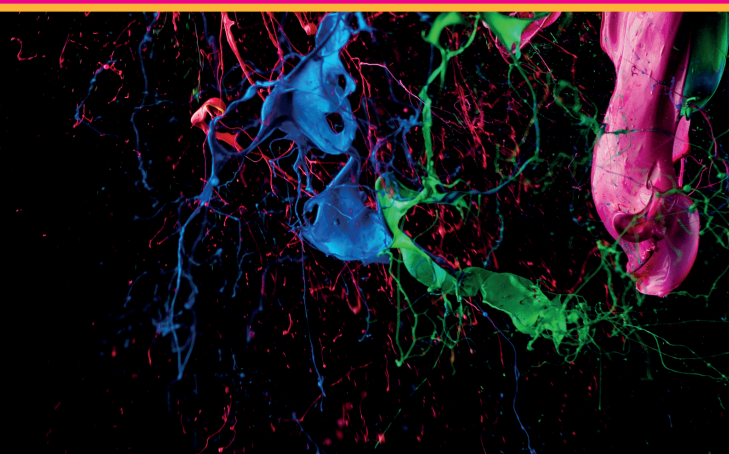
CAMBRIDGE

Brighter Thinking

GCSE for AQA

COMPUTER SCIENCE

Teacher's Resource



CAMBRIDGE

Brighter Thinking

GCSE for AQA

Teacher's Resource

Ann Weidmann

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org

Information on this title:

www.cambridge.org/9781316504116 (Elevate edition)

www.cambridge.org/9781316504123 (Free online)

First published 2016

A catalogue record for this publication is available from the British Library

ISBN 978-1-316-50411-6 Elevate edition

ISBN 978-1-316-50412-3 Free online

Additional resources for this publication at **www.cambridge.org/education**

Contents

Introduction

This book has been written to support you in delivering the **AQA GCSE Computer Science (8520)** specification. It accompanies Cambridge University Press's GCSE Computer Science for AQA student textual and online resources.

The structure of the Teacher's Resource closely matches the Student Book which is divided into chapters and sections covering the specification content. The Teacher's Resource contains a chapter of teaching guidance for each Student Book chapter.

An introduction details **learning outcomes and what your students need to know** in terms of prior learning in order to tackle the contents of the chapter. **Key vocabulary** is introduced. The introduction also describes **common misconceptions**, suggesting how to address them, and offers **'hooks'** to introduce the topics in an engaging way.

A **skills and coding** section lists the **maths** and **coding** skills that are covered in the chapter.

The **Skills and coding for non-specialist teachers** section covers the programming and coding introduced in each chapter in greater depth in addition to explaining specialist concepts such as decomposition and abstraction.

The Teacher's Resource also provides:

- **prompting questions**, to promote discussion of the topic
- suggested activities for:
  - **starters**,
  - **plenaries**,
  - **enrichment** and
  - **assessment** for each chapter of the Student Book
- **full solutions and answers** for all the chapter activities.

A **scheme of work** is available to download, providing learning outcomes, suggested teaching times and showing how the chapter topics cover the items in the specification:

www.cambridge.org/gb/education/samples-and-schemes-work/gcse-computer-science-samples

Advice is also provided for the Non-examination assessment (NEA) component.

Changes to GCSE Computer Science

This chapter offers an overview of the main changes to the GCSE computer science qualification for first teaching from September 2016.

Introduction

The AQA computer science qualification contains the **GCSE Subject Level Conditions and Requirements for Computer Science** (Ofqual/15/5681), published by Ofqual in May 2015.

This can be downloaded from

www.gov.uk/government/publications/gcse-9-to-1-subject-level-conditions-and-requirements-for-computer-science

The document stipulates the assessment objectives, the nature of the non-examination assessment and the subject aims and content that must be incorporated by all examination boards.

Grades

The new GCSE is significantly different from its predecessor. Students will now be graded on a 1–9 scale. If you wish to read more about grades, the latest information can usually be found on the websites of Ofqual and the awarding bodies.

Assessment

The GCSE (9–1) in Computer Science is a linear qualification with a 100% terminal rule.

There are three components: two externally examined components (01 and 02) weighted at 40% each and a Non-examination assessment (03) weighted at 20%, which is assessed by the centre and externally moderated by AQA.

Each examined component consists of an exam paper with a duration of 1 hour 30 minutes. The Non-examination assessment has a duration totalling 20 hours. Students must take all three components.

There will be one examination series available each year in May/June to **all** students and all examined components must be taken in the same examination series at the end of the course.

Students will be able to retake the examination as many times as they wish and can choose either to retake the Non-examination assessment or to carry forward their mark from their previous sitting.

The assessment objectives and their weightings are given in the following table:

	Assessment Objective	Weighting
AO1	Demonstrate knowledge and understanding of the key concepts and principles of Computer Science.	30%
AO2	Apply knowledge and understanding of key concepts and principles of Computer Science.	40%
AO3	Analyse problems in computational terms: to make reasoned judgements to design, program, evaluate and refine solutions.	30%

Subject content

The following tables show the changes in subject content between the new AQA (8520) Computer Science specification and the old 4512 Computer Science specification that it replaces.

Learning objectives in 4512 that are not in 8520	
Specification reference	**Statement**
3.1.1	Understand what is meant by the terms data and information
3.1.6	Be able to discuss and identify the different types of errors that can occur within code (ie syntax, run-time and logical)
	Understand that some errors can be detected and corrected during the coding stage
	Understand that some errors will occur during the execution of the code
	Know how to detect errors at execution time and how to handle those errors to prevent the program from crashing where desirable
	Be able to use trace tables to check their code for errors
	Understand that computer programs can be developed with tools to help the programmer detect and deal with errors (eg Watch, Breakpoint, Step)
3.1.7 Handling external data	Know how to read and write data from an external database in a way that is appropriate for the programming language(s) used and the problem being solved
3.1.8.1 Systems	Be able to define a computer system (ie hardware and software working together to create a working solution)
3.1.8.2 Hardware	Be able to categorise devices as input or output depending on their function
3.1.8.4 Memory	Know the differences between non-volatile and volatile memory
	Understand the purpose of both types of memory and when each should be used
3.1.9 Algorithms	Be able to detect and correct errors in simple algorithms
3.1.11 Software development life cycle	Understand the software development life cycle
	Be able to explain what commonly occurs at each stage of the software development life cycle
	Be able to identify at which stage of the software development life cycle a given step would occur
	Understand that there are several lifecycle models that can be used (eg cyclical, waterfall, spiral)
	Be able to discuss the advantages and disadvantages of these lifecycle models
3.1.11.1 Prototyping	Understand what prototyping is
	Be able to discuss the advantages and disadvantages of using prototyping when developing solutions
	Have experience of using prototyping to create solutions to simple problems
3.1.12 Application testing	Understand the different types of tests that can be used, including unit/modular testing
3.1.13 Networking	Ring network is not now included
3.1.13.1 Client server	Understand the client-server model
	Be able to explain, in simple terms, the handshake process used in most modern networking protocols
	Be able to explain how coding for a client-server model is different from coding for a stand-alone application
3.1.13.2 Web application concepts	Understand the concept of coding at the server and client end
	Know what can be coded at the server end
	Know what can be coded at the client end
	Have experience of coding solutions to simple web application problems

Learning objectives in 4512 that are not in 8520	
Specification reference	**Statement**
3.1.14 Use of external code sources	Know of the existence of external code sources
	Know how to integrate code from these sources into their own code
	Be able to explain the advantages and disadvantages of using such sources
3.1.15 Database concepts	Understand the basic concepts of a relational database as a data store
	Be able to explain the terms record, field, table, query, primary key, relationship, index and search criteria
3.1.15.1 Query methods (SQL)	Be able to create simple SQL statements to extract, add and edit data stored in databases
	Have experience of using these SQL statements from within their own coded systems
3.1.15.2 Connecting to databases from applications and web based apps	Be able to use databases from within their own web based applications
3.1.16 The use of computer technology in society	Be able to evaluate the effectiveness of computer programs/solutions

Learning objectives in 8520 that were not in 4512	
Specification reference	**Statement**
3.1.1 Representing algorithms	Understand and explain the term decomposition
	Understand and explain the term abstraction
	Explain simple algorithms in terms of their inputs, processing and outputs
3.1.2 Efficiency of algorithms	Understand that more than one algorithm can be used to solve the same problem
	Compare the efficiency of algorithms explaining how some algorithms are more efficient than others in solving the same problem
3.1.3 Searching algorithms	Understand and explain how the linear search algorithm works
	Understand and explain how the binary search algorithm works
	Compare and contrast linear and binary search algorithms
3.1.4 Sorting algorithms	Understand and explain how the merge sort algorithm works
	Understand and explain how the bubble sort algorithm works
	Compare and contrast merge sort and bubble sort algorithms
3.2.2 Programming concepts	Use nested selection and nested iteration structures
	Use meaningful identifier names and know why it is important to use them
3.2.4 Relational operations in a programming language	Be familiar with and be able to use: • equal to • not equal to • less than • greater than • less than or equal to • greater than or equal to
3.2.12 Robust and secure programming	Be able to write simple data validation routines
	Be able to write simple authentication routines

Learning objectives in 8520 that were not in 4512	
Specification reference	**Statement**
3.2.13 Classification of programming languages	Know that there are different levels of programming language: • low-level language • high-level language Explain the main differences between low-level and high-level languages
	Know that machine code and assembly language are considered to be low-level languages and explain the differences between them
	Understand that ultimately all programming code written in high-level or assembly languages must be translated into machine code Understand that machine code is expressed in binary and is specific to a processor or family of processors
	Understand the advantages and disadvantages of low-level language programming compared with high-level language programming
	Understand that there are three common types of program translator: • interpreter • compiler • assembler Explain the main differences between these three types of translator Understand when it would be appropriate to use each type of translator
3.3.4 Binary arithmetic	Be able to add together up to three binary numbers
	Be able to apply a binary shift to a binary number
	Describe situations where binary shifts can be used
3.3.5 Character encoding	Understand that character codes are commonly grouped and run in sequence within encoding tables
	Describe the purpose of Unicode and the advantages of Unicode over ASCII Know that Unicode uses the same codes as ASCII up to 127
3.3.6 Representing images	Understand what a pixel is and be able to describe how pixels relate to an image and the way images are displayed
	Describe the following for bitmaps: • size in pixels • colour depth
	Describe how a bitmap represents an image using pixels and colour depth
	Describe using examples how the number of pixels and colour depth can affect the file size of a bitmap image
	Calculate bitmap image file sizes based on the number of pixels and colour depth
	Convert binary data into a black and white image
	Convert a black and white image into binary data
3.3.7 Representing sound	Understand that sound is analogue and that it must be converted to a digital form for storage and processing in a computer
	Understand that sound waves are sampled to create the digital version of sound
	Describe the digital representation of sound in terms of: • sampling rate • sample resolution
	Calculate sound file sizes based on the sampling rate and the sample resolution

Learning objectives in 8520 that were not in 4512	
Specification reference	**Statement**
3.3.8 Data compression	Explain what data compression is. Understand why data may be compressed and that there are different ways to compress data
	Explain how data can be compressed using Huffman coding Be able to interpret Huffman trees
	Be able to calculate the number of bits required to store a piece of data compressed using Huffman coding Be able to calculate the number of bits required to store a piece of uncompressed data in ASCII
	Explain how data can be compressed using run length encoding (RLE)
	Represent data in RLE frequency/data pairs
3.4.2 Boolean logic	Construct truth tables for the following logic gates: • NOT • AND • OR
	Construct truth tables for simple logic circuits Interpret the results of simple truth tables
	Create, modify and interpret simple logic circuit diagrams
3.4.3 Software classification	Explain what is meant by: • stem software • application software Give examples of both types of software
	Understand the need for, and functions of, operating systems (OS) and utility programs Understand that the OS handles management of the: • processor(s) • memory • I/O devices • applications • security
3.4.4 Systems architecture	Explain the Von Neumann architecture
	Explain the role and operation of main memory and the following major components of a central processing unit (CPU): • arithmetic logic unit • control unit • clock • bus
	Understand and explain the Fetch-Execute cycle
	Explain the term 'cloud storage'
	Explain the advantages and disadvantages of cloud storage when compared to local storage
	Understand the term 'embedded system' and explain how an embedded system differs from a non-embedded system
3.5 Fundamentals of computer networks	Describe the main types of computer network including: • Personal Area Network (PAN) • Local Area Network (LAN) • Wide Area Network (WAN)

Learning objectives in 8520 that were not in 4512	
Specification reference	**Statement**
	Understand that networks can be wired or wireless Discuss the benefits and risks of wireless networks as opposed to wired networks
	Define the term 'network protocol'
	Explain the purpose and use of common network protocols including: • Ethernet • Wi-Fi • TCP (Transmission Control Protocol) • UDP (User Datagram Protocol) • IP (Internet Protocol) • HTTP (Hypertext Transfer Protocol) • HTTPS (Hypertext Transfer Protocol Secure) • FTP (File Transfer Protocol) • email protocols: SMTP (Simple Mail Transfer Protocol) IMAP (Internet Message Access Protocol)
	Understand the need for, and importance of, network security
	Explain the following methods of network security: • authentication • encryption • firewall • MAC address filtering
	Describe the 4 layer TCP/IP model: • application layer • transport layer • network layer • data link layer Understand that the HTTP, HTTPS, SMTP, IMAP and FTP protocols operate at the application layer Understand that the TCP and UDP protocols operate at the transport layer Understand that the IP protocol operates at the network layer
3.6 Fundamentals of cyber security	Be able to define the term 'cyber security' and be able to describe the main purposes of cyber security
3.6.1 Cyber security threats	Understand and be able to explain the following cyber security threats: • social engineering techniques • malicious code • weak and default passwords • misconfigured access rights • removable media • unpatched and/or outdated software
	Explain what penetration testing is and what it is used for

Learning objectives in 8520 that were not in 4512	
Specification reference	**Statement**
3.6.1.1 Social engineering	Define the term 'social engineering' Describe what social engineering is and how it can be protected against Explain the following forms of social engineering: • blagging (pretexting) • phishing • pharming • shouldering (or shoulder surfing)
3.6.1.2 Malicious code	Define the term 'malware' Describe what malware is and how it can be protected against Describe the following forms of malware: • computer virus • trojan • spyware • adware
3.6.2 Methods to detect and prevent cyber security threats	Understand and be able to explain the following security measures: • biometric measures (particularly for mobile devices) • password systems • CAPTCHA (or similar) • using email confirmations to confirm a user's identity • automatic software updates
3.7 Ethical, legal and environmental impacts of digital technology on wider society, including issues of privacy	Explain the current ethical, legal and environmental impacts and risks of digital technology on society Where data privacy issues arise these should be considered

Non-examination assessment

For the first examination in 2018, the following programming languages will be supported:

- C#
- Java
- Pascal/Delphi
- Python
- VB.Net

The assessment criteria are compared in the following table:

Criterion	8520 marks	4512 marks
Designing the solution **Design of solution**	9	 **9**
Creating the solution **Solution development**	30	 **9**
Programming techniques used		**36**
Testing the solution **Testing and evaluation**	21	 **9**
Potential enhancements and refinements	10	
Overall quality of the report	10	
Total	80 marks	**63 marks**

Further details of 8520 Non-examination assessment (NEA) are given in the Teacher's Resource and the Student Book.

Chapter 1: Algorithms

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what an algorithm is and create algorithms to solve specific problems
- use sequence, selection and iteration in algorithms
- use input, processing and output in algorithms
- express algorithms using flowcharts and pseudo-code
- analyse, assess and compare different algorithms
- create, name and use suitable variables
- use arithmetic, relational and Boolean operators
- use conditional statements.

What your students need to know

No prior knowledge is expected for this chapter.

Vocabulary

- Algorithm
- Sequence
- Sub-tasks
- Iteration
- Selection
- Flowchart
- Decision
- Process
- Authenticate
- Variable
- Pseudo-code
- Identifier
- Assigning
- Constant
- Comment
- Operator
- Operand
- Parentheses
- Arithmetic operators
- Relational operator
- Logical (Boolean) operator
- Input, output and processing
- Nested operations

Common misconceptions and other issues

Students should be encouraged to use the conventional flowchart symbols for input/output, process and decision boxes, although the AQA specification does not specifically require them to do so.

Students must be familiar with the AQA standard pseudo-code, since any exam question requiring pseudo-code will use this version. Although they don't have to use it themselves, it's probably a good idea for them to do so. That said, any form of pseudo-code is acceptable providing it is clear and unambiguous.

Students should use meaningful identifiers for variables and constants (for example, 'Age' for a variable about age rather than 'X'). They should also use indentation and commenting in their pseudo-code. This will make their code easier to read and facilitate error checking.

Students may be unfamiliar with the arithmetic operators MOD and DIV. Give examples similar to those shown in the arithmetic operators table in the Student Book.

The Boolean operators 'AND' and 'OR' can cause confusion. It's easy to translate a correctly phrased spoken-language statement incorrectly. For example, it's correct to say 'I want a bag of purple and orange fruit'. In all likelihood you'd get a mixed bag of plums and oranges. In code, however, the expression 'bag ← purpleFruit AND orangeFruit' would result in an empty bag, since there are no fruit (as far as I'm aware) that are two-tone, purple and orange. The expression 'bag ← purpleFruit OR orangeFruit' will result in a bag containing fruit that meet either of the two criteria.

Brackets can be used for grouping the parts of complex Boolean expressions. For example: fedUp ← (cold AND hungry) OR (hot AND thirsty).

When using nested selection, care must be taken to complete each block with an 'ENDIF' statement.

Skills and coding

- Maths skills:
  - Arithmetic operators
  - Order of operations (BIDMAS)
  - Calculation of average
- Coding skills:
  - Use of pseudo-code
  - Declaring and assigning variables
  - Selecting suitable identifiers
  - Selection using 'IF…THEN…ELSE' statements
  - Nested 'IF' statements
  - Use of 'CASE' statements

Skills and coding for non-specialist teachers

1 Use of pseudo-code

The pseudo-code syntax and meanings are given in the *AQA Pseudo-code Guide*. In the AQA specification, under **Topic 3.1.1** of the Subject Content, it states that:

'Any exam question where students are given pseudo-code will use the AQA standard version. However, when students are writing pseudo-code they may do so using any form as long as the meaning is clear and unambiguous.'

2 Declaring and assigning variables and selecting suitable identifiers

A variable is defined as a 'named container' for a value that can change as a program is running.

Values are assigned using the '←' symbol. (The '=' symbol is used to compare two values.)

Variables should be given meaningful names (identifiers) and the naming convention should be consistent. For example: firstName or first_name or FIRST_NAME.

3 Selection using an 'IF' statement

The 'IF' statement is explained under the section on 'Relational operators' in the Student Book and its use is demonstrated in the solutions to **Activity 1.9** and **Activity 1.10**.

An 'IF' statement is used to control the flow of a program. The section of code within the 'IF' statement is executed only if the condition is met. The 'ENDIF' command should always be used to terminate an 'IF' statement. For example:

```
IF index = 3 THEN
        OUTPUT("The variable index is equal to 3.")
ENDIF
```

If there are two alternative actions, 'ELSE' is used. For example:

```
IF index = 3 THEN
        OUTPUT("The variable index is equal to 3.")
ELSE
        OUTPUT("The variable index is not equal to 3.")
ENDIF
```

The 'ELSE IF' statement allows for multiple options. For example:

```
IF index = 3 THEN
        OUTPUT "The variable index is equal to 3."
ELSE IF index > 3 THEN
        OUTPUT "The variable index is greater than 3."
ELSE
        OUTPUT "The variable index is less than 3."
ENDIF
```

4 Nested 'IF' statements

A nested 'IF' statement consists of two 'IF' statements, one embedded within the other. This is explained in the Student Book using a worked example.

Care must be taken with indentation when using nested 'IF' statements. It is also important to terminate each 'IF' statement with an 'ENDIF' command.

In this example, the outer 'IF' statement calculates a discount for purchases over £100, while the inner 'IF' statement ensures that the maximum discount given is £20:

```
IF price ≥ 100 THEN
        discount ← price/100*10
        IF discount > 20 THEN
                discount ← 20
        ENDIF
        price ← price – discount
        OUTPUT price
ENDIF
```

5 'CASE' statements

A 'CASE' statement provides another way of handling multi-branching selection. It is especially useful for choosing one item from a list of items. The 'ELSE' command at the end deals with an inappropriate/invalid choice. Here is an example:

```
OUTPUT "Enter a number between 1 and 5."
number ← USERINPUT
CASE number OF
        1: OUTPUT "One Monkey"
        2: OUTPUT "Two Elephants"
        3: OUTPUT "Three Zebras"
        4: OUTPUT "Four Hippos"
        5: OUTPUT "Five Lions"
ELSE
        OUTPUT "Invalid number"
ENDCASE
```

This is explained using a worked example that shows both the 'ELSE IF' and 'CASE' methods in the Student Book and is illustrated in the solution to **Activity 1.12**.

Prompting questions

- What is an algorithm? Have you heard this term before? For which subject, in which context?
- Can you think of any algorithms that you follow in your everyday life?
- Can you think of any activities where you use selection or iteration?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- The human robot: Ask students to consider a routine that is part of their daily life. Can they write down a set of instructions that will allow someone else to follow their routine completely? A useful extension to this activity is to ask students to pair up and role play their partner's instructions. The students can then provide each other with feedback and discuss the quality of those instructions and what can make them better. This is a simple way to enable students to begin writing and testing their own algorithms.
- Divide the class into pairs. One partner sees an image and is going to be the programmer, the other is going to be the human computer and will carry out the instructions they receive. It is important that the human computer doesn't see the image being used by the programmer. The activity is simple: the programmer describes the image they see in front of them and the human computer tries to recreate it. This can be done using paper and pen or by getting creative and using playdough instead!

Enrichment activities

There is magic in computer science. The following resource **www.cambridge.org/links/katd4002** uses magic tricks to explain algorithms and many other computer science concepts. Choose a trick that works for you as a teacher and demonstrate it to the students. Challenge the students to see if they can figure out how the trick works. Encourage them to write the algorithm for it as best they can, refining it as they go and making use of the concepts learnt.

Assessment ideas

- Set the students a challenge using the following scenario: Write an algorithm for a Joke Generator program. When the user runs the program, it begins by displaying a joke on screen. The user is then asked if they would like another joke; if the user answers 'yes', another joke appears on the screen.
- This version of the program makes use of simple selection and sequence statements. Nested 'IF' and 'CASE' statements can be added to enable the user to input a number between 1 and 10. Once the number has been entered, the program will display the joke that corresponds to that number. Ask students to submit the following:
  - A flowchart illustrating how their program will work
  - An algorithm expressed in pseudo-code, annotated to explain how it works.

