

OCR GCSE
(9–1)

FOR THE
J277
SPECIFICATION

COMPUTER SCIENCE

SECOND EDITION

GEORGE ROUSE, LORNE PEARCEY, GAVIN CRADDOCK



An OCR endorsed textbook

 **DYNAMIC**
LEARNING

OCR
Oxford Cambridge and RSA

 **HODDER**
EDUCATION
LEARN MORE



Dynamic Learning is an online subscription solution that supports teachers and students with high quality content and unique tools. Dynamic Learning incorporates elements that all work together to give you the ultimate classroom and homework resource.

OCR only endorses the Student book and Student eTextbook. The other resources referenced here have not been submitted for endorsement.

Teaching and Learning titles include interactive resources, lesson-planning tools, self-marking tests and assessment. Teachers can:

- Use the Lesson Builder to plan and deliver outstanding lessons
- Share lessons and resources with students and colleagues
- Track student progress with Tests and Assessments

Teachers can also combine their own trusted resources alongside those from **OCR GCSE (9–1)**

Computer Science which has a whole host of informative and interactive resources including:

- A ready-made toolkit to deliver the GCSE specification, including a Scheme of Work, lesson plans for each chapter, outline presentations for all topics, and animated explanations for difficult key concepts
- Practical tasks and examples to develop problem-solving, computational thinking and OCR language-specified programming skills in chosen languages
- Opportunities to develop independent learning skills with activity worksheets and answers, and additional programming exercises to enhance understanding of certain topics
- Sample Tasks using Linux style commands (to provide practical experience with the command line interface) and SQL practical exercises (new to the specification)

OCR GCSE (9-1) Computer Science is available as a **Whiteboard eTextbook** which are online interactive versions of the printed textbook that enable teachers to:

- Display interactive pages to their class
- Add notes and highlight areas
- Add double-page spreads into lesson plans

Additionally the **Student eTextbook** of **OCR GCSE (9-1) Computer Science** is a downloadable version of the printed textbook that teachers can assign to students so they can:

- Download and view on any device or browser
- Add, edit and synchronise notes across two devices
- Access their personal copy on the move

To find out more and sign up for free trials visit: www.hoddereducation.co.uk/dynamiclearning

OCR GCSE
(9–1)

FOR THE
J277
SPECIFICATION

COMPUTER SCIENCE

SECOND EDITION

GEORGE ROUSE, LORNE PEARCEY, GAVIN CRADDOCK

The teaching content of this resource is endorsed by OCR for use with specification GCSE Computer Science (J277). In order to gain OCR endorsement, this resource has been reviewed against OCR's endorsement criteria.

This resource was designed using the most up-to-date information from the specification. Specifications are updated over time which means there may be contradictions between the resource and the specification, therefore please use the information on the latest specification and Sample Assessment Materials at all times when ensuring students are fully prepared for their assessments.

Any references to assessment and/or assessment preparation are the publisher's interpretation of the specification requirements and are not endorsed by OCR. OCR recommends that teachers consider using a range of teaching and learning resources in preparing learners for assessment, based on their own professional judgement for their students' needs. OCR has not paid for the production of this resource, nor does OCR receive any royalties from its sale. For more information about the endorsement process, please visit the OCR website, www.ocr.org.uk.

Although every effort has been made to ensure that website addresses are correct at time of going to press, Hodder Education cannot be held responsible for the content of any website mentioned in this book. It is sometimes possible to find a relocated web page by typing in the address of the home page for a website in the URL window of your browser.

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

Orders: please contact Bookpoint Ltd, 130 Park Drive, Milton Park, Abingdon, Oxon OX14 4SE. Telephone: +44 (0)1235 827827. Fax: +44 (0)1235 400401. Email education@bookpoint.co.uk. Lines are open from 9 a.m. to 5 p.m., Monday to Saturday, with a 24-hour message answering service. You can also order through our website: www.hoddereducation.co.uk

ISBN: 978 1 5104 8416 0

© George Rouse, Lorne Pearcey and Gavin Craddock 2020

First published in 2020 by

Hodder Education,

An Hachette UK Company

Carmelite House

50 Victoria Embankment

London EC4Y 0DZ

www.hoddereducation.co.uk

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

Year 2024 2023 2022 2021 2020

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

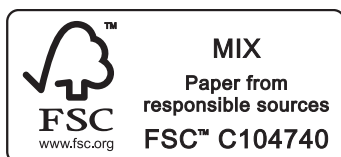
Cover illustration © Patrick P. Palej - stock.adobe.com

Illustrations by Integra Software Services Pvt. Ltd, Pondicherry, India

Typeset by Integra Software Services Pvt. Ltd, Pondicherry, India

Printed in Italy

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



CONTENTS

How to use this book	v
SECTION 1 COMPUTER SYSTEMS	1
1.1 System architecture	2
1.1.1 Architecture of the CPU	2
1.1.2 CPU performance	6
1.1.3 Embedded systems	7
1.2 Memory and storage	12
1.2.1 Primary storage – memory	12
1.2.2 Secondary storage	15
1.2.3 Units	19
1.2.4 Data storage	22
1.2.5 Compression	39
1.3 Computer networks, connections and protocols	52
1.3.1 Networks and topologies	52
1.3.2 Wired and wireless networks, protocols and layers	63
1.4 Network security	73
1.4.1 Threats to computer systems and networks	73
1.4.2 Identifying and preventing vulnerabilities	77
1.5 Systems software	83
1.5.1 Operating systems	83
1.5.2 Utility software	86
1.6 Ethical, legal, cultural and environmental impacts of digital technology	91
1.6.1 Ethical, legal, cultural and environmental impact	91
SECTION 2 COMPUTATIONAL THINKING, ALGORITHMS AND PROGRAMMING	105
2.1 Algorithms	106
2.1.1 Computational thinking	106
2.1.2 Designing, creating and refining algorithms	107
2.1.3 Sorting and searching algorithms	112
2.2 Programming fundamentals	125
2.2.1 Programming fundamentals	125
2.2.2 Data types	134
2.2.3 Additional programming techniques	136
2.3 Producing robust programs	152
2.3.1 Defensive design	152
2.3.2 Testing	157
2.4 Boolean logic	164
2.4.1 Boolean logic	164

2.5	Programming languages and integrated development environments	174
2.5.1	Languages	174
2.5.2	The integrated development environment (IDE)	177
	Appendix	182
	Glossary	190
	Knowledge check answers	195
	Index	205
	Acknowledgements	210

HOW TO USE THIS BOOK

To help you get the most out of it, this textbook uses the following learning features:

Important words

Highlighted in the text **in green**, these are terms that you will be expected to know and understand in your exams.

Important words

You will need to know and understand the following for the exam:

Central Processing Unit (CPU)

Fetch–Execute cycle

Arithmetic Logic Unit (ALU)

Tech terms

Jargon or technical definitions **in blue** that you may find useful.

Tech terms

Identifier The name of a variable or constant.

Reserved keyword A word in a particular programming language that has some special purpose and cannot be used for a variable or constant identifier.

Key point

An important idea or concept.

Key point

You should use multiples of 1000 in the exam. However, if you do use 1024 for calculations in the exam, you will not be penalised.

Worked examples

Used to illustrate an idea or a concept, these will guide you through the reasoning behind each step of a calculation or process.

Worked example



For example, if we have a file that is 2.5 MB, what is that in bytes?

$2.5 \text{ MB} = 2.5 \times 1000 \times 1000$ or 2 500 000 bytes

Beyond the spec

Information that you will not be expected to know or state in an exam but will aid understanding or add some useful context.

Beyond the spec

In a network, latency is a measure of how much time it takes for a packet of data to travel from one device to another. Latency is affected by factors such as the transmission media used.

Knowledge check

Quick check-ins that help you to recap and consolidate your understanding of the previous section.

Knowledge check



- 1 Describe the purpose of the CPU in a computer.
- 2 Describe the Fetch–Execute cycle.
- 3 What is held in the memory address register (MAR)?

Recap and review

A targeted summary of everything you have learned in the chapter. Use this to help you recap as you work through your course.

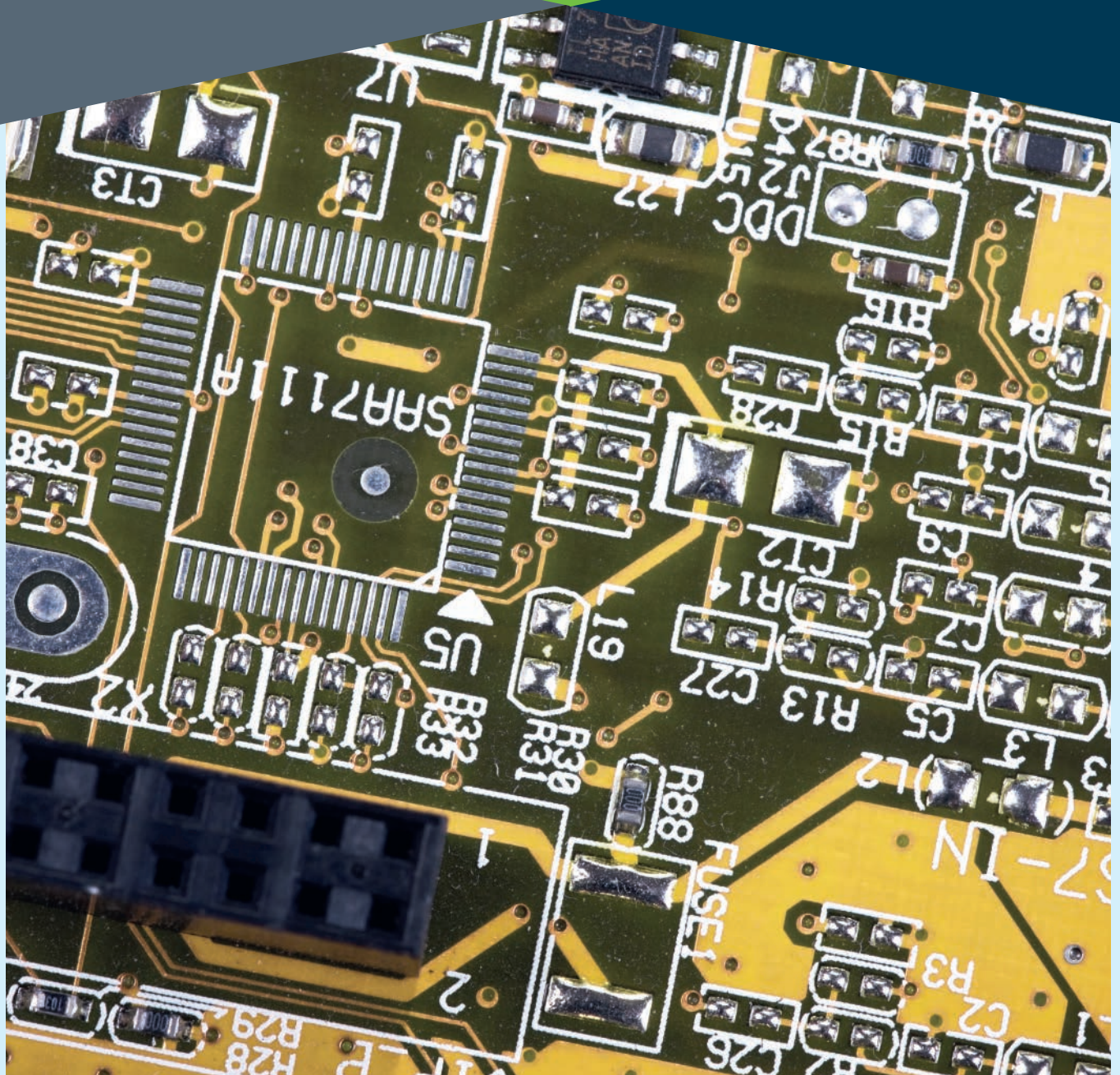
Extra resources

A free series of practice questions accompanies each section and is available online at: www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

SECTION 1

COMPUTER SYSTEMS



SYSTEM ARCHITECTURE

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.1.1 Architecture of the CPU

- The purpose of the CPU
- Common CPU components and their function
- Von Neumann architecture

1.1.2 CPU performance

- How common characteristics of CPUs affect their performance

1.1.3 Embedded systems

- The purpose and characteristics of embedded systems
- Examples of embedded systems

A computer system consists of hardware and software working together to process data. Hardware is the name for the physical components that make up the computer system. Software is the name for the programs that provide instructions for the computer, telling it what to do, and is covered in Chapter 1.5.

A computer system receives information as an input, processes and stores that information, and then outputs the results of that processing.

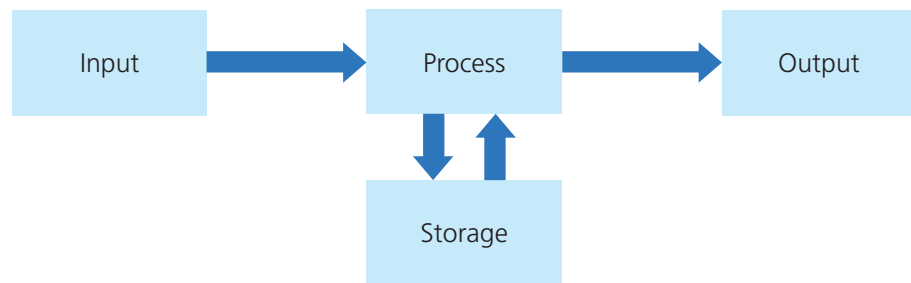


Figure 1.1.1 Input – process – output

Processing and storage is the job of the Central Processing Unit.

1.1.1 Architecture of the CPU

The purpose of the CPU

The **Central Processing Unit (CPU)** is made up of billions of transistors, which are like very small ‘on–off’ switches. The arrangement of transistors creates logic circuits that process data, carry out instructions and control the components of the computer.

The Fetch–Execute cycle

The processor continually:

- fetches instructions from memory
- decodes these instructions
- and then executes them.

This is called the **Fetch–Execute cycle**. There is more detail on this a little later in the chapter.



Figure 1.1.2 A CPU

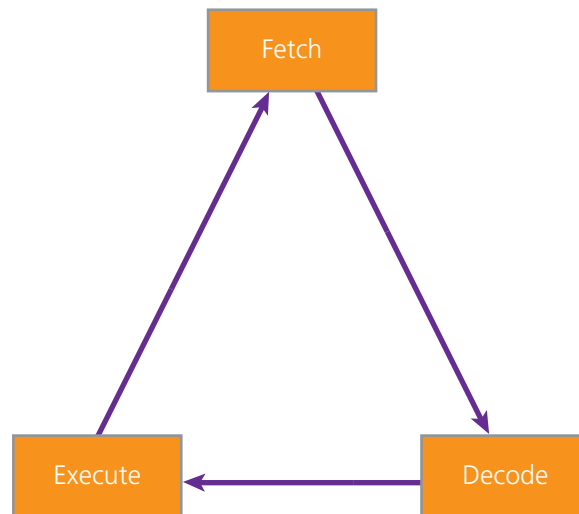


Figure 1.1.3 Fetch–Execute cycle

Common CPU components and their function

The CPU is made up from a number of components:

Arithmetic Logic Unit

The **Arithmetic Logic Unit (ALU)** is responsible for the following:

- arithmetic operations such as add, subtract, multiply and divide
- logical operations such as AND, OR and NOT, and the result of 'less than', 'greater than', 'equal to' comparisons
- binary shift operations (which are used for multiplication or division – see section 1.2.4).

The ALU carries out the calculations and logical decisions required by the program instructions that the CPU is processing.

Control unit

The purpose of the **control unit (CU)** is to co-ordinate the activity of the CPU.

It does this by:

- fetching then decoding instructions from memory
- sending out signals to control how data moves around the parts of the CPU and memory, in order to execute these instructions.

Cache memory

The purpose of **cache memory** is to provide temporary storage that the CPU can access very quickly.

Its role in the CPU is to store instructions and data that are used repeatedly or are likely to be required for the next CPU operation.

Registers

Registers are memory locations within the CPU that hold data temporarily and can be accessed very quickly.

Their role in the CPU is to accept, store and transfer data and instructions for immediate use by the CPU.

These registers are used during the Fetch–Execute cycle.

Von Neumann architecture

CPU architecture describes how the different components in the CPU are laid out and communicate with each other. The **Von Neumann architecture** describes a computer in which the data and instructions are stored in the same area of memory and are indistinguishable from each other. This means that the CPU has to decide what is an instruction and what is simply data.

Beyond the spec

The CPU uses a very low-level language called machine code (see Chapter 2.5). Machine code instructions are represented in binary. The value 1001 may be machine code for the arithmetic operation addition (ADD), but it is also the binary equivalent of the decimal number 9. Depending upon what the computer expects to find, it interprets the value 1001 as **either** an instruction to ADD **or** as the value 9.

Von Neumann architecture is the fundamental design concept behind all modern computer systems.

There are four important registers in a CPU with a Von Neumann architecture:

Program counter (PC): The program counter keeps track of the memory location (an address) for the next instruction. In many cases, the program counter is simply incremented to the next memory location at the Fetch stage of the Fetch–Execute cycle, to allow the program to be executed line by line. (Program instructions can, however, modify the value in the program counter to alter the flow of the program so that it continues from a new location.)

Memory data register (MDR): This register is used to store any data fetched from memory or any data that is to be transferred to and stored in memory.

Memory address register (MAR): This register stores the location in memory (known as an address) to be used by the MDR – that is, where the MDR needs to fetch data from or send data to.

Accumulator (ACC): This register either stores the results of any calculations made by the ALU, or stores the value of inputs and outputs to and from the CPU.

Figure 1.1.4 is a simplified diagram showing the layout of these components and how the CPU communicates with memory and input/output devices. Note that you do **not** need to know about buses for your exam.

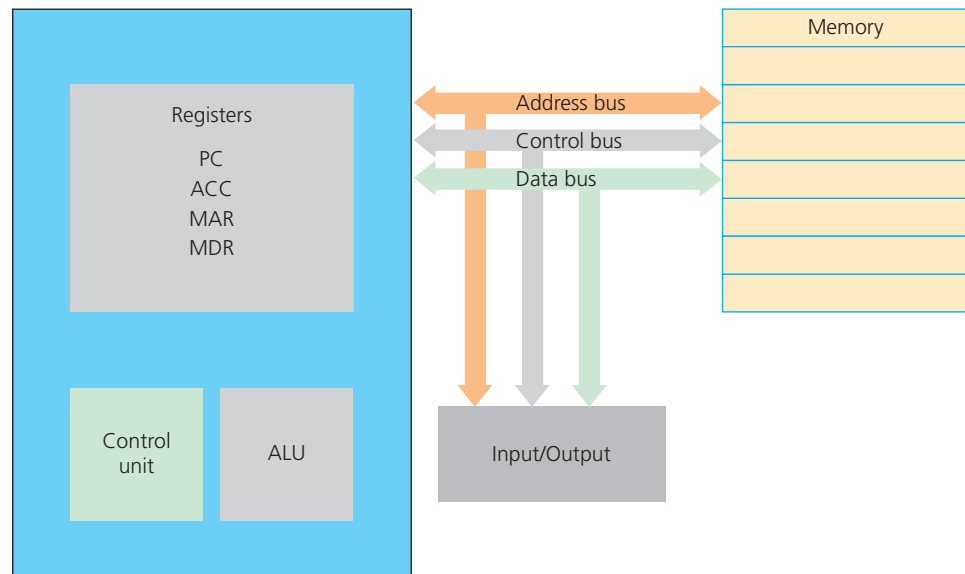


Figure 1.1.4 A CPU with Von Neumann architecture; the arrows represent the flow of data between components

Beyond the spec

To enable data and control signals to move around the CPU and memory, there are a number of buses. Buses are communication channels through which data can be moved.

There are three main buses inside the computer in relation to the CPU:

Data bus: This carries data between the CPU and memory.

Control bus: This carries control signals around the CPU and memory.

Address bus: This carries memory addresses for locations to be read from or written to.

Now that you know the names of the registers in the CPU, we can revisit the Fetch–Execute cycle so you can understand how it works in more detail.

Fetch

- 1 Each instruction in a computer program is stored in a particular location (or address) in memory. The address of the next instruction is copied from the **program counter** and placed in the **MAR**.
- 2 The MAR now contains a memory address. The **control unit** FETCHES the data that is stored at that address and copies it to the **MDR**.
- 3 The **program counter** is incremented to point to the next instruction to be processed in the program, ready for the next Fetch–Execute cycle.

Decode

- 4 The **MDR** now contains either data fetched from memory or an instruction. The **control unit** DECODES the instruction to see what to do.

Execute

- 5 The decoded instruction is EXECUTED. This might mean performing a calculation using the **ALU**, locating some data in memory, changing the **program counter** value – or something else.

Once the Execute part of the cycle is complete, the next Fetch–Execute cycle begins.

Knowledge check



- 1 Describe the purpose of the CPU in a computer.
- 2 Describe the Fetch–Execute cycle.
- 3 What is held in the memory address register (MAR)?
- 4 What is the purpose of the memory data register (MDR)?
- 5 What is the purpose of the program counter (PC) in the CPU?
- 6 State two arithmetic and two logical operations carried out by the Arithmetic Logic Unit (ALU).

1.1.2 CPU performance

How common characteristics of CPUs affect their performance

There are a number of factors that have an impact on the performance of a CPU.

Clock speed

The CPU is constantly fetching and executing instructions and the speed at which it does this is determined by an electronic clock. This clock uses a vibrating crystal that maintains a constant speed. Clock speeds are measured in hertz (Hz), which is the unit of frequency and means ‘number of times per second’ – see section 1.2.4. Typical modern computers work at speeds of up to 4 GHz (or 4 billion instructions per second). Each ‘tick’ of the clock represents one step in the Fetch–Execute cycle. The faster the **clock speed**, the more instructions that can be executed every second.

Size of cache memory

Cache memory is located between the main memory and the CPU. It is used to hold data that needs to be accessed very quickly. Accessing cache memory is much faster than accessing main memory (also known as random access memory (RAM) – see section 1.2.1).

The CPU control unit will first look in the cache for data or instructions, to see if they have already been copied from main memory. If they are not in the cache memory then the control unit will go to the main memory to locate them, and will then copy the data or instructions to cache and then to the CPU registers.

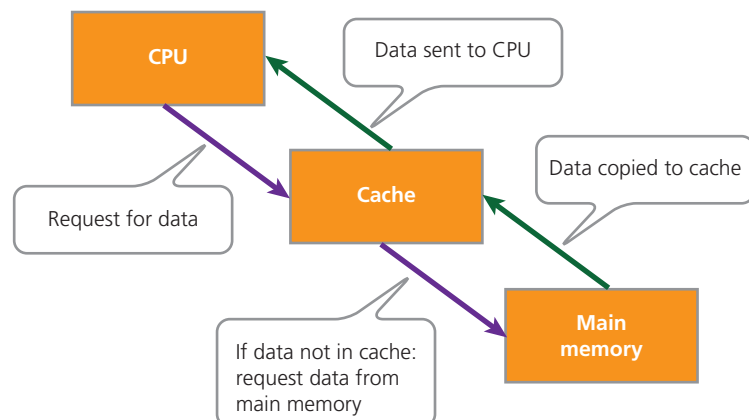


Figure 1.1.5 Cache memory is used to store data waiting to be processed

The more data that can be stored in cache memory rather than main memory, the faster and more efficient the process. Data that is likely to be required will be transferred to cache, ready to be used.

The larger the cache memory the more likely it is that the required data will already have been copied across from main memory. Cache memory is very expensive and while a mid-range laptop may have 8 GB of RAM, it is likely to have just a few KB of cache.

Number of processor cores

Another factor that can affect the performance of the CPU is the number of **processor cores**. Each core can fetch and execute instructions independently so a multiple core processor can handle several instructions at the same time. While these multiple cores can work on separate programs or parts of a program at the same time, this is only possible if the program has been written to take advantage of multiple cores. The task that the program is attempting must also be one that can be split up into subsections to take advantage of multiple cores.

Knowledge check



- 7 What is meant by a quad core processor?
- 8 What is meant by 2.3 GHz when describing a CPU?
- 9 Describe how cache memory is used by the CPU.
- 10 Describe three characteristics of a CPU that affect its performance.

1.1.3 Embedded systems

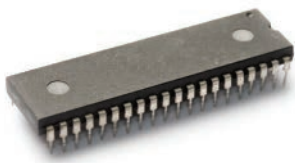


Figure 1.1.6 A microcontroller

An **embedded system** is a computer system that has a dedicated function as part of a larger device. The main components of a computer system are either manufactured onto a single chip (a microcontroller) or separate circuits for processing and memory are combined into a larger device.

Purpose and characteristics of embedded systems

When a computer device is required to perform a single or fixed range of tasks, it can be engineered to reduce its size and complexity in order to focus only on these tasks. Dedicated software will be programmed into the device to complete the necessary tasks and nothing else. The reduction of complexity of the hardware and the dedicated nature of the software will make the device more reliable and cost effective than using a general-purpose computer.

The embedded system will typically include some read-only memory (ROM) to store the dedicated program and some RAM (see section 1.2.1) to store user inputs and processor outputs. For example, in a washing machine, the ROM will store all of the data describing the separate washing cycles while the RAM will store the user's selected options (inputs) and the data used to display choices and progress of the washing cycle (outputs).

Embedded systems have the following characteristics:

- **low power** so they can operate effectively from a small power source such as in a digital camera
- **small in size** so they can fit into portable devices such as a personal fitness device

- **rugged** so that they can operate in harsh environments such as car engine management systems or in military applications
- **low cost**, making them suitable for use in mass-produced, low-cost devices such as microwave ovens
- **dedicated software** to complete a single task or limited range of tasks, such as in computer aided manufacture or control systems.

Examples of embedded systems

Embedded systems are found within common household devices such as washing machines, dishwashers, microwaves, set-top boxes, telephones, televisions, home security and control systems, and so on. Embedded systems are also widely used within larger and more complex systems such as car engine management, aeroplane avionics, computer-controlled manufacturing and military applications such as guidance systems.

Some embedded systems are connected to the internet via Wi-Fi to exchange data with third parties, for example water meters, energy smart meters and home security or heating monitoring systems.

Embedded systems are particularly useful for those with physical disabilities as they can make items more accessible. This can include voice control for gadgets in the home and systems that adapt motorised vehicles so they can be operated using limited physical movements.

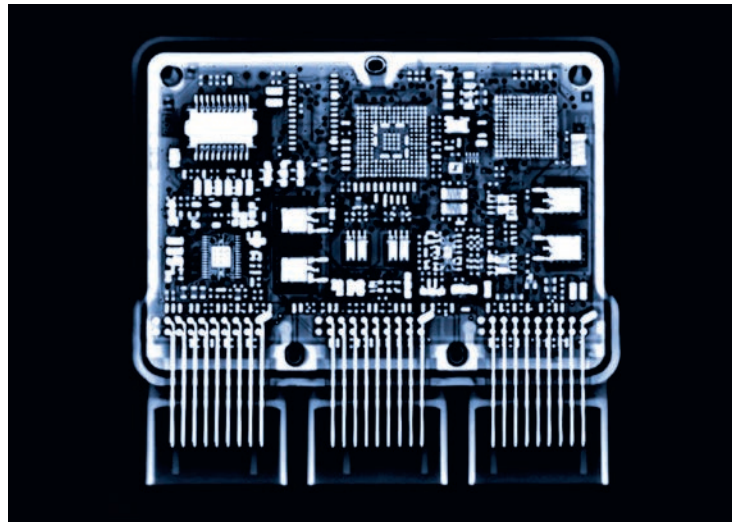


Figure 1.1.7 An x-ray image of an engine control unit in a motorcycle

Knowledge check



- 11** Explain why embedded systems have both RAM and ROM.
- 12** Identify one input and one output from the embedded system in a microwave oven.
- 13** Give two examples of systems that use embedded computer systems and explain why it is the most appropriate type of computer system to use in each case.

RECAP AND REVIEW

1.1 SYSTEM ARCHITECTURE

Important words

You will need to know and understand the following for the exam:

Central Processing Unit (CPU)

Fetch–Execute cycle

Arithmetic Logic Unit (ALU)

Control unit (CU)

Cache memory

Registers

Von Neumann architecture

Program counter (PC)

Memory data register (MDR)

Memory address register (MAR)

Accumulator (ACC)

Clock speed

Processor cores

Embedded system

1.1.1 Architecture of the CPU

Hardware is the term that describes the **physical components** of a computer.

The computer **inputs**, **processes** and **stores**, and **outputs** data.

The computer hardware works with the software to **process data**.

The hardware that processes the data is called the **CPU (Central Processing Unit)**.

Purpose of the CPU

The purpose of the CPU is to carry out a set of instructions that is contained in a computer program.

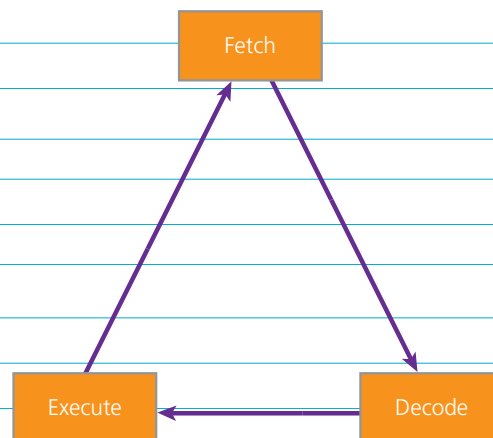
The CPU works at high speeds governed by the clock chip.

- The clock uses a vibrating crystal that maintains a constant speed.
- The clock chip operates at speeds of up to 4GHz (or 4 billion instructions per second).

Fetch–Execute cycle

The CPU continually fetches, decodes and executes instructions.

- Fetch – an instruction in the form of data is retrieved from main memory.
- Decode – the CPU decodes the instruction.
- Execute – the CPU performs an action according to the instruction.



