moving the bits stored in a register a given number of places within the register; there are different types of shift.

Sign and magnitude – binary number system where left-most bit is used to represent the sign (0 = + and 1 = –); the remaining bits represent the binary value.

SIMD – single instruction multiple data, computer architecture which uses many processors and different data inputs.

Single stepping – the practice of running a program one line/instruction at a time.

SISD – single instruction single data, computer architecture which uses a single processor and one data source.

SMTP – simple mail transfer protocol.

Softmodem – abbreviation for software modem; a software-based modem that uses minimal hardware.

Solid state drive (SSD) – storage media with no moving parts that relies on movement of electrons.

Source code – a computer program before translation into machine code.

Spread spectrum frequency hopping – a method of transmitting radio signals in which a device picks one of 79 channels at random. If the chosen channel is already in use, it randomly chooses another channel. It has a range up to 100 metres.

Spread spectrum technology – wideband radio frequency with a range of 30 to 50 metres.

SQL script – a list of SQL commands that perform a given task, often stored in a file for reuse.

Stack – a list containing several items operating on the last in, first out (LIFO) principle.

Star network topology – a network that uses a central hub/switch with all devices connected to this central hub/switch; all data packets are directed through this central hub/switch.

Starve – to constantly deprive a process of the necessary resources to carry out a task/process.

State-transition diagram – a diagram showing the behaviour of a finite state machine (FSM).

State-transition table – a table showing every state of a finite state machine (FSM), each possible input and the state after the input.

Static RAM (SRAM) – type of RAM chip that uses flip-flops and does not need refreshing.

Status register – used when an instruction requires some form of arithmetic or logical processing.

Stepwise refinement – the practice of subdividing each part of a larger problem into a series of smaller parts, and so on, as required.

Stream cipher – the encryption of bits in sequence as they arrive at the encryption algorithm.

Structure chart – a modelling tool used to decompose a problem into a set of sub-tasks. It shows the hierarchy or structure of the different modules and how they connect and interact with each other.

Structured English – a method of showing the logical steps in an algorithm, using an agreed subset of straightforward English words for commands and mathematical operations.

Structured query language (SQL) – the standard query language used with relational databases for data definition and data modification.

State-transition table – a table showing every state of a finite state machine (FSM), each possible input and the state after the input.

Stub testing – the use of dummy modules for testing purposes.

Sub-netting – practice of dividing networks into two or more sub-networks.

Sum of products (SoP) – a Boolean expression containing AND and OR terms.

Super computer – a powerful mainframe computer.

Supervised learning – system which is able to predict future outcomes based on past data; it requires both input and output values to be used in the training process.

Swap space – space on HDD used in virtual memory, which saves process data.

Swarm – connected peers (clients) that share a torrent/tracker.

Switch – hardware used to connect together a number of devices to form a LAN; directs incoming data packets to a specific destination address only.

Symbolic addressing – mode of addressing used in assembly language programming; a label is used instead of a value.

Symmetric encryption – encryption in which the same secret key is used to encrypt and decrypt messages.

Syntax analysis – the second stage in the process of compilation; output from the lexical analysis is checked for grammatical (syntax) errors.

Syntax diagram – a graphical method of defining and showing the grammatical rules of a programming language.

Syntax error – an error in the grammar of a source program.

System clock – produces timing signals on the control bus to ensure synchronisation takes place.

Table – a group of similar data, in a database, with rows for each instance of an entity and columns for each attribute.

TCP – transmission control protocol.

Test plan – a detailed list showing all the stages of testing and every test that will be performed for a particular program.

Test strategy – an overview of the testing required to meet the requirements specified for a particular program; it shows how and when the program is to be tested.

Testing – part of the program development lifecycle; the testing of the program to make sure that it works under all conditions.

Thick client – device which can work both off line and on line and is able to do some processing even if not connected to a network/internet.

Thin client – device that needs access to the internet for it to work; it depends on a more powerful computer for processing.

Third normal form (3NF) – the status of a relational database in which entities are in 2NF and all non-key attributes are independent.

Thrash point – point at which the execution of a process comes to a halt since the system is busier paging in/out of memory rather than actually executing them.

Timeout – time allowed to elapse before an acknowledgement is received.

Touch screen – screen on which the touch of a finger or stylus allows selection or manipulation of a screen image; they usually use capacitive or resistive technology.

Trace table – a table showing the process of dry-running a program with columns showing the values of each variable as it changes.

Tracker – central server that stores details of all other computers in the swarm.

TLB Translation lookaside buffer (TLB) – this is a memory cache which can reduce the time taken to access a user memory location; it is part of the memory management unit.

Translator – the systems software used to translate a source program written in any language other than machine code.

Transport Layer Security (TLS) – a more up-to-date version of SSL.

Truth table – a method of checking the output from a logic circuit; they use all the possible binary input combinations depending on the number of inputs; for example, 2 inputs have 2^2 (4) possible binary combinations, 3 inputs will have 2^3 (8) possible binary combinations, and so on.

Tuple – one instance of an entity, which is represented by a row in a table.

Twisted pair cable – type of cable in which two wires of a single circuit are twisted together; several twisted pairs make up a single cable.

Two's complement – each binary digit is reversed and 1 is added in right-most position to produce another method of representing positive and negative numbers.

Underflow – the result of carrying out a calculation which produces a value too small for the computer's allocated word size.

Unicode – coding system which represents all the languages of the world (first 128 characters

are the same as ASCII code).

Unidirectional – used to describe a bus in which bits can travel in one direction only.

Uniform resource locator (URL) – specifies location of a web page (for example, www.hoddereducation.co.uk).

Universal Serial Bus (USB) – a type of port connecting devices to a computer.

Unlabelled data – data where objects are undefined and need to be manually recognised.

Unsupervised learning – system which is able to identify hidden patterns from input data; the system is not trained on the 'right' answer.

Unwinding – process which occurs when a recursive function finds the base case and the function returns the values.

Upper bound – the index of the last element in an array.

User account – an agreement that allows an individual to use a computer or network server, often requiring a user name and password.

User defined data type – a data type based on an existing data type or other data types that have been defined by a programmer.