Answers

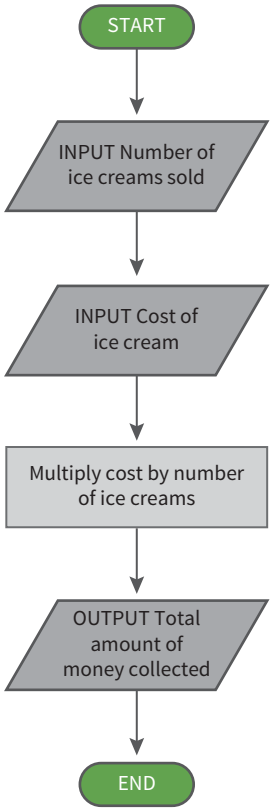
Activity 1.1

- Students will write this as a list of steps. There is no single right answer to this activity and this is important to emphasise. The key thing is to check the logic at this stage: does it work? Encourage students to mime their algorithm to try it out. Often students will forget to write in when to stop pouring the water or milk, for example. The activity emphasises the need for instructions to be specific and precise, concisely written and in a logical order.
- An example of how the algorithm might look is:
  - Fill kettle with cold water to maximum level
  - Check to see if kettle is plugged in
  - If it isn't plugged in, then plug it in
  - Check to see if kettle is switched on
  - Switch it on if it isn't
  - Is kettle boiled?
  - If No, then wait until it has boiled
  - Put 1 teabag into cup
  - Pour boiled water into cup until full
  - Leave for 1 minute
  - Stir for 20 seconds
  - Remove tea bag and place in bin
  - Pour in 10ml of milk
  - Is sugar needed?
  - If No, then end
  - If Yes, then add required sugar

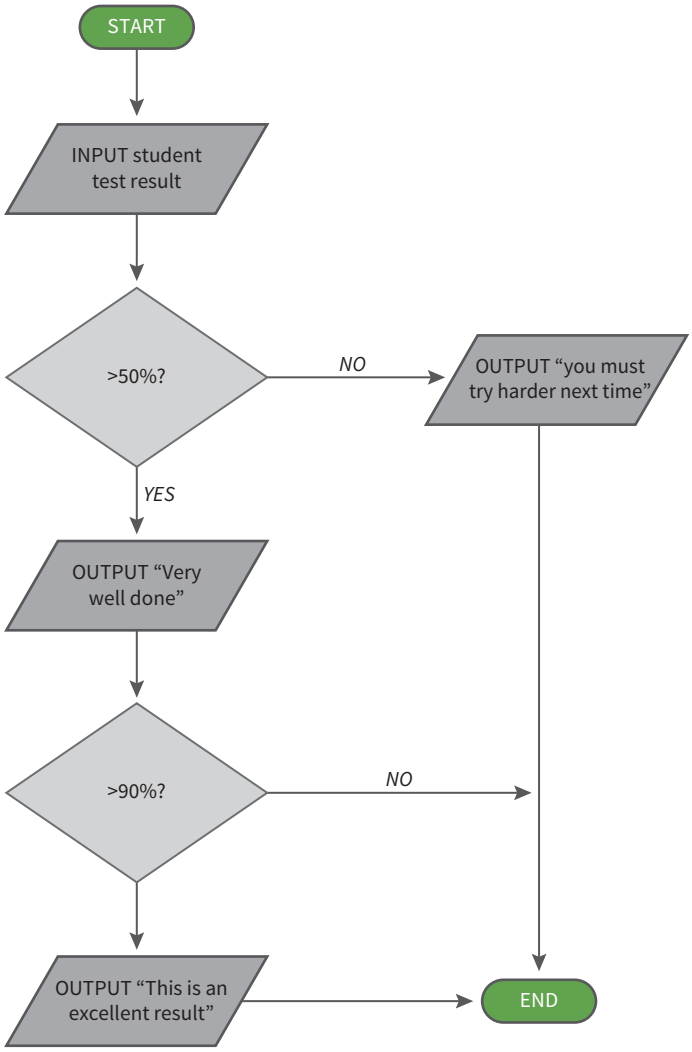
Activity 1.2

- It doesn't matter at this stage how the answer to this activity is presented. The key success criteria are: getting the logic right; writing clear, precise instructions; and correctly identifying them as sequence, selection or iteration.
  - Put plug in the bath # Sequence
  - Turn on hot tap # Sequence
  - Is the water at the correct temperature? # Selection
  - Is it too hot? # Selection
  - Turn cold tap until water is at the correct temperature # Iteration
  - Is the water at the correct temperature? # Selection
  - Is it too cold? # Selection
  - Turn off cold tap until water is at the correct temperature. # Iteration
  - Is there enough water in the bath? # Selection
  - Turn off both taps. # Sequence

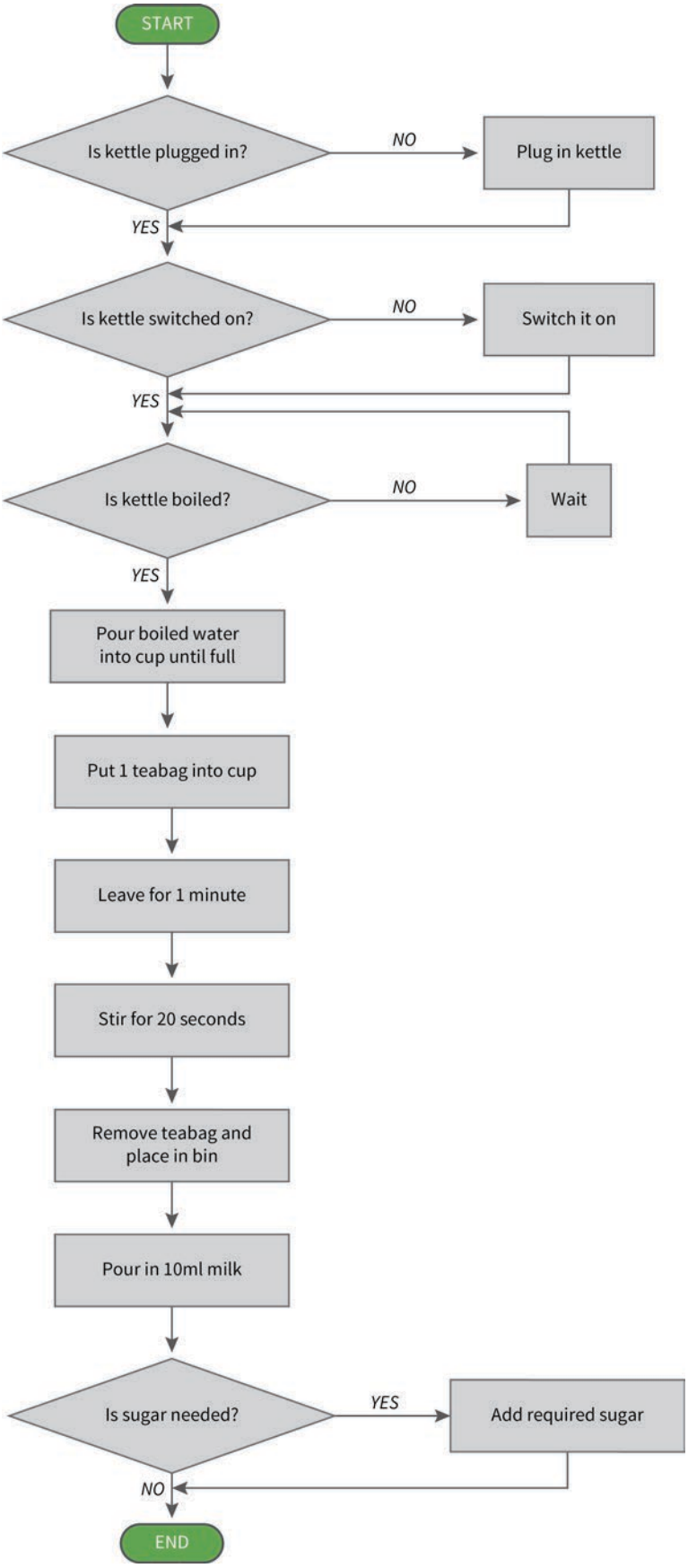
Activity 1.3



Activity 1.4



Activity 1.5



Activity 1.6

The most obvious variables used are: USERNAME, YEAR, FIRST, LAST and X. However, the inputs: *year of entry*, *first name* and *surname* are also variables.

Catherine Jones 2005 becomes '05CJones1'

Fred Green 2006 becomes '06FGreen1'

Username '03SSmith13' tells us the following:

- The user joined in 2003
- Their first name begins with S
- Their surname is Smith
- There are now 13 students in the school with the surname Smith.

Activity 1.7

```
OUTPUT "Please enter your name."
Name ← USERINPUT
OUTPUT "Please enter your age."                    # Asks the user for their age
Age ← USERINPUT
OUTPUT "Hello " + Name + ". You are " + Age + " years of age." # Outputs a personalised message
```

Activity 1.8

```
OUTPUT "Please enter the diameter of the wheel."
Diameter ← USERINPUT
Radius ← Diameter/2                    # Calculates radius as half of the diameter
Pi ← 3.142
Area ← Pi * (Radius * Radius)          # Calculates area by multiplying Pi by radius squared
OUTPUT "The area of the wheel is: " + Area
```

Activity 1.9

```
OUTPUT "Please enter a number between 1 and 10."
Number ← USERINPUT
IF Number ≤ 5 THEN
        OUTPUT Number + " is a low number."
ELSE
        OUTPUT Number + " is a high number."
ENDIF
```

Activity 1.10

- The comment generated would have been "You have gained half marks."
- The algorithm checks first to see if the score entered is less than 5. If this is not the case, it next checks to see if the score is greater than 5. If it is, the message is output and the algorithm ends. In its present form, the algorithm produces the same output for all scores of 5 or above.

A good extension activity would be to ask students to alter the algorithm so that it works as intended. Here is one way of doing this:

```
OUTPUT "Please enter your test score."
score ← USERINPUT
IF score  < 5 THEN
        OUTPUT "You must try harder next time."
ELSE IF score  ≤ 7 THEN
        OUTPUT "Not bad, but there's still room for improvement."
ELSE IF score  = 8 THEN
        OUTPUT "This is a good result."
ELSE
        OUTPUT "This is an excellent result."
ENDIF
```

Activity 1.11

```
OUTPUT "Please enter the student's first name."
FirstName ← USERINPUT
OUTPUT "Please enter the student's surname."
Surname ← USERINPUT
OUTPUT "Please enter the student's year group."
Year ← USERINPUT
IF Year ≥ 7 OR Year ≤ 18 THEN
        OUTPUT "This is a valid year group."
ELSE
        OUTPUT "This year group does not exist."
ENDIF
OUTPUT "Please enter the student's tutor group."
TutorGrp ← USERINPUT
IF TutorGrp = "red" OR TutorGrp = "blue" OR TutorGrp = "green" OR TutorGrp = "yellow" THEN
        OUTPUT "This is a valid tutor group."
ELSE
        OUTPUT "This tutor group does not exist."
ENDIF
```

Activity 1.12

```
OUTPUT "Please enter a month number between 1 and 12."
MonthNo ← USERINPUT
CASE MonthNo OF
        1: OUTPUT "January"
        2: OUTPUT "February"
        3: OUTPUT "March"
        4: OUTPUT "April"
        5: OUTPUT "May"
        6: OUTPUT "June"
        7: OUTPUT "July"
        8: OUTPUT "August"
        9: OUTPUT "September"
        10: OUTPUT "October"
        11: OUTPUT "November"
        12: OUTPUT "December"
ELSE
        OUTPUT "Number entered is out of range."
ENDCASE
```

Chapter 2: Iteration

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what is meant by iteration
- explain the difference between definite and indefinite iteration
- use 'FOR' loops
- use 'WHILE' loops
- use 'REPEAT…UNTIL' loops
- use nested loops
- analyse algorithms using trace tables
- use iteration when designing algorithms.

What your students need to know

Students should:

- be able to create flowcharts using the standard symbols
- be able to use pseudo-code to express algorithms
- be familiar with the *AQA Pseudo-code Guide*.

Vocabulary

- Iteration
- Loop
- Execution
- Syntax error
- Dry run
- Trace table
- Efficiency
- Syntax
- HTML
- Definite iteration
- Indefinite iteration
- 'FOR' loop
- 'WHILE' loop
- 'REPEAT…UNTIL' loop
- Nested loop
- Infinite loop
- Logical error
- Algorithm efficiency

Common misconceptions and other issues

Students should be encouraged to use both comments and indentation in their pseudo-code. This will make their algorithms easier to read and errors easier to spot.

When designing a 'WHILE' loop, a common error is to forget to assign an initial value to the loop control variable before the start of the loop. In this example the variable password should have been assigned the value "" before the start of the loop:

```
storedPassword ← "DaWa11!m"
WHILE password = ""
        OUTPUT "Please enter your password."
                password ← USERINPUT
        IF password ≠ storedPassword THEN
                password ← ""
        ENDIF
ENDWHILE
```

If this algorithm were to be converted into executable code an error would be generated. The correct algorithm is:

```
storedPassword ← "DaWa11!m"
password ← ""
WHILE password = ""
        OUTPUT "Please enter your password."
        password ← USERINPUT
        IF password ≠ storedPassword THEN
                password ← ""
        ENDIF
ENDWHILE
```

Unlike a 'WHILE' loop, a 'REPEAT…UNTIL' loop always executes at least once, since the condition is checked at the end of the loop. For example:

```
storedPassword ← "DaWa11!m"
REPEAT
        OUTPUT "Please enter your password."
        password ← USERINPUT
UNTIL password = storedPassword
```

Students need to ensure that the loop terminating condition will eventually be met. It is surprisingly easy to create a loop that never ends.

When creating a nested loop structure in pseudo-code, care must be taken to indicate the end of each loop by using the appropriate loop terminator command, that is: 'ENDWHILE', 'ENDFOR' or 'UNTIL'.

Students should be encouraged to investigate and be able to use random number generation in the high-level programming language they are studying.

Skills and coding

- Maths skills:
  - Times tables (required especially for **Activity 2.4**)
- Coding skills:
  - Use of pseudo-code
  - Declaring and assigning variables
  - Selecting suitable identifiers
  - 'FOR' loops
  - 'WHILE' loops
  - 'REPEAT…UNTIL' loops
  - User input

The activities will also reinforce the skills acquired in **Chapter 1**.

Skills and coding for non-specialist teachers

1 Definite iteration

Definite iteration is used where the number of iterations is known before the loop starts. A 'FOR' loop is ideal in this situation. Here is an example of a 'FOR' loop that iterates ten times:

```
FOR counter ← 0 TO 9
        OUTPUT counter
ENDFOR
```

(The loop counter could start at 1, but using 0 reinforces the idea that computers start counting at 0 rather than 1.)

A suitable variable name should be used (so, 'FOR counter', as used here, is better than 'FOR x').

'WHILE' loops can also be used for definite iteration. When using a 'WHILE' loop for this purpose, the count variable must be assigned a value before the start of the loop. Here is an example:

```
counter ← 0
WHILE counter ≤ 9
        OUTPUT counter
        counter ← counter + 1
ENDWHILE
```

This will produce the same output as the 'FOR' loop above.

Definite iteration is explained in the Student Book and is illustrated in the solution to **Activity 2.1**.

2 Indefinite iteration

In indefinite iteration, the number of iterations is not known before the loop is started. The loop stops when a specified condition is met.

The 'WHILE' and 'REPEAT…UNTIL' loops are used for indefinite iteration.

A 'WHILE' loop continues **while** a certain condition remains true.

The condition is checked **before** the loop starts and so the variable used in the condition must be assigned an initial value before the start of the loop. Here is an example:

```
answer ← 'no'
WHILE answer = 'no'
        OUTPUT "Please enter 'yes' or 'no'."
        answer ← USERINPUT
ENDWHILE
```

This loop will continue until the variable answer has a value other than 'no'. It doesn't have to be 'yes'; anything but 'no' will stop the loop.

A 'REPEAT…UNTIL' loop also has a terminating condition. However, the comparison is not done until the end of the loop, which means that the loop will always execute at least once. Here is an example:

```
REPEAT
        OUTPUT "Please enter 'yes' or 'no'."
        answer ← USERINPUT
UNTIL answer ≠ 'no'
```

If the user enters anything other than 'no' the loop will terminate.

Indefinite iteration is explained in the Student Book and illustrated in the solutions to **Activity 2.2** and **Activity 2.3**.

3 Nested loops

A nested loop is a loop that runs inside another loop. Here is an example:

```
FOR index ← 1 TO 3
        FOR count ← 1 TO 10
                OUTPUT count
        ENDFOR
ENDFOR
```

This nested loop outputs the numbers 1 to 10 three times. For each turn of the outer loop, the inner loop runs ten times.

Care must be taken that each loop is terminated correctly.

Nested loops are explained in the Student Book and illustrated in the solution to **Activity 2.4**.

4 Trace tables

When creating trace tables, there should be a column allocated to each variable and to each input and output. Entries should be made in the rows to correspond to the values of the variables, inputs and outputs. Four columns would be needed for this algorithm:

```
index ← 1
x ← 0
y ← 0
REPEAT
        x ← index * index
        IF x > 9 THEN
                y ← x * 3
                OUTPUT y
        ENDIF
        index ← index + 1
UNTIL index = 5
```

Here is the trace table for the algorithm. (Obviously, in an actual trace table, the comments column would not be included.)

index	x	y	output	Comments
1	0	0		Before the start of the loop, all three variables are assigned an initial value.
2	1	0		At the first iteration of the 'REPEAT' loop, the value of x becomes 1. Because the value of x is still not greater than 9, the IF statement is not executed, so the value of y remains 0. At the end of the loop, index is incremented by 1. Since the value of index is less than 5, the loop doesn't terminate.
3	4	0		After the second iteration, the value of x is still not greater than 9 and the value of y remains 0. The value of index is still less than 5, so the loop doesn't terminate.
4	9	0		The terminating condition has still not been achieved after the third iteration.
5	16	48	48	The fourth time through the loop, the value of x becomes greater than 9 and so the value of y becomes 3 times the value of x and is output. The index becomes 5 so the loop terminates

Trace tables are explained in the Student Book and illustrated in the solution to **Activity 2.5** and in the practice questions.

Prompting questions

- Can you think of any everyday examples where iteration comes into play?
- Can you think of any computer games that you have played that make use of iteration?
- Why is iteration important/useful? How does it help programmers?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to consider one of the algorithms they wrote for **Chapter 1**. Can they modify their flowchart and/or pseudo-code to include iteration? As an early activity in this chapter, iteration can be illustrated simply by using the word 'repeat' in pseudo-code or a looping arrow on the flowchart. If the activity is used as a plenary, then you can use this task to assess students' understanding of using the appropriate type of iteration.
- Examine a computer game (*note: either let students select their own or pick one for them. The game selected should include some form of iteration and this might be visible in the form of scoring or gaining/losing lives*). Ask students to see if they can find an example of iteration in the game and ask them to explain the reasons behind their decision. This activity can be extended by asking students to:
  - write the algorithm for the example they have found (either as pseudo-code or as a flowchart or both)
  - explain what type of iteration they think is being used.
- Play a game of Word Sneak using words from the 'programming deck'. The full set of resources and activity guides can be downloaded from: **www.cambridge.org/links/katd4003**. Word sneak is a fun activity to help you assess students' knowledge and understanding in a unique way. The aim of the game is to have a normal conversation and sneak your hidden words into what you say without your partner noticing. The first person to use all their hidden words is the winner.
  - A variation of this is Three Word Stories (**www.cambridge.org/links/katd4004**), which requires the player to engineer the story being told (three words at a time) so that their partner uses their hidden word.

Enrichment activities

- Provide students with an algorithm written using only sequential instructions. Ask them to modify the code to include the appropriate iteration in the right places. Does the code work?

Assessment ideas

- Ask students to devise a quick game consisting of a single level. Students will need to explain what the game does as well as write the pseudo-code for the main game functions. Tell students that their pseudo-code should include examples of sequence, selection and iteration.

Answers

Activity 2.1

```
index ← 1
OUTPUT "Please enter a number."
number ← USERINPUT
WHILE index ≤ 12
        OUTPUT index + " x " + number + " = " + number * index
        index ← index + 1
ENDWHILE
```

Activity 2.2

There are lots of ways of solving this problem. Here are two of them:

```
sum ← 0                                  # Initialises counter
OUTPUT "How many numbers do you want to add together?"
total ← USERINPUT                        # User needs to know in advance how many numbers they want to enter
FOR index ← 1 TO total                   # 'FOR' loop used to enter and total the numbers
        OUTPUT "Enter the next number."
        number ← USERINPUT
        sum  ← sum + number
ENDFOR
OUTPUT sum
```

```
sum ← 0
anotherNumber ← "y"                      # User needs to signal after each entry if they want to enter another number
WHILE anotherNumber = "y"
        OUTPUT "Enter a number."
        number ← USERINPUT
        sum  ← sum + number
        OUTPUT "Do you want to enter another number (y/n)?"
        anotherNumber ← USERINPUT
ENDWHILE
OUTPUT "The total is: " +  sum
```

Activity 2.3

```
playAgain ← "yes"                                    # Initialises the loop control variable for outer loop
WHILE playAgain = "yes"                              # Start of outer loop
        mysteryNumber ← RANDOM_INT(1, 100)
        guess ← 0                                    # Initialises the loop control variable for inner loop
        WHILE guess = 0                              # Start of inner loop
                        OUTPUT "Enter a number between 1 and 100."
                        guess ← USERINPUT
                        IF guess > mysteryNumber THEN
                                        guess ← 0
                                        OUTPUT "Your guess is too high."
                        ELSEIF guess < mysteryNumber THEN
                                        guess ← 0
                                        OUTPUT "Your guess is too low."
                        ENDIF
        ENDWHILE                                     # End of inner loop
        OUTPUT "Well done. You guessed correctly!"
        OUTPUT "Do you want to play again (yes/no)?"
        playAgain ← USERINPUT
ENDWHILE                                             # End of outer loop
```

Activity 2.4

The algorithm asks the user to enter the upper and lower limits of the range. These are then stored in two variables. The variables are then used within the loop to define the iterations.

```
OUTPUT "Enter the number for the start of the range of times tables."
lowerRange ← USERINPUT
OUTPUT "Enter the number for the end of the range of times tables."
upperRange ← USERINPUT
FOR index ← lowerRange TO upperRange
    OUTPUT "This is the " + index + " times table.")
    FOR times ← 1 TO 12
        OUTPUT "times " + "x" + index + " = " + index*times)
    ENDFOR
ENDFOR
```

Activity 2.5

TURNS	OUTPUT	X	OUTPUT
0		3	
3		9	
6		27	
9		81	
12		243	
15		729	
18		2187	
21		6561	
24	24	19683	19683

Activity 2.6

The following trace table can be used:

Number	even	odd	sumEven	sumOdd	output
	0	0	0	0	
13	0	1	0	13	
42	1	1	42	13	
3	1	2	42	16	
6	2	2	48	16	
9	2	3	48	25	
0	2	3	48	25	2 48 3 25

- Numbers are input until the number is 0.
- The algorithm decides whether a number is even or odd by finding the remainder of a division by 2 (Modulus). If there is no remainder then it is even.
- The sum of all of the even and odd numbers is calculated.

Chapter 3: Data types and structures

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what is meant by 'data type' and list some common types
- use the correct data types in algorithms
- carry out various manipulations such as finding the length of and slicing and concatenating 'string' data types
- create and work with simple array data structures
- create and work with two-dimensional arrays
- describe other data structures.

What your students need to know

Students should:

- be able to use pseudo-code to create variables and display algorithms
- use suitable and consistent identifiers for variables
- understand the use of 'IF' statements
- be able to use definite and indefinite iteration.

Vocabulary

- Integer
- Real
- Character
- Property
- Index
- String traversal
- Traverse
- Substring
- Concatenation
- Casting
- Static array
- Dynamic array
- Entity
- Field
- Table
- Data type
- Data structure
- Floating point
- String
- Boolean
- Property/attribute
- Array index
- Two-dimensional array
- Database
- Record

Common misconceptions and other issues

In some programming languages, the data type of a variable does not have to be explicitly declared. They assume what the data type of a variable should be from the data that is assigned to it.

In this example, the variable 'index' is being assigned the value 3 and so is implicitly declared as type integer:

```
index ← 3
```

Here,'index' is being assigned a decimal number, so is implicitly declared as type real:

```
index ← 3.33
```

Here, 'firstName' is being assigned a string value, so is implicitly declared as type string:

```
firstName ← "Catherine"
```

Even if the high-level programming language your students are studying does not require them to declare a data type when declaring variables, it is important that they understand the concept of data type, are able to select appropriate data types for variables and are familiar with the operations that can be performed on different types of data.

In AQA pseudo-code, there is a LEN command that returns the length of a string. For example, LEN("rhinoceros") would return the value 10.

When using a loop to iterate through a string, it's important to remember that the first character in a string has the index value 0 and the last character has the index value LEN(string) – 1. (A string with a length of 10 will have characters indexed 0 to 9.)

In this example, an error message would be generated when the loop tries to access the character at index position 17 because it doesn't exist:

```
string ← "This is a string."
FOR index ← 0 TO LEN(string)
        OUTPUT string[index]
ENDFOR
```

When concatenating strings to form compound words, students should remember to add spaces where appropriate to aid readability. In this example, a space is missing between firstName and lastName:

```
firstName ← "Jack"
lastName ← "Smith"
fullName ← firstName + lastName
```

fullName ← firstName + " " + lastName would produce the result 'Jack Smith'.

The standard definition of an array is 'a data structure that contains a collection of elements of the same data type'. However, some languages, such as Python, are more forgiving and will allow elements with different data types to be stored in an 'array-like' structure.

Skills and coding

- Maths skills:

Maximum, minimum and mean (required for **Activity 3.8** and **Activity 3.12**)

- Coding skills:
  - Use of pseudo-code
  - Finding the length of a string
  - Creating a loop to traverse a string
  - Searching a string for a particular character
  - Counting the number of times that a character occurs in a string
  - Splitting strings
  - Finding substrings within a string
  - Concatenating strings
  - Casting
  - Creating, populating, editing and searching arrays
  - Creating, populating, editing and searching multi-dimensional arrays

Skills and coding for non-specialist teachers

1 Strings

The length of a string is found using the LEN command. For example:

```
myString ← "David"
lenString ← LEN(myString)
```

would assign the value 5 to the variable lenString.

A loop to traverse the string would start at 0 (the first index position) and end at lenString – 1, i.e. index position 4.

```
FOR index ← 0 TO lenString – 1
        OUTPUT myString[index]
ENDFOR
```

This would output each character in turn on a new line, i.e.

```
D
a
v
i
d
```

Traversing a string is explained in a worked example in the Student Book and demonstrated in the solution to **Activity 3.2**.

2 Finding substrings

When traversing a string to find a substring, the length of the substring must be taken into account.

For example, if myString has the value "David" and lenString the value 5, a loop traversal through myString would start at 0 and end at 4 (lenString – 1).

But if the substring being searched for is "av", which has a length of 2, the search should end at index position 3 (lenString – 2).

This is explained in a worked example in the Student Book and demonstrated in the solution to **Activity 3.3**.

3 Arrays

The elements of an array have an index position in the same way that characters in a string do. Indexing usually begins at 0, although in some instances might begin at 1. Questions on exam papers will always state whether indexing begins at 0 or 1. This is explained at the start of the section on arrays in the Student Book and demonstrated in the solutions to **Activity 3.6** and **Activity 3.7**. It might be useful for students to practise writing code that uses indexing beginning at 1 and at 0 so that they are familiar with each (an example of each case is in the next section).

4 Creating sub-arrays

A sub-array can be created by copying data items from the main array using the indexes of the data items.

Activity 3.9 asks students to create a sub-array using items that are not contiguous. The students are asked to traverse the main array to find items that are equal to or greater than 5 and then create a new array, called 'Pass', to store them.

The algorithm should:

- create a new array called 'Pass' to store these items
- traverse the main array to find marks equal to or greater than 5
- copy them to the new array.