Fetch–Execute cycle

Common CPU components and their function

Arithmetic Logic Unit (ALU)

Carries out all arithmetic calculations and logical decisions, for example add, subtract, binary shifts to multiply and divide, and comparisons such as equal to or greater than.

Control unit (CU)

Decodes instructions and sends signals to control how data moves around the parts of the CPU and memory to execute these instructions.

Cache

Cache memory sits between the processor and main memory (RAM).

- The CPU looks in the cache for required data.
- If it is not there, it requests it from RAM.
- The data is moved into cache before being accessed by the CPU.

Registers

Memory locations within the CPU that hold data. A register may hold an instruction, a storage address or any kind of data. The data in registers can be accessed very quickly – even more quickly than cache memory.

Von Neumann architecture

CPU architecture refers to the internal logical structure and organisation of the computer hardware.

In Von Neumann architecture, data and instructions are stored in the same memory.

- **Program counter (PC):** Stores the address of the next instruction to be processed.
- **Memory data register (MDR):** Stores data fetched from memory or to be sent to memory.
- **Memory address register (MAR):** Stores the address of the location in memory for data to be fetched from or sent to.
- **Accumulator (ACC):** Stores the results of any calculations carried out by the ALU.

1.1.2 CPU performance

How common characteristics of CPUs affect their performance

Clock speed: The CPU works at high speeds governed by the clock chip. The **faster the clock** the **more instructions that can be completed per second**.

- The clock uses a vibrating crystal that maintains a constant speed.
- The clock chip operates at speeds of up to 4GHz or 4 billion instructions per second.

Cache memory: Cache memory can be accessed very quickly by the CPU, so having more cache will provide the CPU with fast access to more data.

Number of cores: If a CPU has multiple cores it may be able to process more instructions simultaneously.

1.1.3 Embedded systems

Purpose and characteristics

- An **embedded system** is a computer system that has been designed for a **dedicated function as part of a bigger system**.
- Embedded systems are often **manufactured as a single chip**.
- The dedicated hardware and software make embedded systems **more robust and reliable** than general-purpose computers.

Characteristics of embedded systems include:

- Designed and engineered **to perform a limited set of tasks to reduce size and improve performance**.
- Programs are often **uploaded at the manufacturing stage**, directly to the device.
- There are often **very limited options to modify** these programs.
- **Low power consumption** to operate from a small power source.
- **Small in size** to fit in portable devices.
- **Rugged** so that they can operate in hostile environments.
- **Low cost** making them suitable for mass-produced products.

Examples of embedded systems

Embedded systems are found in most consumer products such as:

- washing machines
- microwave ovens
- home security systems
- home heating controls
- car engine management systems
- set-top boxes
- telephones
- televisions
- home security and control systems.

Extra resources

A free set of practice questions accompanies this section and is available online at: www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

MEMORY AND STORAGE

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.2.1 Primary storage—memory

- The need for primary storage
- The difference between RAM and ROM
- The purpose of ROM in a computer system
- The purpose of RAM in a computer system
- Virtual memory

1.2.2 Secondary storage

- The need for secondary storage
- Common types of storage
- The advantages and disadvantages of different storage devices

1.2.3 Units

- The units of data storage
- How data needs to be converted into a binary format to be processed by a computer
- Data capacity and calculation of data capacity requirements

1.2.4 Data storage

- Converting between denary, binary and hexadecimal
- How to add two 8-bit binary integers together and explain any overflow errors
- Binary shifts
- The use of binary codes to represent characters
- Character sets, the relationship between the number of bits per character and the number of characters in a character set
- How an image is represented as pixels and in binary
- Image metadata
- The effect of colour depth and resolution on image quality and size
- How sound can be sampled and stored in digital form
- The effect of sample rate, duration and bit depth on sound quality and file size

1.2.5 Compression

- The need for compression
- Types of compression

For a computer system to be useful, it needs storage: storage for data and programs that are currently in use and storage for data and programs that can be accessed when required.

1.2.1 Primary storage – memory

The need for primary storage

A computer system needs primary storage for any data that it needs to access quickly. This includes the start-up instructions, the operating system, programs that are running and any associated data. There are two main types of primary storage: RAM and ROM.

Random access memory (RAM)

RAM is part of the main memory in a computer system, and a typical laptop will have around 8 GB of RAM available. RAM is required to hold the operating system, applications that are running and any associated data while the computer is on and in use.

When a program is loaded, it is copied from secondary storage, such as a hard disk drive (HDD), into RAM. (For an explanation of secondary storage see section 1.2.2.) Any data associated with the program will also be stored in RAM so that the CPU can access both the data and instructions.

Data is transferred into RAM from secondary storage for use by the CPU. With more RAM available more data and applications can be stored in it. Because RAM has fast data access times this leads to better performance of the system. In practice a system with more RAM can have more programs open at the same time without any noticeable decrease in performance.

As noted in the previous chapter, once data and instructions are in RAM they are then transferred to cache memory, in order to further improve data access speeds for the CPU. Figure 1.2.1 illustrates how access to data and programs is improved by using RAM and cache together.

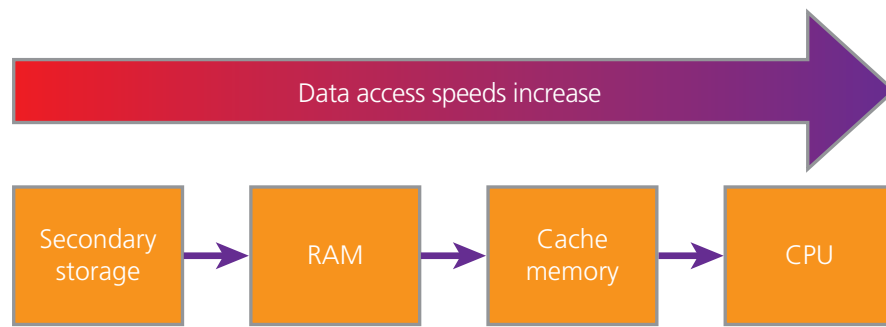


Figure 1.2.1 Data transfer speeds for secondary storage, main memory, cache and CPU

RAM is **volatile**, meaning it needs electrical power to operate. Any data stored in RAM is lost when the power is turned off.

RAM can be read from or written to by the computer. This means RAM is **read and write**.

Read-only memory (ROM)

Typically, in a computer system, **ROM** stores the instructions and data needed to get the system up and running and ready to load the operating system from secondary storage. This special program stored on ROM is called the Bootstrap Loader and we say the process ‘boots’ the computer – this means that it starts it from scratch.

Computers would not be so useful if they had to be switched on all the time. ROM is **non-volatile** memory, which means it does not require power to maintain its contents.

ROM is **read-only**. This means that the data stored in ROM is fixed and cannot be overwritten once it is created. This data is written to ROM either at the manufacturing stage or through a special process later on.

The difference between RAM and ROM

Table 1.2.1 A comparison of RAM and ROM

RAM	ROM
Volatile and needs power to maintain the content	Non-volatile and does not require power to maintain the content
Read and write – data can be read from and written to RAM by the computer	Read-only – the computer cannot overwrite its content
Holds the operating system and any programs and data currently in use by the computer	Holds the data and instructions required to start up (boot) the computer

Knowledge check



- 1 State two differences between RAM and ROM.
- 2 What is held in RAM while the computer is working?
- 3 What is held in ROM on the computer?

Virtual memory

It is not always possible to store all the data and instructions we need in RAM. If a computer is running complex programs processing large amounts of data – for example large images or video files – or lots of programs are open simultaneously, there may be insufficient RAM to hold them all. In this case the computer can allocate a section of secondary storage to temporarily act like RAM. It does this by selecting data in RAM that is not currently required by the CPU and moving it temporarily into secondary storage. Once that data is required by the CPU, it is moved back from secondary storage into RAM.

The area of secondary storage used to temporarily store data from RAM is called **virtual memory**.

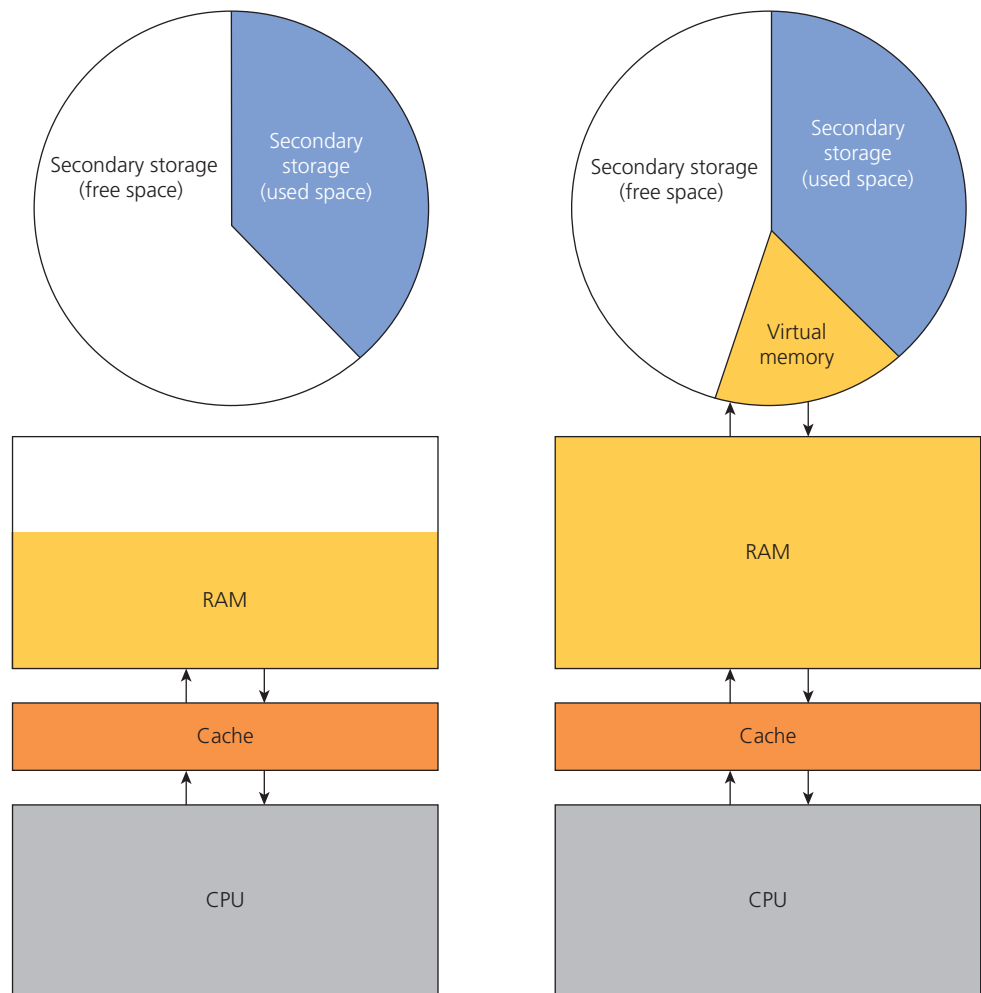


Figure 1.2.2 Virtual memory

In the left-hand diagram there is only one program running which only takes up some of the available RAM, and so the computer does not need to use virtual memory. In the right-hand diagram, more data and programs are open and the computer has run out of RAM. It allocates a section of secondary storage to act like RAM. This is represented by the orange section of secondary storage and is called virtual memory.

Using virtual memory will affect the performance of the computer system because there is a delay when transferring data from secondary storage back into RAM.

A computer with more RAM will need less virtual memory, reducing the number of data transfers between RAM and secondary storage and therefore delivering improved performance.

Knowledge check



- 4 What is virtual memory and why is it needed?
- 5 How might installing more RAM affect the use of virtual memory and how might this affect the performance of the computer?

1.2.2 Secondary storage

The need for secondary storage

Computer systems would be of little value if we lost all of our data and programs every time we switched them off. We need to store the operating system, data, images, programs, documents and various other files on our computers so that they are available the next time we switch on the computer. This kind of data requires a lot of space so we need a low-cost, high-capacity, read-and-write, non-volatile storage medium. This is known as **secondary storage**. Secondary storage needs to keep data safe and must be robust and reliable.

A number of different secondary storage media can be used for a computer system and the choice is based on several factors:

- Capacity: How much data does it need to hold?
- Speed: How quickly can the data be accessed?
 - Typically access times to secondary storage are very slow compared to primary storage (main memory).
- Portability: Does the data stored on the device need to be moved or transported?
 - If so the size, shape and weight of the medium is important.
- Durability: How robust is the medium?
 - Will it be damaged when moved around?
 - Will it be used in a hostile environment subject to shocks or extreme conditions?
- Reliability: Does it need to be used over and over again without failing?
- Cost: What is the cost per GB of data stored?
 - This is also an important factor where the media are being used to distribute data and will not be reused.
 - CDs for example only cost pennies but the cost per GB of storing data is higher than that for a magnetic hard disk drive.

Knowledge check



- 6 Why do we need secondary storage?
- 7 What is stored on secondary storage in a computer system?

Magnetic storage

Magnetic storage uses the principle of magnetism to store data.

Hard disk drives (HDDs) are magnetic and are the most common type of secondary storage.

Beyond the spec

HDDs are made of a stack of rigid disks (called platters) on a single spindle that rotates. Each platter is coated in a magnetic material, which is effectively made up of billions of separate tiny magnets that can either point 'north' or 'south'. Each bit of data (see section 1.2.3) is represented by these tiny magnets – north for '1' and south for '0'. A set of 'heads' moves across the platters, reading or writing data by sensing or changing the north/south alignment of the magnets.



Figure 1.2.3 A hard disk drive showing the platters and heads

The magnetic hard disk is a reliable and cost-effective storage solution, providing high capacity at low cost. This makes them an ideal choice for the typical laptop or desktop computer. Internal and external hard disk drive capacities are currently measured in terabytes (a million megabytes – see section 1.2.3). Large hard disk drives are, however, less portable than solid-state drives or optical disks and are subject to damage if dropped or brought near to strong electric or magnetic fields.

Several drives can be combined in larger commercial systems to provide a significant amount of storage at a reasonable cost. At the other end of the scale, there are small portable hard disk drives that can easily be moved between computers.



Figure 1.2.4 An external hard drive connected to a laptop

Tech term

Magnetic tape An old storage medium that lasts a long time and is very cheap but has very slow read/write times and thus is only really used for archiving.

Another form of magnetic storage is **magnetic tape**, which is still used as a cheap way to archive large amounts of data.

Tech terms

Flash memory A method of storing data that is based on electronics.

Latency A delay before data can be transferred.

Fragmented Data stored in different physical locations across the disk.

Solid-state storage

Solid-state storage uses a technology called **flash memory** that has very fast data access times compared to HDDs, largely because there are no moving parts. Flash memory is, however, relatively expensive compared to hard disk drives and typically has lower capacity. Solid-state storage is widely used for portable devices such as cameras (e.g. memory cards), and comes in a range of physical sizes and capacities to suit a wide range of applications. These portable devices, like USB pen drives, can be used to easily backup or transfer data between devices.

Solid-state flash memory is also used as the basis for **solid-state drives** (SSDs). SSDs have begun to replace magnetic hard disk drives (HDDs) because they have many advantages over them:

- In order to read data, magnetic HDDs have to line up the correct portion of the disk with the position of the read/write head. This means the magnetic disk has to rotate to the correct position and the head has to move across the disk. This in turn means there is a delay before data can be read or written. This delay is called **latency**. SSDs have lower latency times because there are no moving parts and access to the data does not require a platter to rotate or the read/write head to move. This improves access to the data and the performance of the device.
- The lack of moving parts means that SSDs have much lower power requirements and do not generate any heat or noise.
- HDDs can suffer from data being **fragmented** over the surface of the platters, producing very slow access speeds. This is not the case with SSDs.
- SSDs are significantly lighter, smaller and thinner than HDDs, making them particularly suitable for small, thin portable devices such as tablet computers or other portable devices.
- Since there are no moving parts, SSDs are not susceptible to problems caused by sudden movements, making them ideal in hostile environments or in portable devices.

Given the expense of SSDs, they are often combined with a magnetic disk drive to form a hybrid system. Frequently accessed data, such as the operating system, is stored on the SSD, while large, less frequently required data is stored on the magnetic disk. This provides the speed advantage of the SSD with the capacity advantage of the HDD at a reasonable cost compared to high-capacity SSDs.



Figure 1.2.5 Various solid-state devices

Beyond the spec

The surface of each disk is covered in billions of small indentations, known as **pits** and **lands**. When laser light is shone onto the surface of the disk it reflects off the pits and lands, with each pit or land representing a series of zeros, and the transition between pits and lands representing 1.

Tech terms

Pit A tiny indentation on a CD or DVD.

Land Areas on a CD or DVD where there are no pits.

Optical storage

Data can also be stored by using the properties of light. Typical **optical storage** media include CDs, DVDs and Blu-Ray disks. These are optical devices because they are written to and read from using laser light.

Some optical storage media are read-only but others can be written to, by creating pits on the surface of the disk using laser light.

For example, there are three main types of CD/DVD:

- **CD/DVD-ROM:** Read-only device with the data stored at the manufacturing stage.
- **CD/DVD-R:** Recordable media that can be written to once using a suitable drive.
- **CD/DVD-RW:** Rewriteable media that can be written to several times.

Typically, a CD will hold around 700 MB of data and cost pennies. They are used to distribute data and programs or make semi-permanent copies of data.

The DVD is very similar to the CD, but it has a larger capacity of 4.7–8.5 GB. This means that DVDs can store more data than CDs, such as standard resolution movies. A DVD has a faster access time than a CD and costs a little more, but is still only pennies.

Blu-Ray is similar but with significantly larger capacity (25–50 GB) and much faster access speeds. Blu-Ray disks can be used to store large amounts of data and the much higher access speed makes them particularly suitable for high-resolution movies and console games. They are slightly more expensive than DVDs but still reasonably inexpensive.

Table 1.2.2 Comparison of optical devices

Type	CD	DVD	Blu-Ray
Typical cost	18p	60p–80p	£1.80–£3.00
Capacity	700 MB	4.7 GB single layer 8.5 GB dual layer	25 GB single layer 50 GB dual layer

Choosing suitable secondary storage media

When choosing a suitable storage medium all of the following factors need to be considered.

- Capacity: How much data does it need to store?
- Speed: How quickly does the data need to be accessed?
- Portability: Does the device or medium need to be transported?
 - If so the size and weight are important.
- Durability: Will the device or medium be used in a hostile environment?
 - If so, the medium must be resistant to external shocks or extreme conditions.
- Reliability: Does it need to be used repeatedly without failing?
- Cost: What is the cost per unit of storage related to the value of the data?

Table 1.2.3 Capacity and cost of storage media

Media	Capacity	Typical cost	Cost per GB
Magnetic hard disk	Up to 15 TB	A 2 TB HDD costs around £60 and a 10 TB drive about £300	3p
SSD	250 MB up to 2 TB	£30 up to £300 for a 2 TB drive	15p
DVD	8.5 GB	80p	9p
Blu-Ray	50 GB	£3.00	6p
CD	700 MB	18p	23p

The cost per GB varies. All optical media are able to store reasonably small files for pennies per MB. For larger files, SSDs cost significantly more than HDDs for the same capacity. Larger storage requirements can be met more cost-effectively by a magnetic hard disk drive.

Speed

An SSD can transfer data at around 200–550 MB per second. A magnetic HDD transfers data at a much lower rate of 50–120 MB per second. An SSD will provide faster access to data. (For reference, data transfer times for RAM are around 12–20 GB/s.)

Table 1.2.4 Access speeds of storage media

Storage type	Data transfer rates (typical)
SSD	200–550 MB/s
Magnetic hard disk	50–120 MB/s
Blu-Ray disk	72 MB/s
DVD	1.32 MB/s
CD	0.146 MB/s

Table 1.2.5 Portability, durability and reliability

Media	Portability	Durability	Reliability
SSD	Small, with low power requirements Very portable	With no moving parts they are not subject to damage from sudden shocks.	The medium is reliable and will hold data safely for a very long time before failure.
Magnetic hard disk	With moving parts, higher power requirements than SSD. Available as external drives powered from a USB	Subject to damage from being dropped or from exposure to magnetic fields.	Ideal for medium term storage with a reliable life of 5–7 years. Motors and heads are subject to failure over time or from excessive use or mishandling.
CD DVD Blu-Ray	Light and small Very portable – can even be sent through the post	Reasonably robust and resistant to shocks Easily damaged by mishandling and scratches	CDs and DVDs will start to fail after 10 years; Blu-Ray will fail after 20 years.

Knowledge check



- 8 What are the advantages of solid-state storage over magnetic hard disks?
- 9 Why might a user choose magnetic hard disks over solid-state storage?
- 10 What is the most suitable medium for distributing high-definition video films?
- 11 Identify and describe three factors to consider when choosing secondary storage media and devices.

1.2.3 Units

The units of data storage

A computer uses electronic circuits etched onto computer chips to store data and instructions. These circuits contain electronic switches made from tiny transistors. Each switch can be in one of two states: **on** or **off**. The two states are represented by the numbers **1** or **0**. A computer uses combinations of these 1s and 0s to represent data and instructions.

Beyond the spec

In some sources, you will find people referring to one kilobyte as 1024 bytes, one megabyte as 1024 kilobytes and so on. However, there are different prefixes for this – for example, 1024 bytes is called 1 kibibyte. (1024 might seem like a strange number but it is used because it is a ‘neater’ number when written in binary.)

Key point

You should use multiples of 1000 in the exam. However, if you do use 1024 for calculations in the exam, you will not be penalised.

There is a number system that only uses the two values 1 and 0. It is called the **binary** number system and it is used to describe the on/off status of all the switches in a computer. One **binary** digit is called a **bit**, and the symbol for this is **b**. Computers often group 8 bits together as one unit of data. These 8 bits together are called a **byte**, with the symbol **B**.

4 bits grouped together are called a **nibble**, which has no symbol.

In standard scientific notation the prefix ‘kilo’ means 1000 – for instance 1 kilometre is 1000 metres. We use kilo, and a whole set of other units, based on this scientific notation.

8 bits (b)	1 byte (B)
1000 B	1 kilobyte (KB)
1000 KB	1 megabyte (MB)
1000 MB	1 gigabyte (GB)
1000 GB	1 terabyte (TB)
1000 TB	1 petabyte (PB)

Worked example

For example, if we have a file that is 2.5 MB, what is that in bytes?
2.5 MB = 2.5 × 1000 × 1000 or 2 500 000 bytes

Knowledge check

- 12 How many kilobytes are there in 4.5 gigabytes?
- 13 How many gigabytes are there in 32 terabytes?

Data capacity calculations

As discussed, a computer works in binary because the billions of tiny switches that make up its circuits can only be on or off, which is represented by 1s and 0s. This means that all data needs to be converted into binary so that a computer can work with it. In section 1.2.4 we will describe how that conversion is done for sound, but for now we will look at typical file sizes for a range of file types, measured in bytes.

The choice of media to use is often governed by the number and type of file to be stored. The format used to store the file can also make a significant difference. For example, a document with just 250 characters saved as plain text, rich text or a Word document takes from 251 bytes up to 12 KB depending on how it is saved.




Name	^	Date Modified	Size	Kind
 text		31/07/2019	12 KB	Microsof...t (.docx)
 text		31/07/2019	619 bytes	RTF Document
 text		31/07/2019	251 bytes	Plain Text

Figure 1.2.6 A 250-character file saved in different formats

We can estimate how much space we require based on typical or estimated file sizes. For example:

File type	Approximate size
1-page word-processed document with no images	100 KB
Postcard size photograph	6 MB
3-minute MP3 music track	6 MB
3-minute music track on a commercial CD	50 MB
1-minute MPEG video	50 MB
DVD film	4 GB
High-definition film	8–15 GB
Blu-Ray film	20–25 GB
4 k high-resolution film	100 GB or more

Tech term

Metadata Data associated with a file that are used to reconstruct the original document, image or sound.

We are only estimating the sizes and need to make some allowance for small variations and file **metadata**. In our example below, we will allow 10% extra when calculating the size of the medium required to store data, however it is not a requirement for you to do so in your exam.

Worked example



If we need to store 5 minutes of MPEG video, 30 minutes of MP3 music and 5 postcard sized photographs, we can estimate the size as:

5 minutes of video @ 50 MB per minute
= 5 minutes x 50 MB per minute
= 250 MB

30 minutes of MP3 music @ 50 MB per 3 minutes

First, we have to work out how many 3-minute MP3s make up 30 minutes:

= 30 minutes / 3 minutes = 10

The total storage required is:

= 10 x 50 MB
= 500 MB.

5 postcard sized images @ 6 MB each

= 5 x 6 MB
= 30 MB.

Total space

= 250 + 500 + 30
= 780 MB.

10% extra for metadata

= 0.1 x 780 MB
= 78 MB

TOTAL SPACE REQUIRED

= 780 MB + 78 MB
= 858 MB

858 MB is too big to fit on a standard CD but will fit onto a standard DVD or perhaps a small solid-state device.

Knowledge check



- 14 How much space is needed to store 200 postcard sized images and 60 minutes of MP3 music?
- 15 How much space is needed to store 40 pages of word-processed documents and 15 postcard sized images? What medium would be most appropriate to distribute these files?

1.2.4 Data storage

NUMBERS

We have already established that computers use switches to store and process data and that these switches have just two states, either 1 (on) or 0 (off). 1 and 0 are the two numbers in the **binary** number system. This means that, in a computer, all data (numbers, characters, sounds and images) are represented in binary. Before we look at how this is done, we need to understand the concept of binary numbers.

In the **denary** (or decimal) system we are used to using 10 symbols or values: 0,1,2,3,4,5,6,7,8,9. We use these symbols to write numbers.

For example: the number 527 is made up of

5 lots of 100,

2 lots of 10,

7 lots of 1.

Written into a table this can be seen more clearly:

100s	10s	1s
5	2	7

The column values are ten times larger than the previous value as we move from right to left; that is, the 'hundreds' are ten times bigger than 'tens' and 'tens' are ten times bigger than 'ones'.

In binary we just have the two symbols or values, 0 and 1. This means that in the binary number system each column heading is twice as big as the previous one as we move from right to left:

128	64	32	16	8	4	2	1

The column headings in binary, from right to left, are 'ones', 'twos', 'fours', 'eights' and so on.

The leftmost digit in a binary number is called the **most significant bit (MSB)** and the rightmost digit the **least significant bit (LSB)**. In an 8-bit number, the MSB has decimal value 128 and the LSB has decimal value 1.

Key point

You are expected to be able to work with binary numbers with up to eight digits, but also with values containing fewer than eight digits.

How to convert positive denary whole numbers into 8-bit binary numbers and vice versa

Using the table for binary, we can use the same approach as for denary numbers to calculate the denary equivalent of a binary number.

Worked example



What is the binary number 100111 in denary?

First, copy the binary number 100111 into the table, always ensuring that you fill the table from the right. (There should always be a value (0 or 1) in the '1' column.)

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

We have:

1 lot of 32, no lots of 16, no lots of 8, 1 lot of 4, 1 lot of 2 and 1 lot of 1

This is equal to:

$$32 + 4 + 2 + 1 = 39$$

100111 in binary is 39 in denary.

To convert denary numbers into binary, we use the binary column heading values from the table. We decide whether each column heading value, starting at the left-hand side, is smaller than or equal to our decimal number. We record 0 if it is not smaller and then compare it with the next column value.

If it is smaller than or equal to, we record 1 in the table, subtract the number and work out the remainder. We take that remainder and then check if it is smaller than the next column value, and repeat the process above. We continue this process until we are left with the right-hand '1' column.

A worked example will make this clearer.

Worked example



Convert the denary number 142 into binary.

Is 128 smaller than 142? YES it is, so we record 1 in the first column from the left-hand side.

128	64	32	16	8	4	2	1
1							

We subtract 128 from 142, to leave a remainder of 14.

We now check 14 against the next column value, 64. 64 is not smaller than 14, so we record 0 in the second column.

128	64	32	16	8	4	2	1
1	0						

We continue to check 14. We can see that 32 is not smaller than 14, and neither is 16, so we can record 0s in the next two columns.

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

We find that 8 is smaller than 14, leaving a remainder of 6, and so we put a 1 in the column under 8.

We now check 6 against the next column value.

4 is smaller than 6, leaving a remainder of 2. We put a 1 in the column under 4 and now check 2 against the next column value.

2 is equal to 2, leaving a remainder of 0, so we put a 1 in the column under 2.

We have been left to check the number 0 against the last column. 1 is not less than or equal to 0, so the final entry is a 0 under the 1 column.

The final table looks like this:

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

We can check our answer by converting the binary number 10001110 back into denary. We find that:

$$10001110 = 128 + 8 + 4 + 2 = 142$$

We have confirmed that 10001110 is indeed the denary number 142.

Worked example



Convert 83 into binary.

We follow the same process as before. 128 is not smaller than 83 so the 128 column entry is 0.

64 is less than 83, so we enter a 1 in the column under 64. We also note that $83 - 64 = 19$ and we will now check 19 against the next columns.

The next column that is less than 19 is 16, so we enter 0 in the 32 column and 1 in the 16 column.

We then note that $19 - 16 = 3$ and we will now check 3 against the next columns.

The next column that is less than 3 is the 2 column. So we enter a 1 in the 2 column and note that $3 - 2 = 1$. We check this 1 against the 1 column, and 1 is of course equal to 1. So we enter a 1 in the final column.

So, the binary number is:

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

We can check our answer by converting 01010011 back into denary. We find that:

$$01010011 = 64 + 16 + 2 + 1 = 83$$

We have confirmed that 01010011 is indeed the denary number 83.