Utility program – parts of the operating system which carry out certain functions, such as virus checking, defragmentation or hard disk formatting.

Validation – method used to ensure entered data is reasonable and meets certain input criteria.

Variable – a named value that can change during the execution of a program.

Vector graphics – images that use 2D points to describe lines and curves and their properties that are grouped to form geometric shapes.

Verification – method used to ensure data is correct by using double entry or visual checks.

Video Graphics Array (VGA) – type of port connecting devices to a computer.

Virtual machine – an emulation of an existing computer system; a computer OS running within another computer's OS.

Virtual memory – type of paging that gives the illusion of unlimited memory being available.

Virtual memory systems – memory management (part of OS) that makes use of hardware and software to enable a computer to compensate for shortage of actual physical memory.

Virtual reality headset – apparatus worn on the head that covers the eyes like a pair of goggles; it gives the user the 'feeling of being there' by immersing them totally in the virtual reality experience.

Voice over Internet Protocol (VoIP) – converts voice and webcam images into digital packages to be sent over the internet.

Von Neumann architecture – computer architecture which introduced the concept of the stored program in the 1940s.

Walkthrough – a method of testing a program; a formal version of a dry run using pre-defined test cases.

WAN – wide area network (network covering a very large geographical area).

(W)AP – (wireless) access point which allows a device to access a LAN without a wired connection.

Waterfall model – a linear sequential program development cycle, in which each stage is completed before the next is begun.

Web browser – software that connects to DNS to locate IP addresses; interprets web pages sent to a user's computer so that documents and multimedia can be read or watched/listened to.

Web crawler – internet bot that systematically browses the world wide web to update its web page content.

White-box testing – a method of testing a program that tests the structure and logic of every path through a program module.

Wi-Fi – wireless connectivity that uses radio waves, microwaves.

WIMP – windows, icons, menu and pointing device.

Winding – process which occurs when a recursive function or procedure is called until the base case is found.

WLAN – wireless LAN.

WNIC – wireless network interface cards/controllers.

Word – group of bits used by a computer to represent a single unit.

World Wide Web (WWW) – collection of multimedia web pages stored on a website; uses the internet to access information from servers and other computers.

WPAN – wireless personal area network; a local wireless network which connects together devices in very close proximity (such as in a user's house); typical devices would be a laptop, smartphone, tablet and printer.

Write – file access mode in which data can be written to a file; any existing data stored in the file will be overwritten.

Zero compression – way of reducing the length of an IPv6 address by replacing groups of zeroes by a double colon (::); this can only be applied once to an address to avoid ambiguity.

Index

1D arrays 241–2
2D arrays 242–3
3D printers 79

A

A* algorithm 425, 429–34, 541
abnormal test data 294, 298, 541
absolute addressing 121, 125, 541
abstract data types (ADTs) 238, 250–9, 464–89, 541
  binary trees 451, 481–7, 542
  graphs see *graphs*
  implementing one ADT from another ADT 488–9
  linked lists 238, 250–1, 255–9, 464, 469–81, 547
  queues 238, 250–1, 253–5, 464, 466–9, 549
  stacks 238, 250–3, 464–6, 551
abstraction 217, 218–19, 541
acceptance testing 294, 299, 541
access rights
  databases 208, 210, 541
  data security 159, 161, 541
accumulator (ACC) 108, 109, 110, 541
accuracy 323–4
acknowledgement 170, 175, 541
actuators 84
adaptive maintenance 294, 299, 541
addition 3–5
address bus 108, 112, 541
addressing modes 121, 125–6, 541
Advanced Research Project Agency Network (ARPAnet) 28, 29, 541
advertising 193
aggregation (containment) 499, 514–15, 543
Airbus A380 incompatible software issue 184
algorithms 219–35, 450–90, 541
  abstract data types see *abstract data types (ADTs)*
  comparing 489–90
  insertion and bubble sorting methods 458–64
  linear and binary searching methods 451–7
  page replacement 373, 388–9, 548
  shortest path algorithms 425–34
  writing 220–35
alpha testing 294, 299, 541

Amazon 33–4
analogue to digital converter (ADC) 19, 69, 81, 84, 541
analysis 283, 284, 285, 286, 541
AND gates 89, 91, 100
  multi-input 101–2
anti-lock braking systems (ABS) 87
anti-spy software 160, 163, 541
antivirus software 137, 144, 163, 541
append file access mode 525, 531, 533, 541
application layer 329, 330–3
approximations 320–2
arguments 275, 280, 541
arithmetic-logic unit (ALU) 108, 109–10, 541
arithmetic operation instructions 124
arithmetic shift 130, 541
ARPAnet 28, 29, 541
arrays 238, 241–8, 541
artificial intelligence (AI) 189–93, 425–49, 541
  impacts on society, the economy and the environment 190–3
  machine learning, deep learning and 434–45
  shortest path algorithms 425–34
artificial neural networks 435, 439–41, 444, 541
ASCII code 2, 12–14, 541
assemblers 121, 122–3, 150, 151, 541
assembly language 121–9, 541
  instructions 123–5
  simple programs 126–8
  stages of assembly 122–3
Association for Computing Machinery (ACM) 179, 181, 541
asymmetric encryption 410, 413–14, 541
asynchronous serial data transmission 108, 114, 541
attributes
  classes 498, 501, 541
  databases 197, 200, 541
audio compression 21, 541
authentication 159, 160, 541
authenticity 411
auto-documenter 157
automatic repeat request (ARQ) 170, 175, 541