Here is a possible solution that assumes indexing begins at 0:

```
arrayMarks ← [6, 9, 2, 5, 8, 3, 9, 9, 10, 9, 5, 7, 10]
arrayPass ← [ ]
count ← 0
FOR index ← 0 TO LEN(arrayMarks) – 1
    IF arrayMarks[index] ≥ 5 THEN
                        arrayPass[count] ← arrayMarks[index]
                        count ← count + 1
    ENDIF
ENDFOR
```

Here is the same solution but assuming that indexing begins at 1:

```
arrayMarks ← [6, 9, 2, 5, 8, 3, 9, 9, 10, 9, 5, 7, 10]
arrayPass ← [ ]
count ← 1
FOR index ← 1 TO LEN(arrayMarks)
    IF arrayMarks[index] ≥ 5 THEN
                        arrayPass[count] ← arrayMarks[index]
                        count ← count + 1
    ENDIF
ENDFOR
```

5 Two-dimensional arrays

A two-dimensional array is similar to a database table; each row represents a record and each column a field.

This two-dimensional array is declared as:

myArray [6][2]

This stipulates that there will be six index positions and two items of data will be stored at each one.

More than two items of data can be stored at each index position:

myArray [6][3]

This would create a static two-dimensional array with three items of data at each index position.

If there are two items of data, they can be referenced as:

myArray[1][1] and myArray[1][2]

myArray[2][1] and myArray[2][2]

myArray[3][1] and myArray[3][2]

etc.

This assumes that indexing starts at 1.

Prompting questions

- What is data?
- What is the difference between data and information?
- Look at the information around you (e.g. on classroom walls, in textbooks, etc.). What are the different types of data that you can see? How would you categorise them?
- What are arrays?
- What are the different types of array, and when would you use each of them?
- What is a database?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to examine a resource, this can be anything from a poster to a document they have produced, a website, or simply going through their textbook. What are the different types of information that they can see? Ask students to arrive at their own categories and ways of classifying the information. End the activity with groups sharing their work and explaining the reasons behind their categorisation. This will lead on nicely to students then being able to compare their work with the actual data types discussed later in the lesson.
- Get students to play the battleships game to help them understand how two-dimensional arrays are referenced using [row] [column].
- Give students a list of different items of data, and for each one ask them to correctly match the associated data type.

Enrichment activities

- Ask students to review the algorithms for their game design and to see if any would be more efficient if arrays were used. Ask students to make adjustments to the code and then give their reasons for doing so.
- Investigate Databases and SQL. There are many ways you can program in SQL and you don't always need PHP to do it. If your school has a database such as Microsoft Access, you can create a new query and edit it in SQL view. This allows you to enter and execute SQL commands. Ask students to investigate this to create a single table of data storing a range of fields for a number of records. They should run SQL commands to allow them to add/delete/amend and search data within the records.

Assessment ideas

- Give pairs of students a range of topics, including: data types, static array, dynamic array, index, databases, SQL, fields and records. Each pair is given a specific amount of time to research their assigned concept and think of a unique way to 'teach it' to the rest of the class. At an identified time, invite each pair to 'teach' what they have learnt to the class. In turn, the class can then review/evaluate their thoughts on what they have been taught.

Answers

Activity 3.1

Variable	Data type
FirstName	String
LastName	String
Initial	Character
Age	Integer

Activity 3.2

```
times ← 0
OUTPUT "Please enter a piece of text."
myString ← USERINPUT
OUTPUT "Please enter the character you wish to search for."
myChar ← USERINPUT
FOR index ← 0 TO LEN(myString) – 1
        IF myString[index] = myChar THEN
                times ← times + 1
        ENDIF
ENDFOR
IF times = 0 THEN
        OUTPUT "This character is not in the text you entered."
ELSE
        OUTPUT "The character you entered appears " + times + " times."
ENDIF
```

Activity 3.3

```
OUTPUT "Please enter the text you wish to search through."          # Revision notes
myText ← USERINPUT
searchWord  ← "variable"
length ← LEN(searchWord)
times ← 0
found ← "no"
FOR index ← 0 TO LEN(myText) – (length+1)
        testString ← ""
        testString ← testString + myText(index)
        FOR test ← 1 TO length
                testString ← testString + myText(index + test)
        ENDFOR
        IF testString = searchWord THEN
                found ← "Yes"
                times ← times + 1
        ENDIF
ENDFOR
IF found = "Yes" THEN
        OUTPUT searchWord + " was found " + times + " times."
    ELSE
        OUTPUT searchWord + " does not appear in this text."
ENDIF
```

You could extend this activity by challenging students to amend the algorithm so that it handles upper case letters as well as lower case, i.e. 'Variable' as well as 'variable' or 'VARIABLE' or indeed 'vAriABLe'. Most programming languages have a string method that returns a copy of a string converted to lower case.

Activity 3.4

```
OUTPUT "Please enter your first name."
firstName ← USERINPUT
OUTPUT "Please enter your surname."
surname ← USERINPUT
fullName ← firstName + " " + surname
OUTPUT "Hello " + fullName + ", how are you?")
```

Activity 3.5

In this solution we've assumed that indexing begins at 1.

```
cars ← [5]
FOR index ← 1 TO 5
        OUTPUT "Please enter the name of a car."
        response ← USERINPUT
        cars[index] ← response
ENDFOR
OUTPUT "All cars have now been entered."
```

Activity 3.6

In this solution we've assumed that indexing begins at 1.

```
FOR index ← 0 TO LEN(cars)-1
        OUTPUT cars[index]
ENDFOR
```

Activity 3.7

This algorithm assumes that the array 'arrayAlphabet'[26] has already been declared and populated and that array indexing begins at 1.

```
searchString ← "computer"
FOR stringIndex  ← 0 TO LEN(searchString) – 1
        nextChar ← searchString[stringIndex]
        found ← False
        arrayIndex ← 1
        WHILE found = False AND arrayIndex  ≤ LEN(arrayAlphabet)
                IF arrayAlphabet[arrayIndex] = nextChar THEN
                        found ← True
                        OUTPUT nextChar + "has the index value " + arrayIndex
                ELSE
                        arrayIndex ←  arrayIndex + 1
        ENDWHILE
ENDFOR
```

Activity 3.8

This algorithm assumes that the array 'Marks' has already been populated and that array indexing begins at 0.

```
index ← 0
min ← marks[index]
sum ← marks[index]
numbMarks ← LEN(marks) – 1
WHILE index ≤ numbMarks
        IF marks[index] < min THEN
                min ← marks[index]
        ENDIF
        sum ← sum + marks[index]
        index ← index + 1
ENDWHILE
OUTPUT "Your lowest mark is: " + min
OUTPUT "Your mean mark is: " + sum/numbMarks
```

Activity 3.9

In this solution we've assumed that indexing begins at 0.

```
marks ← [6, 9, 2, 5, 8, 3, 9, 9, 10, 9, 5, 7,10]
pass ← []
index ← 0
FOR numb ← 0 TO LEN(marks) – 1
        IF marks[numb] ≥ 5  THEN
                pass[index] ← marks[numb]
                index ← index + 1
        ENDIF
ENDFOR
```

Activity 3.10

In this solution we've assumed that indexing begins at 0.

```
FOR index ← 0 TO LEN(exam1) – 1
        IF exam1[index] < 50 THEN
                exam1[index] ← exam1[index] + 5
        ELSE IF exam1[index] > 50 THEN
                exam1[index] ← exam1[index] + 10
        ENDIF
ENDFOR
```

Activity 3.11

a. 69

b. 76

Activity 3.12

Students' own answers

Chapter 4: Searching and sorting algorithms

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain why sorted lists are of more value than unsorted lists
- describe the bubble sort and merge sort algorithms and compare and contrast them
- use these algorithms to sort lists into ascending and descending order
- understand the linear and binary search algorithms and compare and contrast them
- use these algorithms to search sorted and unsorted lists
- write code for the implementation of these algorithms.

What your students need to know

Students should:

- be able to use pseudo-code to express algorithms
- be able to create and use arrays.

Vocabulary

- Searching
- Sorting
- Compare
- Adjacent items
- Sequential
- Ordered list
- Ascending order
- Descending order
- Bubble sort
- Pass
- Merge sort
- Recursion
- Divide and conquer
- Brute force
- Linear search
- Binary search

Common misconceptions and other issues

Students should be encouraged to work through the stages of each sorting and searching algorithm, showing the results of each stage.

It is worth stressing that when sorting into ascending order using a bubble sort, the highest unsorted value will be in its correct position at the end of each pass.

The efficiency of the different algorithms should be stressed, particularly when comparing linear and binary searches.

Skills and coding

- Maths skills:

Median (**Activity 4.7** and **Activity 4.8**)

- Coding skills:
  - Use of pseudo-code
  - Nested loops
  - Declaring and populating arrays
  - Finding the length of an array
  - Using loops to traverse an array
  - Using comparison operators

Skills and coding for non-specialist teachers

1 Bubble sort

In a bubble sort, the first two items (i.e. items 1 and 2) are compared and are swapped round if they are not in the required order. Then the next pair (items 2 and 3) are compared. This continues until the end of the list.

If they are being sorted into ascending order, the item with the highest value will be in its correct position at the end of the first pass. Passes are repeated until there are no swaps.

The bubble sort is explained with an example in the Student Book. **Activity 4.3** gives students an opportunity to improve the efficiency of the basic algorithm by utilising the fact that after each loop traversal, another number at the end of the list is in the correct position.

2 Merge sort

A merge sort uses recursion. It is said to be **divide-and-conquer**, as it breaks the problem into sub-problems that are similar to the original problem, recursively solves the sub-problems, and finally combines the solutions to the sub-problems to solve the original problem. A 'divide-and-conquer' algorithm has three stages:

Divide the problem into a number of sub-problems that are smaller instances of the same problem.

Conquer the sub-problems by solving them recursively.

Combine the solutions to the sub-problems into the solution for the original problem.

The merge sort is explained in the worked example in the Student Book and is demonstrated in the solution to **Activity 4.4**.

3 Searching algorithms

Linear and binary search algorithms are very straightforward and students should have encountered them in their daily lives.

Comparisons of the best and worst case scenarios provide a good starting point for discussing the efficiency of algorithms.

Searching algorithms are explained in the Student Book and the solutions to **Activity 4.5** to **Activity 4.8**.

The students are not expected to code a binary search algorithm but this could be done as an extension activity. A binary search algorithm is shown below. It assumes that array indexing begins at 1.

Pseudo-code	Explanation
OUTPUT "Please enter a target." target ← USERINPUT	A variable to store the item to be searched for is declared as a 'target'. In this instance, a number is the expected input.
start ← 1	The variable 'start' is set to the index number of the first item.
end ← LEN(list)	The variable 'end' is set to the index of the last item of the list.
found ← False	The Boolean variable 'found' is set to False. This is used to indicate that the search item has not been found.
WHILE start ≤ end AND found = False	A 'WHILE' loop is set up.
middle ← ((start + end)/2)	The median item is found.
IF list[middle] = target THEN OUTPUT target + " is in the list" found ← True	If the median number is the target, then 'found' is set to True and the user is informed.
ELSE IF target < list[middle] THEN end ← middle – 1	If the search item is less than the median, then the variable 'end' is set to the item to the left of the median – the next item less than the median.
ELSE start ← middle + 1	If the search item is greater than the median, then the variable 'start' is set to the item to the right of the median – the next item greater than the median.
ENDIF	
ENDWHILE	

IF found = False THEN
OUTPUT target + " is not in the list."
ENDIF
	The user is informed if the search item has not been found.

Prompting questions

- Why is it important to sort things into an order?
- How many examples can you think of where data has been sorted?
- The bubble sort and merge sort algorithms use different ways to sort data. Explain how they work.

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Provide students with numbered cards in a random order. Tell them to place the cards next to each other and then sort them into order without telling them how. Then ask students how they sorted the data and to write their strategy down. How efficient was their strategy? Could they have done it better? Ask pairs of students to compare their techniques with each other. Which one was faster and why? This can then be used to compare against the sorting techniques discussed in the chapter. It works well either as a starter or a plenary.
- Students play a game. With a deck of sorted cards, a player chooses a number secretly. The 'magician' has to work it out using only questions such as, 'Is it higher or lower?' Another option is to play 20 questions with the class. The ideal questions are those that automatically eliminate at least half the options, such as, 'Is it male or female?' The popular CS4FN/Teaching London Computing initiative, funded by the Mayor of London, has a detailed outline of activities to teach searching algorithms: **www.cambridge.org/links/katd4005**

Enrichment activities

- Ask students to investigate the different search and sort algorithms. Can they find the most effective YouTube video that explains the different algorithms and their differences and characteristics?
- Ask students to search through the CS4FN site and read through the resources. Using inspiration from that style, write an article to explain the different sort algorithms.

Assessment ideas

Students carry out an investigation to code the different sort algorithms and run them with the same set of data. Can students discover which is the most efficient? Ask them to consider their own criteria for comparison and present their findings and justifications at the end.

Answers

Activity 4.1

Students' own answers

Activity 4.2

20	15	3	13	9	2	6
15	3	13	9	2	6	20
3	13	9	2	6	15	20
3	9	2	6	13	15	20
3	2	6	9	13	15	20
2	3	6	9	13	15	20
2	3	6	9	13	15	20

Activity 4.3

The following code assumes that the list of items is an array, with array indexing starting at 1.

```
S ← list of items
N ← (length of S-2)
N ← length of list
IF N ≤ 1 THEN                    # Deals with lists consisting of 0 or 1 items
        swapped ← False
ELSE
        swapped ← True
ENDIF
WHILE swapped = True
        swapped ← False
        FOR X ← 2 TO N
                IF S[X – 1] > S[X] THEN
                        temp ← S[X – 1]
                        S[X – 1] ← S[X]
                        S[X] ← temp
                        swapped ← True
                ENDIF
        ENDFOR
        N ← N – 1
ENDWHILE
```

Activity 4.4

20	15	3	13	9	2	6						
20	15	3	13		9	2	6					
20	15		3	13		9	2		6			
20		15		3		13		9		2		6
15	20		3	13		2	9		6			
3	13	15	20			2	6	9				
2	3	6	9	13	15	20						

Activity 4.5

This algorithm assumes that the array 'popularNames' has already been initialised and populated with the hundred most popular names. Array indexing begins at 1.

```
found ← False
index ← 1
OUTPUT "Please enter the name you want to search for."
name ← USERINPUT
WHILE found = False AND index ≤ 100
    IF name = popularNames[index] THEN
                found ← True
    ENDIF
    index ← index + 1
ENDWHILE
IF found = True THEN
    OUTPUT name + "is in the list."
ELSE
    OUTPUT name + "is not in the list."
ENDIF
```

Activity 4.6

a	d	g	h	k	m	**p**	r	s	u	w	x	z
a	d	g	**h**	k	m							
a	**d**	g										
		g										

Activity 4.7

3	5	6	8	9	12	**15**	21	23	45	56	63	69
							21	23	45	**56**	63	69
							21	**23**	45			
									45			

- Pick median 15
- Too low, so select right-hand side, six numbers left, so choose median 56
- Too low, so select right-hand side, 3 numbers left, so choose median 23
- Too low, so answer is 45.

Activity 4.8

- Enter a number
- Find the length of the array
- Start of (search items) equals 0
- End of search items equals length of array
- Middle equals (start + end)/2
- While start is less than or equal to end
- If middle is equal to number entered, then tell the user and stop the loop
- If middle is less than number entered, then start equals middle + 1
- If middle is greater than number entered, then end equals middle – 1
- End of while loop
- Inform the user that the number is not present

Chapter 5: Input and output

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain why user input is needed
- describe ways in which data input can be validated
- format output
- work with text files.

What your students need to know

Students should:

- be able to use pseudo-code to create variables and display algorithms
- be able to use selection and definite and indefinite iteration
- be able to create and use one- and two-dimensional arrays.

Vocabulary

- Logical error
- Validation
- File handle
- Write mode
- Overwritten
- Closed
- Read mode
- Syntax error
- Presence check
- Range check
- Length check
- Authentication
- Close file

Common misconceptions and other issues

It should be stressed that valid data is not necessarily correct. For example, when the students' details are being entered on to the school admin system, a year group of 10 would be a valid entry, but the student in question might not be in year 10. The entry would be valid, but incorrect.

When checking that data has been entered by a user, a variable initialised as a blank string can be used.

Students should be encouraged to investigate the on-screen formatting commands in the high-level programming language they are studying.

In the Student Book, the AQA pseudo-code commands are used when working with text files.

The examples given in the Student Book only use static arrays, where the numbers of items being read from and written to a file are known in advance. However, in most instances this is not the case. The use of a dynamic array together with an append method enables files of unknown length to be read.

Students should be encouraged to investigate the file handling commands available in the high-level programming language they are studying.

Skills and coding

- Coding skills:
  - Use of pseudo-code
  - Nested loops
  - Creating and populating arrays
  - Finding the length of an array
  - Using loops to traverse an array
  - Opening and closing text files
  - Writing to and reading from text files

Skills and coding for non-specialist teachers

Text files

File handles to open and close a file are not required in the AQA pseudo-code. Nevertheless, it is important that students understand the difference between write and append modes. When an existing file is opened in write mode, any existing data it contains is overwritten. In contrast, the append mode allows data to be added to an existing file without overwriting data that is already there.

The use of text files is explained in the Student Book and demonstrated in the solution to **Activity 5.4**.

Prompting questions

- Can you think of one example where GIGO (garbage in garbage out) is especially important?
- What is the difference between validation and verification?
- Can you think of any examples of validation routines that you have encountered? What data were you entering, and what do you think the validation rule was?
- What is authentication?
  - Why is this important?
  - What are the different ways we can authenticate a user identity?
- What is identity theft?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to list all of the devices they can think of, then categorise them. Are they primarily input, output or something else?
- Give students a program with errors in it. You might even ask them to examine a program they have written that possibly still doesn't work. In pairs, ask them to circle/highlight and label all the syntax and logical errors, clearly stating which error is which and why.

Enrichment activities

- Ask students to examine the last program/algorithm they wrote. Have they included any validation routines? Where and how could validation routines be included? They should modify their program/algorithm to include validation.
- Ask students to investigate the different authentication techniques carried out by social networking sites, banks, online ordering sites, etc. What are the different methods used and why do they work?
- Can students find a recent example of identity theft, or a security breach that might result in identity theft? How did this breach/theft occur and how could it have been prevented?

Assessment ideas

- Ask students to write a program that will enable a record of information to be constructed. This record could be about anything, including: friends' personal and birthday information, music collection, game sales, test and assessment results, etc. The important thing is that a variety of data is possible for input. The program should attempt to validate each field or item of information entered. Check that students have constructed appropriate validation routines.

Answers

Activity 5.1

```
OUTPUT "Please enter your age."

Age ← USERINPUT

IF Age ≥ 17 THEN
        OUTPUT "You can apply for a driving licence."
ELSE
        OUTPUT "You are too young to apply for a driving licence."
ENDIF
```

Activity 5.2

```
OUTPUT "Please enter your password."

password ← USERINPUT

IF LEN(password) < 9 THEN
        OUTPUT "Your password must have at least nine characters. Yours has only " + LEN(password)
ELSE IF LEN(password) ≥ 12 THEN:
        OUTPUT "Your password must have no more than twelve characters. Yours has" + LEN(password)
ELSE
        OUTPUT "The length of your password is OK."
ENDIF
```

Activity 5.3

This algorithm gives the user three attempts to get their password right before locking their account. It assumes the existence of a two-dimensional array called 'users' that hold users' names and passwords. In this instance, it is assumed that indexing begins at 1.

```
userEntry ← ""
foundName ← 0
count ← 1                                   # Allows 3 goes at entering password
WHILE userEntry = "" AND count < 4
        OUTPUT "Please enter your username."
        userEntry ←USERINPUT
        usersLen ← LEN(users)               # Establishes number in list of users/passwords
        FOR index ← 1 TO usersLen           # Checks if username is in list
                IF userEntry = users[index][1] THEN
                        foundName ← 1
                        pwFound ← False
                        WHILE count ≤ 3 AND pwFound = False
                                OUTPUT "Please enter your password."
                                passwordEntry ← USERINPUT
                                IF passwordEntry = users[index][2] THEN
                                        OUTPUT "Username and password are
                    correct."
                                        pwFound ← True
                                ELSE
                                        OUTPUT "Sorry, the password is incorrect."
                                        count ← count + 1
                                ENDIF
                        ENDWHILE
                ENDIF
        ENDFOR
        IF foundName = 0 THEN
                OUTPUT "Username not recognised."
                userEntry ← ""
        ENDIF
ENDWHILE
IF count > 3 THEN
        OUTPUT "Your account has been locked."
ENDIF
```

Activity 5.4

```
# Writing the items from an array 'scores' into a file gameScores.txt.
FOR index ← 1 TO 5
        WRITELINE(gameScores.txt, index, scores[index])
ENDFOR
# Reading the scores from the file gameScores.txt into an array 'leaderBoard'.
FOR index ← 1 TO 5
        leaderBoard[index] ← READLINE(gameScores.txt, index)
ENDFOR
```

Chapter 6: Problem solving

LEARNING OUTCOMES

- By the end of this chapter students should be able to:
- explain what is meant by computational thinking
- explain what is meant by decomposition and abstraction and use these to solve problems
- create algorithms to solve problems that you have analysed
- explain what is meant by top-down and bottom-up problem solving
- create structured programs using procedures
- follow the systems development cycle to analyse problems, design and implement solutions and test the outcomes.

What your students need to know

Students should:

- be able to use pseudo-code to express algorithms
- be able to use selection and definite and indefinite iteration
- be able to ask for and incorporate user input
- be able to use trace tables.

Vocabulary

- Decomposition
- Abstraction
- Subroutine
- Argument
- Parameter
- Local variable
- Global variable
- Menu
- Systems development cycle
- Alpha testing
- Test data
- Testing plan
- Valid test
- Boundary test
- Erroneous test
- Beta testing
- Computational thinking
- Pattern recognition
- Top-down problem solving
- Bottom-up problem solving
- Structured programming
- Modules
- Call a subroutine
- Function
- Procedure

- Identification and analysis
- Design
- Logical errors
- Implementation
- Syntax errors
- Integrated development environment (IDE)
- Source code editor
- Comment
- Evaluation

Common misconceptions and other issues

Abstraction can be thought of as removing unnecessary details to get to the heart or essence of something.

It can be introduced by considering abstraction in everyday situations such as:

- creating mental models of objects, such as cars, houses, animals, etc., so that we can communicate with each other
- our use of machinery without knowing exactly how it works: for example, starting and driving a car without knowing how the combustion engine works
- when we use the print() function, we do not need to know all of the coding involved in making this happen
- when we use a high-level programming language, we do not need to know the actual machine code – a compiler or interpreter translates it for us. We are working at a higher level of abstraction. Assembly language is a low level of abstraction as it is more similar to machine code.

A subroutine is a set of instructions designed to perform a frequently used operation in a program.

It is 'called' by the main program.

- A function returns a value back to the main program.
- A procedure does not return any data to the main program.

When a subroutine is called, the data it needs (the parameters) are passed to it as arguments from the main program.

- The data is passed as an 'argument' and accepted as a 'parameter'.

It is also worth pointing out that a function can be called from within another function.

The first function can pass arguments to the second one that can return values to the first one.

Skills and coding

- Coding skills:
  - Creating functions with parameters
  - Calling functions with arguments

Skills and coding for non-specialist teachers

Functions

When a function is called, the data it needs to process is passed to it as arguments.

In the Student Book, **Activity 6.1** is an exercise on decomposition and abstraction involving the calculation of the approximate cost of a car journey.

This could be coded using a function:

Pseudo-code	Explanation
SUBROUTINE cost (distance, mpg, petrolPriceLitre)	The subroutine is defined with the identifier 'cost' and the parameters 'distance', 'mpg' and 'petrolPriceLitre'. These are local variables used only within the subroutine.
petrolPriceGallon ← petrolPriceLitre * 4.546	Converts petrol price per litre into petrol price per gallon.
cost ← (distance/mpg) * petrolPriceGallon	This statement calculates the cost of the journey.
RETURN cost	This statement returns the value of 'cost' to the main program.
ENDSUBROUTINE	This denotes the end of the function definition.
OUTPUT "Enter the journey distance in miles: " journeyDistance ← USERINPUT	These statements ask for user input. The values are stored in the global variables, journeyDistance, mpg and petrolPriceLitre.
OUTPUT "Enter the average miles per gallon for the car: " mpg ← USERINPUT	
OUTPUT "Enter the price of a litre of petrol: " petrolPriceLitre ← USERINPUT	
journeyCost ← cost(journeyDistance, mpg, petrolPriceLitre)	This statement calls the function 'cost'. The values of the three variables are passed to it as arguments. They are passed in the same order as the three parameters listed in the function. The function will return its result to the variable journeyCost.
OUTPUT "The approximate cost of the journey will be " + journeyCost	The journey cost is displayed.

The values given as arguments from the main program must be in the same order as expected by the parameters in the function.