Note that whilst we only really need 7 bits to represent this binary value, we need to keep the leading 0 for the computer since it uses a fixed number of bits. However, when we write down these values on paper we can leave the column for 128 blank and write down the answer as 1010011.

Key point

You need to show your working in the examination so using a table and showing the key subtractions will demonstrate clearly how you arrived at the answer.

Knowledge check



16 Convert the following binary numbers to denary.

- (a) 1001
- (b) 11101
- (c) 110001
- (d) 10001100
- (e) 11011011
- (f) 11111100

17 Convert the following denary numbers to binary.

- (a) 20
- (b) 46
- (c) 75
- (d) 98
- (e) 147
- (f) 213

How to add two 8-bit binary integers and explain overflow errors which may occur

Adding binary numbers is very similar to the way we add denary numbers. Let's look at the way we add two denary numbers.

Worked example



Adding 357 and 264

$$\begin{array}{r} 3 \quad 5 \quad 7 \\ + 2 \quad 6 \quad 4 \\ \hline \end{array}$$

$$7 + 4 = 11$$

So we write down 1 and carry 1 to be added in the next column. The carry is written in the next column under the line.

$$\begin{array}{r} 3 \quad 5 \quad 7 \\ + 2 \quad 6 \quad 4 \\ \hline \quad 1 \quad 1 \end{array}$$

$$5 + 6 + 1 = 12$$

So we write down 2 and carry 1 to be added in the next column.

$$\begin{array}{r} 3 \quad 5 \quad 7 \\ + 2 \quad 6 \quad 4 \\ \hline \quad 2 \quad 1 \quad 1 \end{array}$$

$$3 + 2 + 1 = 6$$

So we write down 6.

$$\begin{array}{r} 3 \quad 5 \quad 7 \\ + 2 \quad 6 \quad 4 \\ \hline 6 \quad 2 \quad 1 \end{array}$$

So, the key thing to remember is that as soon as a column adds up to a number bigger than 9, we have to carry.

When adding binary numbers we follow the same process. The difference is that as soon as a column adds up to a number bigger than 1, we have to carry.

We also need to note that:

$1 + 1 = 10$ in binary (that is $1 + 1 = 2$ which is written as 10 in binary)

$1 + 1 + 1 = 11$ in binary (that is $1 + 1 + 1 = 3$ which is written as 11 in binary)

Let's look at an example adding 1101 and 1111.

Worked example



$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \\ + 1 \quad 1 \quad 1 \quad 1 \\ \hline \end{array}$$

$1 + 1$ is 10 in binary so we write down 0 and carry 1 to be added in the next column. This carry is written in the next column under the line.

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \\ + 1 \quad 1 \quad 1 \quad 1 \\ \hline \quad 1 \quad 0 \quad 1 \end{array}$$

$0 + 1 + 1$ is 10 in binary so we write down 0 and carry 1 to be added in the next column.

$$\begin{array}{r}
 1 \quad 1 \quad 0 \quad 1 \\
 + 1 \quad 1 \quad 1 \quad 1 \\
 \hline
 \quad \quad 0 \quad 0 \\
 \quad 1 \quad 1
 \end{array}$$

1 + 1 + 1 is 11 in binary so we write down 1 and carry 1 to be added in the next column.

$$\begin{array}{r}
 1 \quad 1 \quad 0 \quad 1 \\
 + 1 \quad 1 \quad 1 \quad 1 \\
 \hline
 \quad 1 \quad 0 \quad 0 \\
 \quad 1 \quad 1 \quad 1
 \end{array}$$

1 + 1 + 1 is 11 in binary so we write down 1 and carry 1 to be added in the next column.

$$\begin{array}{r}
 \quad 1 \quad 1 \quad 0 \quad 1 \\
 + \quad 1 \quad 1 \quad 1 \quad 1 \\
 \hline
 1 \quad 1 \quad 1 \quad 0 \quad 0 \\
 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

The carried 1 is the final digit in the sum giving the answer 11100.

In the worked example we added together two 4-bit numbers and ended up with a 5-bit number. This is a problem if we can only store 4-bit numbers.

Overflow errors

If we are working with 8-bit numbers and all we can store is 8 bits, the following problem might occur.

For example, if we add these two 8-bit numbers:

$$\begin{array}{r}
 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \\
 + 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \\
 \hline
 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \\
 1 \quad \quad \quad 1 \quad 1
 \end{array}$$

The carried 1 in the left-hand column would be the ninth digit in our answer, but we are limited to 8 bits. This carry is lost, which means we have an **overflow error**.

In this example, we tried to add 11000110 (198 in denary) and 11100011 (227 in denary).

198 + 227 = 425 in denary. The answer produces an overflow error because we needed a 9th bit. Without a 9th bit that information is lost, giving an incorrect answer of 10101001 in binary, which is 169 in denary.

The largest value we can store in 8 bits is 11111111, or 255 in denary. So the reason for the overflow error is that 425 is too large a number to store in 8 bits.

Knowledge check

Complete the following additions in binary

18 1100 + 110

19 1011 + 1001

20 110011 + 100111

21 110110 + 111011

22 10100111 + 10110

23 11001100 + 11111

24 Explain what happens when you add 11100011 and 10001110.



Binary shifts

Moving the digits in a binary number left or right is called a **binary shift**.

Each time the value is shifted one place to the left, the number is multiplied by 2.

For example, 40 in binary is 101000:

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

If we shift the digits one place to the left, and add 0 to the right-hand 1 column, we get:

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

The original number that has been moved one place to the left has been highlighted.

In denary this new number has the value $64 + 16 = 80$. This is 40 multiplied by 2.

If we shift another place to the left – which is two places to the left from the original number – we get:

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

In denary this new number is $128 + 32 = 160$. This is the original number 40 multiplied by 4.

If we shift this number one more place to the left we get:

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

The most significant bit (MSB) was 1 and will be lost. This results in the number 64 being stored. This is an overflow error.

Each time the value is shifted one place to the right the number is divided by two.

Starting again with the number 101000, if we shift the digits one place to the right we get:

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

In denary the new number 10100 is $16 + 4 = 20$, which is 40 divided by 2.

If we shift another place to the right we get:

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

In denary this is $8 + 2 = 10$, which is the original number 40 divided by 4.

So far, the digits we have lost from the original number 101000 have been zeroes. However, if the least significant bit (LSB) is a 1 and we lose it, this causes a loss of precision.

For example, if we shift the original number 101000 to the right by 3 places, we get:

128	64	32	16	8	4	2	1
					1	0	1

The least significant bit is now '1'.

If we now shift this number to the right by one place, we get:

128	64	32	16	8	4	2	1
						1	0

In denary, this is 2. We have binary shifted the original number four places to the right, which means that we have divided the original number by 16 (because $2 \times 2 \times 2 \times 2 = 16$). But 40 divided by $16 = 2.5$.

We have a loss of precision. This is because the LSB in the last binary shift was a 1, which we lost. Note that loss of precision and overflow errors only occur when the MSB or LSB is a 1.

Knowledge check



Apply the shifts described using 8 bits to the following binary numbers and state the denary equivalents before and after the shift. Comment on what has happened to the value.

- 25** 1100 shift 2 places to the right
- 26** 11010 shift 1 place to the left
- 27** 101 shift 3 places to the left
- 28** 110000 shift 3 places to the right
- 29** 111 shift 4 places to the left
- 30** 10000000 shift 4 places to the right
- 31** 110011 shift 4 places to the left
- 32** 111 shift 2 places to the right

How to convert positive denary whole numbers into two-digit hexadecimal numbers and vice versa

Computer scientists use another number system called **hexadecimal** (or hex for short). Hexadecimal is a base-16 number system, so it requires 16 separate symbols. It uses 0–9 for the first ten symbols, just like denary. However, it requires extra symbols to represent the denary numbers 10 to 15. It uses the letters A through to F for this purpose.

Base 10 (denary)	Base 2 (binary)	Base 16 (hex)
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

The column values for base 16 (hex) are based on multiples of 16 so the first two columns are:

16	1

To convert denary numbers to hexadecimal we:

- Check if 16 will divide into the number.
- If it does, we write down how many times using the correct hexadecimal symbol in the 16s column.
- We then convert the remainder into its hexadecimal symbol and write it in the 1s column.

Worked example



Convert 189 in denary to hexadecimal.

189 divided by 16 = 11 remainder 13

The hex symbol for 11 is B

The hex symbol for 13 is D

189 in denary is BD in hexadecimal

You can check this: $11 \times 16 = 176$,
and $189 - 176 = 13$

To convert hexadecimal numbers to denary we:

- Convert the individual symbols to their denary equivalent.
- Multiply the denary values by the column values – either 16 or 1.
- Add the results.

Worked example



To convert AF in hexadecimal to denary

16	1
A	F

A is equivalent to 10 and F is equivalent to 15 in denary.

$(10 \times 16) + (15 \times 1) = 175$

AF in hexadecimal is 175 in denary.

Knowledge check



33 Convert the following denary values to hexadecimal

- (a) 52
- (b) 67
- (c) 165
- (d) 191
- (e) 201

34 Convert the following hexadecimal numbers to denary

- (a) 12
- (b) 58
- (c) 5D
- (d) AE
- (e) CA

How to convert from binary to hexadecimal equivalents and vice versa

The binary number 11011000 can be represented in hexadecimal as D8. This is far easier for programmers to use, remember and communicate without introducing errors. So, hexadecimal is used by programmers because it is simple to work with and easy to convert to and from binary.

Programmers often work with bytes, which are 8-bit numbers – for example 11001110. They would like to represent this number using just two hexadecimal digits. So, first they split it into two 4-bit numbers: 1100 and 1110. As we have already seen, 4-bit numbers are known as nibbles.

4-bit numbers can represent 16 different values – from 0000 up to 1111 in binary, which is 0 to 15 in denary. Hence any 4-bit number can be represented by just one hexadecimal number 0–F. To convert binary to hexadecimal, all we do is separate the binary into nibbles and look up the equivalent symbol.

Worked example



Convert 11001110 in binary to hexadecimal.

The binary number is divided into two nibbles and the equivalent hexadecimal symbol identified. This can be done by first converting the binary to denary if necessary. You can use the table given in the previous section.

1100	1110
C	E

11001110 in binary is CE in hexadecimal.

If we have a number with fewer than 8 bits, the same process applies. We split up the number into nibbles, starting from the right-hand side.

Worked example



Convert the binary number 1011101 to hexadecimal.

101	1101
5	D

1011101 in binary is 5D in hexadecimal

We use a similar process to convert between hexadecimal and binary: we replace the hex symbol with the equivalent binary nibble. You can convert hex to decimal first if that is easier, or you can use the table given at the beginning of this section.

Worked example



Convert B8 in hexadecimal to binary.

B	8
1011	1000

B8 in hexadecimal is 10111000 in binary.



Worked example

Convert 3A from hexadecimal to binary.

3	A
11	1010

3A in hexadecimal is 111010 in binary.

Key point

You will be expected to work with and convert between all three number systems in the range:

00–FF in hex

0–255 in denary

00000000–11111111 in binary.

Knowledge check



35 Convert the following binary numbers to hexadecimal. **36** Convert the following hexadecimal numbers to binary.

(a) 10011100

(d) 111001

(a) 95

(d) A3

(b) 110011

(e) 1001110

(b) AB

(e) 56

(c) 11111111

(c) 1D

CHARACTERS

Using binary codes to represent characters

When you press the keys on a keyboard, the computer registers this as a **binary code** to represent each character. This code can then be used to identify and display a character on screen or for printing. It is important for systems to agree on these codes and their meanings if the data is to make any sense. There are agreed international standards that are used to represent the character set for a computer system.

Character sets and bits per character

The **character set** of a computer is all the characters that are available to it. The number of characters in the character set depends upon how many characters can be represented by the associated codes. The first agreed standard was based on contextual messages in English with a limited number of extra symbols. Wider use of computers, and the need for many more languages and other symbols, has led to the development of more advanced coding standards for character sets.

ASCII

In 1960, the American Standards Association agreed a set of codes to represent the main characters used in English. This is called **ASCII** (American Standard Code for Information Interchange). This system was designed to provide codes for the following:

All the main characters, i.e. 26 uppercase and 26 lowercase	52 characters
All the numeric symbols 0–9	10 characters
32 punctuation and other symbols plus 'space'	33 characters
32 non-printable control codes	32 characters

In total, this is 127 characters, or in binary 1111111 (7 bits). Initially, the ASCII character set used 127 codes for the characters, with 0 meaning 'no character'. This gave a total of 128 characters. One additional bit was used for error checking purposes. This meant there were 8 bits overall, and that each character required one byte.

Some ASCII codes are:

Binary	Hex	Denary	Character
0100000	20	32	'space'
1000001	41	65	A
1000010	42	66	B
1000011	43	67	C
1100001	61	97	a
1111001	79	121	y
1111010	7A	122	z
1111111	7F	127	'delete'

Extended ASCII

As the need for more characters became necessary, including non-English characters and mathematical symbols, the extended ASCII set used the full 8 bits in the byte to represent another 128 characters, making the available character set 256, rather than the original ASCII limit of 128.

Unicode

Unicode was developed to use 16 bits rather than a single byte. This provided the ability to store 2^{16} or 65 536 unique characters. Later developments of Unicode use more bits to represent billions of different characters, including graphical symbols and emojis.

To ensure compatibility of all of these systems, the original ASCII and extended ASCII character codes are the same within Unicode. ASCII is considered a subset of Unicode.

The ASCII codes for the main alphabetic characters are allocated to the uppercase characters in sequence, followed by the lowercase characters in sequence. 'A' is 65 and 'B' is one more at 66, and so on. Lowercase characters start with 'a' as 97 and this means that when we sort text, 'Z' comes before 'a'.

For example, if the following animals written as Goat, Bear, ape, Zebra, deer are sorted using ASCII values, they will be in the order:

Bear, Goat, Zebra, ape, deer

Character set	Number of bits	Number of characters	Examples
ASCII	7	128	Upper and lowercase, numbers, punctuation, some control characters
Extended ASCII	8	256	As above plus non-English characters and mathematical symbols
Unicode	16/32 bits	65 000/2 billion +	As above plus all known language characters and different characters including graphical symbols and emojis

The size of a text file can be calculated by taking the number of characters and multiplying this by the number of bits per character.

Worked example

For example, a text file with 100 characters stored in 32-bit Unicode

$$100 \times 32 = 3200 \text{ bits}$$

Dividing by 8 we get 400 bytes

There may be a small overhead to include the metadata, typically we add 10% to be on the safe side, but in this case 400 B would be a reasonable estimate.



IMAGES

How an image is represented as a series of pixels and in binary

A simple image can be made up of black or white blocks. Binary numbers can represent these black and white blocks, using 1 for black and 0 for white. The image in Figure 1.2.7 uses 8 bits, i.e. 1 byte, to represent each row. Those blocks are the smallest element of an image and are called **pixels**.

This image is just 8 pixels wide by 8 pixels high. As each row is represented by one byte, this image requires 8 bytes to store it.

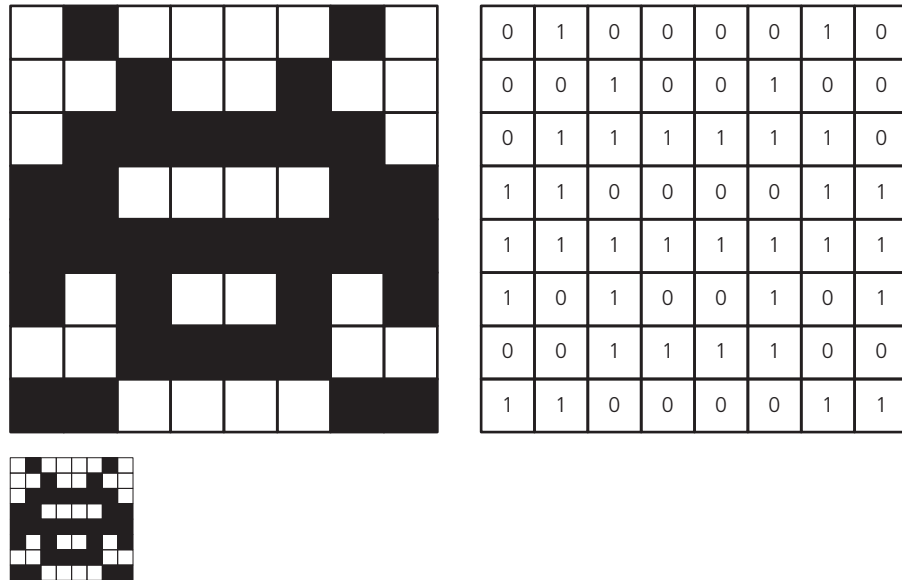
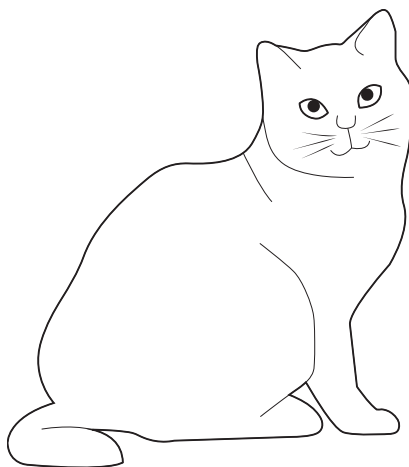


Figure 1.2.7 A simple black and white image

Most images are far more complex than this. For instance, the simple black and white drawing in Figure 1.2.8a is 100×152 pixels. This requires 15 200 bits to store it, which is $15\,200 / 8 = 1900$ bytes, or just under 2 kilobytes. The screenshot in Figure 1.2.8b shows just a small section of the data that is being stored for this image. Like all data, the image is ultimately stored as a series of binary numbers.

When we look at colour images, we need to store more than just a 1 or 0 for each pixel; we need to be able to store enough data to represent a range of colours.



00000000	00	01	02	03	04	05	06	07	08	09	0a	0b	0c
00000000	01000111	01001001	01000110	00111000	00111001	01100001	01111011	00000000	01100100	00000000	11110111	00000000	000000
00000010	00000000	00000000	00110011	00000000	00000000	01100110	00000000	00000000	10011001	00000000	00000000	11001100	000000
00000020	00110011	00000000	00000000	00110011	00110011	00000000	00110011	01100110	00000000	00110011	10011001	00000000	000110
00000030	11111111	00000000	01100110	00000000	00000000	01100110	00110011	00000000	01100110	01100110	00000000	01100110	100111
00000040	00000000	01100110	11111111	00000000	10011001	00000000	00000000	10011001	00110011	00000000	10011001	01100110	000000
00000050	10011001	11001100	00000000	10011001	11111111	00000000	11001100	00000000	00000000	11001100	00110011	00000000	110011
00000060	10011001	00000000	11001100	11001100	00000000	11001100	11111111	00000000	00000000	00000000	11111111	00110110	001100
00000070	00000000	11111111	10011001	00000000	11111111	11001100	00000000	11111111	11111111	00110011	00000000	00000000	001100
00000080	00000000	01100110	00110011	00000000	10011001	00110011	00000000	11001100	00110011	00000000	11111111	00110011	001100
00000090	00110011	00110011	00110011	01100110	00110011	00110011	10011001	00110011	00110011	11001100	00110011	00110011	111111
000000a0	00110011	01100110	00110011	00110011	01100110	01100110	00110011	01100110	10011001	00110011	01100110	11001100	001100
000000b0	10011001	00000000	00110011	10011001	00110011	00110011	10011001	01100110	00110011	10011001	10011001	00110011	100111
000000c0	11111111	00110011	11001100	00000000	00110011	11001100	00110011	00110011	11001100	01100110	00110011	11001100	100111
000000d0	00110011	11001100	11111111	00110011	11111111	00000000	00110011	11111111	00110011	00110011	11111111	01100110	001100
000000e0	11111111	11001100	00110011	11111111	11111111	01100110	00000000	00000000	01100110	00000000	00110011	01100110	000000
000000f0	10011001	01100110	00000000	11001100	01100110	00000000	11111111	01100110	00110011	00000000	01100110	00110011	001100
00000100	01100110	00110011	10011001	01100110	00110011	11001100	01100110	11111111	01100110	01100110	00000000	01100110	001100
00000110	01100110	01100110	01100110	01100110	10011001	01100110	01100110	11001100	01100110	01100110	11111111	01100110	100111
00000120	00110011	01100110	10011001	01100110	01100110	10011001	10011001	01100110	10011001	11001100	01100110	10011001	111111
00000130	01100110	11001100	00110011	01100110	11001100	01100110	01100110	11001100	10011001	01100110	11001100	11001100	011000
00000140	11111111	00000000	01100110	11111111	00110011	01100110	11111111	01100110	01100110	11111111	10011001	01100110	111111
00000150	11111111	10011001	00000000	00000000	10011001	00000000	00110011	10011001	00000000	01100110	10011001	00000000	100111
00000160	10011001	00000000	11111111	10011001	00110011	00000000	10011001	00110011	00110011	10011001	00110011	01100110	100111
00000170	00110011	11001100	10011001	00110011	11111111	10011001	01100110	00000000	10011001	01100110	00110011	10011001	011000

Figure 1.2.8a and Figure 1.2.8b A black and white image and the data that is being used to store it

If we want to store four colours, we need to represent each pixel as one of four different values. We can use 2 bits to do this with the values: 00, 01, 10 and 11. The colour of each pixel will be represented by one of these four binary codes.

So with 2 bits, we can store 2^2 or four colours.

If we use more bits to represent each pixel, we can represent more colours:

With 3 bits per pixel we can store 2^3 or eight colours. (In binary, these eight codes are: 000, 001, 010, 011, 100, 101, 110, 111.)

With 8 bits per pixels we can store 2^8 or 256 colours.

With 16 bits per pixel we can store 2^{16} or 65 536 colours.

The number of bits used to represent the colours is often referred to as the colour depth. The more colours we have, the more data we need to store for each pixel.

Metadata

As we have already seen, image data is stored as a series of numbers. In order to display the image accurately, the computer needs to know exactly how to interpret the numbers. This information about the image is called **metadata**.



Figure 1.2.9 Image of a sign in Portmeirion

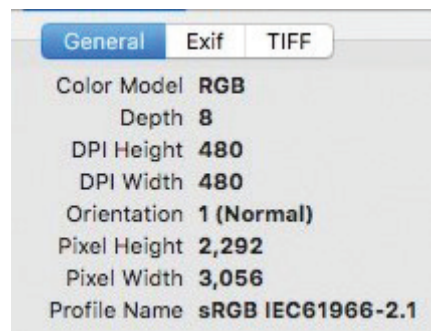


Figure 1.2.10 Metadata stored for this image of the sign in Portmeirion

Metadata includes information such as:

- the number of bits per pixel (the colour depth, as discussed in the previous section)
- the size of the image in bits
- the resolution of the image in dots per inch (DPI).

This information is used to reproduce the image accurately from the data stored.

The effect of colour depth and resolution

Colour depth is the number of bits used per pixel. The more bits per pixel, the larger the range of colours we can have in the image. The more colours we have available, the better the representation of the image.

Look at the image of a sign in Portmeirion in Figure 1.2.11. The original image has a **bit depth** of eight, which is 2^8 or 256 colours, and it is 1.2 MB in size. As we reduce the number of colours available, the image become less well defined but the size of the file reduces, to 787 KB for eight colours and 542 KB for four colours.

The more colours we have, the more bits per pixel we need, and the more data we have to store. Consequently, the higher the colour depth, the larger the file needed to store the image.



Figure 1.2.11 The same image with eight colours and with just four colours

Tech term

Pixelated Blocky

The **resolution** of an image is the number of pixels per unit of distance – usually referred to as dots per inch (DPI). The more dots per inch, the greater the detail stored. This enables us to enlarge the image more without it becoming **pixelated**. However, the more dots or pixels per inch, the more data we need to store – and this means the image file is larger.

Greater colour depth and higher resolution result in higher quality images, but at the expense of larger file sizes. To calculate the size of an image file, we need to know the colour depth and the width and height in pixels:

$$\text{File size} = \text{colour depth} \times \text{image height (pixel)} \times \text{image width (pixel)}$$

Worked example



For example, an 8-bit image that is 1200 pixels high and 2000 pixels wide:

$$\begin{aligned}\text{File size} &= 8 \times 1200 \times 2000 \\ &= 19\,200\,000 \text{ bits}\end{aligned}$$

In bytes this is:

$$\begin{aligned}&= 19\,200\,000 / 8 \\ &= 2\,400\,000 \text{ B} \\ &= 2.4 \text{ MB}\end{aligned}$$

Once again, we would add 10% to account for overheads, including the metadata, to get:

$$2.4 + 0.24 = 2.64 \text{ MB}$$

We could round this to 2.6 MB for our estimate.

Knowledge check



- 37 How does the resolution of the image affect the size of the file used to store it?
- 38 How does the resolution of the image affect the quality of the image?
- 39 What do we mean by colour depth?
- 40 How does reducing the colour depth affect the image?
- 41 How many colours can be represented using a 4-bit colour depth?

Tech term

Discrete Individual distinct values.

SOUND

Sound is a series of vibrations that move through air as waves. These vibrations continuously vary and can take any value, which means they are **analogue**. To store sound on a computer, we need to convert this continuous analogue data into **discrete** digital data. To do this, we convert the analogue data into binary numbers.

How sound can be sampled and stored in digital form

The continuously varying sound wave is sampled at set time intervals. Each sample is a snapshot of the sound wave, represented by discrete digital values that can be stored by a computer. In Figure 1.2.12, the x-axis represents time. The vertical y-axis represents the amount (or amplitude) of vibration. At each time interval, the sound is sampled and the amount of vibration is measured and stored as a digital value.

From the graph, we can read the corresponding amplitude for the curve at each sample point. The first few are shown in the table.

Notice that in the example the amplitude of vibration can only be stored to the nearest ten and the sampling is only done once per second. When the computer uses these values to recreate the sound, we get a shape in Figure 1.2.13 that is similar to the original but not as smooth and missing a lot of the detail. In this case the sampled sound will not be an accurate version of the original sound.

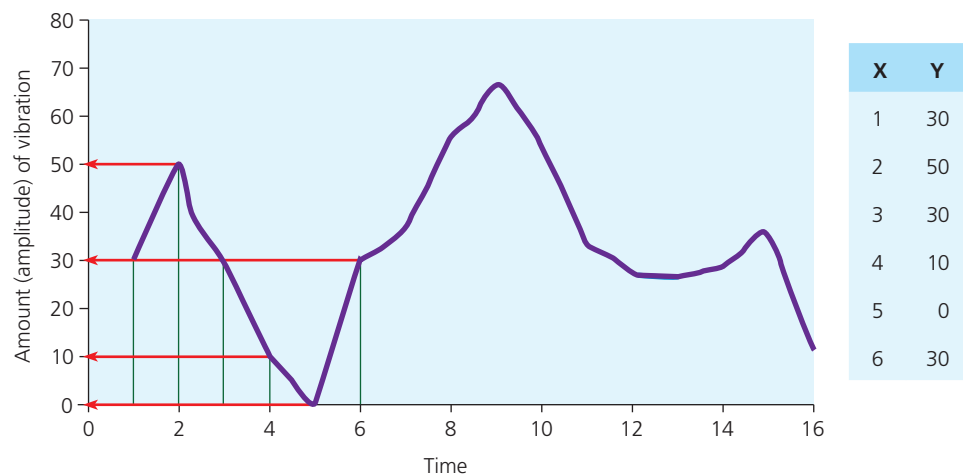


Figure 1.2.12 Sound is sampled at set time intervals

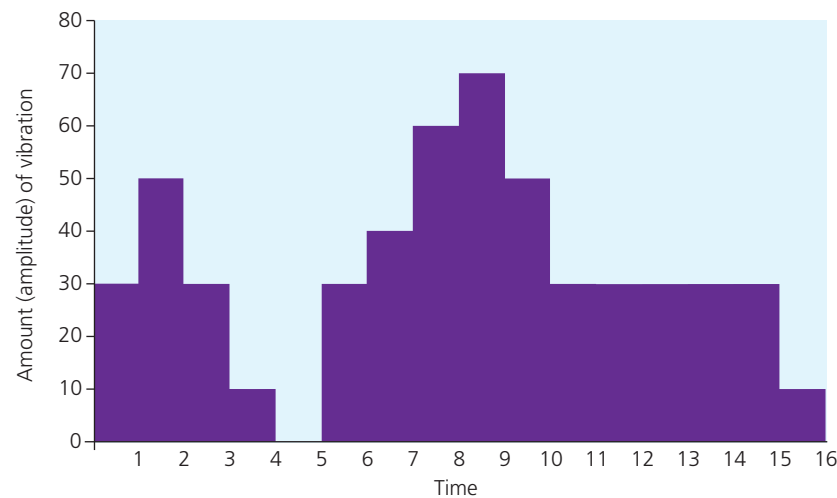


Figure 1.2.13 Digital sound replayed by the computer

Factors affecting playback quality and size of sound file

The **sample rate** is the number of samples taken per second usually measured in hertz (Hz). One sample per second = 1 Hz. A typical commercial CD contains music sampled at 44.1 kHz (44 100 samples per second).

If we sample more frequently, we will get a better approximation of the original sound. However, each sample requires a certain amount of data, so more samples means a larger file to store the data.

The bit depth is the number of bits used to store each sampled value. The more bits we use, the more accurately we can represent the data for that sample point, providing a better representation of the original sound. However, the more bits we use to store each data point, the larger the file needed to store the data. A typical CD recording has a bit depth of 16 – which means 16 bits are used for each sample. This means that the analogue data can take one of $2^{16} = 65\,536$ different digital values.

In Figure 1.2.14, the original sound wave has been sampled again, but this time there are twice as many samples per second and the bit depth has increased so that now the amplitude of vibration can be measured to the nearest '1'. You can see how much more accurate this sample is than the one in Figure 1.2.13, when compared to the original.