B

backing up data 167
back propagation 435, 441, 444–5, 541
back-up utility software 137, 146, 541
Backus-Naur form (BNF) notation 394, 397, 398, 400, 541
bad sectors 137, 143, 541

base case 490–1, 541
basic input/output system (BIOS) 108, 113, 542
Belady’s anomaly 373, 388, 541
beta testing 294, 299, 541
bidirectional buses 108, 112, 541
Big O notation 451, 489–90, 542
binary-coded decimal (BCD) system 2, 10–12, 542
binary files 329, 332, 542
binary floating-point numbers 313–25, 542
  converting denary numbers into 317–25
  converting into denary 314–17
binary number system 2–8, 542
  converting between denary and 2–3
  converting between hexadecimal and 8–9
binary search 451, 454–7, 542
binary shifts 130–1
binary trees 451, 481–7, 542
  finding items 482–3
  inserting items 484–7
  writing programs for 517–21
binder 3D printing 69, 79, 542
biometrics 160, 163–4, 542
BIOS 108, 113, 542
birefringence 69, 76, 542
bit depth (sampling resolution) 15, 16, 20, 24, 542, 550
bit manipulation 130–2
bit-map images 15–18, 542
  calculating file size 17–18
  compared with vector graphics 18–19
  file compression 22
bit rate 21, 22, 29, 52, 542
bits 2, 542
bit streaming 29, 52–3, 542
BitTorrent protocol 329, 335–7, 542
black and white images 23
black-box testing 294, 299, 542
block chaining 410, 411, 542
block cipher 410, 411, 542
blocked state 377–8
Bluetooth 28, 41–2, 542
  protocols 335
Blu-ray® discs 76
Boolean algebra 89, 90–2, 95, 354–6, 542
  and logic circuits 361–8
  simplification using 355–6
bootstrap program 373, 374, 542

bots 165
boundary test data 294, 298, 542
break 273
breakpoint 150, 155, 542
bridges 28, 47, 542
British Computer Society (BCS) 179, 180, 541
broadcast 29, 50–1, 542
bubble sort 238, 245–8, 458–60, 542
buffering 29, 52, 542
bugs 294–6
burst time 373, 376, 542
buses 109, 112–14
bus network topology 28, 37, 39, 542
by reference method 275, 277–8, 542
bytecode 152–3
bytes 6–7
by value method 275, 277, 542

C

cache memory 68, 69, 71, 108, 113, 542, 547
Cambridge Analytica scandal 193
candidate keys 197, 200–1, 542
capacitive touch screens 69, 82–3, 542
carrier sense multiple access with collision avoidance (CSMA/CA) 335
carrier sense multiple access with collision detection (CSMA/CD) 29, 51, 543
car sensors 86–7
CASE statements 222, 223, 271–3
CDs 75–6
cellular networks 56
central processing unit (CPU) architecture 107–20
  components 109–10
  computer ports 108, 114–16, 549
  fetch-execute cycle 108, 116–18, 544
  interrupts 108, 118–19
  registers 108, 109, 110–11, 550
  system buses 109, 112–14
certificate authority (CA) 416, 418, 420, 421, 542
character set 2, 12, 542
chatbots 435, 442–3, 542
check digits 169, 171, 542
checksums 169, 172, 340, 542
ciphertext 410, 411, 542
circuit switching 55, 337, 338, 542
  comparison with packet switching 340–1
CISC (complex instruction set computer) 347, 348, 542
classes 307, 498, 501, 542

classless inter-domain routing (CIDR) 54, 58, 59, 542
CLI (command line interface) 137, 138–9, 542
client/server network model 28, 32–4, 35–6, 542
clock cycle 108, 113, 542
clock page replacement 388–9
close (file processing) 525, 542
cloud software 41
cloud storage 28, 39–41, 543
clusters 347, 352, 543
CMOS 137, 138, 543
coaxial cables 28, 44, 543
code generation 394, 395, 397, 543
codes of conduct 180–3
coding 283, 284, 285, 286, 543
collisions 29, 50–1, 543
colour depth 15, 16, 24, 543
coloured images 24
colouring monochrome photos 442
combination circuits 354, 358, 543
command line interface (CLI) 137, 138–9, 542
commercial software 187
communication 27–67, 328–45
  circuit switching and packet switching 337–43
  internet see *internet*
  networking 28–53
  protocols 328–37
compare instructions 125
compilation, stages in 395–8
compilers 149, 151–2, 155, 394–5, 543
composite data types 238, 240, 306–7, 543
composite key 197, 543
computational thinking skills 450–97
  algorithms see *algorithms*
  recursion 490–4
  skills 217–19
computer-assisted translation (CAT) 441
computer ethics 180–1
conditional instructions 125
conditional loops 456
confidentiality 411
conflicts 29, 50, 543
constants 264, 265–71, 543
constructors 499, 515–16, 543
containment (aggregation) 499, 514–15, 543
context switching 373, 379, 381, 543
contiguous 137, 140, 543

single (contiguous) memory allocation 383
control 85–7, 130, 131–2, 543
control bus 108, 112, 543
control unit 108, 109, 110, 543
copyright issues 186–9
cores 108, 113, 543
corrective maintenance 294, 299, 543
count-controlled loops 274, 275
criminal justice system 192
cross-coupling 354, 358–9, 543
CSMA/CA (carrier sense multiple access with collision avoidance) 335
CSMA/CD (carrier sense multiple access with collision detection) 29, 51, 543
culture 179, 543
current instruction register (CIR) 108, 110, 116, 117, 543
cyclic shift 130, 543

D

database management systems (DBMSs) 208–10, 543
databases 196–208, 543
normalisation 203–7
data bus 108, 112, 543
data definition language (DDL) 211–12, 543
data dependency 209
data dictionary 208, 209, 543
data entry, verification during 170–2
datagrams 330
data hiding 498, 503, 543
data inconsistency 209
data integrity 169–76, 411, 543
data input instructions 124
data-link layer 329, 330, 334–7
data loss
in cloud storage 40–1
preventing 160–4
data management 208, 209, 543
data manipulation language (DML) 211, 213–14, 543
data modelling 208, 210, 543
data movement instructions 123–4
data output instructions 124
data privacy 159, 160, 543
data protection laws 159, 160, 543
data recovery 167
data redundancy 28, 40, 209, 543
data representation 2–15, 304–27
ASCII code 2, 12–14, 541
file organisation and access 308–11