Prompting questions

- Can you think of any examples in your everyday life that illustrate our use of computational thinking?
- Decomposition helps us to solve problems and make them more manageable by breaking them down into smaller parts. Look around you at the things you do every day. What examples can you see that use decomposition?
- "A game such as SimCity is an example of abstraction." Do you agree or disagree with this statement? Why?
- What is structured programming?
- The opposite of the 'top-down' approach is known as the 'bottom-up' approach. What do you think this means? How would this approach work?
- What is beta testing?
- When developers release software (usually for free) in Beta version, what does this usually mean? Why are they doing it?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to carefully consider the activities that they do regularly (these could be anything from jigsaw puzzles to D&T projects). They should pick one that they think uses computational thinking. Ask them to break this down and describe the activity, which strands of computational thinking it covers, how and why.
- Give students an example of some code and ask them to highlight examples of the following (possibly in different colours, or labelled and annotated):
  - local variables
  - global variables
  - subroutines
  - arguments and parameters
  - iteration
  - selection statements
  - array.
- Give students a small program to test. Ask them to design and carry out a test plan to see if it works.
- Choose one of the Computational Word Games from the Playful Computing page on the Digital Schoolhouse website (**www.cambridge.org/links/katd4006**). Select the stack of words under the programming and random categories to test students' knowledge and understanding of some of the key words. You can easily add your own words to this stack. As an interesting variation, ask students: how could the rules of the game be adapted and extended?

Enrichment activities

- Investigate the top-down and bottom-up approaches to computing. Can students find examples of how the approaches have been used in computer science research and development? For example, a top-down approach in robotics generally implies that the researchers have focused on the higher order things first, such as talking and activities closer to human level. The bottom-up approach instead focuses on a single 'sense'. Which robots have been developed as a result of the two approaches?
- The Playful Computing section of the Digital Schoolhouse website (**www.cambridge.org/links/katd4006**) uses unplugged activities to teach computational thinking. Ask students to select one of the activities and investigate it as a group. When reporting back to the class, they should attempt to deliver the activity and explain how and why it maps to computational thinking.
- Ask students to investigate and search for some software that is being released in Beta version. They should find out what the developers are offering and what they expect in return. Ask them to write a brief summary and exchange notes with peers in the class.

Assessment ideas

- Ask students to complete the final challenge for the chapter. Encourage them to follow good practice and guidance when documenting their solution. Students should submit a full testing plan with evidence of testing carried out on their work when they submit their work.

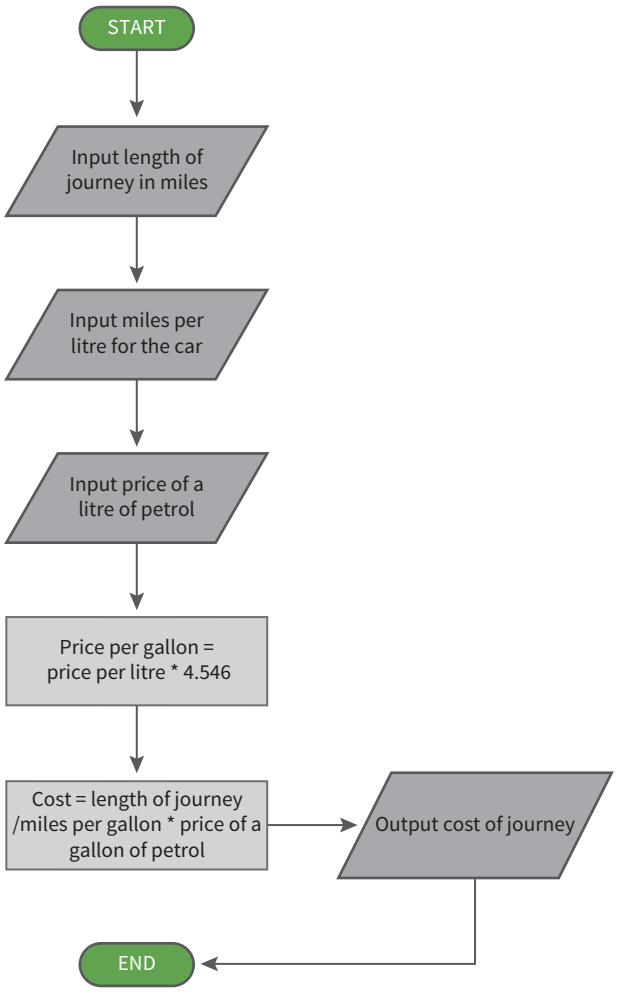
Answers

Activity 6.1

Possible sub-tasks might include among many others:

- Calculate the length of the journey in miles (or km)
- Establish the cost of petrol
- Find out how many miles (or km) can be driven per litre of petrol
- Convert petrol price per litre into price per gallon
- Calculate the cost of the journey

Activity 6.2



Activity 6.3

```
SUBROUTINE diceThrow():                          # Simulates a dice throw
        throw ← RANDOM_INT(1, 6)
            RETURN(throw)
ENDSUBROUTINE

# Start of main program
highestScore ← 0
OUTPUT "Play the game (y/n)?"                    # Keeps track of highest score
anotherGo ← USERINPUT

WHILE anotherGo = 'y' OR anotherGo = 'Y'         # Allows for upper and lower case entry
            total ← 0
            total ← total + diceThrow() + diceThrow() + diceThrow()
            OUTPUT "Your total this time is:" + total

            IF total > highestScore THEN
                                highestScore ← total
            ENDIF

            OUTPUT "Play the game again (y/n)?"
            anotherGo ← USERINPUT
ENDWHILE

OUTPUT "The highest score you achieved was:" + highestScore
```

Activity 6.4

```
SUBROUTINE message(one, two)
            OUTPUT "Hello" + two +  " " + one)
ENDSUBROUTINE

OUTPUT "Please enter your first name."
firstName ← USERINPUT
OUTPUT "Please enter your surname."
surname ← USERINPUT
message(firstName, secondName)
```

Activity 6.5

Input1	Input2	Solution
3	6	2
4	7	28/11
5	8	40/13

Activity 6.6

```
SUBROUTINE dogAge()                    # Calculates human equivalent age of a dog
        OUTPUT "Enter the age of your dog."
        dogYears ← USERINPUT           #Typecasts dogYears as integer
        IF dogYears = 1 THEN
                humanEquivalent ← 12
        ELSE IF dogYears = 2 THEN
                humanEquivalent ← 24
        ELSE
                humanEquivalent ← 24 + (dogYears – 2) * 4
        ENDIF
        RETURN humanEquivalent
ENDSUBROUTINE

SUBROUTINE catAge()                    # Calculates human equivalent age of a cat
        OUTPUT "Enter the age of your cat."
        catYears ← USERINPUT                    # Typecasts catYears as integer
        IF catYears = 1 THEN
                humanEquivalent ← 15
        ELSE IF catYears  = 2 THEN
                humanEquivalent ← 24
        ELSE
                humanEquivalent ← 24 + (catYears - 2) * 4
        ENDIF
        RETURN humanEquivalent
ENDSUBROUTINE
```

```
# Start of main program
anotherGo ← "y"

WHILE anotherGo = "y" OR anotherGo = "Y" # Allows for upper and lower case entry
        pet ← USERINPUT ('1. Cat, 2. Dog')
        IF pet = '1' THEN
                OUTPUT "The human equivalent age of your pet is" + catAge()
        ELSE IF pet = '2' THEN
                OUTPUT "The human equivalent age of your pet is" + dogAge()
        ELSE
                OUTPUT "Invalid choice"
        ENDIF
        OUTPUT "Do you want to use the calculator again (y/n)?"
        anotherGo ← USERINPUT
ENDWHILE
```

Chapter 7: Representing numbers

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain how data is represented by computer systems
- explain why the binary system is essential for computer processing
- convert binary numbers into decimal and vice versa
- carry out binary addition, subtraction, multiplication and division
- use left and right shifts for multiplication and division by powers of 2
- explain why hexadecimal numbers are used
- convert between binary, decimal and hexadecimal.

What your students need to know

Students should:

- have basic maths skills
- be able to use pseudo-code or a programming language to create a program to convert between decimal, binary and hexadecimal for the final challenge in the chapter.

Vocabulary

- Binary
- Number bases
- Binary shifts
- Hexadecimal
- Binary digits
- Base 10
- Place value
- Base 2
- Byte
- Overflow error
- Nibble

Common misconceptions and other issues

The value of any digit in any number system is dependent on its place value.

In binary, place values increase in powers of 2; in decimal, values increase in powers of 10; and in hexadecimal, in powers of 16.

Students often have trouble grasping that 0 is a digit or number and in decimal there are 10 digits, 0 to 9, and in hexadecimal there are 16 digits, 0 to 15.

When converting between Kilobyte, Megabyte, Gigabyte, etc., the specification states that the decimal prefix should be used.

The binary prefix multiplies a value by powers of 2 whereas the decimal prefix multiplies by powers of 10.

Unit	Decimal prefix	
Kilobyte	10^3 bytes	1000 bytes
Megabyte	10^6 bytes	1000 kilobytes
Gigabyte	10^9 bytes	1000 megabytes
Terabyte	10^{12} bytes	1000 gigabytes

Skills and coding

- Maths skills:
  - Place values in binary, decimal and hexadecimal
  - Converting 8-bit binary numbers to decimal
  - Converting decimal numbers up to 255 to binary
  - Binary addition
  - Binary shifts for multiplication and division
  - Converting between hexadecimal, binary and decimal

- Coding skills:
  - Use of pseudo-code or a high-level programming language to create a program to convert between decimal, binary and hexadecimal for the final challenge in the chapter

Skills and coding for non-specialist teachers

The final challenge allows students to create a structured program using procedures.

- A menu can be used for users to select the type of entry required, i.e. binary, decimal or hexadecimal.
- Validation can be used, for example, to ensure that 8 digits are entered for a binary number and they are either 0 or 1. A decimal number should be equal to or less than 255 and a hexadecimal one should be less than or equal to FF.
- Procedures can be called to carry out the conversions into the other number systems.

Prompting questions

- Why do you think binary consists of only two states (1 and 0)?
- When you purchase a device such as a smartphone, tablet or games console, etc., it is usually only available in very specific memory capacities, such as: 16 GB/32 GB/64 GB/128 GB, etc. Why is it always these numbers that are used? What is special about them and how does this relate to binary numbers?
- If computers can only work in binary data, why do we need to devote so much time to converting it to decimal and hex? Why not just work in binary?
- Why and when would developers prefer to use the hex number system rather than binary?
- Where are hex values most commonly seen? Why are they better at representing this data than binary? (*Answer: most commonly used to represent true colour and used as colour references, i.e. 000000 is white and FFFFFF is black.*)

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- An excellent starter to allow students to understand counting in binary is to give them a set of cards and allow them to count the dots. Resources and a lesson plan for the activity can be found at CS Unplugged (**www.cambridge.org/links/katd4007**).
- Also by CS Unplugged is a Card Flip magic trick that actually demonstrates how computers detect errors using parity bits. It's excellent at showing how error detection works, but also allows students to see how a combination of bits that can only ever be in two states can actually store a lot of data. Resources for the activity can be downloaded here: **www.cambridge.org/links/katd4007**
- Students can carry out a web search for animations explaining binary and hexadecimal number systems. Which one do they think is the best one and why?

Enrichment activities

- All numbers systems usually follow specific rules. For example, they all have a base value (in binary it's 2, in hexadecimal it's 16 and in decimal it's 10); they all use positional notation and the number of characters available within that number system is directly linked to its base number. With those factors in mind, it is possible to create our own representation system using any number as a base, for example, 'Septimal' that could be a base 7 number system. Alternatively, we don't have to use numerical characters at all; we could devise our own characters completely. Ask students to devise their own data representation/number system and then present it to the class.
- As an extension of the above activity, can students work out how they could convert data from an existing number system, such as decimal or binary, into their own system? Perhaps they could set their peers some exercises to complete.

Assessment ideas

- Ask students to spend some time looking at the structure of exam questions and mark schemes. Then ask them to devise their own set of exam questions based on what they have learnt in this chapter. For each question, they should also devise a mark scheme. They should then swap papers and answer questions written by one of their peers. The marking and mark schemes should subsequently be distributed among the class for the peer marking exercise. Groups of students should then come together to discuss the quality of the questions, the accuracy of the mark schemes and judgements based on correct/incorrect answers.
  - As an extension to this, students can rate the quality of their peers' questions and mark schemes and these can be used as part of the assessment data gathered from students.
  - Note: this exercise can be repeated for most, if not all, topics on the course.

Answers

Activity 7.1

- 0000
- 0001
- 0010
- 0011
- 0100
- 0101
- 0110
- 0111
- 1000
- 1001
- 1010
- 1011
- 1100
- 1101
- 1110
- 1111

Activity 7.2

a. 205

b. 68

c. 170

d. 240

e. 188

Activity 7.3

a. 00001101

b. 01000101

c. 10000011

d. 11000111

e. 11110101

Activity 7.4

Observe students' attempts at binary counting with fingers.

Activity 7.5

a. There are 1000 bytes in a Kilobyte and 1000 Kilobytes in a Megabyte and so on. Therefore, a Gigabyte is 1000 x 1000 x 1000 x 8, i.e. 8,000,000,000 bits.

b. 2GB

c. 1000 x 1000 x 1000, i.e. 1,000,000,000 Kilobytes

Activity 7.6

a. 100100010 – the first one is a carry over

b. 100111010 – the first one is a carry over

c. 111010110 – the first one is a carry over

Activity 7.7

a. 101101000 (In decimal: 45 x 4 = 360)

b. 100011010000 (In decimal: 141 x 16 = 2256)

c. 1011011100 (In decimal: 183 x 4 = 732)

Activity 7.8

a. 00000101 (In decimal: 45 / 8 = 5.6)

b. 00001000 (In decimal: 141 / 16 = 8.8)

c. 00101101 (In decimal: 183 / 4 = 45.8)

Activity 7.9

Students' own answers

Activity 7.10

a. Decimal: 196; Binary: 11000100

b. Decimal: 70; Binary: 01000110

c. Decimal: 250; Binary: 11111010

d. 60

e. C9

f. 8D

Chapter 8: Representing text, graphics and sound

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain how characters are represented in binary
- calculate the ASCII code for any character
- calculate the size of a text file
- explain how images are represented in binary
- calculate the size of an image file
- explain how sound is represented in binary
- calculate the size of an audio file
- explain the disadvantages of large image and audio files
- explain how file compression reduces the size of files
- explain the differences between lossless and lossy file compression.

What your students need to know

Students should:

- know why the binary number system is used for the operation of computers
- be confident in using binary numbers and converting them to hexadecimal and decimal
- be confident in using the terms bit, byte, kilobyte and megabyte and be able to convert between them
- be able to understand and create algorithms using the AQA pseudo-code
- be able to construct algorithms using sequence, selection and iteration
- be able to apply string manipulation techniques.

Vocabulary

- Character set
- Pixel
- Resolution
- Colour depth
- Sampling
- Compression
- Lossless compression
- Lossy compression
- Redundancy
- Binary tree
- Run
- Analogue and digital
- Sample rate
- Bit depth
- Huffman coding
- Run length encoding

Common misconceptions and other issues

When discussing digital images, there is often confusion around the terms 'image size' and 'resolution'.

The size of an image is determined by the number of pixels and the dimensions are given as width and then length, for example, 640 × 480 would mean 640 pixels in width and 480 pixels in length, so giving a total of 307 200 pixels.

Resolution is expressed in pixels per inch, or ppi, and is therefore influenced by the size of the displayed image. Two images having the same image size would have different resolutions if they were displayed at different numbers of pixels per inch.

Students should understand why it is necessary to compress data and should be familiar with two approaches to lossless data compression – Huffman coding and run length encoding (RLE).

They should be able to draw a Huffman tree and interpret a given Huffman tree to determine the code used for a particular node within the tree. They should also be able to calculate the number of bits saved by compressing a piece of data using Huffman coding. Similarly, they should be familiar with the process of run length encoding to reduce the amount of data stored.

Skills and coding

- Maths skills:
  - Converting between binary, hexadecimal and decimal
  - Converting between bit, byte and megabyte
  - Calculating file sizes of digital images (W × H × D) and digital sound files (sample rate × bit depth × number of channels × length (in seconds)

- Coding skills:
  - Traversing a string
  - Returning an ASCII code for a character in a string
  - Inserting a character using its ASCII code
  - Using subroutines, selection and iteration when creating an algorithm and then coding the program to compress an image file.

Skills and coding for non-specialist teachers

1 ASCII code commands

The AQA pseudo-code includes commands to return the ASCII code for a character or return the character for a code. The command CHAR_TO_CODE(CharExp) converts a character to its ASCII code and the command CODE_TO_CHAR(IntExp) converts an ASCII code to a character.

CHAR_TO_CODE('a') returns 97.

CODE_TO_CHAR(97) returns 'a'.

2 Run length encoding

The final challenge asks students to code an algorithm to carry out run length encoding. This task will consolidate the learning from previous chapters including iteration, selection and working with strings.

A possible solution, in pseudo-code and the Python programming language, is given below:

Pseudo-code	Explanation
OUTPUT "Please enter the text: " text ← USERINPUT	The user is asked to enter the string to be encoded. It is stored in the variable 'text'.
runText ← ""	This variable is declared to hold the 'runs' when the string is evaluated.
run ← 0	This variable will store the length of each run.
code ← ""	The variable code is initially given the value of an empty string.
length ←LEN(text) I	The length of the string is stored in the variable length.
IF length = 0 THEN runText ← ""	This checks that some text has been entered. If not, the 'run' is a blank string.
ELSE IF length = 1 THEN	This checks if there is only one character in the string.
runText ← text	If so, the 'run' is just that character and the length of the run is 1.
run ←1	
ELSE	If the length of the string is greater than 1, then the following code is executed.
index ← 0	The variable index is set to 0 for the first character in the string.
runText ← text[index]	The variable runText is given the value of this character.
run ← 1	As there is at least one instance of this character, the variable run is set to 1.
WHILE index < length - 1	A loop is set up to check the rest of the characters, from the character at index 1 to the last index of the string the length of the string minus 1.
IF text[index + 1] = runText THEN	If the character at the next index position is the same as the present one, then the variable run is incremented by 1.
run ← run + 1	
ELSE	If the next character is different, then the value of the present character and its run length are added to the variable code. The data type of the value stored in the variable run is changed from an integer to a string for this concatenation using the command INT_TO_STRING(IntExp).
code ← code + INT_TO_STRING (run)	
code ← code + runText	
runText ← text[index + 1]	
run ← 1	
ENDIF	
index ← index + 1	The value of the variable index is incremented by 1.
ENDWHILE	The loop is terminated.
ENDIF	
code ← code + INT_TO_STRING(run)	The value of the run is appended to the value of code if it is only 1 character.
code ← code + runText	
OUTPUT code	The result of the run length encoding is printed.

Pseudo-code	Python
OUTPUT "Please enter the text: " text ← USERINPUT	text = input("Please enter the text: ")
run ← 0	runText = ""
run ← 0	run = 0
run ← 0	code = ""
length ←LEN(text)	length = len(text)
IF length = 0 THEN runText ← ""	if length == 0: runText = ""
ELSE IF length = 1 THEN	elif length == 1:
runText ← text	runText = text
run ← 1	run = 1
ELSE	else:
index ← 0	index = 0
runText ← text[index]	runText = text[index]
run ← 1	run = 1
WHILE index < length – 1	while index < length – 1:
IF text[index + 1] = runText THEN	if text[index + 1] == runText:
run ← run + 1	run = run + 1
ELSE	else:
code ← code + str(run)	code = code + str(run)
code ← code + runText	code = code + runText
runText ← text(index + 1)	runText = text[index + 1]
run ← 1	run = 1
ENDIF	
index ← index + 1	index = index +1
ENDWHILE	
ENDIF	
code ← code + str(run)	code = code + str(run)
code ← code + runText	code = code + runText
OUTPUT code	print(code)

Prompting questions

- What is ASCII?
- How does binary affect file size and quality?
- If a single bit is used per pixel to create monochrome graphics, then how are coloured graphics stored?
- What is compression?
- Binary doesn't change: it always consists of 1s and 0s, yet it can represent everything we see on the computer. How does the computer know whether one set of binary is an image or a sound or an application?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to discuss in groups how they think computers are able to understand and manipulate text, sound and images considering that they only work in binary. Each group can be assigned to consider either text or sound or images. You might wish to encourage this to be a simple discussion without access to the Internet so that students can devise their own rules and estimate how systems work. Their discussions might not be wholly accurate but it is a good starter activity to prep them for the information that they are about to learn.
- What is their favourite colour? Ask students to find its hexadecimal code and write the binary equivalent.
- Hex Editor Neo is free software that allows users to view and edit the binary and hex representations of most files. Download it from: **www.cambridge.org/links/katd4008**.
- Ask students to import a file of their choice (graphic or sound) to see the binary/hex representation. Can they spot any patterns?
- Ask students to investigate the different types of compression.
- Ask students to find the binary representation for their name.

Enrichment activities

- Ask students to write messages to each other in binary representation only. It is the job of the recipient to convert the message back to ASCII format and send a reply.
- One useful activity to support data representation is an activity known as 'Paint by Pixels'. Resources for it can be downloaded from: **www.cambridge.org/links/katd4009**
- Ask students to investigate ASCII Art. What is it? How is it related to ASCII and binary representation? Can they collect examples of ASCII Art? Ask students to see if they can develop their own ASCII art work.
- Use Hex Editor Neo to manipulate files in their binary and hex formats. Graphics will work best, by changing the binary or hex colour codes and saving the file. Students will be able to view their altered images.
- JPEG and MP3 files are compressed files. What is the original file format and how are these compressed?

Assessment ideas

- Ask students to write a program that will ask the user to input text and convert it to its binary representation. They should convert the returned binary string into decimal values.
- Using a program such as Audacity, allow students to manipulate sound files, apply compression techniques and explore the impact on file size and quality.

Answers

Activity 8.1

The ASCII code represents characters.

Activity 8.2

```
OUTPUT "Enter the sentence to encode: "
sentence ← USERINPUT
numbChars ← LEN(sentence)

FOR index ← 0 TO numbChars – 1
    asciiCode ← CHAR_TO_CODE(sentence[index])
    OUTPUT asciiCode
ENDFOR
```

Activity 8.3

```
OUTPUT "Enter a sentence or phrase: "
sentence ← USERINPUT
OUTPUT "The size of this sentence/phrase in bytes is: " + LEN(sentence) + "."
```

Activity 8.4

01111110

01111110

01111110

01100110

01100010

01001000

00011000

00111100

Activity 8.5

4220 × 2641 × 24 = 267 480 480 bits = 33 MB

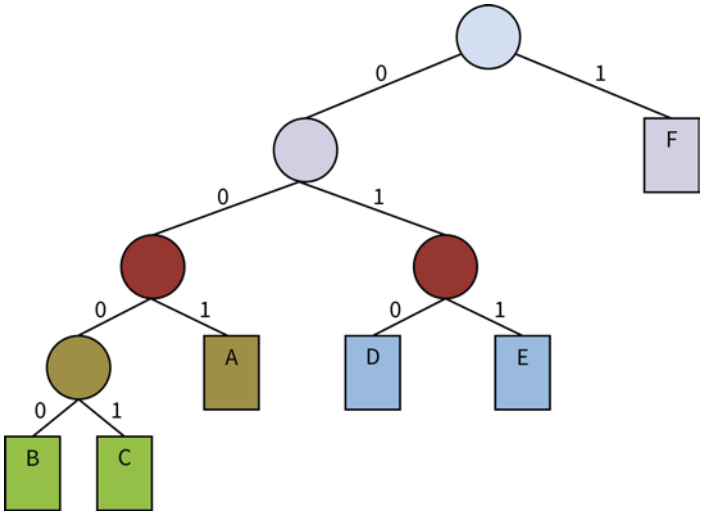
640 × 480 × 8 = 2 457 600 bits = 307 KB

Activity 8.6

44 100 × 24 × 5 × 60 = 317 520 000 bits = 40 MB

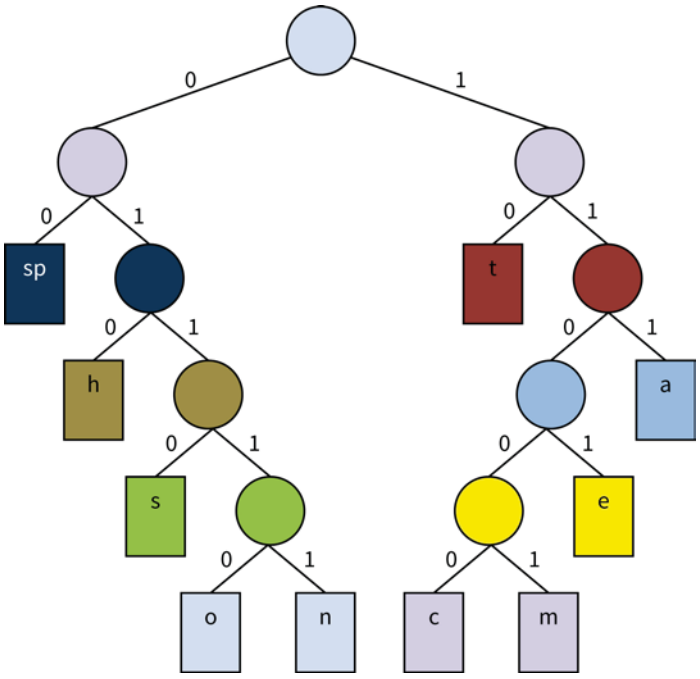
317 520 000

Activity 8.7



Character	Frequency	Bits used in ASCII	Huffman code	Bits used in Huffman code
B	2	16	0000	8
C	7	56	0001	28
A	12	96	001	36
D	13	104	010	39
E	14	112	011	42
F	85	680	1	85
		1064		238

Activity 8.8



a.