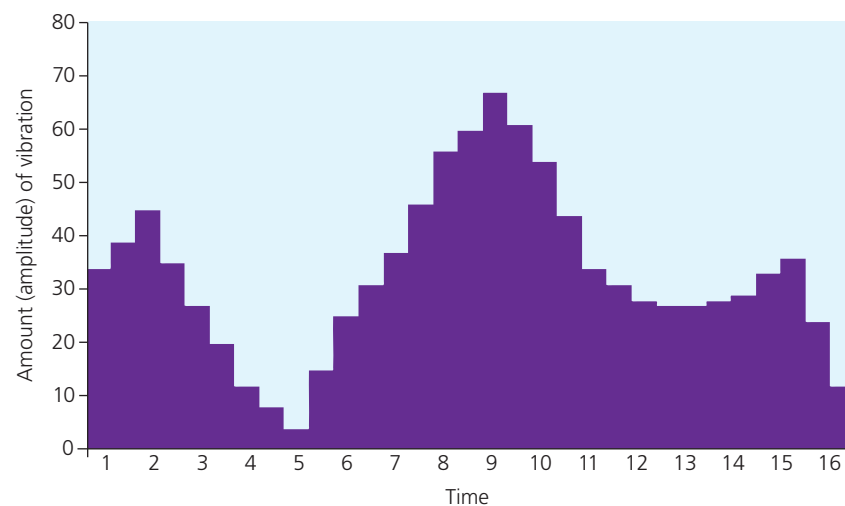


Figure 1.2.14 Higher sample rates and bit depths produce a better approximation of the original

The **duration** of the sampling is how many seconds of sound are sampled. The more seconds of sound that are sampled, the more data we need to store and the larger the file needed to store that data.

The size of the file needed to store sound data depends on these four factors:

- sample rate
- duration
- bit depth
- number of channels.

$$\text{Size of file} = \text{sample rate} \times \text{duration} \times \text{bit depth} \times \text{number of channels}$$

Worked example



A sample of a sound is made at 44.1 kHz for 1 minute at a bit depth of 16 bits. Calculate the size of the resulting sound file.

44.1 kHz means 44 100 samples per second. Each sample consists of 16 bits. So, the number of bits per second

$$= 44\,100 \times 16$$

$$= 705\,600 \text{ bits per second.}$$

The sample lasts for 60 seconds, so the total number of bits

$$= 705\,600 \times 60$$

$$= 42\,336\,000 \text{ bits}$$

Divide by 8 to get the number of bytes:

$$= 42\,336\,000 / 8$$

$$= 5\,292\,000 \text{ bytes}$$

Divide by 1 000 000 for megabytes:

$$= 5\,292\,000 / 1\,000\,000$$

$$= \mathbf{5.29 \text{ MB}}$$

With audio files, the overheads, including the metadata that describes the data, are small and rounding to 5.3 MB would provide a good estimate of the file size.

This calculation is for a recording on one track, which is called mono. A stereo recording uses two tracks and uses twice as much data as a mono recording. In the example above, a stereo recording would use $5.29 \times 2 = 10.58 \text{ MB}$.

Knowledge check



42 What is meant by bit depth?

43 How does the sample rate affect the sound quality and the file size of the file used to store the sampled sound?

1.2.5 Compression

The need for compression

As we have seen, storing images and sound can lead to large file sizes. When large amounts of data are transmitted across the internet, for example sending videos, it can be very slow and may be expensive. If we want to store lots of data on a device with limited memory, for example music and videos on a mobile phone, we might run out of space. However, we can compress data to make the file smaller. This reduces transmission times between devices and allows more files to be stored.

Types of compression

There are two types of compression – **lossy** and **lossless**.

Lossy

Lossy compression is where some data is permanently removed to make the file smaller.



Figure 1.2.15 The same image compressed to 364 KB and to 166 KB

The removed data is chosen as that least likely to be noticed by the human senses. For example, certain frequencies of sound are inaudible or barely audible to the human ear, so these frequencies can be discarded without any significant differences being heard. In images, large areas of very similarly coloured pixels are combined into one block of the same colour, so that the image still looks very similar to the original.

In Figure 1.2.15, you can see that there is some loss of detail in the second image but the essential features of the original image are still clearly visible.

It is important to note that once a file has been compressed using lossy compression techniques, it cannot be reversed and restored to its original condition.

Lossless

For some files, losing any information is simply not possible – for example, computer programs will only work if all the instructions are present. In text files, removing any of the words or characters would alter the sense of the document. For these files lossy compression techniques are not suitable. However, there are other techniques for compressing files without losing any of the original information. This is called **lossless compression**. It involves storing enough data and information to be able to recreate the original file exactly as it was before compression.

For text files, lossless compression can be achieved by creating a dictionary of words used and a list providing the order in which those words were used.

For example, consider the sentence:

Ask not what your country can do for you, ask what you can do for your country.

This sentence can be represented by the following index:

Index	Word
1	ask
2	not
3	what
4	your
5	country
6	can
7	do
8	for
9	you

By referring to the dictionary, the original sentence can be reproduced from the sequence:

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

Storing the index and the sequence requires a smaller file than storing the original sentence. For larger blocks of text with many repeated words, this can reduce the size of a text file quite considerably.

Lossless compression means that files can be restored to their original condition since no data has been removed and all the information is still available.

Knowledge check



- 44** What is file compression?
- 45** Describe the difference between lossy and lossless compression.
- 46** Explain why lossless compression must be used when sending a computer program as an email attachment.
- 47** Explain the importance of using compression for sending image and video files over the internet.

RECAP AND REVIEW

1.2 MEMORY AND STORAGE

Important words

You will need to know and understand the following for the exam:

Primary storage

RAM

ROM

Volatile and non-volatile

Read-only

Read and write

Virtual memory

Secondary storage

Magnetic storage

Hard disk drives (HDDs)

Solid-state storage

Solid-state drives (SSDs)

Optical storages

Binary

Bit (b)

Byte (B)

Nibble

Using prefixes: kilobyte (KB),
megabyte (MB), gigabyte
(GB), petabyte (PB)

Denary

Overflow error

Binary shift

Most significant bit (MSB)

Least significant bit (LSB)

Hexadecimal

Binary code

Character set

ASCII

Unicode

Pixel

Metadata

Colour depth

Resolution

Analogue

Sample rate

Bit depth

Lossy compression

Lossless compression

1.2.1 Primary storage

A computer needs **primary storage** for data it needs to access quickly. This includes:

- start-up or boot instructions
- the operating system
- programs that are running
- any data associated with the operating system or programs.

There are two main types of primary storage: RAM and ROM.

RAM

- Random access memory (**RAM**) is **volatile**, which means it needs power to maintain it. If the power is turned off the RAM loses its contents.
- RAM holds the operating system, and any applications and data currently in use by the computer.
- The CPU can access RAM quickly – much faster than it can access secondary storage such as a hard disk drive.
- The more RAM in a computer the more programs and data it can run at the same time and the better the computer's performance.
- RAM can be read or written to.
- RAM is also known as the main memory in the computer.

ROM

- Read-only memory (**ROM**) is **non-volatile**, which means it does not need power to maintain it. If the power is turned off ROM keeps its contents.
- ROM provides storage for data and instructions needed to start up the computer (also known as the boot process).
- ROM content is **read-only**, which means it cannot be overwritten.
- Information in ROM is often written at the manufacturing stage.

A comparison of RAM and ROM

RAM	ROM
Is volatile and needs power to maintain the content	Is non-volatile and does not require power to maintain the content
Is read and write – data can be read from and written to RAM by the computer	Is read-only – the computer cannot overwrite its content
Holds the operating system and any programs and data currently in use by the computer	Holds the data and instructions required to start up (boot) the computer

Virtual memory

Running several programs at once, or running programs that use large amounts of data, can require more RAM than is available. However, the computer can assign a section of secondary storage to temporarily act like RAM. This section of secondary storage is called **virtual memory**.

- Any data from a running program that is not currently being used by the computer can be temporarily moved from RAM to virtual memory.
- When that data is required by the computer, it is moved back from virtual memory into RAM.
- Moving data between RAM and virtual memory is relatively slow – so using virtual memory slows down the performance of the computer.
- Adding more RAM reduces the need for virtual memory. If less data is held in virtual memory then there are fewer slow data transfers between RAM and virtual memory.
- Therefore, adding more RAM improves the performance of the computer.

1.2.2 Secondary storage

Secondary storage is needed to store files and programs. It needs to be:

- non-volatile, that is, it doesn't lose data when switched off
- low cost
- high capacity
- reliable.

Magnetic storage

Magnetic storage mostly uses **hard disk drives (HDDs)**.

- They are made of a stack of magnetic disks (or platters) that rotate.
- A moving read/write head moves across the surface of the platters to read and write data.
- Magnetic disks are reliable and cost effective and provide high-capacity storage at low cost.

Solid-state storage

Solid-state storage uses a technology called flash memory. It is used in portable hand-held devices and increasingly in computers in the form of **solid-state drives (SSDs)**.

SSDs have the following characteristics:

- Because they use flash memory, SSDs have no moving parts.
- No moving parts means access to data is faster than for a magnetic hard disk drive.
- No moving parts also means power requirements are low and no noise or heat is generated.
- SSDs are robust, lightweight and compact making them ideal for use in portable devices.
- SSDs have a smaller capacity than magnetic hard disk drives and the cost per unit of storage is higher.
- SSDs are commonly used in tablet computers, mobile phones, cameras and general-purpose computers.

Optical storage

- **Optical storage** devices use the properties of light to store data.
 - The most common optical storage medium are optical disks – CDs, DVDs and Blu-Ray disks.
 - They work by reflecting laser light onto the surface of the rotating disk and reading the reflections as 1s or 0s.
 - CD-ROM and DVD-ROM are read-only media.
 - CD-R and DVD-R are write-once/read many times media.
 - CD-RW and DVD-RW are rewriteable media.
 - Blu-Ray disks use a blue laser light that can detect data stored at a higher density than CDs and DVDs.
 - Blu-Ray disks have a much higher capacity than CDs and DVDs, making them ideal for storing and distributing high-definition video films or large amounts of data.
 - Blu-Ray disks can be formatted with up to four layers for very high capacity.
- Optical media are low cost and robust making them an ideal way to distribute data.

Advantages and disadvantages of each media

When selecting the right device or medium for a purpose the following must be considered:

- **Capacity:** How much data does it need to store?
- **Speed:** How quickly does the data need to be accessed?
- **Portability:** Does the device or medium need to be transported?
If so, the size and weight are important.
- **Durability:** Will the device or medium be used in a hostile environment?
If so, the medium must be resistant to external shocks or extreme conditions.

- **Reliability:** Does it need to be used repeatedly without failing?
- **Cost:** What is the cost per unit of storage related to the value of the data?

Media	Capacity	Typical cost	Cost per GB
Magnetic hard disk	Up to 15 TB	A 2 TB HDD costs around £60 and a 10 TB drive about £300	3p
SSD	250 MB up to 2 TB	£30 up to £300 for a 2 TB drive	15p
DVD	8.5 GB	80p	9p
Blu-Ray	50 GB	£3.00	6p
CD	700 MB	18p	23p

Storage type	Data transfer rates (typical)
SSD	200–550 MB/s
Magnetic hard disk	50–120 MB/s
Blu-Ray disk	72 MB/s
DVD	1.32 MB/s
CD	0.146 MB/s

Media	Portability	Durability	Reliability
SSD	Small, with low power requirements Very portable	With no moving parts they are not subject to damage from sudden shocks.	The medium is reliable and will hold data safely for a very long time before failure.
Magnetic hard disk	With moving parts, higher power requirements than SSD. Available as external drives powered from a USB	Subject to damage from being dropped or from exposure to magnetic fields.	Ideal for medium term storage with a reliable life of 5–7 years. Motors and heads are subject to failure over time or from excessive use or mishandling.
CD DVD Blu-Ray	Light and small Very portable – can even be sent through the post	Reasonably robust and resistant to shocks Easily damaged by mishandling and scratches.	CDs and DVDs will start to fail after 10 years; Blu-Ray will fail after 20 years.

1.2.3 Units

Why data needs to be converted into binary

Computers use switches to store data and these switches can be in one of two states: on or off. Because of this we need to convert all data and instructions into **binary**, which can represent on or off using the two digits 0 and 1.

Units of data storage

- Each stored binary digit is called a **bit** (binary digit).
- A group of 8 bits is called a **byte**.
- Half a byte, 4 bits, is called a **nibble**.

4 bits (b)	1 nibble
8 bits (b)	1 byte (B)
1000 B	1 kilobyte (KB)
1000 KB	1 megabyte (MB)
1000 MB	1 gigabyte (GB)
1000 GB	1 terabyte (TB)
1000 TB	1 petabyte (PB)

It is important to use the correct symbol, lowercase b for bit and uppercase B for byte.

Data capacity and calculation of requirements

It is important to be able to calculate the required data capacity when choosing storage media. To do this, we add up the estimated file size for each of the files we need to store and add up to 10% to cover overheads.

For example:

10 pages of word-processed documents @ 100 KB	1 MB
3 postcard sized images @ 6 MB	18 MB
5 minutes of MPEG video @ 50 MB	250 MB
Total	269 MB
10% for overheads	27 MB
Total capacity required	296 MB

sound file size = sample rate (Hz) × duration (s) × bit depth × number of channels

image file size = colour depth × image height (pixels) × image width (pixels)

text file size = bits per character × number of characters

1.2.4 Data storage Numbers

Converting binary to denary

- Our everyday counting system is called **denary** (or decimal). Denary is a base-10 number system.
- Binary is a base-2 number system.

To convert a binary number into denary use the column values and add together the column values for every column with a 1 in the binary number.

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

This is $128 + 32 + 2 = 162$ in denary

The leftmost digit in the number is called the most significant bit (MSB) and the rightmost digit the least significant bit (LSB). In an 8-bit number, the MSB value represents 128 in decimal and the LSB value represents 1.

Converting denary to binary

- Create a table with eight columns representing an 8-bit binary number.
- Starting with the left-hand 128 column, find the first value that can divide into the denary number and write a 1 in that column.
- Subtract that column number from the denary number to get a remainder.
- Repeat the process using the remainder.
- Eventually there will be a remainder of either 1 or 0 to be entered into the right-hand 0 column.

Adding binary numbers

When adding two binary digits together there are four possibilities

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ in binary, or } 2 \text{ in denary.}$$

In the case of $1 + 1$ we write down 0 and carry 1 to the next column.

If we carry a 1 to the next column then there is one more possibility:

$$1 + 1 + 1 = 11 \text{ in binary or } 3 \text{ in denary.}$$

In this case we write down 1 and carry 1.

For example, to add the two numbers 1101 and 1111:

$$\begin{array}{r}
 1 \quad 1 \quad 0 \quad 1 \\
 + \quad 1 \quad 1 \quad 1 \quad 1 \\
 \hline
 1 \quad 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

The carried 1 is the final digit in the sum giving the answer 11100.

Overflow errors occur when the number becomes too large to store in the number of bits available, for example:

$$\begin{array}{r}
 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \\
 + \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \\
 \hline
 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \\
 \hline
 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

When adding two 8-bit numbers, there is no column available to store the carry digit from the 8th column and it is lost.

- The result of the calculation will not fit into 8 bits.
- This is called overflow and means the result will be incorrect.
- In denary, this example has calculated that $231 + 233 = 208$ – which is clearly incorrect and is due to the overflow error.

Overflow can generate further logical errors in a program because the result is not as expected. The program may crash if it cannot deal with the overflow digit.

Binary shifts

Moving the binary digits to the left or right is called a **binary shift**.

Moving to the left multiplies the value by 2 for each place the value is shifted.

Moving to the right divides the number by 2 for each place the value is shifted.

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

This number is 26 in denary.

If we shift the binary number one place to the left, we get:

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

Which is 52 ($= 26 \times 2$) in denary.

If we shift the original binary number one place to the right, we get:

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

This number is 13 ($= 26 / 2$) in denary.

If we shift the original binary number two places to the right, we get:

128	64	32	16	8	4	2	1
					1	1	0

This number is 6 in denary. However, $13 / 2 = 6.5$ so our answer is not quite right. This is because with this shift to the right, we lost a 1. We have lost some information, which has led to a lack of precision.

Shifting the number left when the **MSB** is 1 means that we lose a 1, which results in an overflow error.

Shifting the number right when the **LSB** is 1 means that we lose a 1, which results in a loss of precision.

Hexadecimal

- **Hexadecimal** (or hex) is a base-16 number system.
- We need 16 symbols to represent all the digits in hexadecimal.
- We use 0 to 9, plus the letters A to F to represent the values from 10 to 15.
- You are only expected to work with two-digit hexadecimal numbers in the exam.

Denary	Hex	Binary	Denary	Hex	Binary
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

Converting denary to hex

- Divide 16 into the denary number and write down the correct hexadecimal symbol for the result in the 16s column.
- Convert the remainder to the correct symbol and write that down in the 1s column.

For example, 202 in denary is:

$202 / 16 = 12$ remainder 10.

'12' in denary is 'C' in hex, and '10' in denary is 'A' in hex. The hexadecimal number is therefore:

16	1
C	A

Converting hexadecimal to denary

- Convert each hex digit to its denary equivalent.
- Multiply the column headings by the equivalent denary value.
- Add the two values together.

For example, to convert CA back to denary:

C is 12 in denary and A is 10 in denary.

$$16 \times 12 = 192$$

$$1 \times 10 = 10$$

$$192 + 10 = 202$$

Converting binary to hex

- Split the binary number into nibbles.
- Convert each nibble to the corresponding hex symbol.

For example, 11110 in hex is:

- first nibble is 0001 (remember 11110 is 00011110 as an 8-bit number)
- second nibble is 1110
- 0001 in decimal is 1 which in hex is 1
- 1110 in decimal is 14, which in hex is E.

Hence 11110 is 1E in hex.

In the exam, you will only need to convert between number systems in the following ranges:

- 0 to 255 in denary
- 00000000 to 11111111 in binary
- 00 to FF in hexadecimal.

Characters

- Each character is represented by a numeric **binary code**.
- The **character set** of a computer is a list of all the characters available to the computer.
- It is important computer systems all agree on these codes and there are some agreed standards.

ASCII

- **ASCII** is a 7-bit code able to represent the English alphabet, numbers, some symbols and some control characters.
- There are 2^7 or 128 characters available

Extended ASCII

- Extended ASCII uses the 8th bit in the byte to provide codes for an additional 128 characters to include extra symbols, mathematical symbols and some non-English characters.
- There are 2^8 or 256 characters available.

Unicode

- **Unicode** originally used a 16-bit code to represent many additional non-English characters and a wide range of symbols.
- The 16-bit code has 2^{16} or 65536 characters available.
- Unicode has since been extended to use even more bits to represent billions of characters.
- The original ASCII and extended ASCII codes are the same in Unicode so ASCII can be considered a subset of Unicode.

Text file size = number of characters x number of bits per character

Images

- Images are represented on screen as a series of **pixels**.
- A pixel is the smallest element of an image – these are the dots that make up the image on screen or in a printout.
- Pixels are stored in a computer as binary codes.
- The number of bits used for each pixel determines how many colours each pixel can represent. With 1 bit for each pixel we have just two possibilities: 0 or 1. This means a pixel can only be one of two colours.

To store more than two colours we need more bits per pixel:

- 2 bits to store 4 (2^2) colours per pixel
- 3 bits to store 8 (2^3) colours per pixel
- 8 bits to store 256 (2^8) colours per pixel.

Metadata means 'data about data'.

Image metadata contains information that tells the computer how to reproduce the image from the binary data stored and includes:

- **Colour depth** – the number of bits used per pixel.
- **Resolution** – the number of dots (pixels) per unit of distance, for example dots per inch (DPI).

Higher colour depth:

- a larger number of colours can be represented giving a better quality image
- more data is stored making the image file larger.

Higher resolution:

- more pixels are used to represent the image meaning the quality is better and it can be made larger without losing resolution
- more data is needed to represent the pixel, meaning the file needed to store the data is larger.

Image file size = colour depth x image height (px) x image width (px)

Sound

Sounds are a series of vibrations that continuously vary and can take any value – this means they are **analogue**.

In order to store this on a computer sound is sampled at regular intervals by a device that converts analogue to digital signals, and the digital values are stored as binary.

- The **sample rate** is the number of samples taken per second, measured in Hz (hertz). 1 Hz means one sample per second.
- The **duration** is the length of time that the sound is sampled for, measured in seconds or minutes.
- The **bit depth** is the number of bits used to store each sample.

Sound file size depends on three things:

- The more frequently the sample is taken the better the approximation to the original sound – but the larger the file needed to store the data.
- The longer the duration the larger the file needed to store the data.
- The greater the bit depth the more data is stored about each sample and the better the quality of the sound – but the greater the file needed to store the data.

Sound file size = sample rate (Hz) x bit depth x duration (seconds) x number of channels

1.2.5 Compression

When transmitting files, storing very large files or storing a large number of files we need to compress the data to make it smaller.

Lossy compression

- Some of the data is removed to make the file smaller.
- Algorithms remove data that is least likely to be noticed.
- The original file cannot be restored from the compressed version.

Lossless compression

- None of the data is removed.
- Algorithms look for patterns in the data so that repeated data items only need to be stored once, together with information about how to restore them.
- The original file can be restored.

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

COMPUTER NETWORKS, CONNECTIONS AND PROTOCOLS

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.3.1 Networks and topologies

- Types of network
- Factors that affect the performance of networks
- Client-server and peer-to-peer networks
- Hardware needed to connect computers into a LAN
- The internet as a worldwide collection of computer networks
- Star and mesh network topologies

1.3.2 Wired and wireless networks

- Modes of connection
- Encryption
- IP and MAC addressing
- Standards
- Common protocols
- Layers

A computer network is two or more computers or devices that are linked together, either using cables or wirelessly. This means that they can communicate with one another and can share resources.

As well as computers and printers, devices that may be connected to a network include smartphones, tablets, gaming consoles, fitness trackers, smart watches and internet-connected home appliances.

Advantages of computer networks

One of the main advantages of computer networks is the ability for computers to share hardware resources such as printers and internet connections.

Another advantage of computer networks is the ability to exchange data between computers without needing to use physical media such as memory sticks or external hard drives. This can be particularly useful in a school or office setting and can be achieved using shared drives and folders.

Network users can also communicate easily using email or video calls.

Using networks in larger organisations like schools and businesses allows computers to be managed centrally by a network manager. This enables them to update software remotely and manage security centrally through the use of firewalls and anti-malware software. They can also control access to files and resources for different users, and user activity can be monitored.

Disadvantages of computer networks

Additional hardware is needed to set up a network, which can be expensive, and larger networks will need to be overseen by a network manager.

If one machine on a network gets infected with malware it can quickly spread to other machines on the network if up-to-date anti-malware software and other security measures are not in place.

It is also possible that hackers may target a network specifically in order to gain access to several computers.

1.3.1 Networks and topologies

Types of network

There are two main types of network that you need to know about.

Local area network (LAN)

A **local area network** or **LAN** is the type of network found in homes, schools and single-site companies and organisations. LANs cover a small geographic area as the computers and

buildings are usually located on one site. The hardware is usually owned and maintained by the organisation that uses it, and it will often use both wired and wireless connections.

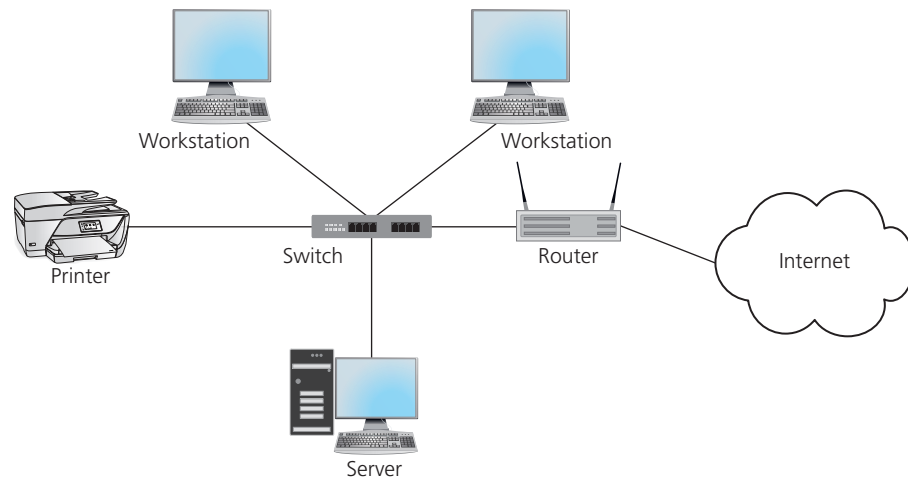


Figure 1.3.1 Basic LAN network

Wide area network (WAN)

A **wide area network** or **WAN** is a type of network used by large organisations such as banks, with offices in different locations that need to be linked together. A WAN typically covers a wide geographic area, which may even be worldwide, and links together the LANs for each different site. The connections between the sites are usually hired or leased from a telecommunication company and may include cable, satellite and telephone lines.

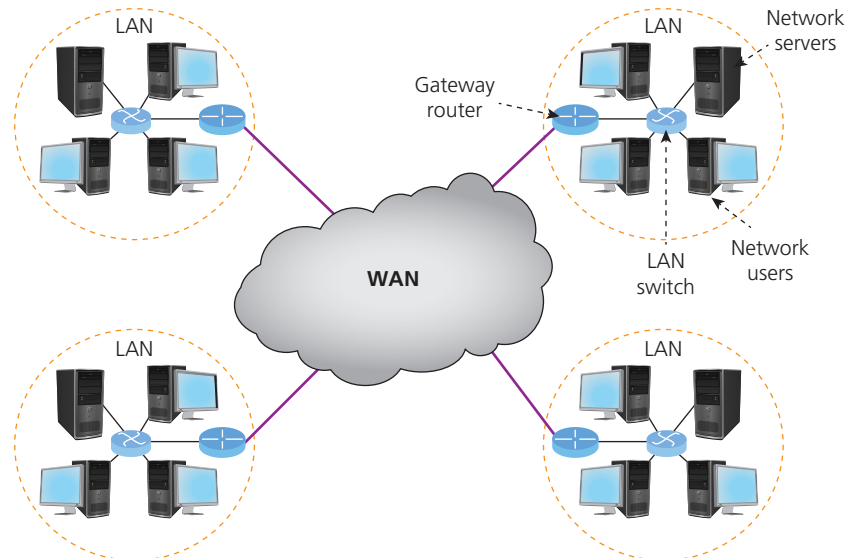


Figure 1.3.2 Basic WAN network

The internet is essentially a huge, worldwide WAN.

Knowledge check



- 1 Describe the characteristics of a LAN.
- 2 Identify two differences between a LAN and a WAN.

Factors that affect the performance of networks

Network performance refers to how quickly data is transmitted through a network, and therefore what the experience is like for the user. There are several main factors which can affect this.

Bandwidth

Bandwidth refers to how much data can be transmitted over a network in a given period of time. It is usually measured in bits (or megabits) per second. The greater the bandwidth capacity, the more data that can be transmitted in that time period. Bandwidth capacity depends on the equipment that makes up the network.

Bandwidth can be compared to water flowing through a pipe. The maximum amount of water that can flow through a pipe in one minute represents the maximum bandwidth. A small pipe will only allow a small amount of water through, whereas a large pipe will allow a lot more water through in the same amount of time. A larger pipe represents a network with a larger bandwidth.

Number of users

The available bandwidth is shared between all of the users on a network. A large number of users can cause network congestion, meaning that **data packets** are queued before they can be transmitted. In addition, some activities, such as streaming HD video, will require a greater amount of bandwidth, which reduces the amount left for other users to share.

Transmission media

Data can be sent around a network through cables or wirelessly. Within a LAN it is usual for copper network cables or Wi-Fi to be used. Copper network cables have a bandwidth of up to 1000 **Mbps** whereas Wi-Fi only has a bandwidth of up to 300 Mbps. Therefore, network cables allow for faster and more reliable transmission. However, despite being slower, wireless connection is very common as it avoids the need to install cables and it is very easy to add and remove new devices.

Traditionally, WANs were connected using copper telephone cables, but as these were designed for carrying voice signals, they do not support fast data transfer. Fibre-optic cable is increasingly used instead and this allows for a much greater bandwidth.

Transmission media will be discussed in more detail later in this chapter.

Error rate

When there is a lot of traffic on a network, it can result in transmission errors due to data packets colliding. This is a bit like two people talking to each other simultaneously and neither being able to hear the other person properly. There may also be transmission errors due to interference from other wireless networks nearby or due to a weak Wi-Fi signal. Errors mean that the packets have to be re-sent, which in turn increases data traffic.

Tech terms

Data packets Small chunks of data that are transmitted across the internet.

Mb (megabit) A megabit is equal to one million bits. Megabits (Mb) should not be confused with megabytes (MB). Remember – there are 8 bits in a byte.

Mbps Megabits per second

Beyond the spec

In a network, latency is a measure of how much time it takes for a packet of data to travel from one device to another. Latency is affected by factors such as the transmission media used.

Knowledge check



- 3 Describe what is meant by bandwidth.
- 4 Identify and explain three elements that can affect the performance of a network.

Client-server and peer-to-peer networks

There are two main ways in which the computers in a network are organised.

Client-server networks

In a **client-server** network there are two types of computers: servers and clients.

Servers are high-end computers that provide services for the rest of the network. There may be more than one server in a network, and servers may be specialised to perform specific functions:

- **File servers** store users' documents and files. This means that they can be accessed from any computer on the network.
- **Authentication servers** check whether a username and password match those stored in a database, and then control the resources that a user can access.
- **Application servers** run programs across the network.
- **Web servers** store and share web pages.
- **Print servers** manage printing across the network.
- **Mail servers** store and handle email.