floating-point numbers 312–25
number systems 2–12
Unicode 2, 14, 15, 552
user-defined data types 304–7, 552
data security 40, 159–68, 410–24, 543
digital signatures and digital certificates 418–23
encryption 160, 163, 410–14, 544
protocols 416–18
quantum cryptography 414–15, 549
when using cloud storage 40–1
data transfer, verification during 172–5
data types 238–41, 543
abstract see *abstract data types (ADTs)*
composite 238, 240, 306–7, 543
non-composite 305–6, 547
debugging 150, 155–6, 543
declarative programming 499, 521–4, 543
decomposition 217, 219, 330, 543
deep learning 434, 435, 439–43, 543
default 273
defragmentation software 144–5
De Morgan's Laws 355
denary numbers 2, 7–8
converting between binary numbers and 2–3
converting binary floating-point numbers into 314–17
converting into binary floating-point numbers 317–25
design 283, 284, 285, 286, 543
destructors 499, 515, 517, 543
developer interface 209, 210, 543
device driver 137, 543
dictionaries 451, 488–9, 544
digest 418, 419, 420, 544
digital certificates 418, 420–2, 544
digital rights management (DRM) 186, 187, 544
digital signatures 162, 418, 419–20, 544
digital to analogue converter (DAC) 69, 80–1, 84, 544
Dijkstra's algorithm 425–9, 544
direct 3D printing 69, 79, 544
direct access 308, 310, 311, 544
direct addressing 121, 125, 544
direct memory access (DMA) controller 373, 375, 544
dirty pages 373, 383, 544
disk compression 137, 145, 544
disk content analysis software 137, 145, 544
disk defragmenter 137, 145, 544
disk formatter 137, 143, 544

disk thrashing 373, 386, 544
DNS cache poisoning 160, 166, 544
DO … ENDWHILE loops 274–5
domain name system/service (DNS) 54, 61–2, 330, 544
double entry 171
dry runs 294, 296–8, 544
dual core 108, 113, 544
dual layering 69, 75–6, 544
DV (digital video) cameras 20, 21
DVDs 75–6
dynamic link files (DLL) 138, 148, 544
dynamic RAM (DRAM) 68, 70–1, 544

E

eavesdropper 410, 411, 544
electromagnetic radiation 42–3
electronically erasable programmable read-only memory (EEPROM) 69, 74, 544
Else 273
embedded systems 72
emulation 392, 544
encapsulation 498, 501, 544
encryption 160, 163, 410–14, 544
entities 197, 200, 544
entity-relationship (E-R) diagrams 197, 202, 544
enumerated data types 305, 544
erasable PROM (EPROM) 69, 72, 544
errors in programs 294–6
Ethernet 29, 50–1, 544
  protocols 334–5
ethical hacking 160, 164, 544
ethics 179–85, 544
even parity 169, 173, 544
exception handling 525, 535–7, 544
exceptions 525, 535–6, 544
exploding laptop computers 184
exponent 313–24, 544
extreme test data 294, 298, 544

F

face recognition software 439–40
facts 499, 521–3, 544
false positives 137, 544
faults in programs 294–6
Federation Against Software Theft (FAST) 186
fetch-execute cycle 108, 116–18, 544

fibre optic cables 28, 44, 544
fields 197, 199–200, 544
file access 308, 309–11, 544
file-based approach 197–9
  how a DBMS addresses limitations of 209–10
file compression 21–5, 145
file organisation 308–9, 544
file processing 525–35
  adding records 531–5
  finding records 535
  storing records 526–31
files 238, 249–50, 544
  bit-map image file sizes 17–18
  management of 142
file server 28, 30, 545
file transfer protocol (FTP) 329, 330, 331–2, 545
fingerprint scans 164
finite state machine (FSM) 287, 292, 545
firewalls 48, 160, 162–3, 545
first come first served (FCFS) scheduling 379, 381
first in first out (FIFO) page replacement 373, 388, 545
first normal form (1NF) 197, 203, 204–5, 545
flags 108, 111, 545
flash memory 69, 74–5, 374, 545
flip-flop circuits 354, 358–61, 545
floating-point numbers 312–25
flooding 38
flowcharts 219, 220, 221, 545
  writing pseudocode from 229–31
foreign keys 197, 200, 201, 545
FOR loops 225, 226
FOR … NEXT loops 274, 275
fragmentation 69, 73, 545
frame rate 15, 21, 24, 545
frames
  memory blocks 373, 383–4, 545
  packets 330
Free Software Foundation 186, 187–8, 545
freeware 186, 189, 545
FTP (file transfer protocol) 329, 330, 331–2, 545
full adder circuits 354, 357–8, 545
functions 264, 269, 545
  string manipulation functions 269–71
  structural programming 278–80

G

gateways 28, 45, 48, 49, 545
general AI 435
general case 490–1, 545
getters 499, 515, 516, 545
graphical user interface (GUI) 137, 138, 139, 545
graphs 451, 487, 545
  shortest path algorithms 425–34
gray codes 354, 363, 364, 545
guest operating system (OS) 392, 393, 545
GUI (graphical user interface) 137, 138, 139, 545

H

hacking 160, 164, 179, 545
half adder circuits 354, 356–7, 545
handshake 416, 417, 418, 545
hard disk drives (HDDs) 69, 73–4, 545
hardware 30, 68–106, 346–71
  Boolean algebra and logic circuits 354–68
  computers and their components 68–89
  logic gates and logic circuits 89–104
  needed to support the internet 55–7
  processors and parallel processing 346–53
  requirements of networks 45–50
hardware management 137, 142, 545
hashing algorithms
  cryptography 418, 419, 420, 545
  file access 308, 309, 310–11, 545
HCI (human-computer interface) 137, 138, 545
headers
  data packets 337, 340–1, 342, 545
  procedures or functions 275, 280, 545
heuristic checking 137, 144, 545
heuristics 425, 430, 545
hexadecimal number system 2, 7–10, 545
high-bandwidth digital copy protection (HDCP) 108, 115, 545
high-definition multimedia interface (HDMI) 108, 115, 116, 545
hop number/hopping 337, 340, 545
host 329, 333, 545
host operating system (OS) 392, 393, 545
host-to-host protocol 329, 333–4, 545
HTTP (hypertext transfer protocol) 329, 330–1, 545
hubs 28, 37, 45–6, 545
  repeating 28, 46–7, 550
human-computer interface (HCI) 137, 138, 545
hybrid cloud 40
hybrid networks 28, 39, 545