Character	Huffman coding
H	010
E	1101
M	11001

b.

Character	Huffman coding
H	010
C	11000
S	0110

Activity 8.9

2w3b3w

3w1b4w

3w1b4w

3w1b4w

3w1b4w

3w1b4w

3w1b4w

2w3b3w

= 6 × 8 bytes, that is, 48 bytes

(Without RLE the character would be 8 x 8 bytes in size, that is, 64 bytes.)

Chapter 9: Computer systems: hardware

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what is meant by a computer system
- explain what is meant by an embedded system
- describe the structure of the central processing unit and the functions of its components
- describe the fetch-execute cycle
- explain the need for and role of multiple cores and cache and virtual memory
- describe secondary storage media and the advantages and disadvantages of each.

What your students need to know

Students should:

- be competent in the use of pseudo-code and a programming language.

Vocabulary

- Processor
- Software
- Von Neumann architecture
- Fetch-decode-execute cycle
- Random access memory
- Read-only memory
- Multi-core processor
- Cloud storage Bus
- Hardware
- System Software
- Application software
- Printed circuit board
- Central processing unit
- Storage location
- Address
- Volatile
- Register
- Control signals
- Heat sink
- Parallel processing
- Multitasking
- Cache
- Secondary storage devices
- Magnetic storage
- Optical storage
- Electrical storage
- Flash memory

Common misconceptions and other issues

Multi-core processors do not produce a proportionate increase in the rate at which programs will run on a computer. For example, programs will not run at twice the speed on a dual-core processor as tasks might be sequential and not run in parallel. One task might not be able to start until another has finished.

Students often confuse ROM with secondary storage. An example to explain the difference is to think of ROM as an old vinyl record, that you could play but not change, whereas a cassette tape could be recorded onto and changed many times.

Because they are usually integral to a computer system, students sometimes think that a hard disk drive is 'primary' storage but it is just another example of a secondary storage device.

Skills and coding

No coding skills are needed for this chapter unless students undertake the final challenge.

Skills and coding for non-specialist teachers

The teaching of this chapter requires no special skills or coding.

Prompting questions

- Computer storage drives currently begin with C:\, which refers to the hard drive, and go upwards with other letters such as D:\, E:\, F:\ onwards referring to CD/DVD drives, removable storage devices and network drives. What happened to A:\ and B:\ drives? What were they used with and why are they no longer referred to?
- What is Moore's Law?
- What is the difference between RAM and ROM?
- When purchasing a new computer or games console, the technical specifications of the device will often tell you about how much RAM is built in, and that it is better to buy a device with more RAM.
  - Why is RAM so important?
  - What is RAM responsible for?
  - How does it affect computer performance?
- What is the job of a computer processor?
- Ask students to name/list common computer processor names/brands.
- What has changed in technology to allow computers to shrink in size over the years?
- What was the world's first computer?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Setting up a role play to demonstrate the fetch-decode-execute cycle is a good way to help students visualise what is happening inside the computer. For example, divide students into groups of four. Student A is responsible for generating the 'input' (i.e. a message or instruction for a task to be done), student B 'fetches' the instruction from student A and hands it to student C, whose job it is to 'decode' the information and instruct student D to carry it out. Student A represents 'input', student B the 'fetch' part of the cycle, student C the 'decode' and student D 'execute'. You can increase the complexity of this task by having more than one person responsible for executing different types of instructions. For example, one student for something written, another for something spoken, and a third for a physical action (each representing a different output device). It would then be the job of student C to decide which of these gets the correct message. The same could be done for input.

- Ask students to list as many storage devices as they can think of and then next to them detail what they are typically used for, their capacity and characteristics.
- What are the similarities between the human brain and a computer processor in the way they carry out instructions/tasks? Discuss with the class.

Enrichment activities

- Give students access to a physical computing device such as a Raspberry Pi, Arduino Board, Galileo Board, etc. Students should investigate and identify the key components on the board and what each element does. Can they identify where the CPU is? How does the device deal with memory or communicate with the other hardware devices attached to it? Ask students to set up a simple circuit with a single input and output. For example, when a motion sensor is activated, it results in a LED lighting up.
- Investigate new and upcoming technologies related to computer processing and data storage. What are the latest developments in computer memory going to be over the next five years?
- Ask students to investigate: What is Moore's law? Can the law continue as predicted? What developments will need to happen in technology in order for memory capacity to continue to increase as predicted?
- Students investigate the timeline of how computer systems have developed over the years from the world's first computers to today's technology and looking forward to future developments. What are the key things that have changed and why?

Assessment ideas

- Divide the class into groups. Assign each group a different concept from the list below:
  - Secondary storage
  - Fetch-decode-execute cycle
  - RAM and ROM
  - Central processing unit and its components
  - Computer systems and embedded systems
- Each group needs to carry out in-depth research into their chosen area to create a short interactive presentation to deliver to the rest of the class. The group's presentation should interact with the class audience and they might choose to 'teach' their peers or set a quiz. They should also create a digital message to highlight their chosen concept. This may be an animation, a program or other such media file. Each group has the opportunity to present their work to the class, and each group should participate in a Q&A session.

Answers

Activity 9.1

1. Embedded devices have been built for a specific and limited purpose. All the components of the system are on a single circuit board. The memory contains the program and the board is contained within a larger device.

2. Washing machine, dishwasher, lift, fridge, coffee maker, navigation systems, etc.

Activity 9.2

Students' own answers. They should have identified the main reason, which is that DRAM has to be refreshed periodically, otherwise it forgets what it is holding. This isn't the case with SRAM.

Activity 9.3

1. ROM: is programmed to perform a specific function when it is manufactured; the BIOS is stored in ROM and that controls what happens when the computer starts up. RAM: temporarily stores program instructions and data that are currently in use so that they can be retrieved by the CPU quickly.

2. Two differences between RAM and ROM are that RAM is volatile and all content stored within it is lost when the computer is turned off, whereas ROM is non-volatile and retains its content, even when the power is switched off. The computer cannot write to a ROM chip, whereas it can write to the RAM chip.

Activity 9.4

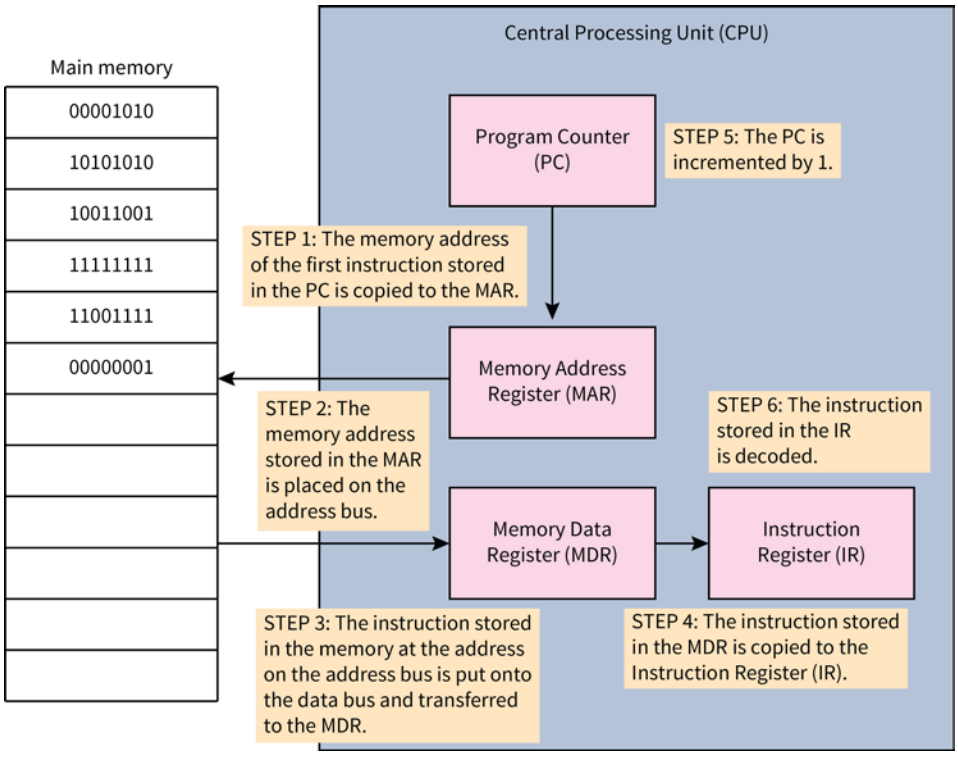
1. ALU – arithmetic and logic unit

2. Registers

3. Control Unit

Activity 9.5

The diagram(s) should illustrate the events that take place during the fetch-decode-execute cycle and the role played by the components of the CPU. Some students may prefer to produce a list of steps rather than a set of diagrams (see below).



The fetch and decode part of the cycle

At the start of the fetch-decode-execute cycle the Program Counter (PC) holds the address in memory of the first instruction to be fetched from random access memory (RAM).

The address stored in the PC is copied into the Memory Address Register (MAR).

The address stored in the MAR is placed on to the address bus. The Control Unit (CU) issues a read signal and the instruction stored at that memory address is put onto the data bus.

The instruction on the data bus is loaded into the Memory Data Register (MDR), which acts as a temporary store (buffer) for anything that is copied from memory ready for the CPU to use.

The instruction in the MDR is copied to the Instruction Register (IR).

The PC is incremented by 1.

The CU decodes the instruction stored in the IR.

Execute

The CU carries out the instruction using the Arithmetic Logic Unit (ALU) for instructions involving arithmetic and logic operations.

Once the instruction has been executed, the cycle is repeated.

Activity 9.6

1. Overclocking

2. The noise is likely to come from the computer's fan, which will be working harder because of the increase in heat resulting from overclocking.

Activity 9.7

1. Quad-core refers to four cores within one CPU. Having a multi-core processor improves performance because the cores all work in parallel either to execute one program or to work on different programs at the same time.

2. It is far quicker to retrieve memory from cache than RAM, so a larger cache means more data can be stored here and therefore be retrieved faster, improving performance.

Activity 9.8

1. The student could use solid state flash memory to back up the photos from their camera. It's portable and fast and they can buy memory cards that fit inside a camera to store the images directly. Alternatively, they could use cloud storage, which would enable them to share their photos with friends and family at home.

2. The owner of the mail order company could use a large-capacity hard disk drive for backing up the order data. It uses magnetic storage and is capable of storing vast amounts of data. Alternatively, a cloud server could be used for storing backups.

3. The school student should use a USB flash memory device, as it is portable and durable and easy to transport between school and home.

4. Magnetic storage such as a tape or hard disk drive would make a good option for a weekly backup due to its capacity and use.

Chapter 10: Computer systems: system software

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what is meant by system software
- explain what is meant by an operating system
- describe the functions of the operating system
- explain what is meant by utility programs
- list some examples of utility software and their functions.

What your students need to know

Students should:

- have knowledge of computer hardware.

Vocabulary

- Operating system
- Memory management
- Swap file
- Process
- Process management
- Multitasking
- Peripheral management
- Security management
- File management
- Permission
- User interface
- Graphical user interface
- Command line interface
- Utility programs
- BIOS
- Motherboard
- RAM
- Driver

Common misconceptions and other issues

Students sometimes do not appreciate that Windows is an operating system and its main function is to manage the operation of the computer and how it communicates with hardware. This is probably because Windows usually comes 'bundled' with applications, such as a calculator, and games, such as solitaire.

Skills and coding

Coding skills:

- Use of arrays
- Use of a high-level programming language to create and test a program to simulate disk defragmentation

Skills and coding for non-specialist teachers

The final challenge requires the students to create and code an algorithm to simulate defragmentation.

The solution, in Python, is given below.

Python	Explanation
disk = [['C', '1'], [',], ['A', '2'], ['B', '3'], ['C', '4'], ['C', '2'], [',], ['A', '3'], [',], ['B', '1'], [',], ['B', '2'], ['C', '3'], ['C', '5'], ['A', '1'], [',]]	A two-dimensional array (list) containing the sectors as shown in the figure for 'Your final challenge' in the Student Book.
letters = 'A', 'B', 'C']	An array (list) to hold the three file names.
change = 0	A variable to hold the sector that has to be swapped.
for letter in range (0, len(letters)):	The loop will go through the letters A, B and then C.
number = 1	The variable 'number' is set to 1. The first search item will therefore be A1.
swapped = 1	The variable 'swapped' signals if a swap has been made.
while swapped == 1:	The 'while' loop will run while a swap has occurred – i.e. while swap is equal to 1.
swapped = 0	The 'swap' variable is now set to 0. It will be changed back to 1 if a swap occurs.
for index in range(0, len(disk)):	The 'disk' array is now searched.
if disk[index][0] == letters[letter] and disk[index][1] == str(number):	If the array contains an entry with the first data item equal to the letter and the second equal to 'number', then it is swapped with the location at index 'choice'. In the first loop, A1 will be swapped with the items at index 0.
temp = disk[index]	This code carries out the swap.
disk[index] = disk[change]	
disk[change] = temp	
number = number + 1	'number' is now incremented, for example, from 1 to 2. Therefore in the second turn of the loop, A2 will be searched for.
change = change + 1	'change' is incremented so that the swap will occur with the next index of the array.
swapped = 1	If a swap has occurred, the variable 'swap' is changed to 1 so that the 'while' loop will turn again. If 'swap' remains at 0, then the while loop will not turn and the next letter will be searched for.
print(disk)	The array is now printed with the items in their sorted positions.

Prompting questions

- What is software?
- Windows 95, Linux, iOS, Android and Windows 10 are all examples of software.
  - What do they have in common?
  - What type of software are they?
  - Can you think of any others that fit into this category?
- Why are backups important?
- Do you use any systems utilities on your home computers?
  - Which ones do you use most?
  - When do you use them?
  - Why do you use them?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to list the different types of software that they can think of. Encourage them to try and think of a diverse range of applications. Then ask students to move into pairs or groups of three and pool their lists. Through discussion, can they group the software into different categories? What would those categories be and why? Why are particular grouping methods chosen by the students? This activity is best carried out as a starter before students have learnt about different software application categories. Follow the activity with a whole-class discussion. Did more than one group devise similar categories?
- Ask students to describe the factors that an IT systems manager would need to consider in order to select the most appropriate type of backup system for their company.
- In pairs, students discuss: what is the role of a 'driver' and how does it work? Why is it important to have the correct drivers installed on your machine? What might happen if the correct drivers are not installed?

Enrichment activities

- Investigate the 'onion diagram' for operating systems (this is the diagram at the beginning of the 'Operating system' section in the Student Book). This diagram shows the relationship between the computer hardware, the different aspects of the operating system, software applications and the user. Students should research the variations on this diagram and the relationship between the components described and then create their own improved onion diagram to illustrate this relationship.
- Investigate the terms multi-user and multitasking. What do these terms mean in relation to operating systems? Can students find examples of computer systems that are:
  - multi-user only
  - multitasking only
  - multi-user **and** multitasking
  - neither multi-user **nor** multitasking?
- Investigate different types of user interface. Find an example of each, and for each example state the software which uses that interface and discuss why this is/isn't appropriate for that application.

Assessment ideas

- Use the animation tools in a software package such as PowerPoint to create a simple animation to illustrate the concept of:
  - why a fragmented hard drive can slow a computer's performance
  - the role of the BIOS
  - the role of drivers
  - the importance of backups.

Students choose or are assigned one or more of the above.

Chapter 11: Boolean logic

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- create and interpret truth tables for Boolean operators
- draw 'AND', 'OR' and 'NOT' logic gates
- create and interpret logic circuits
- create truth tables for logic circuits.

What your students need to know

Students should:

- have knowledge of Boolean/logical operators
- be able to use and understand algorithms expressed in pseudo-code that use Boolean/logical operators.

Vocabulary

- Logical operator
- Boolean logic
- Transistor
- AND gate
- OR gate
- NOT gate
- Compound statement
- True or false
- Truth table
- Logic gate
- Logic circuit

Common misconceptions and other issues

Care should be taken when formulating and evaluating NOT statements.

For example, Q = NOT(A AND B) is not the same as Q = NOT(A) AND NOT(B).

A truth table for Q = NOT(A AND B) would be:

A	B	Q
T	T	F
T	F	T
F	T	T
F	F	T

The outcome is the reverse of a truth table for an AND statement. Any situation where both A AND B are not true would cause Q to be true.

A truth table for Q = NOT(A) AND NOT(B) would be:

A	B	Q
T	T	F
F	T	F
T	F	F
F	F	T

This statement requires both A and B to be false for Q to be true.

Skills and coding

- Coding skills:
  - Use of pseudo-code
  - Using and interpreting logical operators in pseudo-code

Skills and coding for non-specialist teachers

1 Truth tables

Truth tables can be used to check the logic of all statements using logical operators, from simple statements such as:

```
IF password = "Password!" THEN
```

where the outcome can be assessed depending on whether the statement is true or false, to compound statements such as:

```
IF (colour = "red" OR colour = "blue") AND size = "M" AND distance <= 10 THEN
```

where a table such as the one below could be used.

red	blue	M	<=10	Result
T	T	T	T	T
T	F	T	T	T
T	F	F	T	F
T	F	T	F	F
F	T	T	T	T
F	T	F	T	F
F	T	T	F	F

2 NOT statement

The NOT statement reverses the logic of AND and OR operators.

This is demonstrated before **Activity 11.1** in the Student Book and reinforced in **Activity 11.1**.

In this activity, the statement to be evaluated is:

```
IF NOT (X = 3 OR Y = 6) THEN
        OUTPUT "Conditions are met."
ENDIF
```

If either of the conditions is true, that is, if X is equal to 3, OR Y is equal to 6, then the compound statement is false.

Logic without NOT		
X	**Y**	**X OR Y**
T	T	T
F	T	T
T	F	T
F	F	F

Logic with NOT		
X	**Y**	**X OR Y**
T	T	F
F	T	F
T	F	F
F	F	T

3 Logic gates

Logic gates are built of transistors to electronically represent Boolean logic.

When using truth tables for logic gates and logic circuits, in which they are combined, true and false should be represented by 1 and 0 to signify whether there is or is not an input or output.

Prompting questions

- What's the difference between AND/OR/NOT?
- Can students come up with any examples of AND/OR/NOT?
- One example of AND/OR/NOT in everyday life might be "We are going to the circus if the day is Saturday OR Sunday AND the circus is in town". Can students come up with any other examples of their own?
- What is a compound statement?
- Why are truth tables important? What do they help us do?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to write down three examples of situations where they see Boolean logic in action. These can then be shared with a partner before feeding back to the group.
- We can often see Boolean logic in games. For example, we only want the score to double if: the character has collected the object AND the object is a diamond. Ask students to consider another example of Boolean logic within a game and to write a truth table for it.

Enrichment activities

- Ask students to go through the algorithms for the game they designed in **Chapter 2**. Identify where Boolean logic applies and draw out a truth table for each one.
- Choose one of the algorithms from the previous enrichment activity and use it to draw out the logic gate for the truth table.

Assessment ideas

- Provide students with a scenario for which they should attempt to draw out a truth table and an associated logic gate. Some examples of possible scenarios are:
  - A leisure club will charge the concession rate if visitors are under 16 or over 65.
  - A school's electronic security gates will open only if: it is a weekday and the ID card is a valid student card OR it is an administrator ID card.

Answers

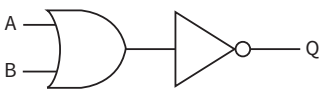
Activity 11.1

X	Y	NOT(X OR Y)
T	T	F
F	T	F
T	F	F
F	F	T

Activity 11.2

Q = NOT (A OR B)

A	B	Q
T	T	F
T	F	F
F	T	F
F	F	T



Activity 11.3

Inputs			Output
A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Because of the NOT gate, the circuit will output 1 only if both inputs A and B are 0 and input C is 1.

If there is an input of 1 at either or both of the inputs A and B, then the OR gate will output 1, which will be reversed by the NOT gate.

This can be written as:

Q = NOT(A OR B) AND C

Activity 11.4

Inputs			Output
A	B	C	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

The logic circuit will output 1 whenever input A is 0 as there will be an output of 1 at the NOT gate and this output becomes an input of 1 at the OR gate.

When input A is 1, there will be no output from the NOT gate and therefore there will be an output from the circuit only if inputs B and C are both 1.

This can be written as:

Q = NOT(A) OR (A AND B)

Chapter 12: Programming languages

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain the difference between low- and high-level languages
- explain the advantages of using high-level languages
- explain how program instructions are encoded in low-level languages
- explain why high-level languages need to be translated
- explain the characteristics and use of:
    - an assembler
    - a compiler
    - an interpreter.

What your students need to know

Students should:

- be competent in the use of pseudo-code and a programming language.

Vocabulary

- Applet
- Execute
- Machine code
- Instruction set
- Instruction
- Opcode
- Operand
- Mnemonic
- Assembler
- Compiler
- Interpreter
- Machine language
- Assembly language
- High-level language
- Low-level language

Common misconceptions and other issues

The instructions for a microprocessor must be presented in machine code that consists of strings of 1s and 0s. Each type or family of processor has its own set of instructions, known as its instruction set.

To assist programmers, programming languages have been developed that use commands more related to human languages. They are at a higher level of abstraction.

Assembly language is at a low level of abstraction and the instructions used have a one-to-one relationship with those of machine code.

High-level languages, for example, Python, Java and C, are at a higher level of abstraction.

The code from these languages must be translated for the processor. An assembler is used for assembly language and compilers and interpreters for high-level languages.

Skills and coding

- Coding skills:
  - The final challenge introduces students to a CPU simulator with a limited instruction set. The students are encouraged to investigate the simulator and code simple programs to sort and multiply numbers.

Skills and coding for non-specialist teachers

The Little Man Computer simulator is at: **www.cambridge.org/links/katd4010**

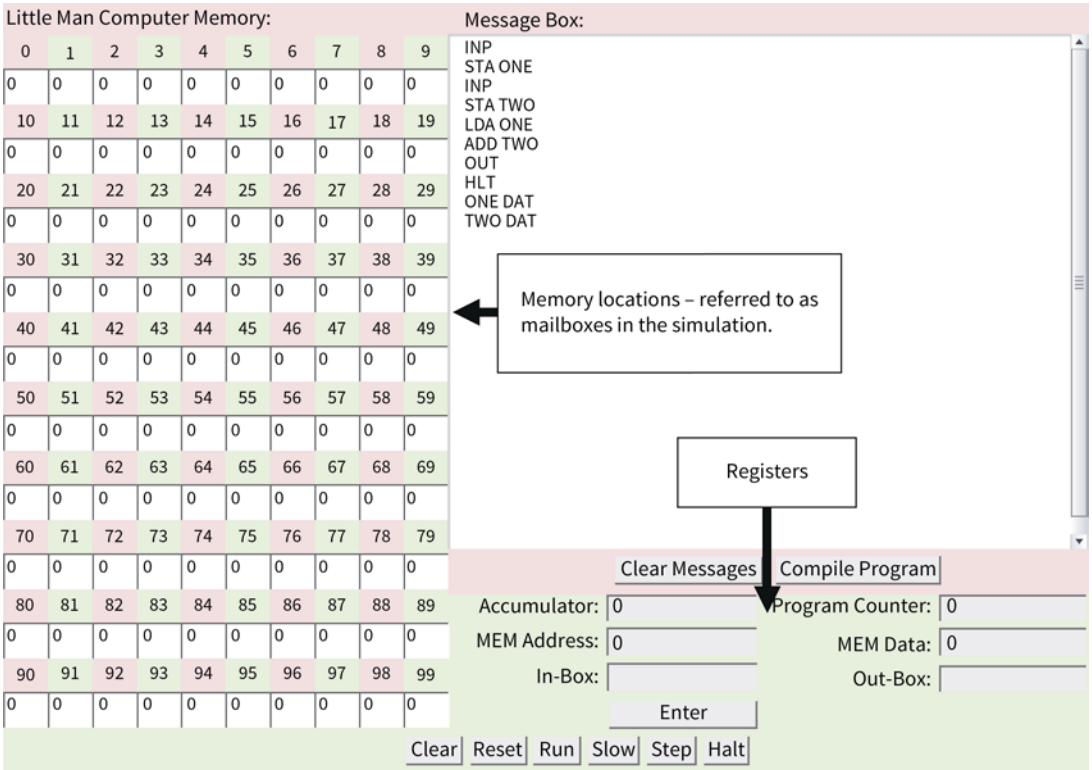
Simulators provide a good introduction to assembly language programming and Little Man Computer simulator also illustrates the fetch-execute cycle, which is explored in more detail in **Chapter 9**.

The simulator has nine instructions that can be entered using mnemonics.

The following is a simple program to input and store two numbers and then add them together.