The computers that connect to servers are known as clients. These request the services and resources that they require from the servers, such as software and files. The server processes this request and then sends a response back to the client. Clients do not normally store data.

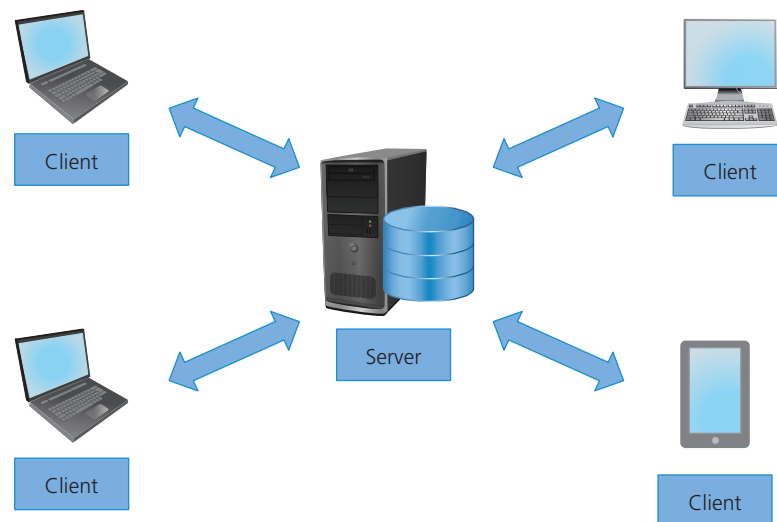


Figure 1.3.3 Client-server network

Client-server networks are the most common way to organise a LAN, and are well-suited to schools or organisations with large numbers of computers, or where users need to access the same software or files.

Advantages of client-server networks

The main advantage of client-server networks is that activities can be managed and controlled centrally. Users' files can be stored on the server so they can be accessed from any connected client device. Backups can also be managed centrally ensuring that all files are included. Software and security updates can be managed by network managers without the need to update every client computer individually, and activity on client machines can be monitored. User accounts can be managed centrally, including the changing of passwords and recovery of lost files, and access levels can be controlled for different categories of user.

Disadvantages of client–server networks

There are, however, some disadvantages of client–server networks. If the server becomes unavailable for any reason, users will not be able to access their files, and servers can become overwhelmed by too many requests, preventing clients from accessing their services. Server hardware is typically more expensive than ordinary computers, which increases the upfront cost of setting up a client–server network. A cyber attack, such as a ransomware attack, only needs to focus on the server rather than on each individual client.

Peer-to-peer networks

In a **peer-to-peer (P2P)** network all of the computers have equal status and are connected directly, using cables or wirelessly, without a central server. Each computer is called a peer and stores its own files. Peers are configured so that specified files and folders can be accessed by other peers on the network.

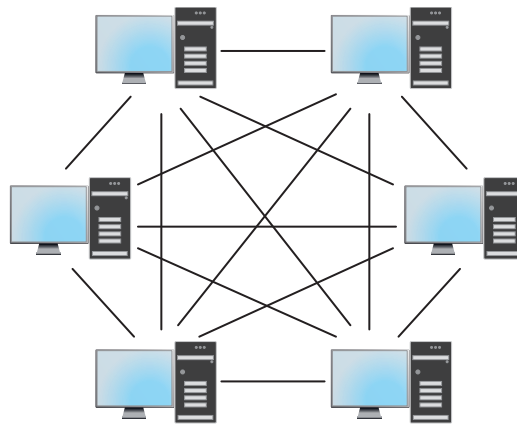


Figure 1.3.4 Peer-to-peer network

P2P networks are easy to set up and can be suitable for small organisations with few computers or less need to share data, and are the model most often found in home networks. Examples of P2P network activities include:

- wireless printing from a laptop, tablet or phone
- ad hoc file sharing, such as the use of AirDrop on iOS devices
- streaming audio from a device to a Bluetooth speaker
- sharing internet connections via personal hotspots.

Advantages of peer-to-peer networks

The main advantage of peer-to-peer networks is that they are easy to set up as they do not require any expensive or dedicated hardware. P2P networks are also more robust as there is no single point of failure, meaning that if one device fails the rest of the network will still continue to operate normally.

Disadvantages of peer-to-peer networks

Without a central server there is no central management or maintenance of a P2P network. This means that software and security updates have to be carried out individually on each peer device. In addition, there is no centralised backup of files as the files are distributed among the peer devices, so each peer needs to be backed up separately. Files are duplicated when they are transferred between devices, which often leads to multiple versions of the same document, which can lead to out-of-date versions being used. It is also possible for peers to go offline when they are being accessed. This might happen if a device loses its Wi-Fi connection, or simply because it is switched off.

Knowledge check



- 5 Describe the characteristics of a client–server network.
- 6 Explain two advantages and two disadvantages of using a client–server network.
- 7 Describe the characteristics of a peer-to-peer network.
- 8 Explain two advantages and two disadvantages of using a peer-to-peer network.

Network hardware

There are various pieces of hardware that are needed to connect together the devices in a network.

Network interface controller/card (NIC)

All devices need some form of **network interface controller/card (NIC)** to connect to a network. These are usually integrated into the motherboard of modern devices, but they used to be separate hardware components (which were called network interface cards).

The NIC formats the data to be sent on a network using the correct protocol, and receives incoming signals. The most common types of NIC are an ethernet port to connect to a wired network, or a radio transmitter/receiver to connect to a wireless network.

Every NIC has a **Media Access Control (MAC) address**, which is a unique identifier used when transmitting data around a network.

Transmission media

On a network, data is transferred between devices either through **cables** (wired) or by **radio waves** (wireless).

Copper wire

Standard network cables consist of eight individual copper wires that are arranged in pairs. Each pair of cables is twisted together to reduce interference from other signals and therefore improve transmission.

Data is transmitted as electrical signals and there are different ratings that indicate how quickly the cable can reliably transmit data and over what range. The bandwidth is generally between 100 Mb and 1 Gb per second, for a distance of up to about 100 metres.

Most PCs have built-in wired LAN port and so connecting computers using copper wire can be a cost-effective option if the bandwidth is adequate.

Fibre-optic cable

Fibre-optic cables are made up of many thin glass strands (or fibres), which transmit data as pulses of light. As they use light to transmit data, they do not suffer from electromagnetic interference. Fibre-optic cables do not break easily as they are strong, flexible and do not corrode.

Fibre-optic cables have a very high bandwidth of up to 100 Tb per second and are capable of transmitting data over distances of 100 kilometres or more. For this reason, they are often used to connect WANs across large geographic areas. The cables that cross the oceans to connect different continents to the internet are fibre-optic cables.

Tech term

Electromagnetic spectrum The range of frequencies of electromagnetic radiation.

Beyond the spec

Hubs are similar to switches but do not hold any information about the devices connected to them. Therefore, all data packets are transmitted to all devices, which affects network performance because many unnecessary transmissions are made.

Radio waves

Wireless networking technologies, such as Wi-Fi and Bluetooth, use radio waves to connect devices. Radio waves form part of the **electromagnetic spectrum** and their use is strictly controlled by governments. The most commonly used frequencies for data transmission are 2.4 GHz and 5 GHz.

The strength of a radio wave decreases as it moves further away from its transmission source, and so radio waves at 2.4 GHz and 5 GHz are only suitable for relatively short distances of up to 100 metres. Radio waves are also subject to interference from other radio signals of the same or similar frequencies, and are partially blocked by physical objects such as walls. They generally have a bandwidth of about 300 Mb per second.

Wireless access point

A **wireless access point (WAP)** is a piece of hardware that connects to a network switch and allows Wi-Fi devices to connect to a network.

WAPs broadcast a **Service Set Identifier (SSID)**, which is effectively the name for the network so that Wi-Fi devices can connect to it. The WAP then sends the wireless data that it receives on to the main wired network.

The WAP is usually connected to a network switch via a cable. However, WAPs can also be used to extend the range of a wireless network by transmitting or receiving data from other WAPs.

Switch

A network **switch** is the piece of hardware that allows multiple devices to connect together to form a wired network.

A switch stores the MAC address of every device connected to it in a table. When the switch receives a packet of data it looks at the destination address and forwards it on to the intended device.

It is possible to connect multiple switches together to increase the number of devices on a network.

Router

A **router** is the piece of hardware that connects networks of different types together. Most commonly, routers are used to connect a LAN to the internet.

Routers inspect the destination IP address (explained in the next section) of a data packet to determine whether it is located on the local network. If it is not, the data packet is passed on to the connected network. Routers collect data about all of the available routes to transmit data and then determine the most appropriate route for each individual data packet.

Home broadband 'routers' often combine the features of a switch, WAP and router in one device, and are referred to as hybrid devices.

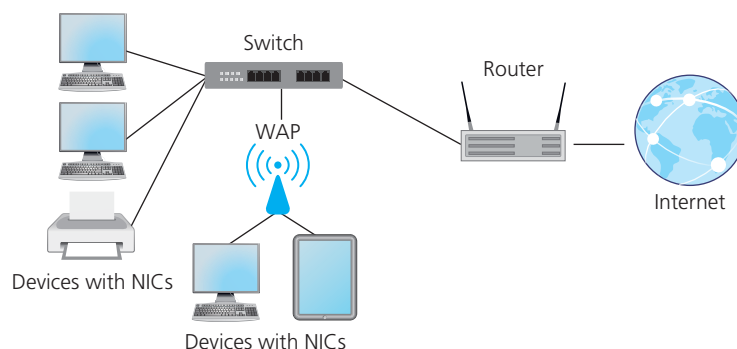


Figure 1.3.5 Basic network diagram

Knowledge check



- 9 State the purpose of a network interface controller.
- 10 Identify two ways in which a desktop computer may be connected to a home network.
- 11 Explain how an individual device is uniquely identified on a LAN.
- 12 State the name of the hardware device that allows other devices to connect to a LAN wirelessly.
- 13 Describe the purpose of a router in a home network.

The internet

The **internet** is a worldwide collection of computer networks that are all linked together as a WAN.

The set of rules that ensures that devices can work together on the internet is called the **Internet Protocol (IP)**. Every computer using the internet has a unique **IP address**, such as 212.58.244.26, and this IP address is used to send data from one device to another across the internet, in the same way that a MAC address is used on a LAN.

Websites are hosted on web servers and are accessed via their IP address. However, addresses like 212.58.244.26 are hard for humans to remember. Therefore, we use a **Uniform Resource Locator (URL)**, such as www.mywebsite.co.uk since these are much easier to remember. The URL includes the **domain name**, in this example [mywebsite.co.uk](http://www.mywebsite.co.uk).

Domain Name Server (DNS)

The **Domain Name Server (DNS)**, or domain name system, keeps a record of the IP address for each web server that is associated with a particular domain name. It is like a worldwide phone book that relates domain names to IP addresses.

When a user types the URL for a website into a browser, a request is sent to a DNS server to ask for the matching IP address. If the DNS server doesn't have a record for that domain name, it makes a request to another DNS server that it is connected to. The process continues until a DNS server is reached that can return the appropriate IP address. The DNS then sends the required IP address back to the user's browser. If the IP address cannot be found then an error message is returned.



This site can't be reached

www.mywebsite.co.uk's server IP address could not be found.

- Did you mean <http://www.mywebsite.com/>?
- Search Google for [mywebsite co uk](http://www.mywebsite.co.uk)

ERR_NAME_NOT_RESOLVED

Figure 1.3.6 Error message returned if IP address cannot be found

Once the user's browser knows the required IP address, it can make a request to the appropriate web server using its IP address.

Hosting

An internet **host** is a company that runs servers to provide different types of content via the internet. This may include file hosting, web hosting, email hosting, video hosting and game servers. Content can be accessed from any device connected to the internet.

Often, hosting companies will charge a monthly fee in order to use the facilities provided by their servers, although it is sometimes to access a limited amount of content for free. For example, a number of hosts provide a limited amount of space to store files in the cloud for free, but additional space has to be purchased.

The cloud

The cloud is a generic term that refers to storage, services and applications that are accessed via the internet rather than being stored locally on your computer, tablet or phone.

The cloud is effectively a network of servers, some that store data and others that run applications. These servers are housed in giant data centres around the world and users do not actually need to know the geographical location where their data is stored.

Examples of cloud services and applications include:

- **file storage and sharing**, for example Google Drive, Dropbox, iCloud Drive, OneDrive
- **software and applications**, for example Google Docs, Office 365, Gmail
- **processing power**, for example Amazon Web Services (AWS).

Advantages of cloud computing

One of the main advantages of cloud computing is that files and applications can be accessed from anywhere in the world with an internet connection. Cloud applications are always the latest, most up-to-date version and users do not have to update anything themselves. This reduces the need for network managers and technical support staff. The amount of storage space is flexible, and users can buy additional storage when they run out of space. All data that is stored in the cloud is regularly backed up and kept secure by the hosting company, and data can be shared easily with colleagues anywhere in the world.

Disadvantages of cloud computing

However, there are some drawbacks, most notably that an internet connection is required to access files and services. Users have little control over the security of their data and it is possible that data stored in the cloud could be targeted more easily by hackers than data stored locally. It can also be unclear who legally owns the data that is stored. Cloud providers can change their terms and prices with little notice, and ongoing fees may become expensive in the long run.

Web servers and clients

Every website is hosted on a **web server**. This is a dedicated computer on the internet that responds to HTTP and HTTPS requests by returning web pages. It is possible for anyone to set up their own web server, but people usually use a web hosting company.

The hosting company will generally charge a monthly fee to host the website in its data centres. Larger or more popular sites that consume more bandwidth will be charged more each month than a smaller site that requires less bandwidth.

Web hosting often includes domain name registration, which ensures that the chosen name for the website has not already been taken, and then it is registered on the domain name server (DNS).

In addition to storing the files, web hosting companies will usually offer service guarantees, ensuring that a website will always be available to visitors. They will also make regular backups of the data, and apply appropriate security measures to keep a website safe from cyber attacks.

Tech term

Web server A dedicated computer on the internet that stores web pages.

Client computers request files from a server. When you visit a website, the browser on your device requests the web pages from a web server and they are downloaded to your device.

Knowledge check



- 14 Explain why a computer needs an IP address to access the internet.
- 15 Describe how a domain name is used to access a website.
- 16 State what is meant by cloud computing.
- 17 Identify two services that can be accessed via the internet.
- 18 Explain two disadvantages of storing your data in the cloud.

Tech term

Network topology

The arrangement of connections in a network.

Network topologies

The way in which devices in a network are arranged and connected together is called its **topology**. Any device connected to a network is referred to as a node.

Star network topology

In a **star network** each computer or client is connected individually to a central point, usually a switch or hub.

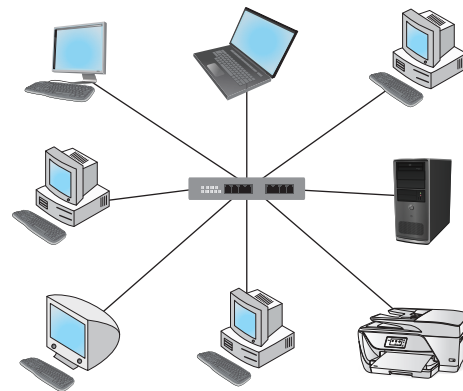


Figure 1.3.7 Star network topology

A star topology is the most common network layout, and it tends to be fast and reliable because each client has its own connection to the central node. As data is only directed to the intended computer, it helps to keep network traffic to a minimum, and in turn this reduces data collisions.

The switch can screen data packets, rejecting any that are corrupt, which can increase security on the network, and it is easy to add new devices as they simply need to be connected to the switch. If the connection to one device on the network fails, the rest of the network will be unaffected.

However, star networks require a lot of cabling, as every computer is connected individually, which can be expensive. If the central server or switch fails then so will the entire network.

Star networks tend to be found in large organisations such as schools and businesses. They are also found in home networks, especially those that are wireless, with all of the devices connecting to a central router with a built-in wireless access point.

It is important to note that star network diagrams can appear to look the same as client-server networks. However, the devices in a star network could run applications in either a client-server or peer-to-peer model.

Mesh network topology

In a **mesh network**, all of the devices are connected either directly or indirectly without the use of a central switch.

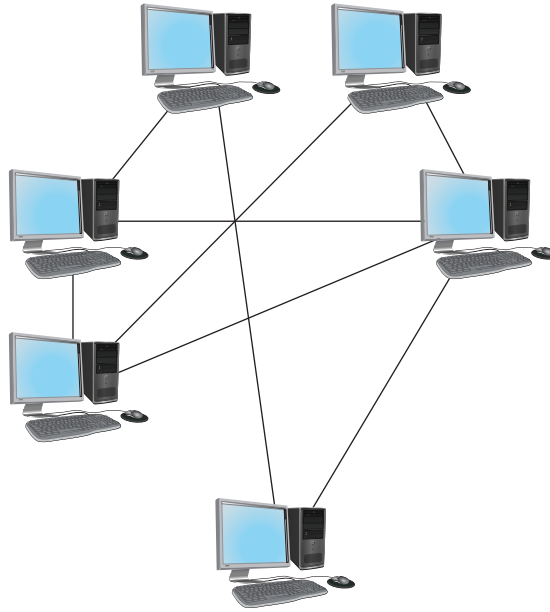


Figure 1.3.8 Mesh network topology

In a full mesh network, every device is connected directly to every other device. In a partial mesh network, direct connections only exist between some of the devices, although all devices are able to communicate with one another indirectly. This means that computers send their own data, and also relay (or pass on) data from other devices.

Mesh networks are very robust as there is no single point of failure, and so are used where the reliability of network communication is very important. For example, the military and the emergency services, such as the police and fire service, often use mesh topologies to avoid any breakdown in communications. If any component fails, data can simply be sent down a different route. Data can also be transmitted from different devices simultaneously allowing high volumes of data traffic to be handled.

Full wired mesh networks are generally too impractical and too expensive to set up, and require a lot of maintenance owing to the many connections. On the other hand, wireless mesh networks are being used increasingly as they are relatively cheap and simple to set up, with each wireless device being used to route data through the network and thus extend its range.

Mesh network diagrams can look similar to peer-to-peer networks. However, a mesh topology network can also support a client–server model where one of the devices is the server.

Knowledge check



- 19** State what is meant by a network topology.
- 20** Describe a star topology.
- 21** Explain one advantage and one disadvantage of using a mesh topology.

1.3.2 Wired and wireless networks, protocols and layers

All networks work in essentially the same way. A device prepares a data signal to send to another device, and this is transmitted along cables or wirelessly until it reaches its destination address. This transmission is controlled by **protocols**, which are essentially sets of rules that all manufacturers and devices use.

Modes of connection

Data can be transferred from one device to another via cables or wirelessly and different protocols are used for each type of transmission.

Wired

Ethernet is the traditional protocol used to connect devices in a wired LAN. It defines how data should be physically transmitted between different devices, using MAC addresses to determine which device the data should be sent to. Ethernet also defines what should happen if collisions occur on the network.

Wireless

Wi-Fi is a set of protocols that defines how network devices can communicate wirelessly using radio waves. The Wi-Fi standards determine the frequency band and channel that should be used, data transmission rates and how devices should be authenticated when they attempt to join a network.

Most Wi-Fi standards transmit data using radio waves in one of two frequency bands, either 2.4 GHz or 5 GHz. Signals transmitted on the 2.4 GHz frequency have a greater range but lower data transmission rates than those using the 5 GHz frequency.

Beyond the spec

In order to maximise the amount of data that can be sent, each frequency band is subdivided into smaller frequency ranges called channels. However, most of these channels overlap, causing significant interference between wireless networks. There are only three channels that don't overlap in the 2.4 GHz band, which reduces its effectiveness for supporting numerous Wi-Fi networks.

The 5 GHz frequency band has 24 non-overlapping channels, which enables many more wireless networks to exist side by side without interfering with one another. However, it is less able to penetrate through walls and other obstacles than the 2.4 GHz frequency.

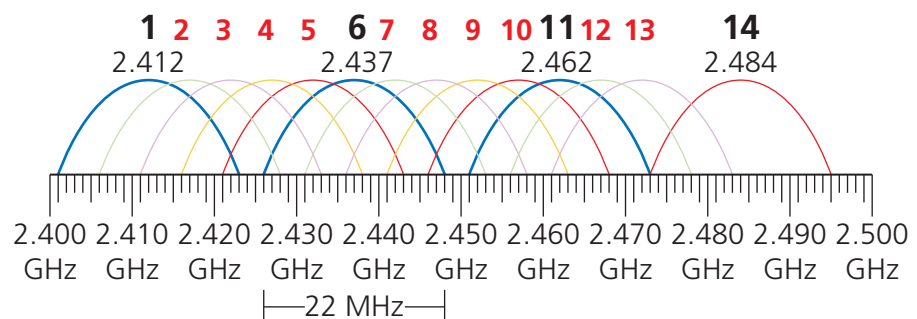


Figure 1.3.9 2.4 GHz channel diagram

Bluetooth is another form of wireless protocol that can be used over short distances using ultra high frequency (UHF) radio waves. It is very secure, and does not require a clear line of sight between the two devices. Common uses of Bluetooth are connecting Bluetooth headphones to devices, streaming audio to Bluetooth speakers and sharing files between mobile devices.

	Advantages	Disadvantages
Ethernet	<ul style="list-style-type: none"> • Stable connection. • More secure. • Faster data transfer speed. • Signal quality usually better. 	<ul style="list-style-type: none"> • More expensive to install. • More difficult to add extra devices. • Devices can only be used in one location.
Wi-Fi	<ul style="list-style-type: none"> • Cheap set-up costs. • Can connect multiple devices without the need for extra hardware. • Less impact on the physical environment as no wiring needed. • User can work in multiple locations. 	<ul style="list-style-type: none"> • More vulnerable to hacking. • Signal quality will reduce through walls and other obstructions. • Interference can occur. • The connection is not as stable. • Tends to have slower data transfer speeds.
Bluetooth	<ul style="list-style-type: none"> • Free to use if the device has Bluetooth installed. • Quick to pair devices. • Data communication is more secure than open Wi-Fi. • Avoids interference from other wireless devices. 	<ul style="list-style-type: none"> • Only works over short distances. • Lower bandwidth than Wi-Fi.

Encryption

It is very easy to intercept wireless signals and so most wireless networks are encrypted. When a device wants to connect to a secure wireless network, a Wi-Fi password needs to be entered. This wireless network password (or network security key) must be entered correctly to authenticate the device and allow it to connect to the wireless network. The Wi-Fi network then uses a unique encryption key to scramble the data that is sent over the network so that if it is intercepted, it doesn't make sense and cannot be used by others. Only devices with the correct encryption key can decode the data.

Knowledge check



- 22** Explain what is meant by a protocol.
- 23** Identify the protocol used to send messages across a wired network.
- 24** State the two common frequencies used for Wi-Fi networks.
- 25** A smart watch needs to connect to an application on a smartphone. Identify the protocol most likely to be used to transfer data from the smart watch to the phone and give two reasons why it is appropriate.

Beyond the spec

The IP address is usually a dynamic address, which means that it may change from time to time. It is possible to get a static address that is permanently assigned to a device and does not change, but these are usually reserved for ISPs and important servers.

IP addressing and MAC addressing

In order to send data across a network, an address is needed, much the same way that an address is written on the front of an envelope to send a letter. There are two types of address used on networks.

IP addresses

Every device in a network has a unique Internet Protocol (IP) address. This represents the location of the device on the network, just like your postal address indicates where your home is. Every time a device connects to the internet, it is assigned an IP address to use for that session by the **Internet Service Provider (ISP)**.

IPv4

There are currently two different IP address systems in use. IPv4 uses a 32-bit number.

The address is broken down into four 8-bit sections, each of which represents a number between 0 and 255. Each section is separated by a full stop, for example: 194.83.249.5.

In theory, IPv4 provides 2^{32} or just over 4 billion different addresses. However, this will soon not be enough to satisfy global demand.

IPv6

IPv6 is the new standard but is not yet implemented everywhere. IPv6 addresses use 128 bits, which are divided into eight 16-bit sections.

IPv6 addresses are written using hexadecimal, and as each hexadecimal number represents 4 bits, there are four numbers to each section or group. Each group is separated by a colon, for example: 2001:0db8:3c4d:0015:0000:1234:1a2f:1a21

This allows for 3.4×10^{28} , or 34 000 trillion trillion, different addresses. If we had been assigning IPv6 addresses at a rate of 1 billion per second since the earth was formed 4.5 billion years ago, we would have used up less than one trillionth of the addresses by now!

MAC addresses

A Media Access Control (MAC) address is a unique number that identifies the actual device that is connected to a network. The MAC address is part of the NIC inside the device and is assigned when the NIC is manufactured and cannot ever be changed.

A MAC address is made up of 48 bits, shown as six groups of two hexadecimal digits, for example: b8:09:8a:b8:57:17

Knowledge check

- 26** Explain the purpose of an IP address.
- 27** An IPv4 address consists of four denary numbers, each between 0 and 255. State how many bits are needed to store the IP address.
- 28** Describe the format of an IPv6 address.
- 29** Explain the purpose of a MAC address.

Standards

Computer **standards** are official definitions or rules that exist for various aspects of computing. For example, standards exist for operating systems, programming languages, communications protocols, hardware devices and data formats. Some standards are set and agreed by official organisations, and others have become established by common use.

Some common software standards include HTML for creating websites, and the MP3 file format for encoding audio files. The standards for USB connectors enable them to be used to connect hardware from different manufacturers to a wide range of different computer systems – for example, connecting a memory stick to a laptop or desktop running different operating systems.

Hardware and software standards are very important as they allow different manufacturers to make components and programs that are compatible and will work with each other. Without standards, only hardware and software made by the same company could be used together. A program or product is likely to be more popular if it is widely compatible and can be used with a range of different systems.

Knowledge check



30 Explain why standards are used in computing.

Common protocols

There are a number of different **network protocols**, or sets of rules, that define how data is transmitted between devices to ensure that devices from different manufacturers can communicate with each other.

Hypertext Transfer Protocol (HTTP)

HTTP defines how web browsers (clients) should request and web servers should deliver resources such as HTML files.

Hypertext Transfer Protocol Secure (HTTPS)

HTTPS is a secure version of HTTP that adds **Secure Socket Layer (SSL)** encryption to the communications. It is used for activities such as internet banking and online shopping.

File Transfer Protocol (FTP)

FTP is used to transfer computer files between a client and a server. It is commonly used for uploading web pages to web servers.

Post Office Protocol (POP)

POP is used to retrieve emails from an email server. When the user logs in to the email server all emails are downloaded to the device being used and are then deleted from the server. It is more or less obsolete these days.

Internet Message Access Protocol (IMAP)

IMAP is an alternative protocol for accessing email messages. However, the messages are read rather than downloaded, and can be organised into folders or flagged as important. IMAP allows multiple devices to have synchronised access to the same inbox.

Simple Mail Transfer Protocol (SMTP)

SMTP is used to send email to an email server, or between servers, for example where the sender and recipient have different email service providers.

Transmission Control Protocol/Internet Protocol (TCP/IP)

TCP is a protocol that splits the data from applications into smaller data packets that can be sent across a network.

Each packet is made up of a **header** and **payload**. The header contains the sequence number of the packet and a checksum to allow the recipient device to check that it has been sent correctly. The payload contains data from the application that needs to be sent.

The Internet Protocol (**IP**) defines how data packets should be sent between networks. An IP header is added to each packet containing the source and destination IP addresses for that packet. Routers use this information to determine whether the packet's destination is on the local network or whether the packet needs to be passed onto another network.

TCP and IP almost always work together.

Tech terms

Header Contains details about the sequence of the data packet in relation to all the other data packets, and error checking codes.

Payload Is the data that the data packet is transmitting.

Knowledge check



- 31 Identify the most appropriate protocol to be used when uploading a file from a computer to a web server.
- 32 Identify the most appropriate protocol to use when communications between a client and host need to be encrypted.
- 33 Explain the purpose of the SMTP protocol.
- 34 TCP and IP are two protocols used in network communications. State what the initials TCP and IP stand for and describe the function of each protocol.

The concept of layers

In order to simplify the network communication process, the different activities involved in sending data packets are divided into **layers**. Each layer is concerned with a different task, and the relevant protocols are assigned to each layer. The layers are organised into the order in which their rules must be applied.

For example, one layer is concerned with allowing web browsers, web servers and email clients to communicate. This requires the data to be encoded in a way that will be understood by both the client and the server, and involves protocols such as HTTP, HTTPS, FTP, POP, IMAP and SMTP.

Another layer is concerned with splitting the data into packets and adding packet information such as the packet number, size and the total number of packets. This information is needed so that the data can be reassembled correctly by the recipient device. The protocol involved at this level is TCP. The TCP layer does not need to know anything about the data that is being sent, it is just concerned with packaging it up.

Another layer adds the sender and recipient IP addresses, so that the network knows where to send the data. This is done by the Internet Protocol.

A final layer adds the MAC addresses of the sender and recipient to allow the data packet to be directed to a specific device. The data is then converted into electrical signals and sent via Ethernet or Wi-Fi using the relevant protocol.

Beyond the spec

The layer names have been provided for reference but this detail is not covered in the specification.

Tech term

Encapsulation

Wrapping data with more information as it passes through the layers.

Layer	Function of the layer	Relevant protocols
Application Layer	Encodes data to allow communications between clients and servers	HTTP/HTTPS FTP POP IMAP SMTP
Transport Layer	Splits the encoded data into packets to allow them to be sent over a network	TCP
Internet Layer	Adds the IP address of the sender and recipient to each packet	IP
Network Access Layer	Adds the MAC address of the sender and recipient, and converts the data into electrical signals	Ethernet Wi-Fi

So, as data is passed from one layer to another additional information is added by the relevant protocols. This process of wrapping the data with more information as it passes through the layers is called **encapsulation**. When the data reaches its destination, it is then unwrapped and the information read one layer at a time.