HyperText Mark-up Language (HTML) 54, 55, 545
  scripting in 62–4
hypervisor 392, 545

I

icons 137, 545
identifier 238, 239, 545
identifier tables 221, 227, 230, 233, 244, 246, 265, 290
IEEE 50, 179, 180–1, 546
IF statements 222, 223, 224, 271, 457
image resolution 15, 16–17, 24, 546
images
  general file compression methods 24
  run-length encoding with 23–4
IMAP (internet message access protocol) 329, 330, 332–3, 546
immediate access store (IAS) 108, 110, 546
immediate addressing 121, 126, 546
imperative programming 498, 500–1, 546
in demand paging 373, 385–6, 546
index
  array 238, 241, 546
  database 197, 202, 546
indexed addressing 121, 125, 546
indirect addressing 121, 125, 546
infrared radiation 42–3
inheritance 499, 505–9, 514–15, 546
inkjet printers 78
input data instructions 124
input devices 81, 84–7
input/output (I/O) system 374–5
insertion sort 451, 461–4, 546
instances 498, 502–4, 546
Institute of Electrical and Electronics Engineers (IEEE) 50, 179, 180–1, 546
instructions 121–2, 546
  assembly language instructions 123–5
instruction set 121, 122, 546
integrated development environments (IDEs) 150, 151, 153–7, 546
integration testing 294, 299, 546
integrity, data 169–76, 411, 543
intellectual property rights 179, 180, 546
  copyright issues 186–9
internet 54–65, 187, 546
  communication and internet technologies 328–45
  hardware and software needed 55–7
  IP addresses 57–61
  TCP/IP protocols 57, 329–37

internet message access protocol (IMAP) 329, 330, 332–3, 546
internet/network layer 329, 330, 334–7
internet protocols (IPs) 54, 57–61, 334, 546
internet service providers (ISPs) 54, 55, 546
interpreters 149, 151–2, 155, 394–5, 546
interrupt dispatch table (IDT) 373, 382, 546
interrupt priority 108, 119, 546
interrupt priority levels (IPL) 373, 382, 546
interrupts 108, 118–19, 349–50, 382, 546
interrupt service routine (ISR) (interrupt handler) 108, 118, 119, 546
IPv4 addressing 54, 57–8, 546
IPv6 addressing 54, 58–9, 546
iterative model 283, 286, 546

J

Java 228, 239, 271
  binary search 456, 457
  bubble sort 460
  case statements 273
  constants and variables 266, 267, 268, 269
  exception handling 537
  file processing 529–31, 533
  functions 278, 280
  IF statement 224
  insertion sort 463
  linear search 453–4
  linked lists 473, 476, 480
  loops 274, 275
  OOP 503, 504, 508–9, 512, 514
  procedures 275, 276, 277, 278
  queues 466, 467, 468
  recursion 492
  stacks 464, 465, 466
  writing programs for binary trees 518, 520, 521
JavaScript 54, 63, 546
JK flip-flops 360–1
JPEG 21, 22, 546

K

Karnaugh maps (K-maps) 354, 363–7, 546
kernel 373, 375–6, 546
key distribution problem 410, 412, 546
keyword table 396

L

LA airport shutdown 184
labelled data 434, 437–8, 440, 441, 546
language translation 149–57, 394–402
LANs (local area networks) 28, 29, 31, 32, 546
laser printers 77
latency 69, 73, 74, 351–2, 546
least recently used (LRU) page replacement 373, 388, 546
leeches 329, 336, 337, 546
left shift 130, 131, 546
legality 179, 546
lexical analysis 394, 395–7, 546
library programs 138, 147–8, 546
library routines 138, 147–8, 264, 271, 546
linear search 238, 243–4, 451–4, 546
linked lists 238, 250–1, 464, 469–81, 547
  deleting items 477–81
  finding items in 469–74
  inserting items 474–7
  linked list operations 255–9
local area networks (LANs) 28, 29, 31, 32, 546
logical memory 373, 383–4, 547
logical schema 208, 210, 547
logical shift 130, 547
logic bombs 165
logic circuits 89, 92–101, 356–68, 547
  Boolean algebra and 361–8
  flip-flop circuits 358–61
  half adder and full adder circuits 356–8
  in the real world 99–101
  simplification 101
logic errors 150, 155, 295, 547
logic gates 89–92, 547
  multi-input 101–4
loops 274–5, 456
  writing algorithms 220–9
lossless file compression 21, 547
lossy file compression 21, 547
lower bound 238, 241–2, 547
low-level programming 498, 499–500, 547
low level scheduling 373, 377, 547
lurkers 329, 336, 547

M

machine code 121–2, 547

machine learning 193, 434, 435, 436–9, 443, 547
maintenance 283, 284, 285, 286, 294, 299, 547
malicious hacking 160, 164, 547
malware 160, 162, 164–6, 547
MANs (metropolitan area networks) 28, 30, 32, 547
mantissa 313–24, 547
mask 130, 547
massively parallel computers 347, 352, 547
memory 69–77
  measurement of size 6–7
memory cache 68, 69, 71, 108, 113, 542, 547
memory dumps 2, 9–10, 547
memory management 137, 140–1, 373, 382–5, 389, 547
memory optimisation 137, 140, 382, 547, 548
memory organisation 137, 140, 547
memory protection 137, 140–1, 547
memory sticks (flash memories) 69, 74–5, 374, 545
mesh network topology 28, 38, 547
metadata 329, 335, 547
methods 498, 501, 547
  object methods 515–17
metropolitan area networks (MANs) 28, 30, 32, 547
microphones 81
microwave radiation 42–3
MIMD (multiple instruction multiple data) 347, 351, 352, 547
MIME (multi-purpose internet mail extension) protocol 329, 332, 547
MISD (multiple instruction single data) 347, 351, 547
modems 28, 48–9, 547
modulo-11 169, 171, 547
monitoring 85–7, 130, 131–2, 547
morality 179, 547
motion JPEG 20
movie files 20–1, 24
MPEG-3 (MP3) files 21–2, 547
MPEG-4 (MP4) files 21, 22, 547
multi-input logic gates 101–4
multimedia 15–21
multitasking 373, 376, 547