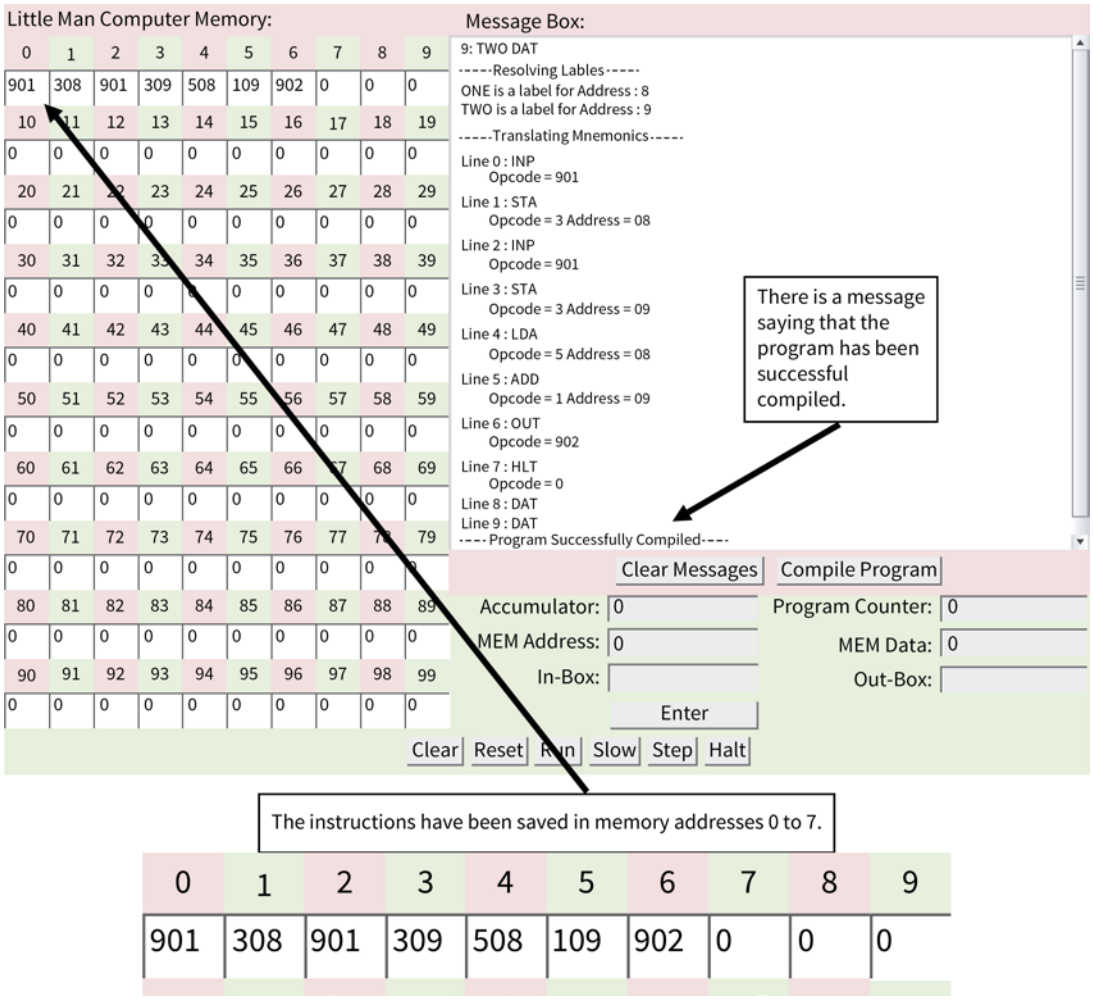
INP	Mnemonic for user input for the first number.
STA ONE	Store the first number in memory location labelled ONE.
INP	User input for the second number.
STA TWO	Store the second number in memory location labelled TWO.
LDA ONE	Load the contents of memory location ONE into the accumulator.
ADD TWO	Add the contents of memory location TWO to the accumulator.
OUT	Output the contents of the accumulator.
HLT	Stop execution of the program.
ONE DAT	These commands reserve data locations for the two numbers to be entered.
TWO DAT	

The following diagram shows the program in the message box.



Before the program can be run, it must be compiled into machine code.

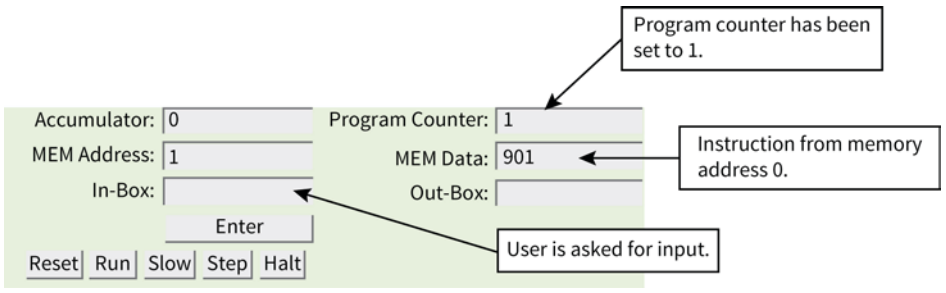
This is done by clicking on 'Compile Program'.



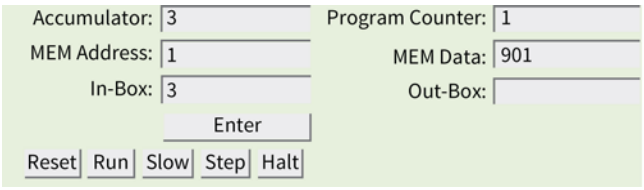
Address	Opcode	Explanation
0	901	This is the opcode for the mnemonic INP-user input.
1	308	3 is the opcode for STA. 08 is the memory location where it will be stored.
2	901	This is the opcode for the mnemonic INP-user input.
3	309	3 is the opcode for STA. 09 is the memory location where it will be stored.
4	508	5 is the opcode for LDA and 08 is the location of the data to be loaded into the accumulator.
5	109	1 is the opcode for ADD and 09 is the location of the data to be added to the accumulator.
6	902	902 is the opcode for OUT-the contents of the accumulator will be output for the user.
7	0	0 is the opcode for HLT-execution of the program will stop.

The program can be run one statement at a time by clicking the 'Step' button.

0

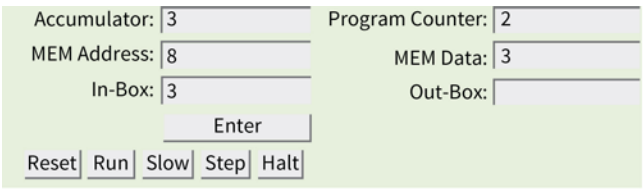


1



After data has been input and enter button clicked, the number is copied to the accumulator.

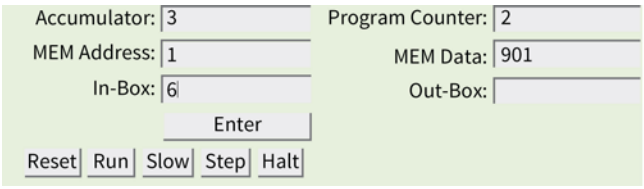
2



The second instruction (308) has copied the contents of the accumulator to memory location 8.

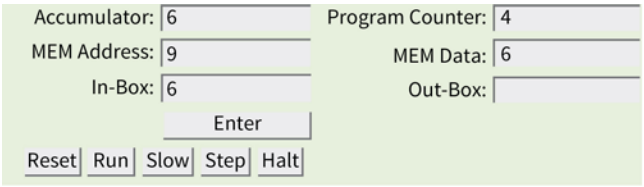
0	1	2	3	4	5	6	7	8	9
901	308	901	309	508	109	902	0	3	0

3



The user is asked to enter the second number by the instruction at memory location 2 (901).

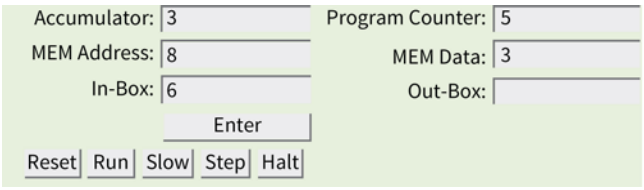
4



It is copied to the accumulator and then memory location 9 by the instruction at memory location 3.

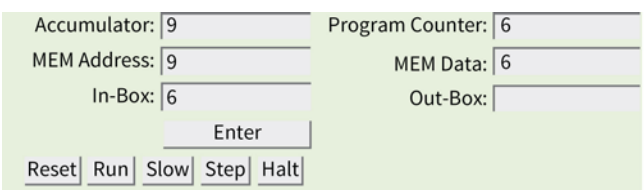
0	1	2	3	4	5	6	7	8	9
901	308	901	309	508	109	902	0	3	6

5



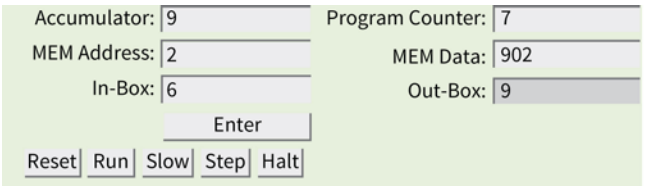
The data at memory location 8 is copied to the accumulator by the instruction at memory lacation 4 (508).

6



The data at memory location 9 is now added to the accumulator by the instruction at memory location 5 (109).

7



The contents of the accumulator are now copied to the out-box by the instruction at memory location 6 (902). The instruction at location 7 (0) then halts program execution.

Prompting questions

- How many programming languages can you name?
- What is the difference between a high-level and a low-level language?
- What is an interpreter?
- What is a compiler?
- What is machine code?
- Does the programming language being learnt by the class use an interpreter or a compiler? Why do you think this is the case?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to list as many programming languages as they know. Are different languages recommended for different purposes?
- Can they find examples of machine code?
- Students investigate which programming languages use compilers and which use interpreters.

Enrichment activities

- The term 'programming generations' refers to a classification applied to programming languages. Investigate what the different generations are and what they refer to. Can students categorise the different languages into the different categories?
- HTML is a mark-up language. PHP and JavaScript are scripting languages. Do they use interpreters or compilers? Why do students think this is the case? Ask them to justify their answer.
- SQL is often considered to be a fourth generation language. Ask students to investigate this to find out what the general consensus is on the issue. What are the most common SQL commands and what is the language commonly used for?
- Prolog is considered to be a fourth generation language. Ask students to investigate why this is the case and what the most common commands are. What is the language most commonly used for?

Assessment ideas

- Write a program that allows the user to enter a series of numbers and return the total value.
  - Write a program in your chosen language to run the above program.
  - Write a program using 'Little Man Computer' to execute the program.
  - Write a machine code to execute the above program.

Answers

Activity 12.1

Load the number found at memory location 6:	0010 00000110
Add the number 113:	0100 01110001
Load the number 10:	0001 00001010
Add the number 21:	0100 00010101
Store the result at memory location 30:	0000 00011110

The number 31 would be found at memory location 30.

Activity 12.2

Students' own answers

Chapter 13: Computer networks

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain what is meant by a computer network and list the different types of network
- discuss the benefits and risks of computer networks
- explain the functions of the hardware needed to connect computers
- explain how computers communicate using cable and wireless
- describe network topologies
- explain how users connect to and use the Internet
- explain how data is transmitted across networks
- explain the use of protocols.

What your students need to know

This chapter does not involve any coding and no specific skills or knowledge are required.

Vocabulary

- Personal area network (PAN)
- Local area network (LAN)
- Wide area network (WAN)
- Cables
- Microwaves
- Protocols
- Ethernet
- Frequency
- Wi-Fi
- Bandwidth
- Topology
- Node
- Modem
- IP address
- Domain name
- Packet
- Packet switching
- Computer network
- Network interface card (NIC)
- Transmission media
- Wireless access point
- Bus topology
- Star topology
- Switch
- Internet
- Word wide web (WWW)
- Router
- Networking layers

Common misconceptions and other issues

Hubs and switches: hubs relay messages received from each computer to all of the others on a single network, whereas switches inspect the messages and relay them only to the intended recipients.

They can do this as they build tables recording the MAC addresses of each computer.

Wireless access points are similar to hubs in that they do not relay messages to specific computers.

Routers connect different networks and, like switches, they can direct messages as they inspect each message.

Ethernet and Wi-Fi are both suites (or families) of protocols for communication within a network. Wi-Fi can be thought of as the wireless equivalent of Ethernet.

Wi-Fi is only one standard for wireless communication. Others include Bluetooth, 3G and 4G.

Students are often confused over the difference between the Internet and the World wide web.

It should be stressed that the Internet is a huge wide area network that allows communication between computers. The WWW is one of the services that run on the Internet and others include email, file transfer, instant messaging and chat rooms.

The WWW is a system of interconnected documents formatted in HTML.

Skills and coding

Coding skills:

No specific coding skills are required.

Skills and coding for non-specialist teachers

The chapter does not require any skills or coding.

Prompting questions

- What is a network?
- Why is a network useful?
- When have you used a network?
- Who has a network at home? Why did you choose to set up a network? What benefits have you received?
- What is the difference between a Personal Area Network and a Local Area Network?
- What is the difference between a MAC address and an IP address?
- In a school network, which device would be preferable, a hub or a switch? Why?
- How is information sent across the Internet?
- What is the relationship between cloud computing and networks?
- How does cloud computing work?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Draw a diagram that illustrates what a network is. This is a quick activity and students should not need more than a minute or so to do a rough sketch. After they have their drawing, ask them to compare results. Are there any similarities or differences? Ask students to justify or explain why their diagram looks the way it does. If used as a starter activity, this will help highlight any prior understanding or misconceptions about networking that students may have.
- Find and describe an example of:
  - Personal Area Network
  - Local Area Network
  - Wide Area Network.
- Networks Unplugged is a series of activities that teaches the concepts of networks without using computers. It includes activities asking students to role play packet switching and a library mystery hunt among others. Resources can be downloaded from: **www.cambridge.org/links/katd4011**
- Investigate common protocols and what they do.

Enrichment activities

- Using animation software (or the animation tools in a presentation software package), ask students to create an animation that demonstrates how a network is structured and in particular how data is sent across a network.
- Ask students to write an essay on the following scenario: 'Imagine waking up tomorrow and you find that networks no longer exist. Although computers exist, each machine is a stand-alone machine and incapable of directly communicating with another device. Networks do not exist in any form. Describe your day: how different would your life be?'
- Ask students to investigate the school network. What type of network is it? What topology does it use? Can they identify which hardware components are used in the computer suite?
- Taking the students on a 'school trip' to the server room to meet the IT manager works well with this topic. Invite the IT manager to explain the structure of the network, how it is set up and his daily responsibilities. Students can prepare questions beforehand to ask on the day.

Assessment ideas

- Ask students to draw and annotate a diagram of their home (or school) network. The diagram should be labelled to show the main hardware components used and should illustrate network structure. Each student should be given the opportunity to present their network diagram and explain it.

Answers

Activity 13.1

1. a. PAN stands for personal area network. A PAN is a user-oriented wireless network covering a relatively small geographical area – typically no more than 10 square meters. Bluetooth is the most well-known communication medium for a PAN. A PAN enables a user to connect various personal digital devices such as a wireless mouse and a laptop, a wireless headset or a sports band with a smart phone, or a camera with a tablet. It may also allow devices to connect to the Internet.

b. LAN stands for local area network. Although a LAN spans a wider geographic area than a PAN, it is still confined to a single site such as a school, an office or a home. A LAN can be wired or wireless or in some instances a combination of the two. Computers and other digital devices, such as TVs, printers and speakers, connected on a LAN can communicate and share data with each other. A LAN is usually owned and managed by a single organisation.

c. WAN stands for wide area network. A WAN connects separate networks over a large geographical area, enabling computers in one location to communicate with computers in other locations. The biggest WAN of all is the Internet. WANs typically use a number of different communications media, including fibre optic cables, microwave and satellite. Ownership of a WAN is likely to be shared by several organisations.

2. Three benefits for a school using networked rather than stand-alone computers are:

- Data stored on the school network can be accessed by authorised users from anywhere in the school. It's also considerably easier and much less time-consuming to back up data stored centrally compared with having to go round each stand-alone computer in turn separately backing up the content of its hard drive.
- The school doesn't need to purchase as much equipment since printers and other devices can be connected to the network and shared.
- Software installation is also much more straightforward and less time-consuming. Software is installed centrally and then copied to all computers over the network. It's also easier to keep software up to date and to prevent students or members of staff from installing illegal software.

Three risks for a school of using networked rather than stand-alone computers are:

- Having all the computers connected on a network allows viruses to spread more rapidly.
- If someone successfully hacks into the school network, they have unimpeded access to all its data, including personal information about students and members of staff.
- There are cost implications – additional hardware and infrastructure are needed to set up a network and the school will need to employ a network manager with the necessary technical expertise.

Activity 13.2

Benefits and drawbacks of setting up a wired home network:

- Cables allow for higher bandwidth, which is great for media streaming.
- Setup is much more difficult: hiding cabling so it doesn't become a safety hazard will need planning. It's easier to do this when a house is being built rather than retrospectively.
- Better security: you need to be physically plugged in.
- Not as portable since you need to be directly plugged in; for example you can't work from your garden unless you have a cable.

Benefits and drawbacks of setting up a wireless home network:

- Usually lower bandwidth than cable.
- Easy and cheap to set up; you just need WAPS.
- Signal can be affected by interference and distance so you might lose your Wi-Fi signal or data transfer slows down.
- Very portable; you can work from anywhere.
- Security is poor. Anyone within range can see the network and connect to it and use it. The access point must be secured with a security password. Some form of encryption must be set up.

An alternative would be to have a mixture of the two, with devices such as the TV and the printer that have a fixed location wired in to the router, and other devices such as smart phones and tablets connected wirelessly.

Activity 13.3

Benefits of using a star topology:

- If one computer fails, the other devices on the network will be unaffected and will carry on working.
- Adding or removing devices is easy and can be done without affecting the entire network.
- Data packets can be directed to the intended node directly without having to pass along the complete network. Consequently, there is less network traffic and fewer collisions.

Chapter 14: Cyber security

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain the need for and importance of cyber security
- describe the different strategies that criminals use to attack computer networks
- explain how people are the greatest security risks to networks
- describe the threats posed to networks
- explain how these threats can be identified, prevented and combatted.

What your students need to know

Students should:

- be able to use a high-level programming language to create a menu-based information system about the security risks faced by computer users and how they can be avoided.

Vocabulary

- Cyber security
- Social engineering
- Antivirus software
- Malware
- Trojan horse
- Input sanitisation
- Authentication
- Penetration test
- Bot
- Biometric authentication
- Patch
- Blagging
- Phishing
- Pharming
- Antivirus software
- Shouldering
- Virus
- Worm
- Spyware
- Adware
- Brute force attack
- Denial of service
- Data interception
- SQL injection
- Zero day attack
- Physical security
- Encryption
- Firewall
- MAC address filtering

- Network policies
- CAPTCHA

Common misconceptions and other issues

The material in this chapter is very straightforward and should not give rise to any misconceptions.

Skills and coding

- Coding skills:
  - The use of a high-level programming language to create a menu-driven information system.

Skills and coding for non-specialist teachers

Final challenge

This will give students experience of applying some of the programming skills learnt in earlier chapters to create a structured program, using subroutines called from within the main program.

Prompting questions

- Describe a recent computer crime that you might have heard of in the news or via your social network.
  - Why was it a criminal act?
  - Could it have been prevented?
  - How?
- What is meant by the term 'network forensics'? What would this team of experts do? When would they be used?
- What is shouldering? Has anyone ever been a victim or perpetrator of this?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Ask students to research a recent news report describing a computer crime. They need to find out what happened and then in pairs/small groups discuss if it could possibly have been prevented. Can they discover which legal Act is enforced here and why?
- Compare the school's acceptable use policy with one from another organisation (for example, this one from the University of Bath: **www.cambridge.org/links/katd4012**). Ask students to identify common features and differences.
- Think-Pair-Share. Think: of one thing that you have understood well during the lesson and one question that you would like answered. Pair: tell your partner about your area of confidence and your question: can you answer each other's question? Share: your common areas of confidence and any unanswered questions. Can someone else in the class answer them?
- Evaluate the security of your own devices and home network. What strategies do you employ? Could you improve the security? Should you improve security?
- What possible threats might you face with your personal devices and home network? Think-Pair-Share your thoughts and in groups discuss ways that these can be prevented.

Enrichment activities

- Ask students to research a criminal attack on a network and to prepare a two-minute presentation to the class describing the attack and important details surrounding it. What computer security features were breached?
- An excellent activity to explain public key encryption is available on CS Unplugged (**www.cambridge.org/links/katd4013**). The activity is a good one to carry out with students, or the video listed below it also helps students to understand the nature of the process.
- Ask students to create a digital message that describes computer crime/misuse and various ways to prevent these from happening.
- Ask students to write a program that will test the strength of a user's password upon entry. The program should output to the user whether their password is 'weak', 'medium strength' or 'strong'. It should also give appropriate suggestions about what can be done to improve the password strength.

Assessment ideas

- You have been given a training position with a network department of a new school that is just starting. Your task is to write a report outlining the range of risks that the school network will face and the security measures that should be put in place to prevent them.

Answers

Activity 14.1

a. Phishing email

b. Clue 1 – The email is not addressed to Catherine in person.

Clue 2 – The writing style is careless; for example, 'is terminated' rather than 'will be terminated.'

Clue 3 – Urgency: they want Catherine to respond within 24 hours or else.

Another clue is the inclusion of a link that Catherine is asked to click on. This will almost certainly lead to a website controlled by the criminals.

Activity 14.2

1. A virus is a program that finds its way into a user's computer via another program or file. Once there, it can attach itself to other programs and files. In contrast, a user has to actively install a Trojan. They do this unwittingly either by opening an email attachment or by being fooled into thinking the software is legitimate. Both viruses and Trojans are harmful. They can corrupt data, delete files and (in the case of Trojans) enable criminals to access personal information including IDs and passwords.

2. Precautions users should take include:

- Installing firewalls to ensure software isn't downloaded without their knowledge.
- Keeping their computer's operating system up to date and installing the latest security patches.
- Installing anti-virus, adware removal and anti-spyware protection software.
- Avoiding opening emails and attachments from unknown sources.
- Only downloading programs from trusted websites and taking proper note of all security warnings, licence agreements and privacy statements.

Chapter 15: Ethical, legal and environmental impacts of digital technology on wider society

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- investigate and discuss the following issues in relation to the development and impact of digital technologies:
    - environmental
    - ethical
    - legal
    - cultural
- discuss issues of data collection and privacy
- describe the legislation relevant to digital technology.

What your students need to know

This chapter does not involve any coding and no specific skills or knowledge are required.

Vocabulary

- Ethics
- Legal
- Hacking
- Cracking
- Autonomous
- Regulation of Investigatory Powers Act 2000
- Wearable technology
- Copyright
- Patent
- Environmental impact
- Data centre
- E-waste
- The digital divide
- Legislation
- Data Protection Act
- The Computer Misuse Act
- Copyright Designs and Patents Act
- Creative commons licensing

Common misconceptions and other issues

Students often have difficulty in understanding the difference between 'ethical' and 'legal'.

They should be encouraged to discuss examples of situations which are legal but might not be considered ethical, for example: capital punishment, refusing to help an injured person.

Skills and coding

- Coding skills:
  - No specific coding skills are required.

Skills and coding for non-specialist teachers

No specific skills or coding are required for this chapter.

Prompting questions

- Would you prefer to take your exam on the computer or using paper? Why?
- What benefits do you think computing has made to your life?
- Computer legislation is often difficult to enforce. Why do you think this is the case?
- 'Technology has made the world a smaller place.' What do you think this statement means?
- Why do you think people prefer to use pirated copies of entertainment media rather than purchase the actual product?
- Is current computer legislation successfully tackling computer crime in the UK? If not, what do you think needs to change?
- What impact do you think wearable technology is having or will have on our lives?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- In groups, ask students to discuss how their lives today are different from the lives and childhood of their parents.
- Give students a short paper-based quiz, then ask them to complete an online version (one can be chosen, or specifically designed using the many tools available). In groups, students should compare their experiences and explore the benefits and drawbacks of both approaches. This task should end with a class discussion summarising the differences.
- What are the 10 commandments of computer ethics? Can they be legally enforced? The website **www.cambridge.org/links/katd4014** can be given to students if needed.
- What is Digital Rights Management?
- Write three key points about each legal act dealing with computer misuse.
- Write a poem/song/short story that discusses one of the computer laws or an ethical issue discussed in the chapter.
- Find examples of wearable technology. Choose two from your list and consider the potential benefits and drawbacks of using such technology.
- If you could write a quote about computers and the ethical, legal or environmental concerns surrounding them, what would you say? Write your own quote. Note to teacher: these student quotes, when printed, would make a good display feature for the classroom if presented in the right way.

Enrichment activities

- Ask students to find out by speaking to older friends and family members, such as aunts, uncles, parents, grandparents, etc., what childhood was like for them.
  - What did lessons look like in school?
  - What did they do for fun?
  - How did they keep in touch with their friends?
- Students should compare these findings with their own lifestyle and answer the questions for

themselves. This task can be extended by asking students to create a digital message highlighting the similarities and differences between the two age groups.

- Students should select and investigate one of the topics below. They should consider both sides of the argument and end with their own opinion and justification of it. Their research findings should be summarised to be delivered as an interactive presentation to the class.
  - E-waste
  - Climate change
  - Energy production
  - Social communication
  - Drones
  - Security and surveillance
- Computer piracy is a growing issue. A news story (**www.cambridge.org/links/katd4015**) has highlighted the success of Netflix in Brazil, a country known for its widespread piracy issue. Ask students to read the news story. Why has Netflix been so successful? In groups, ask them to discuss what measures they think the entertainment industry needs to take to tackle and reduce computer piracy in the UK.
- The digital divide still exists in the UK. Ask students to investigate what the government intends to do to tackle it.
- Artificial intelligence has increased and improved significantly over recent years. Ask students to investigate examples of current technology that uses artificial intelligence. How many examples can they find? What impact do they think these examples have had on our lives?
- Wearable technologies are increasing and perhaps the most famous examples are Google Glass and the Apple iWatch. Yet the Apple iWatch has been more socially acceptable than Google Glass. There was lots of excitement around the project before release, but this quickly dwindled when the product came to market. Google even released guidance for its users. Ask students to investigate the reviews and news reports around Google Glass. What were the issues and debates surrounding the technology? Why did Google have to undertake a review of the entire project?