The benefits of using layers

Layering is useful as it breaks down complex problems into smaller, more manageable parts. It is a concept that is not confined to computer science, but is frequently found in engineering. For example, many cars are built using interchangeable parts, with the same components being used on various makes and models of car. Different suppliers or developers can work to improve individual components without needing to modify other parts of the system.

Each layer uses specific hardware and software to perform particular tasks. This means that each layer is self-contained, but is able to send and receive data to and from the layers above and below it.

This is helpful as it means that one layer can be developed or changed without affecting the other layers. Software and hardware manufacturers only have to be concerned with understanding how one layer works when developing new products. Therefore, different companies can develop products that will work together.

It is also helpful when trying to identify and correct networking errors and problems as the issue can usually be narrowed down to one layer of the process.

Knowledge check



- 35** TCP/IP is a set of protocols based on layers. Explain what is meant by a 'layer'.
- 36** Describe two advantages of having the protocols arranged in layers.

RECAP AND REVIEW

1.3 COMPUTER NETWORKS, CONNECTIONS AND PROTOCOLS

Important words

You will need to know and understand the following for the exam:

Local area network (LAN)
Wide area network (WAN)
Network performance
Bandwidth
Ethernet
Client-server networking
Peer-to-peer networking
Network interface controller/card (NIC)
Media Access Control (MAC) address
Transmission media
Cables
Radio waves
Wireless access point (WAP)
Service Set Identifier (SSID)
Internet
Internet Protocol (IP)
Uniform Resource Locator (URL)
Domain Name Server (DNS)
Domain name
Internet Protocol (IP) address
Web server
The Cloud
Star network topology
Mesh network topology
Protocols
Wi-Fi
Internet Service Provider (ISP)
Standards
HTTP/HTTPS
Secure Socket Layer (SSL)
FTP
POP / IMAP / SMTP
TCP/IP
Layers

1.3.1 Networks and topologies

Types of network

A **network** is formed when two or more computers and devices are linked together so that they can communicate and share resources.

A **local area network (LAN)** consists of devices connected together in a single building or site, such as a school or office, using ethernet and wireless connections.

A **wide area network (WAN)** is formed by connecting together LANs. They may be spread across a wide geographic area and use telephone lines, fibre-optic cables and even satellites. The largest WAN is the internet.

Network performance

Network performance concerns how quickly data is transmitted through a network. **Bandwidth**, measured in bits per second (bps) or megabits per second (Mbps), refers to how much data can be transmitted over a network in a given period of time.

Factors that can affect performance and potential bandwidth include:

- the number of users
- the transmission media being used
- interference and error rates.

Client-server and peer-to-peer networks

A **client-server** network has at least one main computer, the server, which controls access to the system and stores programs and files. Client machines connect to the server and request the resources they require.

All of the computers in a **peer-to-peer** network have equal status. Computers store their own programs and files, but security permissions shared across the network allow users to access the contents of another user's computer and peripheral devices attached to it.

Network hardware

Various hardware devices are needed to create a LAN:

- **Network interface controller (NIC)** to allow a device to communicate with a network. Each NIC has a unique **MAC address** that allows it to be identified on the LAN.
- **Transmission media** to allow devices to connect together:
 - **Copper wire** is used in local wired networks.
 - **Fibre-optic cables** are used to connect LANs over longer distances.
 - **Radio waves** are used to connect devices wirelessly.
- **Wireless access points (WAPs)** allow wireless devices to connect to a network.
- **Switches** allow data to be passed between devices on a network.
- **Routers** connect networks together.

The internet

The internet is a worldwide collection of computer networks linked together as a WAN.

The **domain name server (DNS)** enables websites to be accessed via their text-based address (**domain name**), for example www.hoddereducation.co.uk. The DNS links the domain name to its **Internet Protocol (IP) address**.

Websites are **hosted** on **web servers**. These are dedicated computers that are always connected to the internet and are usually provided by web hosting companies.

The Cloud is effectively a network of servers that store data and run applications. Files and applications can be accessed from any internet-connected location.

Network topologies

A network topology is the way in which devices are arranged and connected together.

In a **star network topology**, each computer is connected to a central point, which can be a switch or a server. It is the most common network layout.

In a **mesh network topology**, every device is connected to every other device, either directly or indirectly, and network traffic is shared between all devices. It is more commonly used for wireless networks.

1.3.2 Wired and wireless networks, protocols and layers

Modes of connection

Protocols are rules or standards that must be followed when data is sent between devices on a network.

Ethernet is the protocol used to connect devices in a **wired** LAN.

Wi-Fi is a set of protocols that defines how network devices can communicate **wirelessly** using radio waves. Wi-Fi can transmit at the 2.4GHz frequency, which has a greater range but lower transmission speeds compared to transmission at the 5GHz frequency.

Bluetooth is another form of **wireless** protocol, which can be used over short distances using ultra high frequency (UHF) radio waves.

IP addressing and MAC addressing

There are two types of address used in the transmission of data across a network.

Every device connecting to the internet is assigned an **IP address** to enable it to be located. The address can change each time a device re-connects.

IPv4 addresses use a 32-bit number, which is broken down into four 8-bit sections each representing a number between 0 and 255. An IPv4 address is usually written in denary with a full stop between each section.

IPv6 addresses use a 128-bit number, which is broken down into eight 16-bit sections. IPv6 addresses are written using a hexadecimal character to represent each set of 4 bits, and with each section separated by a colon.

Every network interface controller has a **MAC address** programmed into it when it is manufactured.

A MAC address is made up of 48 bits, shown as six groups of two hexadecimal digits separated by colons, and it can never be changed. MAC addresses are used to identify specific devices on a LAN.

Standards

Standards are sets of rules that exist for various areas of computing, including file formats and hardware. Some standards are set out by official organisations, and others become established through common use.

Standards enable manufacturers to build hardware components or write software that will work on different systems. Without standards you would probably only be able to use hardware and software made by the same manufacturer.

Common protocols

There are a range of standard protocols used by applications such as web browsers and email clients.

HTTP	Hypertext Transfer Protocol defines the rules to be followed by a web browser and a web server when requesting and supplying information.
HTTPS	Hypertext Transfer Protocol Secure uses SSL to encrypt communications between a web browser and a web server to ensure that they are secure.
FTP	File Transfer Protocol defines the rules for transferring files between computers.
POP	Post Office Protocol is used by a client to retrieve emails from a mail server. The emails are downloaded to the device and then deleted from the server.

IMAP	Internet Message Access Protocol allows multiple devices to have synchronised access to mail on a mail server. Messages are read rather than downloaded and can be organised and flagged.
SMTP	Simple Mail Transfer Protocol defines the rules for sending email messages from a client to a server, and then from server to server.

TCP/IP are the protocols used to transmit data across a network.

TCP	Transmission Control Protocol splits the data from applications into smaller data packets that can be sent across a network.
IP	Internet Protocol adds a header to each data packet including the source and destination IP addresses.

The concept of layers

Protocols are assigned to **layers**, each of which has a specific purpose to enable communication to take place. When data is being sent, it passes through each layer in turn and is encapsulated with more information. When a data packet is received each layer of information is read and decoded in reverse order.

Function of the layer	Relevant protocols
Allow communications between clients and servers	HTTP/HTTPS FTP POP IMAP SMTP
Splits the data into packets	TCP
Adds the IP address of the sender and recipient	IP
Adds the MAC address of the sender and recipient, and converts the data into electrical signals	Ethernet Wi-Fi

There are several advantages to organising protocols in layers:

- 1 one layer can be developed or changed without affecting other layers
- 2 enables hardware and software manufacturers to develop different products that will all work together
- 3 it is easier to identify and correct networking errors and problems.

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

NETWORK SECURITY

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.4.1 Threats to computer systems and networks

► Forms of attack

1.4.2 Identifying and preventing vulnerabilities

► Common prevention methods

Network security is about keeping networks, computers and the files, data and programs stored on them safe from attack, damage and unauthorised access.

1.4.1 Threats to computer systems and networks

Threats to networks and computer systems can come both from internal and external sources, and there are a number of different ways in which networks can be attacked. These include the use of malware, social engineering and other direct attacks on a network.

Malware

Malware is any kind of malicious program that is installed on a computer system with the intention to cause damage and disrupt its functionality or to steal information. It is usually installed without the user's knowledge.

Viruses

A virus is a computer program that is hidden within another program. The virus code is only run when the host program is executed.

Viruses can delete data or change system files so that data becomes corrupted. Some viruses fill up the hard drive so that the computer runs very slowly or even becomes unresponsive.

Viruses can insert themselves into other programs that can then be passed on. They are often spread through attachments to emails, but may also be spread through files, programs or games downloaded from a web page or by loading an infected memory stick or CD/DVD.

Worms

Worms are different to viruses as they do not need to be hosted in another program. They often create a 'back door' so that a hacker can take over an infected computer.

Worms are self-replicating, which uses up the computer's resources and causes other programs to run slowly. They usually spread by sending themselves in emails to everyone in a user's address book. They can also travel to other computers within a network, which consumes network bandwidth and affects performance.

Trojans

Trojans are programs that users are tricked into installing under the pretence that they are legitimate and useful.

Some Trojans are just annoying, changing the desktop layout and adding new icons, but they can also delete files and use back doors to send screenshots and key presses to a hacker's computer, allowing them to access your personal information.

Ransomware

Ransomware is malware that interferes with a user's operation of a computer unless a sum of money (ransom) is paid.

It encrypts the files on an infected computer and only decrypts them once payment has been made. Sometimes the malware doesn't actually encrypt anything but still scares users into thinking that it has so that they hand over payment. Even when payment is made, there is no guarantee that the files will be decrypted, and often they are not.

Spyware

Spyware is malware that comes packaged with other software such as free software that a user downloads.

It gathers information about a user and sends it to the criminal. It includes programs such as keyloggers that record all the user's keystrokes to obtain passwords and other login details.

Pharming

Pharming is a form of attack where users are directed to a fake website.

There are two ways in which this might happen. Malware installed on a computer can send lookup requests to a rogue **DNS server** rather than their ISP's genuine DNS server, or malware can infect the DNS server itself (known as DNS spoofing) so that everyone is directed to the bogus site.

The rogue or 'poisoned' DNS server responds with the IP address for a server hosting a fake copy of the website being visited. When users then enter their login details at the fake site these are captured so that they can be used by hackers. Often, the fake website then redirects the user onto the real version of the website and logs them in using the details they have just provided. This means that they are unaware they have been attacked.

Social engineering

The weakest point of any computer system is the people that use it. **Social engineering** is a form of security attack that involves tricking or manipulating people into giving away critical information or access details. Fear is often used to put people off guard and make them more likely to comply with instructions.

Phishing

Phishing uses fake emails and websites to trick people into giving away their sensitive data and information. Emails usually claim or appear to be from a bank or building society, an e-commerce site or an email provider.

They often ask the user to verify their account by clicking on a link or taking some other similar action. Links often then take the user to a fake version of the website where login details, and possibly credit and debit card details, can be captured.

Tech term

DNS server Relates domain names to IP addresses.

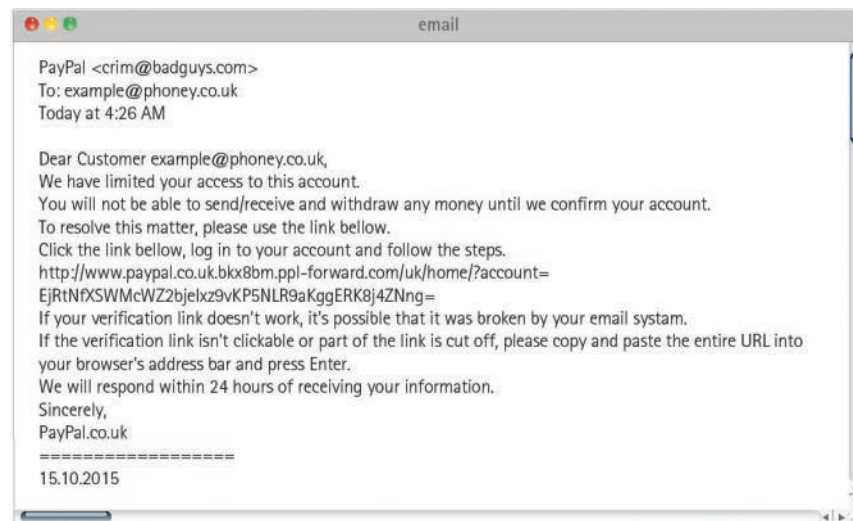


Figure 1.4.1 Example of a phishing email

Pretexting

Pretexting, also known as blagging, is often done by phone but can also be carried out face to face. Here, the criminal invents a scenario to persuade the victim to divulge information that they might not do otherwise. Often, they will pretend to be from an official organisation such as a bank, insurance company or the police, or to be another employee of the company or a network administrator.

Tech term

PIN Personal identification number; used for credit and debit cards.

Shouldering

Shouldering, or shoulder surfing, involves finding out login details, passwords and **PINs** by watching people enter them.

This could happen by looking over someone's shoulder as they enter their PIN at a cashpoint or checkout, or even by using recording equipment.

Brute force attacks

A **brute force attack** is where a hacker attempts to crack a password by systematically trying different combinations of letters and numbers until the correct one is found.

Automated software is generally used to try millions of different passwords every second. Often brute force attacks begin with a dictionary attack, where lists of previously cracked passwords from other sites are tried before attempting every possible combination of characters.

Success is based on the amount of computing power available rather than any specialist techniques or algorithms.

Denial of service (DoS) attacks

Denial of service (DoS) attacks are designed to bring down servers or websites by flooding them with superfluous bogus requests such as repeated attempts to login. This uses up internet bandwidth and prevents the servers from responding to legitimate requests.

A **distributed denial of service (DDoS)** attack uses a large number of compromised machines that have been infected with malware. These 'zombie' computers can be used to form a **botnet** so that a huge number of login requests can all be sent at the same time.

DoS or DDoS attacks may be used to extort money from a firm to stop the attacks, or may be used by **hacktivists** to punish organisations that they deem to be unethical.

Tech terms

Botnet A collection of computers infected by malware and controlled by hackers.

Hacktivists Individuals who misuse computers for a socially or politically motivated reason.

Data interception and theft

Data is a very valuable commodity. Personal data can be used to access bank accounts or in identity theft, while the financial data or trade secrets of a company can be exploited by others to gain a competitive advantage. **Data interception and theft** can occur in several ways.

Tech terms

Data packet A small chunk of data that is transmitted across a network.

Network switch The hardware that allows multiple devices to connect together to form a network.

Packet sniffing

Packet sniffing involves intercepting data using packet analysers as it is being transmitted across a network. These analysers read and display the contents of each **data packet**, enabling sensitive data such as login names, passwords and credit card details to be stolen.

The software used can manipulate the **network switch** so that all packets are sent to the sniffing device, which then sends them on to their intended destination once they have been intercepted and read. This means that no one is aware that any interception has taken place.

Packet sniffing can be carried out with relative ease on wireless networks as the signals can be accessed from distances of up to 300 metres, making it easy for the perpetrators to avoid detection.

Man-in-the-middle (MITM) attacks

A MITM attack involves intercepting a device's connection to the internet. Often this is achieved by luring users into using a fake Wi-Fi hotspot.

The operator of the fake Wi-Fi network can then sniff all of the packets to gain personal information and to see which websites are being visited. This method also allows phone numbers to be captured, which can then be used in further attacks.

Tech term

SQL A language specifically designed for interactions with databases.

The concept of SQL injection

Many websites use databases to store the details of users. **Structured Query Language (SQL)** is used to search these databases, for example to check that a user's login details and password are correct.

SQL injection can be used to bypass security and circumvent the need to enter legitimate login credentials, thus allowing hackers to gain access to the database. From here they can steal valuable data such as names, addresses and bank details.

SQL statements often operate on data input into fields on online forms, and so inputting a syntactically valid SQL expression instead of a username can cause the commands to be executed.

Beyond the spec

SQL queries are covered in Chapter 2.2 and the information covered there **is** needed for the examinations.

However, here is an example of an SQL statement on a server script to select a user with a given user id:

```
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserID;
```

If a user enters **9 or 1=1** into the UserID field, the SQL statement will look like:

```
SELECT * FROM Users WHERE UserId = 9 OR 1=1
```

(The 9 could be any value.)

This will cause all rows from the 'Users' table to be returned because * is a wildcard that means 'everything' and 1=1 is always TRUE so the statement is valid.

Knowledge check



- 1 Match the type of malware to the description of how it is spread:
Virus Malware disguised as legitimate software
Spyware Malware that comes packaged with other software
Worm Malware that is spread through infected files
Trojan Malware that is self-replicating and spreads via email
- 2 Explain what is meant by pharming.
- 3 Pretexting is a form of social engineering. Describe how it is used to gain personal data.
- 4 Explain what is meant by a brute force attack.
- 5 Identify and describe two ways in which data transmissions on a network may be intercepted and stolen.
- 6 Describe how SQL injection can be used to access a database.

1.4.2 Identifying and preventing vulnerabilities

There are a number of different methods that can be used to limit threats and protect networks and computer systems from unauthorised access.

Penetration testing

Penetration testing is used to test a system or network in order to identify vulnerabilities in its security that an attacker could exploit.

Testers take on the role of hackers and try to gain unauthorised access in a controlled attack. Good penetration testing also assesses the security awareness of users to see how likely they are to fall for social engineering ploys, and demonstrates the effectiveness of network security policies. It may also include checking the organisation's ability to respond to security incidents and to recover any data that has been lost or compromised following an attack.

Anti-malware software

Anti-malware software is designed to detect and remove malware. It protects systems in several ways:

- It performs real-time scans of incoming network traffic to detect whether they have been infected with a virus.
- It performs periodic scans of the whole system looking for malicious applications.
- If a virus or other malware is detected or manages to install itself it is quarantined. This prevents it from running and allows users to attempt to clean or remove it.

Tech terms

Heuristic analysis A method of detecting viruses by examining code for suspicious properties.

Sandbox A test environment that isolates untested code from production environments.

Beyond the spec

- ◆ Anti-malware software often uses **heuristic analysis** which is designed to detect previously unknown viruses as well as new variants of viruses already in circulation. It involves de-compiling a suspect program to examine the source code and compare it to that of viruses already in the database. The code is flagged as a probable threat if a particular percentage of the code matches anything already known about.
- ◆ Suspect code can also be tested in a **sandbox** to simulate what would happen if the program was allowed to run. The code is examined to look for suspicious actions such as overwriting code and self-replication.

Anti-malware software needs to be able to get regular updates from the internet as it relies on using up-to-date definitions of the viruses and malware that are known about and how to identify them by their code.

Firewalls

A **firewall** is designed to prevent unauthorised access to a network, and can be provided by either a hardware device or a piece of software.

Firewalls inspect and filter incoming and outgoing data packets to ensure that they meet the security criteria that have been configured. If a packet does not meet the security criteria it is not allowed through. Criteria may include:

- the **MAC address** of the computer sending the data
- the type of data been sent or received, for example .exe files
- **IP address** filtering to prevent users and programs from accessing specific internet sites.

Firewalls protect a network or computer from attempts by hackers to break in from the outside. However, they also protect against attempts by malware to send data packets out of the network from infected machines.

User access levels

Users of a network are often arranged into user groups. Each group has different user **access rights** that determine what software, hardware and files they are permitted to access. For example, on a school network staff may be able to access certain folders that pupils cannot.

User access levels are an important way of avoiding attacks caused by the careless actions of users. Preventing normal users from installing new software means that malware cannot be installed even if a user is lured into clicking on a suspicious link. In addition, access to confidential information can be limited to only those who need it, which helps to protect against insider attacks.

Passwords

Passwords help to prevent unauthorised access to a network or computer. However, they are only effective if they remain secret and are not easy to crack by brute force attacks. They should not be used for multiple accounts and should never be written down.

Long passwords that use a combination of letters, numbers and symbols will take longer to guess in a brute force attack. As hackers can potentially find out information about users, such as their dates of birth and names of family members, these should never be used in passwords.

Tech terms

Biometric authentication A security process that relies on the unique biological characteristics of an individual to verify that they are who they say they are.

Key A string of randomly generated characters (similar to passwords except they are generated by algorithms rather than users).

Two-factor authentication (2FA) can be used to add an extra layer of security to the use of passwords. In addition to providing a username and password, the user has to enter a code that only they have access to. Usually this is a code that has been sent to another device they have possession of, such as a mobile phone.

Passwords are gradually being replaced by **biometric authentication** methods such as fingerprint readers and facial recognition. Biometric factors can also be used as the second step in two-factor authentication.

Encryption

Encrypting data means that it cannot be read by unauthorised persons, even if they manage to access the data. Encrypted data requires the correct **key** to be used in order to be decrypted. Wi-Fi networks should use secure encryption, such as WPA2, to ensure that network packets cannot be intercepted and read. Files on a network or removable storage device can also be encrypted so that they cannot be read if someone manages to gain access to them.

Physical security

Physical security is about protecting hardware, software, networks and data from physical actions that could cause harm. These include:

- burglary and theft
- fire, flood and natural disasters.

Security measures might include keeping servers in a locked room that can only be accessed by network managers, and ensuring that backups are kept off-site in a different secure location.

Threats and how they can be prevented

A range of different prevention methods can be used to guard against each different type of threat.

Threat	Prevention methods
Malware	<ul style="list-style-type: none">• Install anti-virus and anti-spyware software.• Ensure that the operating system is up to date.• Implement user access levels to prevent standard users from being able to install software.• Only download programs from trusted websites.• Educate users about the risks of opening emails and attachments from unknown sources.
Social engineering	<ul style="list-style-type: none">• Educate users so that they are aware of the tactics of criminals and can guard against them.• Ensure that network and security policies are followed.
Brute force attacks	<ul style="list-style-type: none">• Use long passwords that include special characters.• Use complex passphrases rather than single words.• Use a password manager.• Networks and websites can limit the number of login attempts allowed.• Networks and websites can use two-factor authentication.
Denial of service attacks	<ul style="list-style-type: none">• Install a firewall to reject packets that originate from the same source or that have identical contents.• Configure a firewall to restrict the number of packets that can be accepted within a particular time frame.
SQL injection	<ul style="list-style-type: none">• Use input validation to set password and username rules that don't permit characters which can be used in SQL injection attacks.• Use input sanitisation to remove special characters and SQL command words from an input before processing it.
Data interception and theft	<ul style="list-style-type: none">• Use strong encryption, especially on Wi-Fi networks; do not use unencrypted free public Wi-Fi networks• Use MAC address authentication on networks so that only known devices can connect.• Ensure that websites are using HTTPS connections so that if data is intercepted it cannot be read.

Knowledge check



- 7 Explain how anti-malware software helps to protect a system.
- 8 Identify two forms of attack that the use of a firewall could help to prevent.
- 9 Explain why encryption should be used on wireless networks.

RECAP AND REVIEW

1.4 NETWORK SECURITY

Important words

You will need to know and understand the following for the exam:

Malware
Social engineering
Brute force attack
Denial of service (DoS)
Data interception and theft
SQL injection
Penetration testing
Anti-malware software
Firewall
Access rights
Passwords
Encrypting
Physical security

1.4.1 Threats to computer systems and networks security

There are various ways in which a computer system or network can be attacked including:

- **malware**
- **social engineering**
- **brute force attacks**
- **denial of service attacks**
- **data interception**
- **SQL injection.**

Some attacks disrupt the functionality of the computer or network, while others are designed to gather sensitive personal information.

Data is a very valuable commodity. Stolen usernames and passwords allow criminals to access bank accounts and gain private information, which can then be used to commit crimes against the victims without their knowledge.

There are several different types of malware:

Type of malware	What it does
Virus	Is hidden inside, or attached to, another file or program. Deletes or corrupts data and files.
Worm	Is self-replicating. Slows the computer and creates back doors.
Trojan	Looks like legitimate software. Slows the computer and creates back doors.
Ransomware	Denies a user access to their system until a ransom is paid.
Spyware	Is often bundled with free software. Logs activity and keystrokes and sends these back to a criminal.
Pharming	Redirects a user to a spoof website without their knowledge by modifying DNS entries.

Social engineering involves tricking or manipulating people into giving away critical information or access details. Methods include **phishing**, **pretexting** and **shouldering**.

Brute force attacks involve the use of automated software to crack passwords in order to gain access to a system.

Denial of service attacks occur when a server is flooded with bogus requests in order to bring it down.

Network communications can be intercepted on their way to their destinations. **Data interception and theft** can occur through packet sniffing or the use of fake Wi-Fi hotspots.

SQL injection uses SQL commands entered into input fields on online forms to gain access to databases.

1.4.2 Identifying and preventing vulnerabilities

Penetration testing is used to identify weaknesses and vulnerabilities in computer systems so that they can be addressed.

User-based security includes:

- the use of **strong passwords**
- the use of **user access rights**
- preventing the use of removable storage devices.

Network-based security includes the use of:

- **firewalls** to control the transmission of data in and out of the network and to manage which devices can be connected
- **anti-malware software** to detect and eliminate malicious software
- **encryption** on wireless networks to protect files and data
- regular **backups** to avoid data loss.

Physical security involves protecting hardware, software, networks and data from physical actions that could cause harm, such as theft, fire, flood and natural disasters.

Extra resources

A free set of practice questions accompanies this section and is available online at: www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

SYSTEMS SOFTWARE

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.5.1 Operating systems

- The purpose and functionality of operating systems

1.5.2 Utility software

- The purpose and functionality of utility software
- Utility software: encryption, defragmentation and compression

There are two main categories of software used on computers: systems software and application software. **Systems software** controls the hardware inside the computer and provides an interface for users to interact with it and is comprised of the operating system and utility software. **Application software** is the end-user programs that are designed to perform specific tasks, such as word processing, photo editing, or used for entertainment such as playing games or watching videos.

1.5.1 Operating systems

Operating systems are found in almost all computing devices, from video game consoles to mobile phones, tablets to desktops, data servers to supercomputers. They control the general operation of the system or device and provide a way for users to interact and run programs.

Well-known operating systems include:

- Windows
- macOS
- Linux
- Ubuntu
- Android
- iOS.

The purpose and functionality of operating systems

The operating system (OS) manages the hardware in a computer and provides an environment for applications to run. It is essential to the function of any computer.

The functions controlled by the OS include:

- a **user interface** to allow the user to interact with the system
- **memory management** to control the use of the RAM and to share processor time between different programs and processes
- **peripheral management** to control peripheral devices using drivers
- **user management** to control who can access the computer and what resources they can use
- **file management** to allow users to organise their work into folders and subfolders.

User interface

The user interface allows the user to interact with the computer system. Most modern OSs provide a **graphical user interface (GUI)**, which uses small icons to represent the files, devices and applications and allows the user to interact by clicking, dragging or touching them.

Most systems use the **Windows, Icons, Menus and Pointers (WIMP)** interface. Here applications are shown in windows, with programs and files represented by icons. Menus allow you to access features and other options, and interaction is provided by moving a pointer (or cursor) which is controlled by a mouse or touch pad.

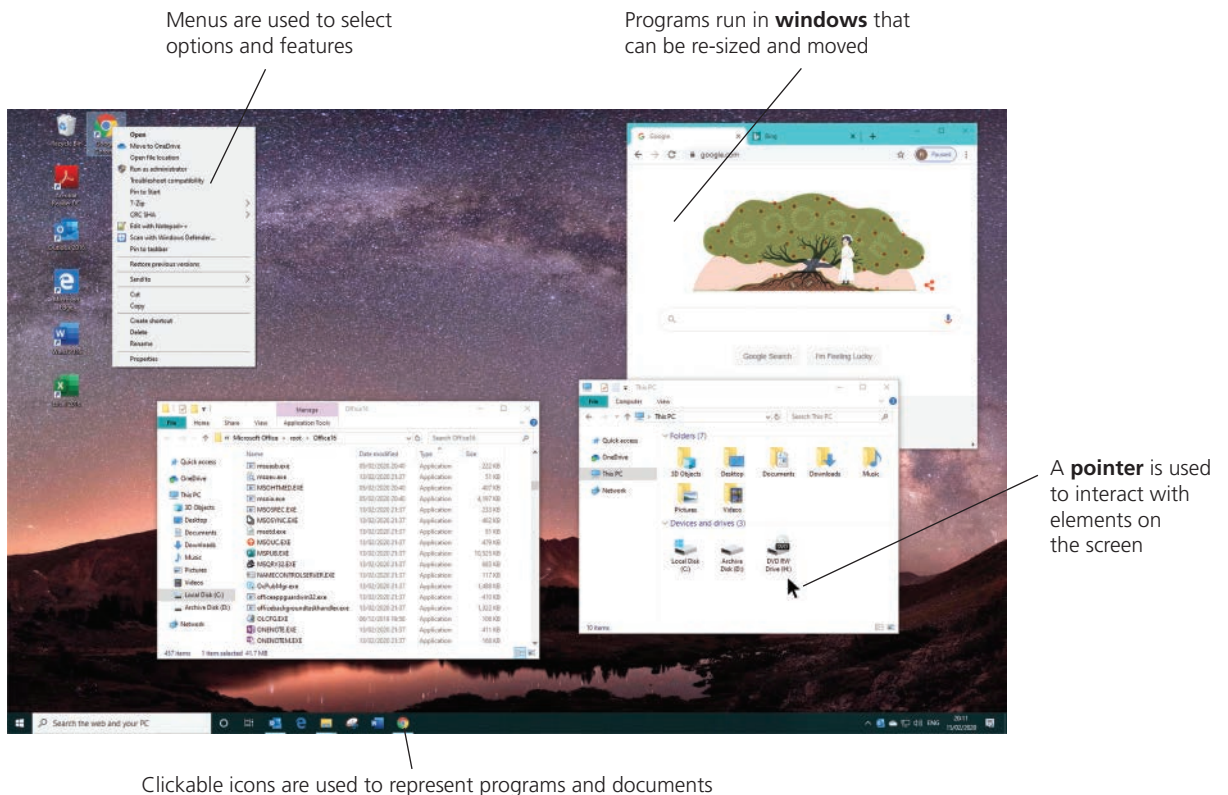


Figure 1.5.1 A graphical user interface

[Google and the Google logo are registered trademarks of Google LLC, used with permission]

Before GUIs were developed, users had to use the **command line interface (CLI)**, which involved typing in all of the commands via a keyboard. This included the commands to load programs, copy, delete and move files, and print. This meant that users had to learn all of the commands in order to be able to type them in correctly and use them.