N

NAND gates 89, 91, 100
narrow AI 435
negative numbers 3–4
  converting binary floating-point numbers into denary 315–17
  converting denary numbers into binary floating-point numbers 319–20, 323
  normalisation 322

network/data-link layer 329, 330, 334–7
networking 28–53
  bit streaming 29, 52–3, 542
  client/server model 28, 32–4, 35–6, 542
  devices 29–32
  Ethernet 29, 50–1, 544
  hardware requirements 45–50
  peer-to-peer model 28, 34–5, 548
  public and private cloud computing 39–41
  topologies 36–9
  wired and wireless 41–5
network interface cards (NICs) 28, 49, 547
  wireless 29, 50, 552
nodes
  networks 28, 34, 547
  vertices (in graphs) 425–34, 547
non-composite data types 305–6, 547
non-preemptive scheduling 373, 376, 547
non-repudiation 411
NOR gates 89, 91
normalisation
  databases 197, 203–7, 547
  floating-point numbers 313, 322, 547
normal test data 294, 298, 547
NOT gates 89, 90, 100
number systems 2–12
  BCD 2, 10–12, 542
  binary 2–8, 8–9, 542
  hexadecimal 2, 7–10, 545

O

object code 121, 123, 548
object-oriented programming (OOP) 498, 501–21, 548
  containment 499, 514–15, 543
  inheritance 499, 505–9, 514–15, 546
  object methods 515–17
  polymorphism and overloading 509–14
  writing a program for a binary tree 517–21
objects 307, 498, 502–4, 547
odd parity 169, 173, 548
on demand (bit streaming) 29, 53, 548
one’s complement 2, 3, 548
opcode 121, 122, 548
open (file processing) 525, 533, 548
Open Source Initiative 186, 187–9, 548
operand 121, 122, 548

operating systems (OS) 136–49, 372–92, 548
  memory management 137, 140–1, 373, 382–5, 389, 547
  need for 138–9
  page replacement 373, 388–9, 548
  process management 137, 142, 376–7, 389, 549
  process states 373, 377–82, 549
  program libraries 138, 147–8, 549
  resource maximisation 374–6
  tasks 140–2
  utility software 137, 143–6, 552
  virtual memory 373, 385–7, 552
optical storage 69, 75–6, 548
optimal page replacement (OPR) 373, 388, 548
optimisation
  compilation 394, 395, 398, 548
  memory management 137, 140, 382, 547, 548
organic light emitting diode (OLED) 69, 81–2, 548
OR gates 89, 91, 100
  multi-input 102–3
OTHERWISE 271–3
output data instructions 124
output devices 77–84
overclocking 108, 113, 548
overflow errors 313, 325, 548
overloading 499, 509, 513–14, 548

P

packets 28, 37, 329, 330, 548
packet switching 56, 337, 339–43, 548
  compared with circuit switching 340–1
page fault 373, 388, 389, 548
page replacement 373, 388–9, 548
pages 373, 383–4, 548
page tables 373, 383–4, 548
paging 373, 383–4, 385, 548
  using virtual memory 385–7
PANs (personal area networks) 28, 32, 548
parallel processing 347, 350–3, 548
parameters 275, 276–8, 280, 548
  functions with and without 279
parity bit 169, 173, 548
parity blocks 169, 174, 548
parity byte 170, 174–5, 548
parity checks 169, 173–5, 329, 548
partial compiling and interpreting 152–3
passwords 161–2

pattern recognition 217, 219, 548
peers 329, 335, 548
peer-to-peer file sharing 335–7
peer-to-peer network model 28, 34–5, 548
perceptual music shaping 21, 22, 548
perfective maintenance 294, 299, 548
personal area networks (PANs) 28, 32, 548
pharming 160, 166, 548
phishing 160, 165–6, 548
phone calls 55–7
photographic (bit-map) images 22
photographs
  enhancing 442
  turning monochrome photos into colour photos 442
PHP 54, 63–4, 548
physical memory 373, 383–4, 548
pieces 329, 335, 548
piezoelectric technology 78
pinching and rotating 137, 139, 548
pipelining 347, 348–50, 548
piracy 186, 548
pixel density 15, 17, 440, 548
pixels 15–16, 82, 548
plagiarism 179, 180, 548
plaintext 410, 411, 548
pointer data types 305, 306, 548
polymorphism 499, 509–13, 549
POP (or POP3/4) (post office protocol) 329, 330, 332–3, 549
ports 108, 114–16, 549
positive feedback 84, 354, 359, 549
positive numbers
  converting binary floating-point numbers into denary 314–15
  converting denary numbers into binary floating-point numbers 317–19, 323, 324
  normalisation 322
post-condition loops 274
post office protocol (POP or POP3/4) 329, 330, 332–3, 549
post-WIMP 137, 139, 549
precision 323–4
pre-condition loops 274–5
pre-emptive scheduling 373, 376, 549
prettyprinting 149, 154, 549
primary keys 197, 200–1, 549
primary memory 70–3
printers 77–9
privacy 179, 549
  data privacy 159, 160, 543

software copyright and 186–7
private cloud 40
private IP addresses 54, 61, 549
private keys 410, 413–14, 549
private networks 31
procedures 264, 271, 275–8, 280, 549
process control block (PCB) 373, 377, 549
processes 373, 376, 549
process management 137, 142, 376–7, 389, 549
processors 107–35, 346–53
assembly language 121–9, 541
bit manipulation 130–2
CPU architecture 107–20
parallel processing 350–3
RISC and CISC processors 347–50
process priority 377
process states 373, 377–82, 549
product key 186, 549
professional ethical bodies 180–3
program counter (PC) 108, 116, 117, 549
program design 287–93
program development lifecycle 283–7, 549
different development lifecycles 285–7
purpose 284
stages 284–5
program libraries 138, 147–8, 549
programmable ROM (PROM) 69, 72, 549
program maintenance 283, 284, 285, 286, 294, 299, 547
program testing 283, 284, 285, 286, 293–4, 296–9, 551
programming 264–82, 498–540
basics 264–71
constants and variables 265–71
constructs 271–5
exception handling 525, 535–7, 544
file processing operations 525–35
library routines 264, 271
structured 275–80
programming paradigms 498–525, 549
declarative programming 499, 521–4, 543
imperative programming 498, 500–1, 546
low-level programming 498, 499–500, 547
OOP 498, 501–21, 548
properties 498, 502, 549
protocols 328–37, 549
security and 416–18
prototyping 287