Assessment ideas

- Ask students to write an essay on: 'Everyone needs to learn to code.' Discuss.
- Choosing one of the quotes from the list below, ask students to investigate the quote. What did they think the speaker was referring to? Do they agree with what was said? Using the quote as a guiding point, ask the students to write an essay to discuss the issues highlighted in the quote.
  - *Computers themselves, and software yet to be developed, will revolutionize the way we learn.* Steve Jobs
  - *Personally, I rather look forward to a computer program winning the world chess championship. Humanity needs a lesson in humility.* Richard Dawkins
  - *Home computers are being called upon to perform many new functions, including the consumption of homework formerly eaten by the dog.* Doug Larson
  - *I do not fear computers. I fear the lack of them.* Isaac Asimov
  - *Computer science is no more about computers than astronomy is about telescopes.* Edsger Dijkstra
  - *Security is, I would say, our top priority because for all the exciting things you will be able to do with computers – organizing your lives, staying in touch with people, being creative – if we don't solve these security problems, then people will hold back.* Bill Gates
  - *Every piece of software written today is likely going to infringe on someone else's patent.* Miguel de Icaza
  - *The Internet is not just one thing, it's a collection of things – of numerous communications networks that all speak the same digital language.* Jim Clark
  - *Supercomputers will achieve one human brain capacity by 2010, and personal computers will do so by about 2020.* Ray Kurzweil

Answers

Activity 15.1

This activity focuses on the role of computer scientists in combating global climate change. Students should avoid broadening out the discussion to include other ways in which computer science can help protect the environment, such as animal conservation or natural resource management.

They should begin by defining the term 'global climate change', that is, a long-term shift in weather patterns including temperature, rainfall and winds, and identifying factors that contribute to climate change (in particular the build-up of greenhouse gases caused by human activity), notably the burning of fossil fuels.

They should then describe a number of different ways in which computer scientists can play a role in combating climate change, such as:

- developing computer systems that combine satellite observations, ground-based data and forecast models to monitor and predict changes in the weather and climate
- producing computer models to explore the impact of climate change on water resources, crop yields, fish stocks, etc. and using computer models to aid planning
- contributing to the development of renewable energy sources to reduce our dependence on fossil fuels
- developing smart systems for homes, offices, cars, etc. that use energy more efficiently
- developing battery technology so that energy generated (when the sun is shining or on a windy day) can be stored and used when required.

They should find appropriate examples to support their discussion.

This activity could provide useful exam practice, demonstrating how to tackle a question that requires an extended, essay-style answer.

Activity 15.2

Students' own answers. They should pick up on the fact that there was inadequate testing and documentation, an over reliance on just one person and failure to react quickly enough when problems were reported.

Activity 15.3

This activity is designed to generate some lively classroom debate. There is no need for students to delve too deeply into the Luddite movement. Instead, the discussion should focus on the impact of new technology. For example, students could review the effects of automation on labour-intensive repetitive jobs in sectors such as car manufacturing, food processing and retail sales. They might also want to consider how advances in artificial intelligence and robotics could impact on 'professional' jobs in fields such as Medicine, Transport and Finance. This theme is picked up again in **Activity 15.8,** which focuses on digital inclusion.

Activity 15.4

There is a whole host of ethical dilemmas associated with the growing use of robots that students might want to consider. These include issues associated with privacy, security and human dignity as well as the moral obligations of society towards its robots. A starting point for this discussion could be to investigate Isaac Asimov's 'three laws of robotics', drawn up in the 1940s.

Students might find it helpful to focus on a few specific examples, such as driverless cars, home robots, combat robots or robotic surgery.

Hopefully students will conclude that robotics is a good thing, with the potential to transform the way we live and work, but that it is important to put in place appropriate limitations and controls on their use.

Activity 15.5

This follows on from **Activity 15.4** and is designed to elicit further lively classroom debate. Students could investigate the codes of conduct issued by the three main professional bodies, The British Computer Society (BCS), Association for Computing Machinery (ACM) and Institute of Electrical and Electronic Engineers (IEEE), to see if they shed any light on this ethical dilemma. All three require their members to treat people fairly, honour contract agreements and adhere to company policy, which provides something of a steer in this case.

Students could try to find examples of serious software bugs and then rank them in order of severity. This should provoke some interesting debate. What algorithm did they apply to decide on the order? Should bugs that threaten human life always be dealt with first? What happens if there's a very slim chance of this happening?

Activity 15.6

This is another interesting topic for a class discussion. Students should begin by summarising what the Regulation of Investigatory Powers Act (RIPA) was designed to do; that is, to govern the interception and use of electronic communications. This is to ensure that the way investigatory powers are used by organisations such as councils and government departments complies with human rights law, in particular the European Convention on Human Rights.

In response to the increased threat from terrorist organisations, the police and security services are calling for even greater powers to monitor Internet traffic, mobile devices and other forms of electronic communication, as well as to extend the use of CCTV surveillance in public places. Students might want to consider whether this increased intrusion on privacy is justified. Do law-abiding citizens have anything to worry about? Should we be concerned that surveillance could be used for reasons other than those stated?

The RIPA was passed back in 2000. Since then, surveillance technology has moved on apace. Students might also want to investigate the implications of the draft Investigatory Powers Bill currently going through parliament – regarded by some as a snoopers' charter.

Activity 15.7

a. Examples of how new technology impacts on the way in which people interact with one another could include:

- Individuals are more likely to use social networking sites, such as Facebook and Twitter, to socialise and keep in touch with friends and family members, than to interact with them face to face.
- Social networking sites and other online forums enable individuals to communicate with more people across greater distances faster than ever before. However, information that might once have been shared privately is now often published online for anyone to read and see.
- Social interaction within the family can suffer when individual family members are preoccupied with their smart phones, tablets, games consoles, etc. Instead of spending time interacting with each other, they are engrossed in their own digital world.

b. Examples of how new technology impacts on work and employment could include:

- Although demand for highly skilled workers is increasing, some jobs are disappearing as automated systems eliminate the need for human workers.
- Technology supports more flexible working practices, enabling people to work anywhere at any time, but potentially making it more difficult for them to maintain a healthy work–life balance.
- Technology makes it possible to outsource some jobs away from areas where wages are high and workers have well-established employment rights to developing parts of the world where employment costs and overheads are lower.

c. Examples of how new technology impacts on education could include:

- Textbooks (like this one) are no longer limited to merely text and pictures. They have links to additional web-based materials, including animations, videos, practice assessments, etc. to support the learning of new content and make learning more interactive.
- Online education and training are available to anyone anywhere, providing they have access to the Internet.
- It is much quicker and easier to find information online than it is to visit a library. But information overload can be a problem. Good information handling skills are required to sift through the huge amounts of information available online.

d. Examples of how new technology impacts on leisure could include:

- Digital entertainment, including music, films and books, is available on demand 24/7 via the Internet.
- Multi-user online games enable users to play with other people from all over the world.
- Fitness gadgets such as the Apple Watch enable users to monitor their activities and motivate them to do more.

Activity 15.8

Chapter 15 stresses the benefits of new technology. The premise of this activity is that these benefits should be available to all, irrespective of wealth, gender, ethnicity, etc., and that society should attempt to ensure that this is the case.

Students may want to summarise factors contributing to the digital divide and evaluate efforts being made by governments to promote digital inclusion.

Activity 15.9

a. Five reasons why data stored online is less secure than paper-based storage are listed below.

- It could be hacked.
- It could be maliciously damaged or destroyed by malware.
- It could be stored on servers belonging to a third party over whom the data owner has no control/jurisdiction.
- It could be damaged or destroyed as a result of hardware or power failure or human error (though a sensible back-up and recovery regime should prevent this).
- Large-scale theft of digital records is much easier than stealing large quantities of paper records – thieves would need an articulated lorry to steal 10 000 paper records.

b. Five advantages of online storage over paper-based storage are listed below.

- Data stored online can be accessed from anywhere, providing there is an Internet connection.
- Multiple users can have access to the same files simultaneously.
- In the event of a fire, flood or other disaster that affects your place of work, data stored online will remain unaffected.
- There's no need to set aside physical space to store paper files or to predict how much storage space is likely to be required in the future. Cloud storage is much more flexible: you only use and pay for the amount you need and can have more or less, as and when required.
- Encryption can be used to add an additional layer of security to digital data stored online.

Activity 15.10

a. Intentional and unauthorised destruction of software or data

b. Unauthorised access with intent to commit further offences.

c. Unauthorised access to computer material

Activity 15.11

Students' own answers

Non-examination assessment (NEA)

The non-examined assessment component targets primarily Assessment Objective AO3 but also AO1 and AO2. It is marked out of 80 and has a weighting of 20 per cent.

AQA will issue the assessment task at the start of the final academic year of assessment.

It requires students to design, implement, test, refine and evaluate a computer program to solve it. It will then be assessed in the following areas: Designing the solution (9 marks), Creating the solution (30 marks), Testing the solution (21 marks), Potential enhancements and refinements (10 marks), Overall quality of the report (10 marks). Students may complete the assignment over multiple sessions, up to a combined duration of 20 hours.

The program must be written in one of the high-level languages stipulated by AQA:

- C#
- Java
- Pascal/Delphi
- Python
- VB.Net

All work submitted by a student must have been done under observation by their teacher and the final report must be only their own work. External sources can be used but must be referenced and no marks can be awarded for materials submitted which are not the learner's own.

The students are not allowed to take the NEA tasks nor any work associated with them home.

Teachers cannot give detailed advice and suggestions as to how the work may be improved in order to meet the assessment criteria. This includes indicating errors or omissions and personally intervening to improve the presentation or content of the work.

The report can be submitted in either hard copy or electronic format.

The project should be marked according to the marking criteria using a 'best fit' approach. For each of the marking criteria sections, teachers must select one of the three band descriptors provided in the marking grid that most closely describes the quality of the work being marked.

Extra guidance for non-specialist teachers

When completing the final challenge in each chapter, and for any other programming project, you may want to check your students' understanding of the following points.

Starting a programming project

- Identify the main requirements of the solution – what it has to do. Students should be prepared to justify these in their project report.
- Decompose the problem listing all of the sub-tasks.
- Make a **requirements specification**, listing what each of the solutions for the sub-tasks should achieve. Students will need these when they are writing the enhancements and refinements section of the report.
- List the **performance or success criteria** that will be used to judge whether or not the solution successfully solves the problem.
- List all of the inputs and outputs required for the solution and the validation techniques that will be required.
- List the data structures that will be required (e.g. variables and arrays).

Completing a programming project

You may want to encourage your students to create a checklist to ensure they have covered all aspects needed in a programming project. Below is an example of a checklist; this can be scaled up and down according to the type of project being worked on.

Item	Completed
Pre-project preparation	
outline of the problem, stating what the solution is expected to achieve	
list of all the sub-tasks	
requirements specification, listing what the sub-task solutions should achieve	
performance or success criteria that will be used to judge the success of the solution	
identification of inputs, outputs and validation	
Designing the solution	
designs for all of the components	
identification of all variables and assignment of meaningful names to each of them	
identification of all data structures	
design for a user interface	
use of a modular design, using functions and/or procedures	
algorithms for all of the components	
display of algorithms as flow diagrams	
display of algorithms as pseudo-code	
commented pseudo-code to explain what it is intended to do	
Creating the solution	
explanation of how the program developed at each stage	
screen prints showing development	
list of all the problems encountered and changes made as a result	
tests carried out on each component with results and any changes made	
list of all resources used	
The code	
modular – use of functions and/or procedures	
comments explaining each module	

Item	Completed
all variables have meaningful names consistently displayed	
validation and error-handling routines	
The following techniques have been used (if required):	
• variables with descriptive names in a consistent style (e.g. camel case)	
• different data types, including Boolean, string, integer and real	
• arrays	
• operators	
• the three basic programming constructs used to control the flow of a program: sequence, conditionals and iteration	
• suitable loops, including count and condition-controlled loops	
• string and array manipulation	
• file-handling operations: open, read, write and close	
all techniques used appropriately	
Testing the solution	
test plan covers all of the performance criteria	
all tests include test data, expected results and actual results	
normal, extreme and erroneous data used	
explanation of how errors were rectified	
use of screen prints to illustrate test results	
Potential enhancements and refinements	
evaluation compares all the success criteria against the finished solution and the test results	
discussion of efficiency and robustness	
discussion of potential improvements	
The overall quality of the report	
report is complete	
all sections are in ordered sequence	
all spellings have been checked	
range of technical terms used correctly	
consistent style and layout	
table of contents	
numbered captions, diagrams and screen prints	
references for any sources used	

Worksheets: Answers

ANSWERS

Worksheet 1.1

1 (a) An **algorithm** is a step-by-step procedure for solving problems. It is a set of instructions that can be followed by humans and computers.

(b) There will be many possible solutions, but the sequence should be correct.
This is an example answer.

Leave the house
Walk to the bus stop.
Wait for a bus to arrive.
If it is the correct bus, then get on.
Sit on the bus until it reaches the school.
Get off the bus.
Walk into school.

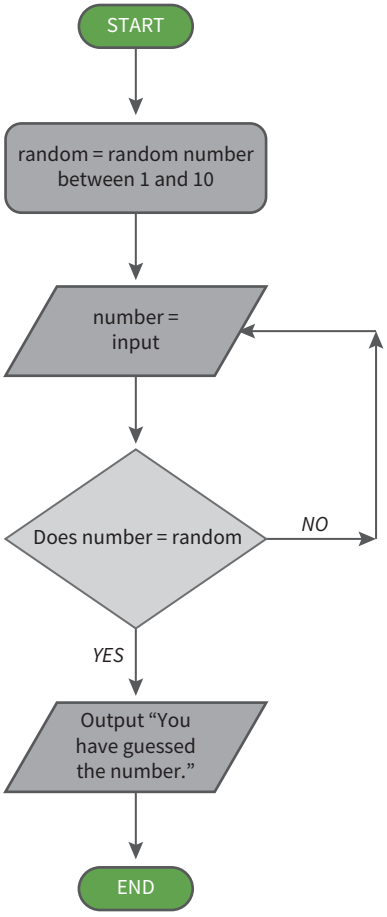
(c) Sequence

Worksheet 1.2

1 (a) A **selection** in an algorithm is where a question is asked, and depending on the answer, the program takes one of two courses of action.

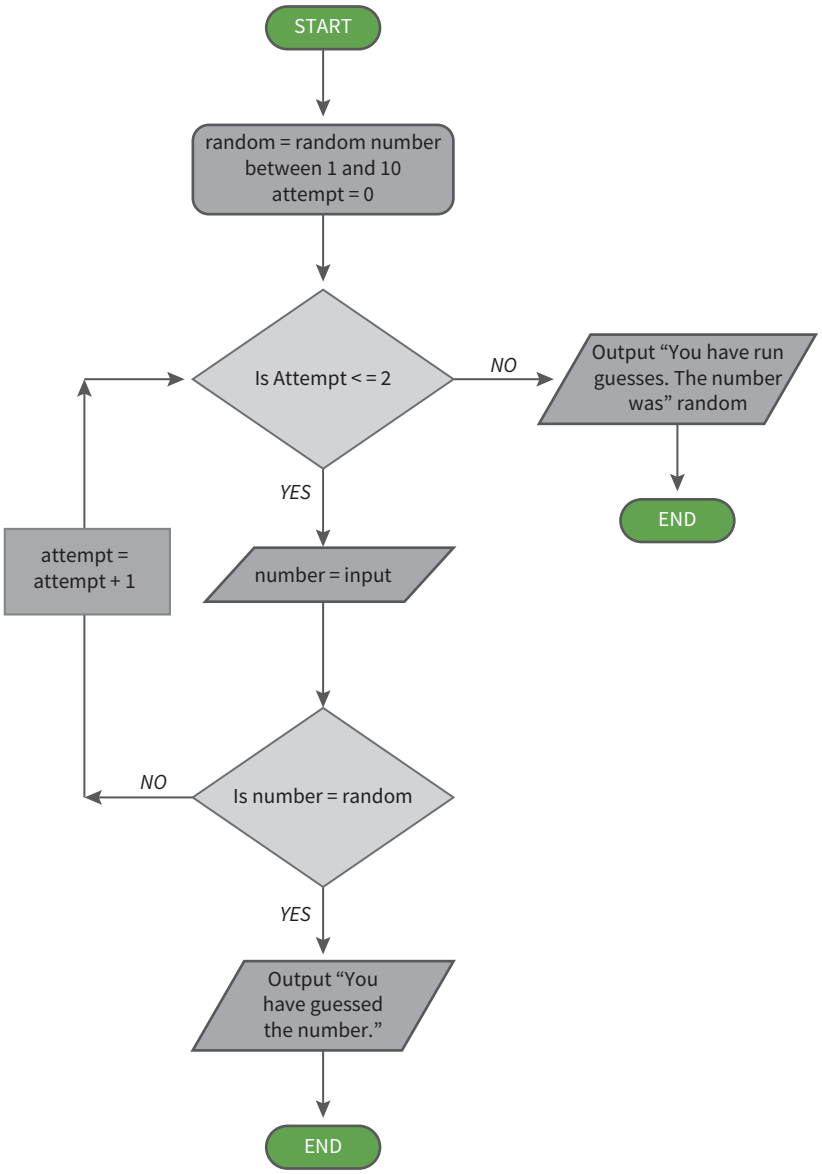
(b) An **iteration** in an algorithm is where a task is repeated for a set number of times or until there is a required outcome.

2



Worksheet 1.3

1



Worksheet 1.4

1 (a) Result = 7

(b) Result = 1

(c) Result = 27

2 (a) (i) True
(ii) False
(iii) True

(b) (i) false
(ii) true
(iii) false

Worksheet 1.5

1

```
OUTPUT  (“Please enter the first number.”)
number1 ← USERINPUT
OUTPUT  (“Please enter the second number.”)
Number2 ← USERINPUT
OUTPUT  (“Please enter the third number.”)
Number3 ← USERINPUT

IF number1 > number2 THEN                # number1 is compared with number2
       IF number1 > number3 THEN         # if it is greater, then it is compared
                                         with number3
              highMark ← number1         # if it is greater, then it must be
                                         the largest
       ELSE
              highMark ← number3         # if not, then number3 must be the largest
       ENDIF
ELSEIF number2 > number3 THEN            # if number2 is greater than number1 it is
                                         compared with number3
       highMark ← number2                # if it is greater then it must be
                                         the largest
ELSE
       highMark ← number3                # if not then number3 must be the largest
ENDIF
OUTPUT “The largest number is “ + highMark
```

Worksheet 2.1

1 (a) An **iteration** in an algorithm where a task is repeated for a set number of times or until there is a required outcome.

(b) A **definite iteration** is when the number of iterations is known before the execution of the loop is started, for example, it may be set to three or five times and it will execute that number of times, whatever the conditions, unless there is a command to break out of the loop.

2

```
OUTPUT “Please enter a number.”
number1 ← USERINPUT
OUTPUT “Please enter a higher number.”
number2 ← USERINPUT
FOR index = number1 TO number2
       print index
ENDFOR
```

Worksheet 2.2

1 (a) In **indefinite iteration** (also known as conditional iteration), the number of iterations is not known before the loop is started. The iterations stop when a certain condition becomes true or false.

(b)

```
numberEntered ← 0
total ← 0
reply ← "y"
WHILE reply = "y"
        OUTPUT "Please enter a number."
        number ← USERINPUT
        numberEntered ← numberEntered + 1
        total ← total + number
        OUTPUT "Enter 'y' to enter another number or 'n' to stop."
        reply ← USERINPUT
ENDWHILE
OUTPUT "You entered" + numberEntered + "numbers and the total is " + total
```

Worksheet 2.3

1

```
# Nested loops are used to check all of the possible combinations
# Nested loops are used to check all of the possible combinations
FOR numberCoffees = 1 to 100
    FOR numberTeas = 1 to 100
        total ← (numberCoffees * 1.9) + (numberTeas * 1.2)
        IF total = 285 THEN # if the total equals 285 the number
                            of teas and coffees are stored
            teas ← numberTeas
            coffees ← numberCoffees
            totalTaken ← total
        ENDIF
    ENDFOR
ENDFOR
OUTPUT totalTaken
OUTPUT teas
OUTPUT coffees
```

Worksheet 2.4

1

Array				Index	Total	Output
3	9	6	13	0	3	
3	9	6	13	1	12	
3	9	6	13	2	18	
3	9	6	13	3	31	31

2

a	b	c	output
10	0	0	
9	1	8	
8	2	14	
7	3	18	
6	4	20	
5	5	20	
4	6	18	
3	7	14	
2	8	8	
1	9	0	1, 9, 0
0	10	-10	0, 10, -10

Worksheet 3.1

1 (a) An **integer** is a whole number without decimals (can be positive or negative).

(b) A **real** number is any number that exists including their decimals and fractions.

(c) A **string** is a list of characters of any length. It can include alphanumeric data and symbols.

2 (a) Boolean

(b) String

(c) Integer

(d) Integer

Worksheet 3.2

1 In the algorithm there are two logical errors. For each, state the line number and how the error could be corrected.

Line number: 1

Correction:

```
length ← LEN(string) - 1
```

Line number: 3

Correction:

```
found ← false
```

```
Before the WHILE statement the variable 'found' must be declared as found ← false
```

Worksheet 4.1

1 (a)

Pass 1						
Jackson	Brown	Morrison	Andrews	Fleming	Smith	Wilkinson

Pass 2						
Brown	Jackson	Andrews	Fleming	Morrison	Smith	Wilkinson

Pass 3						
Brown	Andrews	Fleming	Jackson	Morrison	Smith	Wilkinson

Pass 4						
Andrews	Brown	Fleming	Jackson	Morrison	Smith	Wilkinson

(b) Another pass is needed as the algorithm must continue until there are no swaps in a pass and in the next pass, there will be no swaps.

Worksheet 4.2

1 A **merge sort algorithm** is used to sort an unordered list by repeatedly (recursively) dividing a list into two smaller lists until the size of each list becomes one. The individual lists are then merged with the items in the correct order.

2

Morrison	Jackson	Brown	Wilkinson	Smith	Fleming	Andrews

Morrison	Jackson	Brown	Wilkinson		Smith	Fleming	Andrews

Morrison	Jackson		Brown	Wilkinson		Smith	Fleming		Andrews

Morrison		Jackson		Brown		Wilkinson		Smith		Fleming		Andrews

Jackson	Morrison		Brown	Wilkinson		Fleming	Smith		Andrews

Brown	Jackson	Morrison	Wilkinson		Andrews	Fleming	Smith

Andrews	Brown	Fleming	Jackson	Morrison	Smith	Wilkinson

Worksheet 4.3

1 A **linear** search algorithm is a simple, sequential search. It starts at the beginning of the list and moves through the items, one-by-one, until it finds a matching value or reaches the end without finding one.

A **binary** search algorithm can only search an ordered list to find an item by looking at the middle (median) item and comparing it with the search value. If the search item is smaller than the median then the median of the sublist to the left is searched and vice versa, if it is larger. This continues until the search item is found or there are no items left to search.

2 (a) For a **linear** search, the best case would be 1 selection if the search item was at the start of the list and 1000 if it was the last item.

(b) For a binary search, the best case would be 1 if the search item was the median and 10 for the worst case as the following medians could be chosen – 500, 250, 125, 63, 32, 16, 8, 4, 2, 1.

Worksheet 4.4

1 The item to be searched for is stored in the variable 'item'.

```
low ← 0                                  # 'Low' is the first item in the
                                         array named 'list'.

high ← LEN(list) - 1                     # 'High' is the last item in the list.

found ← False

WHILE low <= high AND found = False      # The search will continue
                                         while there are items in the list and
                                         the search item has not been found.

       midpoint ← (low + high)/2         # Finds the mid point.

       IF list[midpoint] = item THEN

              found ← True

       ELSEIF item < list [midpoint] THEN

              high ← midpoint - 1        # If the item is lower than the
                                          median, then the sublist to the
                                          left is searched.

       ELSE

              low ← midpoint + 1         # If the item is higher than the
                                          median, then the sublist to the
                                          right is searched.

       ENDIF

ENDWHILE

IF found = True THEN

       OUTPUT "The item is in the list."

ELSE

       OUTPUT "The item is not in the list."

ENDIF
```

Worksheet 5.1

1 **Validation** is the process through which the program checks that data is sensible, reasonable and appropriate to be processed by the program.

(a) A **range check** makes sure that the data is in a certain range or falls between two points (e.g. is less than 20 or is between A and F).

(b) A **length check** ensures that the data has a certain number of characters (e.g. that a password has a minimum of eight alphanumeric characters).

(c) A **presence check** is usually used when users have to enter data, ensuring that they have actually entered something and that the data is present.