It is still possible to use a CLI on modern operating systems by running the Command Prompt in Windows, or opening the Terminal on Mac or Linux systems. They tend to be used by network managers who want to do things that are not possible using the standard GUI. Commonly used sequences of commands can be grouped together into **batch files** (Windows) or **shell scripts** (Linux) so that a single command can cause multiple actions.

Increasingly, we are able to control devices by speaking to them. Natural language interfaces include Siri on macOS and Cortana on Windows, and these allow the user to control some software, as well as dictating text rather than needing to type it. Alexa integrates with some streaming cloud-based services as well as devices around the home, allowing you to control them by voice.

Memory management and multitasking

One of the key roles of the operating system is to manage the available memory. Any program running on a system needs to be loaded into RAM, and the memory manager controls whereabouts the program and its data will go, allocating memory blocks according to the needs of the program. The CPU program counter is then set to the memory location of the first instruction in the program. When a program finishes or the data is no longer needed the

Tech terms

Batch files, shell scripts

Files that execute a series of command line interface instructions.

memory manager frees up the space for reuse. Applications can only access the memory that has been allocated to them, and cannot access memory that is reserved for other programs.

The OS uses **buffering** to set aside memory for the temporary storage of data. For example, a process may output data which can be temporarily stored in the buffer so that the next process can be executed.

It is normal for several applications to be loaded into RAM and be running simultaneously and this is called **multitasking**. The activities that the programs perform are called processes. Although the programs appear to be running concurrently, only one process can be run by the CPU at any one time so the OS shares out processor time between each application. The process manager prioritises all of the tasks and allocates CPU time for each process to be completed.

Peripheral management and drivers

Peripherals are any hardware components that are not part of the CPU. These include input devices such as a keyboard, mouse, trackpad, microphone, webcam and scanner, and output devices such as a monitor, printer and speakers. Storage devices such as external hard drives and USB memory sticks are also peripheral devices.

All peripheral devices are controlled by the OS using programs called **device drivers**. These act as a translator to allow the CPU and the devices to communicate correctly. Without drivers the computer would not be able to send and receive data correctly between hardware devices. Each device has its own driver and the correct driver has to be installed for the specific model and operating system being used.

The device driver handles the translation of requests between a device and computer, and defines where incoming or outgoing data will be stored. When data is received from an input device, or sent to an output device, it is usually temporarily stored in a buffer. For example, a document being sent to a printer may be held in a print buffer from where the printer can draw the characters at its own pace. This is known as spooling. The device driver will also wake up a device when it is needed and put it back to sleep when it is not.

User management

The ability for a user to access a system is also controlled by the operating system. The OS can allow individual users to be created or deleted. Access to the system is usually based on a username and password that match those of known users. The OS will then allocate permissions for access to files and folders, applications, settings and other features based on the user's **access rights**.

Control of access rights is often used to ensure that a standard user cannot install applications or change a master password. It may also be used to restrict access to the internet.

File management

The operating system also controls all of the different files on the system, such as documents and executable programs. It creates a folder and file structure that makes it easier for users to organise and find data in a systematic way. It allows users to create, name, move and save files within this structure. The file manager then keeps track of where the specific files and folders are physically located in secondary storage. It also determines the file type, so that it can either be executed or sent to an application to be opened.

On networks and shared computers, file management is also used to control **file permissions**. These control who can see or open a file, write to a file or edit it, and who can delete a file. This helps to keep a system secure.

Beyond the spec

A File Allocation Table (FAT) is used to keep track of where files are stored, by maintaining a table that provides a map of the clusters of memory that a file has been saved in.

Knowledge check



- 1 State two functions of an operating system.
- 2 Explain how a graphical user interface allows a user to interact with a computer.
- 3 Identify another type of user interface found on operating systems.
- 4 Explain how an operating system manages memory in a computer system.
- 5 Explain what is meant by device drivers.
- 6 Explain why file management is used.

1.5.2 Utility software

Utility software is a collection of programs, each of which does a specific housekeeping task to help maintain a computer system. Most computers have utility software installed alongside the operating system.

The purpose and function of utility software

Utility software is not essential for the computer to work, but it helps to configure the system, analyse its performance and make changes to ensure that it is running efficiently.

Types of utility software include:

- encryption
- defragmentation
- data compression.

Tech term

Ciphertext Encrypted text.

Encryption software

Encryption is the scrambling of data into a form that cannot be understood if it is accessed by unauthorised users. It is used to protect data from unauthorised access. The encryption process uses an algorithm and a key to transform the plaintext into **ciphertext**. The same algorithm and key are needed to decode the information.

Modern operating systems have built-in encryption utilities that enable the user to encrypt specific files or entire drives – for example BitLocker on Windows and File Vault on macOS. Files on systems like these are automatically decrypted when they are accessed by an authorised user.

It is especially important that sensitive data copied onto laptops and portable storage devices, such as flash memory drives and external hard drives, is encrypted as there is more likelihood that these devices could be lost or stolen. This usually requires the use of separate encryption software which can be used to encrypt specific files with a chosen key. It is then possible for someone else to decrypt the files providing they have the same encryption software and the correct key.

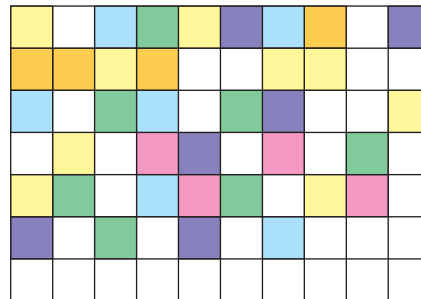
Defragmentation

When data is stored on a magnetic hard disk drive, it is saved to different areas of the disk depending on where there is space available. If the data file is larger than the free space available in one part of the disk, it is split into separate blocks of data that are saved in different places. This is called fragmentation.

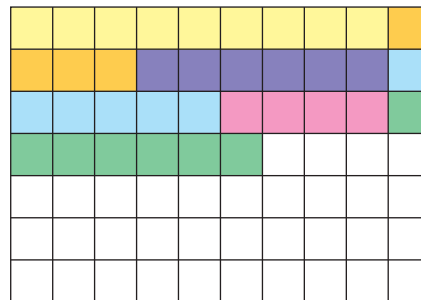
Over time the contents of a hard disk become increasingly fragmented and this begins to affect performance. This is because hard disk drives are mechanical devices and the read/write head has to move to the correct physical location on a disk to read its contents (see section 1.2.2). When the data is fragmented, the disk needs to be accessed more frequently to read all of the data.

Defragmentation is the process of organising and moving the data so that data belonging to the same file is grouped together in adjacent blocks and stored in one location on the hard disk drive, which makes it quicker to access. Defragmentation also groups all of the free disk space together so that new files can be stored in one place. This improves the performance of the computer.

Fragmented data



Defragmented data



Over time, the contents on a hard drive becomes fragmented.

Defragmentation reorganises and moves the data blocks.

Data that belongs to the same file is grouped together in adjacent blocks.

Free space is grouped together at the end of the disk.

Figure 1.5.2 The process of defragmentation

It is important to note that there is no need to perform defragmentation on solid-state drives. This is because SSDs do not have any moving parts and so having data split up around the memory locations does not affect the read/write times. In fact, because SSDs have a limited number of read/write cycles, performing defragmentation is likely to reduce the life of an SSD.

Data compression

Data compression uses algorithms to reduce the size of files so that they take up less storage space. There are two types of compression, which are known as **lossy** and **lossless**.

Lossy compression reduces the file size by deleting some of the data. It can be used on items such as photographs where the loss in detail will not be noticeable. As the data is completely removed from the file it can never be restored back to its original form.

Lossless compression must be used if a file, such as a text file or document, needs to be restored exactly back to its original form. Lossless compression uses algorithms to look for patterns and repeated elements in a file. These are then stored in a dictionary from where they can be referenced by a number. This enables data to be compressed and then restored to its original form.

Compressed files can be transmitted much more quickly over the internet as the file size is smaller and therefore requires less bandwidth. Compression can also be useful when emailing files as attachments as there is usually a limit to the size of a file which can be transmitted. Compressing, or zipping, the file can reduce it to an acceptable size.

For more on file compression see section 1.2.5.

Knowledge check



- 7** Define what is meant by utility software.
- 8** Give three examples of utility software.
- 9** Explain why encryption software is used.
- 10** State one problem caused by a fragmented disk.
- 11** Identify two situations where compression software might be used.

RECAP AND REVIEW

1.5 SYSTEMS SOFTWARE

Important words

You will need to know and understand the following for the exam:

Systems software
Application software
Operating systems
User interface
Memory management
Peripheral management
User management
File management
Graphical user interface (GUI)
Windows, Icons, Menus and Pointers (WIMP)
Command line interface (CLI)
Buffering
Multitasking
Device drivers
Access rights
File permissions
Defragmentation
Lossy compression
Lossless compression

Extra resources

A free set of practice questions accompanies this section and is available online at:

www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

1.5.1 Operating systems

The purpose and functionality of operating systems

The operating system (OS) manages the hardware in a computer and provides an environment for applications to run. The OS controls different aspects of the running of the computer or device.

User interface

This allows the user to interact with the computer system. Most modern OSs provide a **graphical user interface (GUI)** which uses windows, icons and menus that can be controlled by a pointer.

Before GUIs were developed users had to use the **command line interface (CLI)** into which commands are typed, and these still tend to be used by network managers who want to do things that are not possible using the standard GUI.

Increasingly we are able to use our voices to control aspects of computer devices.

Memory management and multitasking

The OS manages the available memory in the RAM. The validity of a request is verified and then blocks of RAM are allocated to a program and its data. When a program finishes, or the data is no longer needed, the memory manager frees up the space for reuse. The OS uses **buffering** to set aside memory for the temporary storage of data so that it can be used by different hardware devices.

Several programs running at the same time is known as **multitasking**. Only one process can be completed at a time and so the process manager prioritises all of the tasks and allocates CPU time for each process to be completed. As this happens very quickly it appears as if the programs are running simultaneously.

Peripheral device management and drivers

All peripheral hardware devices are controlled by the OS using programs called **device drivers**. These act as a translator to allow the CPU and the devices to communicate correctly.

User management

User management controls who can access a system. Accounts can be created or deleted for individual users, with different **access rights** controlling the files and software they can access. This helps to keep the system secure.

File management

The OS creates a folder and file structure that makes it easier for users to organise and find data in a systematic way. It then keeps track of the location of files on disks and other storage devices so that they can be retrieved when needed.

1.5.2 Utility software

The purpose and function of utility software

Utility software is a collection of programs each of which does a specific housekeeping task to help maintain a computer system.

Encryption software

This is used to encode data so that it cannot be understood if it is intercepted by unauthorised users. The encryption process uses an algorithm and key to transform the plaintext into ciphertext. The same software and key are needed to decrypt the data.

It is possible to encrypt specific files or whole drives depending on what is needed. In addition, data on portable and removable storage devices should be encrypted.

Defragmentation

Over time, data on a hard drive becomes fragmented. Parts of a file are saved to different areas of the disk where there is free space. This slows down the computer as more disk accesses are needed to read all of a file. **Defragmentation** software reorganises the files, putting all of the free space together and all of the parts of the same file together. This improves performance.

Data compression software

Data compression software uses algorithms to reduce the size of files so that they take up less storage space. There are two types of compression known as **lossy** and **lossless**. Compression is useful in freeing up storage space. It is also useful when transferring files across a network as smaller files require less bandwidth.

ETHICAL, LEGAL, CULTURAL AND ENVIRONMENTAL IMPACTS OF DIGITAL TECHNOLOGY

CHAPTER INTRODUCTION

In this chapter you will learn about:

1.6.1 Ethical, legal, cultural and environmental impact

- Impacts of digital technology on wider society
- Legislation relevant to computer science

1.6.1 Ethical, legal, cultural and environmental impact

The impacts of digital technology on wider society

The widespread use of computer technology in all aspects of daily life has brought many benefits for the individual and society. Computer systems are involved in most human activities. The list of how computer systems affect us is endless but a few examples will show something of the extent:

Safety	Guiding aircraft, controlling trains, supporting signalling systems, monitoring patient body signs
Travel	Smartphone apps, GPS, train and bus timetables, flight bookings
Business	Orders, stock control, payroll
Retail	Online ordering of more or less anything, logistic systems that control the delivery of goods
Entertainment	DVDs, Blu-Ray, online film and television services for home and mobile
Communication	Email, chat, social networks, business transactions, smartphones
Education	Virtual Learning Environments, online exam marking, distance learning, unlimited sources of information on the web
Politics and government	Online campaigns, voting (in some countries), payment of taxes, passport and visa applications
Science	Number crunching, simulations, visualisations, distributed processing of data for research (e.g. research into genes/DNA)

However, alongside these benefits the widespread use of computer technology has also generated various problems, from computer crime to issues with the freedom and privacy of the individual. The fact that we depend upon computer technology in so many aspects of our daily lives makes us all vulnerable to at least some of these problems.

Ethical issues

Ethics refer to what is right and wrong. While most people probably agree about most things that constitute ethical behaviour, there is not always a definitive definition. The ethical use of computer technology is about acting in a responsible way and ensuring that no harm is caused to others. Ethics are not the same as legalities – something may be immoral but not illegal. However, a good legal system will be based on an ethical approach.

Ethics are to some extent a personal thing but there are codes of ethics laid down by various organisations including associations of computing professionals. BCS (British Computer Society) has some fairly typical ethical standards that it recommends computing professionals should adhere to.

They include not undertaking work that is beyond the individual's capability, not bringing the profession into disrepute, avoiding injuring others and not taking bribes.

Another organisation, the Computer Ethics Institute, lists ten commandments for computer ethics:

- 1 Thou shalt not use a computer to harm other people.
- 2 Thou shalt not interfere with other people's computer work.
- 3 Thou shalt not snoop around in other people's computer files.
- 4 Thou shalt not use a computer to steal.
- 5 Thou shalt not use a computer to bear false witness.
- 6 Thou shalt not copy or use proprietary software for which you have not paid.
- 7 Thou shalt not use other people's computer resources without authorisation or proper compensation.
- 8 Thou shalt not appropriate other people's intellectual output.
- 9 Thou shalt think about the social consequences of the program you are writing or the system you are designing.
- 10 Thou shalt always use a computer in ways that ensure consideration and respect for your fellow humans.

Stakeholders are groups of people who have an interest in, or might be affected by, a particular use of technology. Different stakeholders may have different views about the uses of technology. This will have an impact on whether they consider that use to be ethical.

Here are some examples of uses of digital technology and the positive and negative ways in which they might be viewed by different stakeholders – and, therefore, whether or not they would be considered ethical.

The use of CCTV cameras in public places to monitor behaviour:

- + It is a form of security that can be used to solve crimes and keep us safe on the streets.
- + Live facial recognition can be used to identify and locate criminals.
- It is a 'big brother' approach, constantly tracking what we do and where we go.
- It is an invasion of privacy.

The ability to track mobile phone signals:

- + It is useful for finding friends.
- + It can provide valuable evidence for the police.
- When we take a picture the location and time are recorded.
- There are apps that track our location at all times.

Logging the activities and the use of computers and phones in the workplace:

- + It enables the employer to monitor the effectiveness of an employee.
- + It provides insight into working patterns.
- + It can reduce the risk of employees using the organisation's facilities for illegal or unacceptable purposes (e.g. monitoring social media posts).

- It is an intrusion into the lives of the employee.
- It results in constant pressure to perform at work.

Using social media to make friends and keep in contact with other people:

- + You can share recent activities to keep people up to date with what you like and what you are doing.
- Trolling and cyber bullying are attempts to cause someone distress by posting insulting or threatening messages and can be very unpleasant.
- Unguarded comments or inappropriate images posted on social media are available to a wide audience and may be seen by family, friends, work colleagues or employers.

Knowledge check



- 1 Identify two ways that individuals might be monitored in their daily life.
- 2 What issues may result from unwise posts on a social media site?

Legal issues

The widespread use of computers has had a **legal** impact and new laws have had to be constructed. Computers are associated with a wide range of existing and new criminal activities including:

- unauthorised access to data and computer systems for the purpose of theft or damage
- identity theft
- software piracy
- fraud
- harassment, such as trolling.

Various laws describe the rules that computer users must obey. These laws are designed to prevent the misuse of computer systems, and they can vary from country to country. It is a criminal offence to not follow laws. Laws in the UK will be considered in greater detail later on in this chapter.

The internet does not belong to anyone nor is it based in one country, so it is not governed by one country's set of laws. There is a lot of debate about whether the internet should be policed – and even whether it could be.

Cultural issues

Culture refers to the ideas, behaviour, beliefs and values of a group of people. The use of computers has had a widespread impact on culture and society and it is now almost impossible to imagine life without them. In particular they have had a huge impact on social interaction and how we access information and entertainment.

- Mobile devices are used by most people and have changed the way that people interact with each other.
- Almost every industry has been affected or disrupted by digital technology. For instance, streaming services provide an increasing proportion of our entertainment including television, gaming and music. These industries are completely different now compared to 20 years ago.
- The use of contactless payment methods has increased dramatically, reducing the need for cash.
- Increased use of online services and retailing is having a dramatic effect on city centre high streets, with many bank branches and retail outlets closing.

- There are widespread effects on our daily and working lives, with our online activity monitored and computers fully embedded in the workforce.
- It could be argued that a desire for the latest technology – particularly mobile phones – encourages a 'disposable' culture.

Computers are used throughout the workforce, in many cases making old roles redundant but introducing new ones:

- Automated assembly lines: for instance, instead of a human welder making cars, factories now require a technician to maintain the robots.
- Online shopping: instead of a shop assistant we now use automated warehouses with workers collecting and packing objects.
- Online banking: we now use automated telephone systems and no longer need as many high street banks and the associated workforce.
- Will driverless cars in the future mean we no longer need people to drive taxis and lorries?

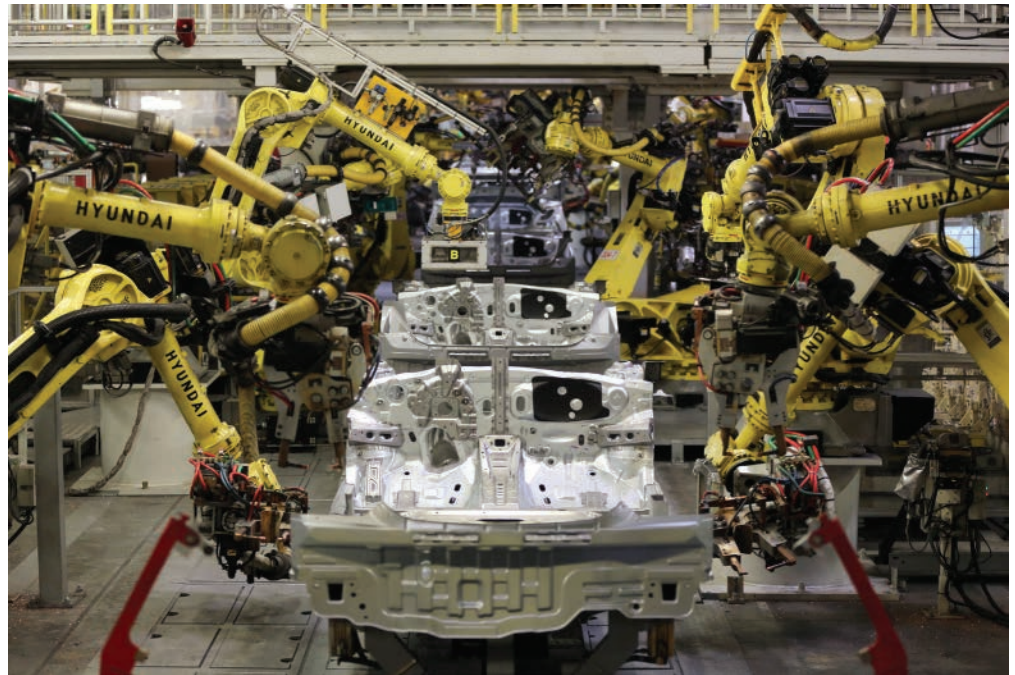


Figure 1.6.1 Robots building a car

Many organisations collect data about individuals:

- Searches on the internet and social media are monitored and analysed by companies to understand consumer behaviour and focus their advertising for individuals.
- Social media activity, financial activity (e.g. use of credit cards), use of passports and internet search history details may be monitored by governments to track criminal or terrorist activity.
- Medical data is collected and analysed by medical researchers to develop new treatments for various conditions.

Censorship and the internet

Censorship is the deliberate suppression of material by an organisation or government. This may include:

- material considered to be socially unacceptable
- information that the organisation or government regards as dangerous.

Access to websites is controlled by blacklists, which keep a record of unacceptable websites and monitor web pages to see if some or all of the content should be blocked.

The extent to which the internet is censored varies from organisation to organisation and from country to country. Governments often take differing views about what is acceptable and what is not.

The debate is about where to draw the line between protecting the public and infringing their rights to free speech and access to information.

At a local level, schools may filter content to protect children from unsuitable content. At a national level some countries impose very strict filtering of content in order to prevent the population from debating political or cultural issues that it does not approve of.

Automated decision-making

Computers are used for automated decision-making by analysing data to reach a decision about what action to take. It is used in situations where decisions have to be made frequently and rapidly based on electronic evidence. These systems use complex algorithms to reach a decision. It is important to get these algorithms right in order to avoid incorrect decisions, which in some circumstances have serious consequences. However, automated decision-making can also make some processes very impersonal.

- Electrical power distribution requires rapid responses to changing demand.
- Emergency services use algorithms when responding to major incidents in order to deploy resources quickly and effectively.
- Stock market trading, also known as algorithmic or automated trading.
- Industrial plant automation, for example chemical plants or distribution centres.
- Airborne collision avoidance systems in planes.
- Driverless cars.
- Banks use algorithms to decide whether someone can have a bank account or credit card based on their financial history. In the past a person would make this decision.

Knowledge check



- 3 Write down two advantages and two disadvantages of using social media every day.
- 4 Identify two advantages of monitoring an individual's internet searches for an online store.

Environmental issues

Most modern computers consume low levels of electricity but are often left running permanently. Data centres, which are large facilities that store all sorts of data (like Instagram accounts, YouTube videos, etc.), account for around 2% of all energy used on the planet – this is the same as air travel. In addition, energy is of course also used to manufacture a computer in the first place.

As with all consumer electronics, computers are made from valuable physical resources such as metals and minerals, some of which are very rare and non-renewable. Computers also include some pretty toxic material, such as: airborne dioxins, polychlorinated biphenyls (PCBs), cadmium, chromium, radioactive isotopes and mercury. Their disposal raises **environmental** issues and needs to be handled with great care.

Unfortunately, old computer equipment is often shipped off to countries with lower environmental standards, to reduce the cost of disposal, and they end up in landfill sites. In some cases, children pick over the waste to extract metals that can be recycled and sold thus exposing them to significant danger



Figure 1.6.2 Picking over discarded computer equipment to extract metals

However, there are also a number of positive effects on the environment from computer use, including:

- A reduction in the use of paper, with on-screen reading replacing the need for physical copies of documents.
- Laptops and the internet allow people to work from home, reducing the need for travel which reduces energy consumption and CO₂ emissions.
- Computers are essential tools for scientific research into the development of renewable energy and more energy-efficient devices.
- Smart metering constantly monitors and accurately reports energy and water use.
- Computers are also essential for the management of renewable and low energy-use technology.

Knowledge check



- 5** Discuss the environmental impact of computer use.

Privacy issues

Most people agree they have a right to some degree of privacy. However, we often provide lots of personal information to organisations whenever we access the internet, particularly when people sign up to services with accounts. Organisations may even share this information with third parties – which we may have accidentally agreed to when we opened an account.

Personal details and details of activities are often willingly shared on social media. People do not always realise how much personal information they are sharing and exactly what can be seen by whom.

Other examples of computer technology used to monitor behaviour include:

CCTV cameras around our towns and cities used to monitor behaviour	To some this represents an intrusion of privacy. Others feel CCTV provides added security and safety assurances. Facial recognition algorithms can be used in conjunction with CCTV footage on private land, e.g. in shopping centres. There are concerns about how this technology could be used in the future.
Automatic number plate recognition (ANPR)	ANPR is widely used on UK roads and in car parks to determine who has parked and for how long.
Electronic tags on those who have committed a criminal offence	These tags can identify when they are not in the agreed location at the agreed time or, with GPS, identify their location at any time.
Black boxes installed in cars which monitor how and when people drive	These can reduce insurance premiums by rewarding safe driving.
Mobile phone signal and location tracking by apps	Mobile phone signal data is stored by mobile phone companies, which can provide very accurate location information. Mobile call records are also stored and can be requested as part of law enforcement activities. Individuals' locations can also be tracked by mobile apps or mobile websites without their knowledge or explicit permission.
Workplace logging systems	These can be used to monitor activity at work such as any online activity, phone calls, work patterns, work quality, social media activity, etc. Unacceptable posts, such as trolling, racist or sexist comments, can be traced back to the organisation and reflect badly upon them.

Many organisations collect data about individuals and this is often shared with partner organisations.

- Whenever we check in on social media the location and time is logged.
- Many apps track the location of your mobile phone.
- Whenever we take a picture with our phone's camera the location and time are logged. When such images are uploaded to social media sites, the companies are able to access this information and automatically scan images to try and work out who was in the picture through facial recognition algorithms.

When browsing the internet, websites can, quite legally, track a lot of information about you – this might include your location, your browser, your IP address, your operating system, what websites you have visited and what you have searched for. Data is a valuable commodity and companies use computer algorithms to analyse your behaviour and build up a detailed profile about you. This profile is often used to target advertising.

There are ways you can limit such tracking but this normally requires changing the default settings of your browser and websites that you log into.

Knowledge check



- 6 A distribution warehouse uses computer technology to monitor its workforce. Describe two ways the distribution centre management might monitor the workforce.
- 7 Is it reasonable for organisations to demand access to and monitor social network pages where the content is posted from private computers?

Legislation relevant to computer science

The Data Protection Act 2018

Computers hold vast amounts of data and it is important this data is collected, stored and processed in ways that protect the individual. The Data Protection Act 2018 sets out rules for

Tech term

GDPR A Europe-wide law enforcing an individual's rights over their data.

handling this personal data and is the UK's implementation of the **General Data Protection Regulation (GDPR)**. Every organisation, apart from those with specific exemptions, holding personal data must register with the Information Commissioner's Office and disclose what data they are holding, why they are collecting it and how it will be used.

Exemptions are granted to specific sectors including national security, scientific research, financial services and the Home Office.

There are some key exceptions to these regulations for organisations dealing with national security, crime detection or taxation.

The Data Protection Act 2018 sets out seven key principles that should be central to processing personal data:

Lawfulness, fairness and transparency

- There must be valid reasons for collecting and using personal data.
- Nothing must be done with the data in breach of any other laws.
- Personal data can only be used in a way that is fair. This means data must not be processed in a way that is damaging, unexpected or misleading.
- The person or organisation collecting the data must be open and honest with people from the start about how they will use their personal data.

Purpose limitation

- The purpose for processing the data must be clear from the start.
- The purpose must be documented and specified in privacy information that is communicated to individuals.
- Data must not be used for a new purpose unless this is compatible with the original purpose. If it is not then additional consent must be obtained unless there is a clear obligation or function set out in law.

Data minimisation

Data being processed must be:

- adequate
- relevant
- limited to what is necessary.

Accuracy

- All reasonable steps must be taken to ensure the personal data held is not incorrect or misleading.
- Data must be kept up to date.
- If any data is incorrect or misleading it must be corrected or erased as soon as possible.
- Challenges to the accuracy of personal data must be considered.

Storage limitation

- Data must not be kept for longer than necessary.
- How long data is kept for must be justified and specified in a policy statement.
- There must be a periodic review of the data held, and data no longer required should be erased or anonymised.
- Individuals have a right to have data erased if it is no longer required.
- Data can be kept for longer if it is only kept for public interest archiving, scientific or historical research, or statistical purposes.

Security

There must be adequate security measures in place to protect the data held.

Accountability

The data holder must take responsibility for how the data is used and for compliance with the other principles.

Computer Misuse Act 1990

Under the provisions of the Computer Misuse Act 1990 it is a criminal offence to make any unauthorised access to computer material ...

- ... with intent to commit further offences (for example blackmail)
- ... with the intent to modify the computer material (for example distributing viruses).

The first provision refers to unauthorised access (commonly called hacking).

The second provision refers to anything that impairs the performance of a computer system including the distribution of viruses.

For information on how to minimise these threats see section 1.4.2.

Knowledge check



- 8 What are the seven principles of the Data Protection Act?
- 9 What is the purpose of the Computer Misuse Act?

Copyright, Designs and Patents Act 1988 (CDPA)

The CDPA protects the intellectual property of an individual or organisation. Under the act it is illegal to copy, modify or distribute software or other intellectual property without the relevant permission. This act also covers video and audio where peer-to-peer streaming has had a significant impact on the income of the copyright owners. Using the internet to download free copies of copyright material (e.g. software, films, books, music) without the consent of the author is illegal since no money or credit will have been passed on to the original creator.