pseudocode 219, 220, 221–33, 549
structure charts 289–91
writing algorithms using 221–9
writing from a flowchart 231–3
writing from a structured English description 229–31
public cloud 40
public impact of hardware or software 183–5
public IP addresses 54, 61, 549
public key infrastructure (PKI) 416, 418, 549
public keys 410, 413–14, 549
public networks 31
public switched telephone network (PSTN) 54, 55, 549
pull protocols 329, 332–3, 549
push protocols 329, 332, 549
Python 228, 239, 271, 273
binary search 456, 457
bubble sort 459
constants and variables 266, 267, 268
exception handling 536
file processing 527–8, 533
functions 278, 280
IF statement 224
insertion sort 463
linear search 452
linked lists 471, 476, 479
loops 274, 275
OOP 502, 503, 506, 510, 513
procedures 275, 276, 277, 278
queues 466, 467, 468
recursion 491
stacks 464, 465
writing programs for binary trees 518, 519, 520

Q

quad core 108, 113, 549
quantum 373, 376, 379, 381, 549
quantum cryptography 414–15, 549
quantum key distribution (QKD) 414, 549
quarantine 137, 144, 549
qubit 414, 549
query processor 209, 210, 549
queues 238, 250–1, 464, 466–9, 549
queue operations 253–5

R

radio waves 42–3
random access memory (RAM) 68, 70–1, 72, 374, 385, 549
random file organisation 308, 309, 310, 549
  adding records to random files 533–5
  finding records in random files 535
range 323–4
rapid application development (RAD) 283, 286–7, 549
read file access mode 525, 526–7, 549
read-only memory (ROM) 68, 70, 71–2, 549
ready state 377–8
real-time (bit streaming) 29, 53, 549
record protocol 417
records
  database 196, 199–200, 549
  data type 238, 240–1, 549
recursion 490–4, 549
referential integrity 197, 201, 549
refreshed 68, 71, 550
registers 109, 110–11, 550
Register Transfer Notation (RTN) 108, 117–18, 550
regression 435, 445, 550
reinforcement learning 434, 439, 550
relational databases 196, 198–207, 550
relationships 197, 201–2, 550
relative addressing 121, 126, 550
removable hard disk drives 69, 74, 550
repeaters 28, 46–7, 550
repeating hubs 28, 46–7, 550
REPEAT … UNTIL loops 225, 226, 227, 274
report window 150, 155–6, 550
resistive touch screens 69, 83, 550
resolution 15, 17, 550
resource management 374–6
retina scans 164
RETURN 279
Reverse Polish notation (RPN) 394, 400–1, 550
reward and punishment 434, 439, 550
right shift 130, 131, 550
RISC (reduced instruction set computer) 347–8, 550
robotics 190
ROM (read-only memory) 68, 70, 71–2, 549
rounding errors 320–2
round robin scheduling 373, 378–9, 381, 550
routers 28, 47–8, 49, 330, 550
routing 38
routing tables 337, 341–2, 550

rules 499, 521–4, 550
run-length encoding (RLE) 21, 22–4, 550
  with images 23–4
  with text data 23
running state 377–8
runtime environment with a debugger 155–6
run-time errors 294, 296, 550

S

sampling rate 15, 20, 24, 550
sampling resolution/bit depth 15, 16, 20, 24, 542, 550
satellites 43, 56–7
scalable vector graphics (SVG) 22
scheduling 373, 374, 376–82, 550
  routines 379–81
screen resolution 15, 16–17, 69, 82, 550
screens 82–4
secondary keys 197, 200, 550
secondary storage 70, 72–7
second normal form (2NF) 197, 203, 205, 550
Secure Sockets Layer (SSL) 416–17, 417–18, 550
  digital certificate 421
security see *data security*
security management 137, 141, 550
seeds 329, 336, 337, 550
segmentation 384–5
segment map table 373, 384, 550
segment numbers 373, 384, 550
segments
  memory 373, 384–5, 550
  transport layer 329, 330, 550
semi-supervised (active) learning 434, 439, 550
sensors 69, 84–7, 550
sequential access 308, 309–10, 550
sequential circuits 354, 358, 550
sequential file organisation 308, 309–10, 550
  adding records to sequential files 531–3
  storing records in sequential files 526–31
serial access 550
serial file organisation 308, 309, 531, 550
  storing records in serial files 526–31
services 30
session caching 416, 417, 550
sets 305, 307, 550
setters 499, 515, 516, 551
shareware 186, 189, 551

shifts 130–1, 551
shortest job first (SJF) scheduling 379–80, 381
shortest path algorithms 425–34
shortest remaining time first (SRTF) scheduling 379–80, 381
sign and magnitude 2, 3, 551
SIMD (single instruction multiple data) 347, 350, 352, 551
simple mail transfer protocol (SMTP) 329, 330, 332, 351
simplification
  of logic circuits 101
  using Boolean algebra 355–6
single (contiguous) memory allocation 383
single pass assemblers 122
single stepping 150, 155, 551
SISD (single instruction single data) 347, 350, 551
SMTP (simple mail transfer protocol) 329, 330, 332, 551
softmodem 28, 49, 551
software 30, 136–58
  cloud software 41
  copyright and privacy 186–7
  language translation 149–57, 394–402
  licensing 187–9
  needed to support the internet 55–7
  operating systems see *operating systems*
software development 283–303
  program design 287–93
  program development lifecycle 283–7
  program testing and maintenance 293–300
Software Engineering Code of Ethics 181–3
solid state drives (SSDs) 69, 74–5, 551
sound files 19–20
source code 121, 122, 551
source code editor 154–5
space complexity 490
speakers 80–1
spread spectrum frequency hopping 28, 41–2, 551
spread spectrum technology 28, 31, 551
spyware 165
SQL scripts 211–14, 521, 551
SR flip-flops 358–60
stacks 238, 250–3, 464–6, 551
  stack operations 251–3
star network topology 28, 37–8, 39, 551
starving a process 373, 376, 551
state-transition diagrams 287, 292–3, 551
state-transition tables 287, 292, 551
static libraries 148