Worksheet 5.2

1 The array is **friends.**

```
myFile ← openWrite("Friends.txt")
FOR index = 0 TO LEN(friends) - 1 # The loop could also be 0 to 6 as there
                                  are seven items.
myFile.writeLine(friends[index])
ENDFOR
myFile.close()
```

Worksheet 6.1

1 **Decomposition** means breaking a problem down into smaller, more manageable parts which are then easier to solve. It is an example of the 'divide and conquer' approach to problem solving.

2 Decomposition of the problem should identify some of the following sub-problems:

- Design of interface showing the 3×3 grid.
- How to keep track of which squares have been selected by 'X' and '0' and which are free.
- How the 'computer' will decide which square to select when it is its turn.
- How the 'computer' will decide when the game is over.
- How the 'computer' will decide who has won a game and keep count of the score.
- How the user can ask for another game or quit the program.

Worksheet 6.2

1 **Abstraction** is the process of removing unnecessary details so that only the main, important points remain. When creating an algorithm to model a real-life action or activity, abstraction identifies essential elements that must be included in the computer models and discards inessential ones.

2 (a)

```
SUBROUTINE dice()                  # This is the function definition.
                                   No parameters are passed to it.
      dice1 ← RANDOM_INT(1, 6)     # This generates a random number between
                                   1 and 6.
      dice2 ← RANDOM_INT(1, 6)
      totalDice = dice1 + dice2
      RETURN totalDice
ENDSUBROUTINE


score ← dice()                      # This is the statement in the main
                                   program which calls the function.
                                   No arguments are passed to it.
```

(b) The important feature or property of a dice is that it generated a random number between 1 and 6. This is the only property that needs to be represented in the model. Other features such as colour or the material it is made from are unimportant.

Worksheet 6.3

1 A **subroutine** is a self-contained module of code that can be 'called' by the main program when it is needed.

2 **Functions** are called from an expression in the main program and return a value to it. **Procedures** are called from a statement in the main program but do not return any value to it. They just do something and then the main program continues.

3

```
SUBROUTINE finalPrice(price)      # This defines the function with
                                  the parameter 'price'.
      IF price > 200 THEN
            payment ← price - (price/100*10)
      ELSEIF price >= 100 AND price <= 200 THEN
            payment ← price - (price/100*5)
       ELSE
            payment ← price
      ENDIF
      RETURN payment              # Payment is passed back to the main
                                  program where it is stored in the
                                  variable 'customerPayment'.
ENDSUBROUTINE

cost ← total of all the goods bought by the customer.

customerPayment ← finalPrice(cost) # 'cost' is passed to the function as
                                  an argument. In the function, it is the
                                  parameter 'price'.
```

Worksheet 6.4

1 The answer should contain the following points:

- Subroutines are a natural way of implementing computational thinking because some of the tasks identified can be allocated to a subroutine. They therefore assist with decomposition and abstraction.
- Repeated sections of code need only be written once and called when necessary. This shortens the development time of a program and means that the finished program will occupy less memory space when it is run.
- Subroutines improve the structure of the code, making it easier to read through and follow what is happening.
- It's easier to check the program because each subroutine can be coded and tested independently.
- The program is easier to debug as each subroutine can be inspected independently.

- If changes have to be made at a later date, it is easier to change a small module than having to work through the whole program.
- In large development teams, different members can be working independently on different subroutines.
- Standard libraries of subroutines can be built up and they can be reused in other programs.

Worksheet 7.1

1 The microprocessor contains the central processing unit, which carries out all of the program instructions by carrying out millions of calculations each second.

These calculations are performed by billions of transistors acting as switches. They are either 'on' or 'off'. They have only two states: they either transmit an electric current or they do not.

Any system involving two states is called a binary system.

As there are only two states ('off' or 'on'), they can be represented by the two digits of the binary system: 0 and 1.

All computer programs are lists of instructions switching transistors 'off' or 'on' and therefore they can be represented by the digits 0 and 1.

2

Decimal prefix is used.

Unit	Number of bits
Megabyte	8 000 000
Nibble	4
Byte	8
Terabyte	8 000 000 000 000
Kilobyte	8 000
Gigabyte	8 000 000 000

3 213 624 133 bits = 26 703 016.6 bytes
= 26 703 kilobytes
= 26.7 megabytes

Worksheet 7.2

1

(a) (i) 213
(ii) 110
(iii) 170

(b) (i) 11101001
(ii) 10011001

Worksheet 7.3

1 An overflow error occurs when a calculation produces a result that is greater than the computer can deal with or store in a single byte.

2 (a)

0	0	1	1	1	1	0
1	0	1	0	0	1	0
0	**1**	**1**	**0**	**0**	**0**	**0**

(b)

	0	1	1	1	1	1	0	1
	1	1	0	0	1	1	1	1
1	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**

(c)

0	0	1	0	1	1	0	0
1	**0**	**1**	**1**	**0**	**0**	**0**	**0**

Worksheet 7.4

1 Computers do not understand or use hexadecimal; they only understand and use binary.

Hexadecimal is used by computer scientists because people get confused with large binary numbers, so we simplify binary numbers by representing them in hexadecimal notation, meaning that fewer numbers are used.

2 (a) DD

(b) 92

3 11001101

Worksheet 8.1

1 The character set is the complete list of binary codes that can be recognised by computers as being usable characters.

2 (a) C

(b) a

3

```
FOR index = 0 TO  LEN(myString) - 1
      OUTPUT myString(index) + " " + CHAR_TO_CODE(myString(index))
ENDFOR
```

Worksheet 8.2

1 (a) The **image size** is the number of pixels in the width and then the number of pixels in the height, for example 640 × 480 or 2048 × 1536.

(b) The **image bit depth** or colour depth is the number of bits used to encode the colour of each pixel in an image. The greater the number used, the greater is the image quality.

2 Image size = 4096 × 2480 × 24 = 243 793 920 bits = 30.4 megabytes.

3 The **sample rate** is the number of samples of the sound taken each second. The higher the sample rate, the more accurately the sound will be represented. A sample is a physical measurement of the sound recorded as a binary value.

4 File size = 180 × 2 × 44 100 × 16 = 254 016 000 bits = 31.7 megabytes

Worksheet 8.3

1 (a) During **lossless compression**, no data is lost and the file can be decompressed with all its information intact.

(b) During **lossy compression**, data is lost in the compression process and when the file is decompressed, it will not contain all the original information

2 A lossless compression algorithm would more successfully compress a black-and-white image file than a true colour one because there are long runs of only two colours, either black or white, but a colour photograph has short runs of many different colour variations (16 777 216).

3 wwwwwwbbbwwwwwwbbbbbbwwbbbwbbbwbbwbbb = 6w3b6w6b2w3b1w3b1w2b1w3b

Worksheet 9.1

1 The **hardware** consists of the physical devices of a computer system, such as the keyboard, processor and storage devices.

The **software** consists of all the programs that the hardware uses to operate.

2 (a) An **embedded system** is a computer system built into another device in order to control it. The components are on a single printed circuit board.

(b) It monitors the water temperature so that it can turn the heating element on and off to maintain the correct temperature.

(c) Any of these three are suitable: television, microwave, cooker.

Worksheet 9.2

1 (a) The **control unit** co-ordinates the actions of the computer by sending out control signals to the other parts of the CPU, such as the ALU and registers, and to the other components of the computer system, such as the input and output devices.

(b) The **arithmetic and logic unit (ALU)** performs arithmetic and logical operations. It carries out activities such as, addition and subtraction, multiplication and division, logical tests using logic gates and comparisons such as whether one number is greater than another.

(c) **Registers** are storage locations within the CPU itself. They can be accessed even more quickly than the main memory. Some of these registers serve specific functions, but some of them are general purpose registers used for the quick storage of data items.

(d) **The clock** regulates the timing and speed of all computer functions. Within the clock is a quartz crystal which vibrates at a particular frequency when electricity is applied to it. Pulses are sent out to the other components to co-ordinate their activities and ensure instructions are carried out and completed. One instruction can be carried out with each pulse of the clock, and therefore the higher the clock rate, the faster the CPU will be able to carry out the program instructions.

Worksheet 9.3

1 (a) A **quad core processor** has four core processing units within the CPU.

The advantages of multiple core processors over single core processors are:

- the cores can work together on the same program; this is called parallel processing.
- the cores can work on different programs at the same time; this is called multitasking.

However, not all programs will run at four times the speed with a quad-core processor. The tasks required might not be able to be carried out in parallel. They might be sequential so that one task requires output from a previous task, so the second task cannot start until the first has finished.

(b) **L1, L2** and **L3** cache are memory modules consisting of fast dynamic RAM within or very close to the CPU. The fastest is the Level 1 cache and is smaller than the Level 2 and Level 3 caches.
The caches speed up the processing speed of the CPU by storing recently used data and data likely to be frequently used so that so that data does not have to be collected from the slower RAM when it is needed. The L1 cache is checked first followed by the L2 and then L3 caches. In a multi-core processor, the cores have their own L1 and L2 caches.

Worksheet 9.4

1 Secondary storage devices are necessary because RAM is volatile. Data is stored on devices called secondary storage devices so that it is not lost when the computer is switched off. The secondary storage devices also allow data to be transferred between devices.

2

Use	Magnetic, optical or solid state	Reason why this is the most appropriate
Storing images in a digital camera	Solid state	It is light and as there are no moving parts; it is not damaged when devices are knocked or dropped.
Storage devices on a school's main fileserver	Magnetic	Store large amount of data and are relatively cheap Fast data access speed
Videos of a school production to be given to parents	Optical	Cheap and easy to transport data between locations
Handheld devices used by students for fieldwork	Solid state	It is light and as there are no moving parts; it is not damaged when devices are knocked or dropped.

Worksheet 10.1

1 (a) The **memory manager** is in charge of the RAM. Programs often need to use the RAM throughout their operation. Some programs will use the RAM extensively whereas other smaller programs will use it less frequently. The memory manager checks that all requests from programs for memory space are valid and allocates them accordingly.

(b) The **peripheral manager** controls all the computer input and output by managing requests from programs to use devices such as printers, speakers, keyboards and hard disk drives.

The peripheral manager communicates with the devices through software called 'drivers', which translate the instructions sent by the device manager into a more understandable format.

(c) The **file manager** controls all the different files on the system. It controls file permissions, such as a user's ability to see or open a file, write to a file or delete it. It is therefore important for the security of the system. File management also helps organise and control files so that they are as user-friendly as possible'. It helps to protect the user from accidental mistakes too.

Worksheet 10.2

1 (a) **Utility systems software** perform specific tasks related to computer functions, resources, files and security. They help to configure the system, analyse how it is working and optimise it, improving its efficiency.

(b) (i) **Disk defragmentation** tools are used to rearrange the parts of files on the disk drive. When a file is saved to a disk, parts of the file might be saved in different areas of the disk. These tools try to move all the parts to the same area so that they can be accessed more quickly.

(ii) **File compression** software is used to make the files smaller so they take up less storage space and can be transmitted to other users more easily.

(c) (i) **Encryption software** uses an algorithm to encrypt (scramble) a file according to the key which is used; the key is needed to decrypt the file back to its original form. The file can therefore only be opened by authorised access.

(ii) **Firewalls** are either software or hardware devices that protect against unauthorised access to a network, and are primarily used to prevent unauthorised access from the Internet. They can be configured to prevent communications from entering the network and also to prevent programs and users from accessing the Internet from the network.

Worksheet 11.1

1 (a) AND gate

(b) NOT gate

(c) OR gate

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

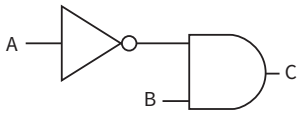
Worksheet 11.2

1

A	B	C	D	E	Q
0	0	0	1	0	1
0	0	0	1	1	1
0	1	0	1	0	1
0	1	0	1	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1

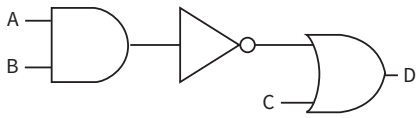
Worksheet 11.3

1



A	B	C
0	0	0
0	1	1
1	0	0
1	1	0

2



A	B	C	D
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	0
1	1	1	1

Worksheet 12.1

1 The **instruction set of a computer** is the set of instructions for its particular processor that it can understand and is able to process.

2 **Machine code** or **machine language** presents the instructions in a form that the processor can execute; strings of 0s and 1s.

3 (a) **Assembly languages** are called **low level languages** because they are very similar to machine code in concepts and structure. There is a one-to-one relationship between the commands in assembly language and those in machine code. Assembly language is at a low level of abstraction from the machine language.

(b) The binary opcodes which are binary in machine code are represented as descriptive words called **mnemonics** in assembly language (e.g. SUB, MUL and DIV). These are easier for humans to remember and use.

4 An **assembler** translates the mnemonics of assembly language programs into machine language instructions for the microprocessor to execute.

Worksheet 12.2

1 (a) and (b) Answers could include:

- Faster program development: it is less time consuming to write and then test the program.
- It is not necessary to remember the registers of the CPU and mnemonic instructions.
- Portability of a program from one machine to other: each assembly language is specific to a particular type of CPU, but most high-level languages are generally portable across different CPUs.

2 (a) Programs written in high-level languages (source code) must be converted into machine code by a translator into machine code (object code) before the processor can execute them as it can execute instructions only if they are presented in machine code.

(b) (i) A **compiler** translates the source code into a standalone, machine code program (object code) which is output as a new file that can then be executed by the processor.

An **interpreter** translates the high-level code line-by-line into machine code each time it is run.

(ii) **Benefits of compiler:**

- The program that is run is already translated into machine code, so it can be executed more rapidly.
- It protects the software from competitors who would otherwise be able to see the source code.

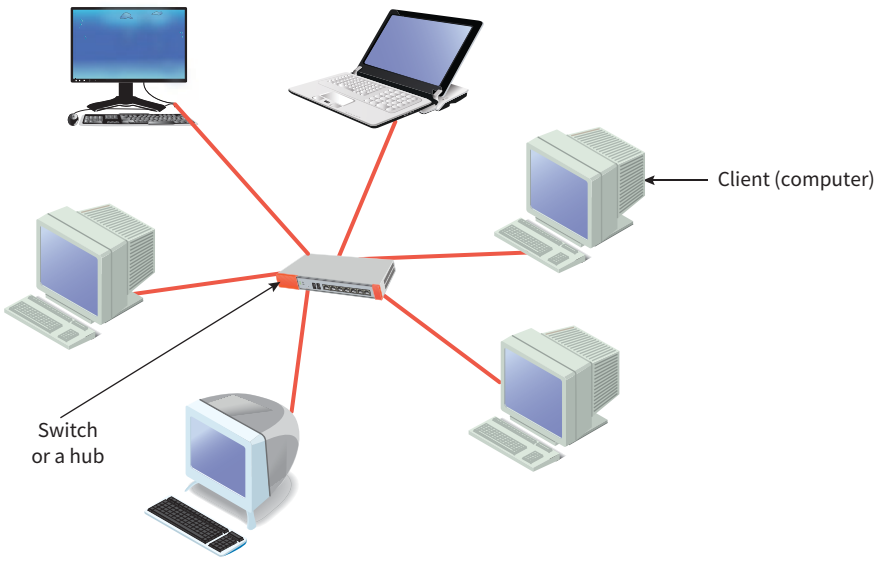
- If it encounters any errors, it carries on.

Benefits of interpreter:

- When an error is found, the interpreter reports it, stops and pinpoints the error, so that the programmer knows where it has occurred.
- The code is not platform-specific and can be run on different operating systems and platforms if there is an interpreter.
- The program can be easily edited as it always exists as source code.

Worksheet 13.1

1 (a) Diagram to show computers connected to a single point.



Using a star topology, each computer is connected individually to a central point, which can be a file server or switch. The computers are individually connected by cable to a switch. The cable is connected to the network adapter in each device.

Required hardware: a network adapter for each computer, cabling, and a switch or a hub.

(b) **Advantages**:

- Easy and cheap to install as only access points are required.
- Very mobile. Users can access the network from anywhere on the site. They can move from room to room and remain connected.

Disadvantage:
Any one from:

- Far lower bandwidth leading to lower download and upload speeds.
- Security is poor. Anyone within range can see the network and connect to it to use it. The access point must be secured with a security password and some form of encryption must be set up.

- The signal can be affected by walls and electronic equipment such as microwave ovens. It is also affected by distance from the access point and the number of connected computers.

Worksheet 13.2

1 (a) **Domain names** are used to identify IP addresses as they are more convenient to use and easier to remember than the four octets of binary numbers (e.g. mysite.org is easier to remember and enter than 192.100.321.003).

(b) **The domain name service (DNS)** returns the IP address when a client requests access using a domain name. When a browser requests access to a host using its domain name, the client computer contacts a DNS server. The DNS server contains a database of domain names which allows it to look up the domain name and returns the IP address. This is known as resolving the domain name.

2 When devices transmit data, the data are broken down into small pieces called packets. These are sent separately, and then joined up at the end so that the message is complete. This process is called packet switching.

- These packets are then sent onto the network using cables or microwaves as in a wireless network.
- Routers on the network inspect each packet and decide on the most efficient path for the packet to take on the next stage of its journey.
- To do this, each router has a configuration table containing information about which connections lead to particular groups of addresses.
- The routers can balance the load across the network on a millisecond-by-millisecond basis.
- If there is a problem with one part of the network while a message is being transferred, packets can be routed around the problem, ensuring the delivery of the entire message.
- The final router can direct the packet to the correct recipient.

Worksheet 13.3

1 (a) **TCP** (Transmission Control Protocol)/**IP** (Internet protocol).

(b)

- Transport layer
- Internet layer or Network layer
- Data link layer or Network access layer.

(c) (i) and (ii) any two from:

- **FTP (File Transfer Protocol):** provides the rules that must be followed when files are being transmitted between computers.
- **HTTP (Hypertext Transfer Protocol):** the rules to be followed by a web server and a web browser when requesting and supplying information. HTTP is used for sending requests from a web client (a browser) to a web server and returning web content from the server back to the client.
- **HTTPS (Secure HTTP):** ensures that communications between a host and client are secure by ensuring that all communication between them is encrypted.

- **SMTP (Simple Mail Transfer Protocol):** the protocol for sending email messages from client to server and then from server to server until it reaches its destination.
- **POP (Post Office Protocol):** used by a client to retrieve emails from a mail server. All of the emails are downloaded when there is a connection between client and server.
- **IMAP (Internet Message Access Protocol):** unlike POP, the messages do not have to be downloaded. They can be read and stored on the message server. This is better for users with many different devices as they can be read from each one rather than being downloaded to just one.

(d) Any two from:

- It simplifies the overall model by dividing it into functional parts.
- Each layer is specialised to perform a particular function.
- The different layers can be combined in different ways, as required.
- One layer can be developed or changed without affecting the other layers.
- It makes it easier to identify and correct networking errors and problems.
- It provides a universal standard for hardware and software manufacturers to follow, so that they will be able to communicate with each other.

Worksheet 14.1

1 Answer to include:

- Users often pose the greatest threats to networks, either through their direct actions or by allowing criminals access to the networks illegally.
- Users can pose a threat to network security by using their own portable devices. These devices pose a threat because they can introduce malware to the network or remove data.
- If data is removed illegally, the company may be sued if this data is covered by the Data Protection Act.
- Network policies should cover the use of removable media such as USB flash drives, smartphones, CDs, DVDs, MP3 players and digital cameras.
- Network policies may state that only devices provided by the company can be used.
- Network policies should cover user authentication and the use of passwords.
- Network policies should ensure that users do not use passwords which are easy to remember and are based on personal details such as birth dates and the names of family members.
- All user passwords are at least eight characters long and contain numbers, letters (upper and lower case) and non-alphanumeric characters such as exclamation marks.
- Passwords should be changed regularly and old ones should never be reused.
- Users may be susceptible to social engineering (tricking people into divulging secret

information or doing things that they would not otherwise do).

- Social engineering techniques include:
  - o 'blagging', where a criminal uses a voice call or email to try to get a user to divulge information
  - o 'phishing', which is the use of fraudulent emails
  - o 'shouldering', where a user can be watched or filmed entering user names and passwords.

- Network policies should ensure that users do not divulge any details.

- Users should be given information about, and training on how to deal with, suspicious emails.

- Users can be a risk to the networks they use and they should be given security training, reinforced by strict network policies.

Worksheet 14.2

1 (a) **Denial of service** attacks flood a network or website with useless network communications, such as repeated login requests, which prevent legitimate users from gaining access to the network or website. They are caused by hackers taking over thousands of computers which they then use in the attack.

(b) Many websites use databases to store users' details such as names, addresses, credit card details, etc. SQL (structured query language) is used legitimately to analyse this data and carry out business activities. In **SQL injection**, criminals input specially created commands instead of a username or password. These commands gain access to the database so that the criminals have access to users' data.

(c) During **data interception** attacks, criminals use software called packet analysers or packet sniffers to intercept network packets. The packets are analysed and decoded. This allows criminals to steal sensitive data such as logins, passwords, credit card numbers and PINs.

Worksheet 14.3

1 (a) **Encryption** is the scrambling of data into a form that cannot be understood by unauthorised recipients. The encrypted data must be decrypted back to its original form.
A common method is the use of a 'public' and a 'private' key. The public key is freely available to anyone, but the private key is only known to the owner. Messages encrypted by a particular public key can only be decrypted with the corresponding private key.

(b) **Penetration testing** is the testing of a computer system, network or web application to find vulnerabilities that an attacker could exploit. The test then indicates how those vulnerabilities could be exploited to demonstrate the security risks. The main objective is to determine security weaknesses. It can also be used to test an organisation's security policy, the security awareness of the users, and the organisation's ability to identify and respond to security incidents.

There are two types:

- White-box penetration testing, which simulates hacking by 'insiders', people who have full knowledge of the network. It is also called 'full disclosure' testing as the testers are given details of items such as IP addresses, source code, network protocols and even login names and passwords.

- Black-box penetration testing, which is also called 'blind testing' because testers are given very little or no information. The testers must find their own way into the network without any knowledge or normal means of access. Black-box testing is more realistic to everyday penetration attacks.

(c) Firewalls are either software or hardware devices that protect against unauthorised access to a network, and are primarily used to prevent unauthorised access from the Internet.

They can be configured to prevent communications from entering the network and also to prevent programs and users from accessing the Internet from the network. A firewall, for example, can inspect the incoming packets and reject those that are not from a trusted IP address list or block communication to certain external IP addresses.

Worksheet 15.1

1 Answer to include:

Energy

- Manufacturing involves energy-intensive mining and processing of minerals.
- The use of devices involves the energy used by the devices themselves, but also by data centres. These data centres generate heat, so energy is needed to keep them cool.
- Much of the energy used comes from non-renewable sources such as gas and coal.
- Computer science is used in efficient energy production.
- Computer software is used to design, model and test efficient devices to produce electricity from wind, wave and solar power.
- Energy use can be reduced using smart technologies, such as light- sensitive switches that turn off lights when they are not needed.
- Efficient transport planning using computer modelling and analysis can reduce fuel use.

Sustainability

- Digital devices use many different chemical elements. Some of these are rare and will be in short supply as they are used up.
- It is difficult to recycle devices to reuse these elements.

Waste

- Electronic devices are difficult to recycle and are often disposed of in landfill sites as e-waste.
- Landfill sites take up areas of land that could be used for other purposes.
- Toxic substances such as lead, mercury and cobalt can get into the soil and the water supply from the landfill sites and so cause health problems.
- Often old computing devices are dumped illegally in third world countries where toxic fumes (caused by reprocessing and leakage) cause health problems and death.

Data analysis

- Computer science technology can be used to monitor environmental factors by transmitting and analysing data.
- This data can be shared by scientists around the world who can collaborate to find solutions to problems.
- Computers can be used to develop models to forecast environmental behaviour and identify options for action.

Worksheet 15.2

1 (a) Any three from:

- Data will be kept securely.
- Data must be accurate and up-to-date.
- Data can only be used for the purpose for which it was collected.
- Only data that is actually needed should be held.
- Data must not be held longer than it is needed for.
- Data will be used in accordance with the rights of the data subjects.

(b) Any three from:

- A right of access to a copy of the information contained in their personal data.
- A right to object to useage that is likely to cause or is causing damage or distress.
- A right to have inaccurate personal data rectified, blocked, erased or destroyed.
- A right to claim compensation for damages caused by a breach of the Confidentiality/ Protection Act.
- A right to prevent usage for direct marketing.

Worksheet 15.3

1 Answer to include the following:

- Greater broadband access to the Internet allows fast online communications from more locations and also when 'on the move'.
- Convergence technology allows many more devices (e.g. smartphones, tablets, games consoles) to access the Internet and to be used for communications.
- Mobile devices are faster and more capable of high speed Internet access.
- Protocols allow users to communicate using different devices, access the same conversations and sync data between them.
- Fast broadband access allows video and voice communication as well as emails and texts.
- Instant chat with video (as well are purely text) is available.
- The building of large server farms and cheaper storage enables the infrastructure behind large social networking websites.
- Communication with a group of friends is accessible and the uploading of images, videos, music etc.
- Software is increasingly easier to use and easily adopted by the younger generation.

Acknowledgements

The authors and publishers acknowledge the following sources of copyright material and are grateful for the permissions granted. While every effort has been made, it has not always been possible to identify the sources of all the material used, or to trace all copyright holders. If any omissions are brought to our notice, we will be happy to include the appropriate acknowledgements on reprinting.

Screenshots in Chapter 12 used with permission from Stephen Chen, Associate Professor, School of Information Technology.