static RAM (SRAM) 68, 70–1, 551
status register 108, 109, 110, 111, 551
stepwise refinement 219, 233–5, 551
storage devices 69–77
stream cipher 410, 411, 551
strings 269
  manipulation functions 269–71
strong AI 435
structure charts 287, 288–92, 551
structured English 219, 220, 551
  writing pseudocode from a structured English description 229–31
structured programming 275–80
structured query language (SQL) 209, 210, 211, 551
  SQL scripts 211–14, 521, 551
stub testing 294, 299, 551
sub-netting 54, 59–61, 551
subtraction 5–6
sum of products (SoP) 354, 361, 551
super computers 347, 352, 551
supervised learning 434, 438, 551
swap space 373, 385, 551
swarm 329, 336, 551
switches 28, 37, 46, 551
symbolic addressing 121, 126, 551
symmetric encryption 410, 411–12, 551
syntax analysis 394, 395, 397, 551
syntax diagrams 394, 398–400, 551
syntax errors 150, 155, 295, 551
system buses 109, 112–14
system clock 108, 109, 110, 113, 551
system software 136–58
  language translation 149–57, 394–402
  operating systems see *operating systems*

T

tables 196, 199–200, 551
TCP (transmission control protocol) 329, 333–4, 551
TCP/IP protocols 57, 329–37
terminology databases 441
test data 298
testing 283, 284, 285, 286, 293–4, 296–9, 551
test plans 294, 296, 298, 551
test strategy 294, 296, 551
text data, RLE on 23
text files 249–50
text mining 441

thermal bubble technology 78
thick clients 28, 35–6, 551
thin clients 28, 35–6, 551
third normal form (3NF) 197, 203–4, 206–7, 551
thrash point 373, 386, 551
time complexity 489
timeout 170, 175, 551
tokenisation 396
touch screens 69, 82–3, 552
trace tables 128, 294, 295, 297, 552
tracker 329, 336, 552
translation lookaside buffer (TLB) 373, 383, 552
translation memories 441
translation software 149–57, 394–402
translators 149, 150–1, 552
transmission control protocol (TCP) 329, 333–4, 551
  TCP/IP protocols 57, 329–37
transport 192
transport layer 329, 330, 333–4
Transport Layer Security (TLS) 416, 417–18, 552
Trojan horses 165
truth tables 89, 90–8, 552
tuples 197, 200, 552
twisted pair cables 28, 44, 552
two pass assemblers 122–3
two's complement 2, 3–4, 552

U

unconditional instructions 125
underflow errors 313, 325, 552
Unicode 2, 14–15, 552
unidirectional buses 108, 112, 552
uniform resource locators (URLs) 54, 55, 61, 552
Universal Serial Bus (USB) ports 108, 114–15, 552
unlabelled data 434, 437, 440–1, 552
unsupervised learning 434, 438–9, 552
unwinding 490, 491, 552
upper bound 238, 241–2, 552
USB ports 108, 114–15, 552
use of data 193
user accounts 159, 160–1, 552
user-defined data types 304–7, 552
utility programs 137, 143–6, 552

V

validation 169–70, 552
variables 264, 265–71, 552
VB 228, 239, 271
  binary search 456, 457
  bubble sort 459–60
  case statements 273
  constants and variables 266, 267, 268
  exception handling 536–7
  file processing 528–9, 533
  functions 278, 280
  IF statement 224
  insertion sort 463
  linear search 453
  linked lists 472, 476, 479–80
  loops 274, 275
  OOP 502, 504, 506–7, 510–11, 513–14
  procedures 275, 276, 277, 278
  queues 466, 467, 468
  recursion 492
  stacks 464, 465
  writing programs for binary trees 518, 519, 520
vector graphics 15, 18–19, 552
  file compression 22
verification 169, 170–6, 552
  during data entry 170–2
  during data transfer 172–5
vertices (nodes) 425–34, 547
video 20–1, 24
Video Graphics Array (VGA) 108, 115–16, 552
virtual machines (VMs) 392–4, 552
virtual memory 373, 385–7, 552
virtual memory systems 137, 140, 552
virtual reality headsets 69, 83–4, 552
virus checkers 144
viruses 164–5
visual check 171
Voice over Internet Protocol (VoIP) 54, 55, 56, 552
Von Neumann architecture 108, 109, 552

W

walkthrough 294, 298, 552
WANs (wide area networks) 28, 29–30, 32, 552
WAPs (wireless access points) 28, 31, 552
waterfall model 283, 285, 552
web browsers 54, 55, 61, 552
web crawler 435, 439, 552

WHILE … DO … ENSEMBLE 274–5
white-box testing 294, 299, 552
wide area networks (WANs) 28, 29–30, 32, 552
Wi-Fi 28, 41–2, 552
WiMax (worldwide interoperability for microwave access) 335
WIMP (windows, icons, menu and pointing device) 137, 138, 552
winding 490, 491, 552
wired networking 43–5
  vs wireless 44–5
wireless access points (WAPs) 28, 31, 552
wireless LANs (WLANs) 28, 31, 552
wireless networking 41–3, 44–5
wireless network interface cards/controllers (WNICs) 29, 50, 552
wireless personal area networks (WPANs) 28, 42, 552
wireless (Wi-Fi) protocols 335
word 108, 112, 552
World Wide Web (WWW) 54–5, 187, 552
worms 165
WPANs (wireless personal area networks) 28, 42, 552
write file access mode 525, 526–7, 552

X

XOR gates 89, 92

Z

zero compression 54, 58–9, 552
zero value 325