Software licences

Most commercial software will come with a **licence** agreement specifying how the purchaser may use the product. In most cases a licence key will be required to access the software to prevent unauthorised copying and distribution.

Proprietary software

Much of the software we buy is written by organisations trying to make a profit. The source code is kept securely and versions of the software are distributed as executable programs. The user is not able to access the source code and cannot modify it. Copyright laws also forbid this modification. In return for the money paid these organisations fully test the product and regularly provide upgrades to fix bugs or improve features of the program. If the product has faults in it then the user can contact the organisation for an upgrade or fix. These products usually have a significant amount of online support available for them.

Open-source software

Software developed under open standards has its source code freely available so that others can access it and make changes to it to develop their own version of the product. **Open-source software** is often regularly updated by a community of developers. These updated versions are then made available to anyone for little or no cost. Despite being free, this software is often of very high quality because of the community of highly skilled developers who regularly test, fix and improve the product. On the downside there is no one to blame if it goes wrong.

Open-source software	Proprietary software
Access to the source code	No access to source code
May be free of charge	Almost always some cost involved
Users can modify the software	User cannot modify the software
Can be installed on as many computers as necessary	Extra licences must normally be obtained before installing on additional computers
No one is responsible for any problems with the software	Full support from the software developer
Usually only community support	Commercial and community support available



Figure 1.6.3 Libre Office is a suite of office programs available for free download

Examples of open-source software include: Libre Office, Mozilla Firefox, the Android operating system, Linux and the Python programming language.

There are no 'typical' users for open-source or **proprietary software** – many large organisations such as Amazon rely on open-source software because they have the in-house expertise to manage it effectively. The choice will be based on the level of expertise balanced against the need for commercial support.

An individual who relies on a piece of software but has limited expertise in finding and understanding the community support available for open-source software will find the readily available support to solve problems with proprietary software essential.

Creative Commons

Creative Commons is an organisation that issues licences that are less restrictive than proprietary licences. The licences include:

- public domain, which has no restrictions on use for any purpose
- attribution, where the work can be freely reproduced but the original creator must be credited
- attribution non-commercial, where the work can only be used for non-commercial purposes.

This form of licence is often referred to as 'some rights reserved' as opposed to the CPDA's 'all rights reserved'.

Knowledge check

- 10 What is the difference between proprietary and open-source software?
- 11 Describe Creative Commons.



RECAP AND REVIEW

1.6 ETHICAL, LEGAL, CULTURAL AND ENVIRONMENTAL IMPACTS OF DIGITAL TECHNOLOGY

Important words

You will need to know and understand the following for the exam:

Ethics
Legal
Culture
Environmental
Licence
Proprietary software
Open-source software

1.6.1 Ethical, legal, cultural and environmental impacts

Ethical and legal issues

Ethics refer to what is right and wrong and how people should behave. Computer organisations such as BCS have codes of conduct that prescribe suitable ethical behaviour for its members.

Legal issues are laws drawn up to govern activities and control computer crime such as:

- unauthorised access to data and computer systems for the purpose of theft or damage
- identity theft
- software piracy
- fraud
- harassment, such as trolling.

Cultural issues

Culture refers to the ideas, behaviour, beliefs and values of a group of people. Cultural issues relate to how computers have impacted our lives, including:

- the widespread use of 'disposable' digital devices
- the ways in which people interact with each other – for instance, social media
- changes in the workplace
- replacing human roles in organisations
- changing human roles within organisations
- widespread data collection about individuals
- access to entertainment and social interaction
- workforce monitoring.

Environmental issues

The negative **environmental** impacts of widespread computer use include:

- large global energy requirements to run computer systems and data centres
- the use of rare and non-renewable metals and minerals
- some computer components are made from toxic materials that are a hazard to the environment and to human health if not disposed of properly.

The positive environmental impacts of widespread computer use include:

- homeworking reduces the need to travel, which reduces CO₂ emissions
- more on-screen documents means a reduction in the use of paper and other resources
- computers enable scientific research that leads to more environmentally friendly technologies, such as electric cars, the design of solar panels, and so on.

Privacy issues

Using computers raises concerns about individual rights to privacy. Ways in which individuals are monitored include:

- Companies can monitor exactly what their workforce are doing on their computers.
- Use of CCTV and facial recognition.
- Automatic number plate recognition (ANPR).
- Websites can track a lot of information about your internet activities: your location, your browser, your IP address, your operating system, what websites you have visited and what you have searched for. This data might be used to provide insights, for example to target advertising.
- Mobile phone companies are able to track an individual's location from their mobile phone, even if they are not using it.
- Mobile phone call records are also stored and can be accessed by law enforcement agencies if requested.
- With the wrong privacy settings, social media activity is available for anyone to see.

Legislation

The Data Protection Act 2018

The Data Protection Act 2018 sets out seven key principles that should be central to processing personal data:

- lawfulness, fairness and transparency
- purpose limitation
- data minimisation

- accuracy

- storage limitation

- security

- accountability.

Exemptions are granted to specific sectors including national security, scientific research, financial services and the Home Office.

Computer Misuse Act 1990

This act makes it a criminal offence to access or modify computer material and includes hacking and the distribution of malware.

Copyright Designs and Patents Act 1988

Protects the intellectual property of an individual or organisation making it illegal to copy, modify or distribute software or other intellectual property such as music and video.

Software licences

Most commercial software will come with a **licence** agreement.

Proprietary software is written by organisations trying to make a profit.

- The source code is kept securely and versions of the software are distributed as executable programs so that the user is not able to access the source code or modify it.

- The software is copyright protected, making it illegal to modify or distribute it.

- The software is usually licensed for a fixed number of computer systems.

- The software is fully tested and supported by the organisation.

Open-source software uses a community of developers.

Software developed under open standards makes the source code available to everyone.

- It is developed and updated by a community of programmers.

- It can be installed on as many computers as necessary.

- Others can modify the code and distribute it.

- Versions are made available at no or very little cost.

- It relies upon the community for testing and support; modified versions may not be supported or fully tested.

Open-source software	Proprietary software
Access to the source code	No access to source code
May be free of charge	Almost always some cost involved
Users can modify the software	Users cannot modify the software
Can be installed on as many computers as necessary	Extra licences must normally be obtained before installing on additional computers
No one is responsible for any problems with the software	Full support from the software developer
Usually only community support	Commercial and community support available

Creative Commons is an organisation that issues licences allowing a user to modify or distribute parts of the software under certain conditions. Also known as 'some rights reserved'.

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

SECTION 2

COMPUTATIONAL THINKING, ALGORITHMS AND PROGRAMMING

Ensure no other block

```
def row_available(block, row)
```

```
# Determine which of the main
```

```
20 boardRow = int(block / 3);
```

```
21 good = True
```

```
22 for b in range(boardRow * 3, (t
```

```
23 if b != block:
```

```
24 if num in board[b][row]:
```

```
25 good = False
```

```
break
```

```
return good
```

ALGORITHMS

CHAPTER INTRODUCTION

In this chapter you will learn about:

2.1.1 Computational thinking

- Principles of computational thinking

2.1.2 Designing, creating and refining algorithms

- Identify the inputs, processes and outputs for a problem
- Structure diagrams
- Create, interpret, correct, complete and refine algorithms
- Identify common errors

- Trace tables

2.1.3 Searching and sorting algorithms

- Standard sorting algorithms
- Standard searching algorithms

2.1.1 Computational thinking

Computers are simply electronic machines that carry out instructions given by human programmers; although a computer can solve many complex problems, it can only do so if it is given the instructions to tell it how to solve the problem. Writing these instructions in a form that is suitable for a computer to carry out can be a challenging task! **Computational thinking** involves three key principles to help with this.

Principles of computational thinking

- **Decomposition** is breaking a problem down into smaller sub-problems. Once each sub-problem is small and simple enough it can be tackled individually.
- **Abstraction** is removing or hiding unnecessary details from a problem so that the important details can be focused on or more easily understood.
- **Algorithmic** thinking involves deciding on the order that instructions are carried out and identifying decisions that need to be made by the computer.

For example, if a programmer wishes to create a system to sell T-shirts, this could be a relatively large and complex problem. Where would they start? By using these tools, the programmer can begin to work out how to tackle the problem.

Decomposition	Abstraction	Algorithmic thinking
How can the T-shirt sales system be split up? It perhaps involves: <ul style="list-style-type: none"> • A login system for users • A search function to find particular T-shirts • A system to allow users to buy T-shirts. • A reordering system 	What can we focus on and what can we ignore? Considering the login system and search function of our example: <ul style="list-style-type: none"> • The login system could focus on a customer's email address and password but ignore everything else. • The search function could focus on the T-shirt details (colour, size, material, price, how many in stock). 	What steps need to be taken in which order? One example may be: IF a customer enters a correct username AND correct password, show them the catalogue of T-shirts. ELSE display an error message, then redirect back to the login screen.

Key point

Computational thinking doesn't immediately give us all of the answers, but it certainly helps to begin to think in a way that makes a problem easier to solve.

A particularly powerful example of abstraction is Harry Beck's well-known London Underground map. The map focuses on the connections between stations and how lines intersect. It is a useful tool for anyone who wants to plan a journey in London.

However, it also ignores many of the realities of tunnels, distances between stations and relation to streets passing overhead; these are not important for underground travellers and would make the map more confusing if included.



Figure 2.1.1 Abstraction – Harry Beck's London Underground map, 1931

Knowledge check



- 1 Define the following terms:
 - (a) decomposition
 - (b) abstraction
 - (c) algorithmic thinking.
- 2 A programmer is creating software to send and receive encrypted messages via email. Describe how decomposition can be used to allow this problem to be solved.
- 3 A chess club develops a system to store details about games played. For each game, the winner's and loser's names are stored alongside the date that the game was played. Explain how abstraction has been used in the development of this system.

2.1.2 Designing, creating and refining algorithms

An **algorithm** is a step-by-step sequence of instructions that are used to solve a problem. Each instruction in an algorithm must be precise enough to be understood independently. The algorithm must also clearly show the order that each instruction is carried out in and where decisions are made or sections repeated.

Identify the inputs, processes and outputs for a problem

An algorithm can be represented by the diagram in Figure 2.1.2.

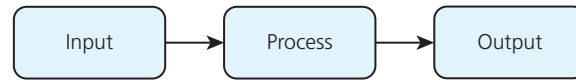


Figure 2.1.2 Input–Process–Output

Input refers to data that is given to the algorithm by the user. This may be typed in via the keyboard, entered with another input device such as a microphone or read in from an external data file.

Output is the data that is given back to the user by the algorithm. This may be done with a message on the screen, using another output device such as a printer or written to an external data file.

Processes are the steps that the algorithm takes in order to solve the problem. This stage involves using the input given in order to produce the desired output.

Going back to the example of the system to sell T-shirts, what are the inputs, outputs and processes for the section of the system that allows the user to search for suitable T-shirts?

The inputs include everything that the system requires the user to enter: for example, the size, colour and style of T-shirts that they require. Another input would be an external file containing details of all the T-shirts available.

With these inputs, the system will then carry out processes to find T-shirts to match the user's search criteria, such as only selecting those in the correct size and colour and excluding any that are out of stock.

Once this has been completed, a list of matching T-shirts will be produced as an output to the user.

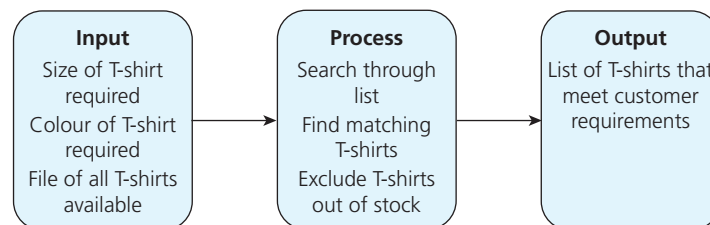


Figure 2.1.3 Example of Input–Process–Output

Structure diagrams

A **structure diagram** is a form of top-down design that helps us to decompose a problem. At each level, the problem is broken down by asking the question: 'What is involved in carrying out that step?'

For example, we can further investigate the T-shirt sales system. As we decided when looking at decomposition, a successful system would perhaps need to have a login system for users, a search function, a sales function and a reordering function. We can represent this in a structure diagram:

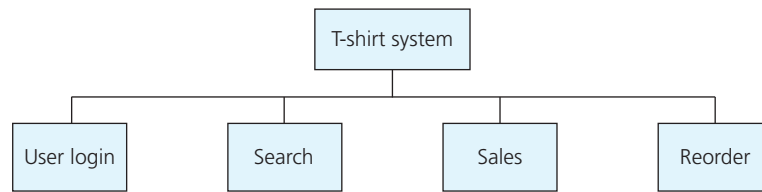


Figure 2.1.4 Two-level structure diagram

However, what is involved in each of these steps? If we further decompose the sales step, we may decide that we need to have a shopping basket for the user to select the T-shirts to buy. We may also need a way to pay for the T-shirts and a confirmation email perhaps needs to be sent. We can add this to our structure diagram at a lower level, beneath sales.

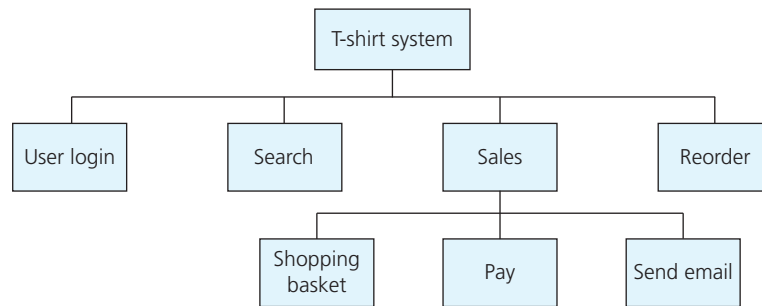


Figure 2.1.5 Three-level structure diagram

We could then decompose each of the other steps in a similar way to help us to decide what is involved in carrying out that step. We could also decide that we need to understand a little bit more about what is involved in setting up a shopping basket or paying for the T-shirts, and so add further levels beneath them; structure diagrams can have as many levels as required until each step is clearly defined and understood.

Create, interpret, correct, complete and refine algorithms

Flowcharts

The inputs, processes and outputs can be put together into an algorithm by using a **flowchart**. This is a graphical representation of an algorithm and uses symbols to denote each step, with arrows showing how to move between each step.

A flowchart may use any of the following symbols:

	Line	An arrow represents control passing between the connected shapes.
	Process	This shape represents something being performed or done.
	Subroutine	This shape represents a subroutine call that will relate to separate, non-linked flowchart.
	Input/output	This shape represents the input or output of something into or out of the flowchart.
	Decision	This shape represents a decision (Yes/No or True/False) that results in two lines representing the different possible outcomes.
	Terminal	This shape represents the 'Start' and 'End' of the process.

Figure 2.1.6 Flow diagram symbols

All flowcharts begin and end with the terminal shape, indicating the start and end of the flowchart. Inputs and outputs are represented by a parallelogram, with decisions using a diamond shape. Decision boxes must have two possible outputs, a True/False or Yes/No decision having taken place. All other processes are shown as a rectangle.

Where algorithms are decomposed into separate **subroutines**, a rectangle with two additional vertical lines is used to show a call to a different subroutine.

The following flowchart shows part of the algorithm for the T-shirt system that deals with reordering when T-shirt stocks get low.

Tech term

Subroutine A separate piece of code that can be called within a program.

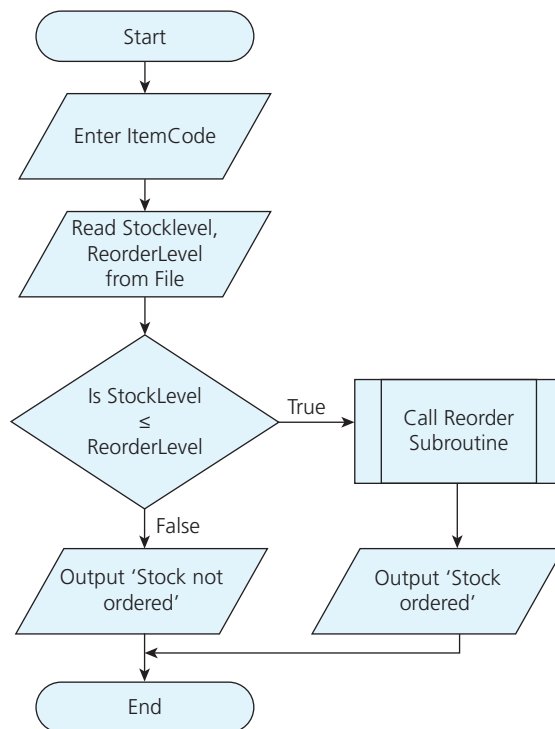


Figure 2.1.7 Flow diagram for T-shirt reordering system

Key point

Because pseudocode is not strictly defined, many variations would be acceptable. For example, the keyword **PRINT** could instead be replaced with **OUTPUT**, or even **DISPLAY** – as long as the steps to be taken are clear, that is sufficient.

Key point

A trace table shows the values of each variable after each step has been executed. Although the item code is only entered on line 1, it still holds the same value throughout the rest of the program and so its value is repeated in later lines of the trace table.

Pseudocode

Alternatively, an algorithm may be represented using **pseudocode**. Pseudocode is a textual, English-like method of describing an algorithm. It is much less strict than high-level programming languages, although it may look a little like a program that could be entered directly into a computer. The same T-shirt reordering system could be represented in pseudocode as follows:

```
01 INPUT ItemCode
02 READ StockLevel, ReorderLevel From File
03 IF StockLevel <= ReorderLevel THEN
04   Call ReOrder()
05   PRINT "Stock ordered"
06 ELSE
07   PRINT "Stock not reordered"
08 END IF
```

Trace tables and identifying common errors

When an algorithm has been defined, it needs to be checked for correctness. A **trace table** is a tool that can be used to follow each line of an algorithm through, step by step. The trace table will show the contents of each variable after each line has been carried out and will also show any output. By manually following through the algorithm in this way, we can see if it produces the correct result and if not, help us to identify where any errors have occurred.

For example, we can trace through the pseudocode algorithm shown previously to reorder stock for item 009, for which we currently have 8 in stock and has a reorder level of 3.

Line	Item Code	Stock Level	Reorder Level	Output	Comments
01: INPUT ItemCode	009				Item code 009 entered by user
02: READ StockLevel, ReorderLevel From File	009	8	3		Values read in from external file
03: IF StockLevel <= ReorderLevel THEN	009	8	3		8 is NOT smaller than or equal to 3, so line 7 executed next
07: PRINT "Stock not reordered"	009	8	3	'Stock not reordered'	Correct output printed

By tracing this algorithm through using a trace table, we can see that it does work correctly for an item that has enough items in stock. We would also have to trace through values that were out of stock and perhaps also values that were just equal to the reorder level to be sure that this algorithm works perfectly.

Beyond the spec

The GCSE Computer Science specification covers three sorting algorithms and two searching algorithms, but there are many, many more that each work in different ways. Why not investigate how Quicksort, Shellsort or even Bogosort work?

Knowledge check



- 4 The following algorithm has been designed to decide which of two numbers is the largest. Complete the trace table to check that the algorithm works correctly when the values 8 and 5 are entered by the user.

```
01 INPUT NumOne
02 INPUT NumTwo
03 IF NumOne >= NumTwo THEN
04   PRINT NumTwo
05 ELSE
06   PRINT NumOne
07 END IF
```

Line	NumOne	NumTwo	Output	Comments

2.1.3 Sorting and searching algorithms

One of the great things about an algorithm is that it can be reused; once a computer scientist has written down a clever set of instructions for how to do something, other people can simply follow those instructions to solve the same problem.

A sorting algorithm is a set of instructions used to put a list of values into order. A searching algorithm is used to find a value within a list, or to confirm that value is not present in the list.

Sorting algorithms

Bubble sort

The **bubble sort** algorithm works by comparing pairs of values. If the two values are in the wrong order with respect to each other, they are swapped over. This is then repeated for each further pair of values. When the last pair of values has been compared, the first pass of the bubble sort algorithm is complete.

The algorithm will repeat until a pass has been completed with no swaps occurring. Once this happens, the list is guaranteed to be in order.

Worked example



The following list of numbers will be sorted into ascending order using the bubble sort algorithm.

7	2	9	4	3
---	---	---	---	---

Firstly, 7 and 2 are compared as the first two values in the list. These are in the wrong order, so they are swapped over.

2	7	9	4	3
---	---	---	---	---

Now 7 and 9 are compared. These are in the correct order and so no swap is necessary.

2	7	9	4	3
---	---	---	---	---

On the next step, 9 and 4 are compared. These are in the wrong order and so are swapped.

2	7	4	9	3
---	---	---	---	---

Finally, 9 and 3 are compared, which are in the wrong order and so are swapped.

2	7	4	3	9
---	---	---	---	---

The first pass of the bubble sort algorithm has been completed. However, at least one swap has taken place and so the algorithm is repeated. After this pass, the numbers will be in the following order, with two swaps having taken place:

2	4	3	7	9
---	---	---	---	---

Again, because swaps have taken place, the algorithm must repeat. This time, only one swap is needed, giving the following list:

2	3	4	7	9
---	---	---	---	---

Key point

The list now appears to be in numerical order. However, the algorithm only stops when a pass is completed without any swaps taking place. This is not yet the case.

The final pass of the algorithm compares each pair of numbers and finds no numbers that need to be swapped. The algorithm is therefore complete and the values are in order.

The bubble sort algorithm gets its name because numbers 'bubble' to the top after every pass.

Key point

We have covered how the bubble sort algorithm works. However, a full implementation of the bubble sort in OCR Exam Reference Language is given in the Appendix at the back of the book. In the exam YOU MAY BE ASKED ABOUT THIS ALGORITHM IN OCR EXAM REFERENCE LANGUAGE – so ensure you read and understand the Appendix.

Writing the bubble sort algorithm out in pseudocode could be done as follows:

```

REPEAT until NoSwapsMade
  REPEAT for each pair in list
    IF list[x] > list[x+1] THEN
      #swap values
      temp = list[x]
      list[x] = list[x+1]
      list[x+1] = temp
    END IF
  END LOOP
END LOOP

```

Insertion sort

The **insertion sort** algorithm splits the list to be sorted in two parts: a sorted side and an unsorted side. Initially, the sorted side contains just the first item in a list, with everything else on the unsorted side.

7	2	9	4	3
---	---	---	---	---

Sorted
Unsorted

Beyond the spec

If a list contains just one value, it can always be considered to be in order. This is a helpful concept for other sorting algorithms at A Level.

Each item on the unsorted side is then taken and inserted into the correct place on the sorted side, one by one. In this example, 2 would next be taken and inserted before the value 7 in the sorted list.

2	7	9	4	3
---	---	---	---	---

Sorted
Unsorted

This process is repeated for the next item in the unsorted list, with 9 inserted into the sorted list.

2	7	9	4	3
---	---	---	---	---

Sorted
Unsorted

The penultimate number (4), is then inserted into the sorted list.

2	4	7	9	3
---	---	---	---	---

Sorted
Unsorted

When the final number (3) is then inserted into the sorted list. At this point, all numbers have been inserted and the list can be said to be in order.

2	3	4	7	9
---	---	---	---	---

Sorted

Key point

We have covered how the insertion sort algorithm works. However, a full implementation of the insertion sort in OCR Exam Reference Language is given in the Appendix at the back of the book. In the exam YOU MAY BE ASKED ABOUT THIS ALGORITHM IN OCR EXAM REFERENCE LANGUAGE – so ensure you read and understand the Appendix.

Unlike a bubble sort, an insertion sort does not require multiple passes to check that the values are in order; once each value has been inserted into the sorted list and the unsorted list is empty, the list as a whole will be in order. Writing the insertion sort algorithm out in pseudocode could be done as follows:

Split the list into a sorted section (containing the first item) and unsorted section (containing the rest of the values)

REPEAT for each item in the unsorted section

Insert each item in turn into the sorted section, placing it in the correct place.

Merge sort

The **merge sort** algorithm uses a 'divide and conquer' approach to split data up into individual lists and then merge it back together in order. The way that the lists are merged back together is key to understanding how this algorithm works.

7	2	9	4
---	---	---	---

First, in the 'divide' stage, the original list is split into two separate sublists – [7 2] and [9 4]. Each of those sublists are themselves each split into two sublists. Each sublist now only contains one element.

7	2	9	4
---	---	---	---

Key point

You will be expected to know how to perform a merge sort on a list with an even or odd number of elements. With an odd number of elements there are two choices of where to divide lists – you can choose either but you must apply your choice consistently. Each of the merge stages must resemble the divide stages, just with elements in a different order.

Key point

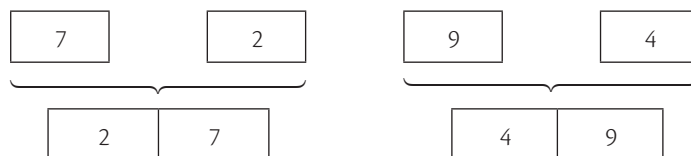
At any point in the merging stage of this algorithm, the merged lists are always in order. This means that it is only the first numbers from each list that need to be considered when deciding on how to merge them.

Key point

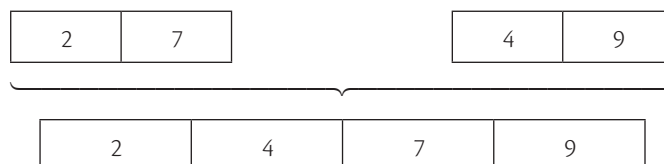
Unlike the other algorithms, you do not need to recognise an implementation of merge sort in OCR Exam Reference Language. However you DO need to understand and be able to carry out a merge sort in the exam.

Then each pair of lists are then merged together in the 'conquer' stage. Where there are an uneven number of lists, the odd list will simply remain unmerged until the next step in the process.

When two lists are merged together, the first number in each of the lists are compared and whichever should be first is taken to be first in the new list. This process is repeated until all numbers have been inserted into the new list.



Here, 7 and 2 are compared, with 2 being inserted into the new list before 7. Similarly, 4 is inserted before 9 in the next merged list.



The merging process is again repeated to merge pairs of lists together. 2 and 4 are compared to decide which value will be first in the new list, with 2 being inserted. 7 and 4 are then compared, with 4 being inserted. 7 and 9 are compared, with 7 being inserted before 9 is finally inserted. It is important to note that each sublist in the merge stage should contain the same elements as the corresponding sublist in the divide stage – just in a different order.

The list is now in order. Writing the merge sort algorithm out in pseudocode could be done as follows:

REPEAT until all lists are merged

REPEAT for each pair of lists

REPEAT until all numbers in pairs of lists are merged

Compare the first value in both lists

Insert larger of two values into new merged list

Remove value from old list

Comparison of sorting algorithms

All of the above algorithms will result in a sorted list, but they will do it in very different ways. Bubble sort is generally thought of as a simple but slow algorithm; as the size of the list of values increases, it slows down significantly because it requires multiple passes over the same data.

An insertion sort can be more efficient, but a merge sort is much more efficient than both of these for large lists of values. However, a merge sort may not be the best sorting method for nearly-sorted or small lists.

Knowledge check



- 5 Explain how a bubble sort would sort the values [6, 9, 2, 5, 8] into order.
- 6 Show the stages of an insertion sort being used to sort the words ['Dog', 'Cat', 'Mouse', 'Ant'] into alphabetical order.
- 7 A merge sort is an example of a divide and conquer algorithm. State what happens during the divide stage.

Searching algorithms

A searching algorithm is used to find an item of data in a list, or to confirm that it is not in the list. The simplest searching algorithm is a linear search.

Linear search

Key point

We have covered how the linear search algorithm works. However, a full implementation of the linear search in OCR Exam Reference Language is given in the Appendix at the back of the book. In the exam YOU MAY BE ASKED ABOUT THIS ALGORITHM IN OCR EXAM REFERENCE LANGUAGE – so ensure you read and understand the Appendix.

A **linear search** is carried out by inspecting each item of the list in turn to check if it is the desired value. If so, we have found the item; if not, the next item in the list must be checked. If the algorithm gets to the end of the list without finding the item, then it is not in the list.

7	2	9	4	3
---	---	---	---	---

To find the value 9 in this list, 7 and 2 would be checked first before finally finding 9. If the value 8 was to be searched for using this algorithm then every item (7, 2, 9, 4 and 3) would be checked. Upon checking the last value, we can be sure that the item is not in the list.

A linear search is simple but inefficient. If we have a list of a million values, we have to check all one million of them before being sure that the value is not in the list. Writing the linear search algorithm out in pseudocode could be done as follows:

REPEAT until number is found or end of list is reached

Check one value (from the start)

IF value matches what is being searched for:

Output value

Stop

ELIF end of list reached:

Output "not found"

Stop

END IF

Move to next value

Binary search

A much more efficient algorithm to find values in a list is a **binary search**. However, this algorithm has the **pre-requisite** that the list must be in order – it is impossible to use a binary search on an unsorted list.

The middle value in the sorted list is picked. 'Middle' means there are equal numbers of values either side of it. If there are an even number of values then there isn't an exact middle value – however, generally the value to the left of the middle is chosen. (Either side can be picked as long as we are consistent.)

If the middle value is the one we are searching for then the algorithm finishes. However, if not, we can discard the bottom half of the list if the middle value is smaller than the one we are searching for, or discard the top half of the list if the middle number is larger than the one we are searching for. Either way, we always also discard the middle value.

If we get to a situation where the list only has one item and it is not the one that we are searching for, then the value is not in the list.

Worked example

The list below is in alphabetical order and so can be used in a binary search. We will look for the value Q in this list.

A	C	D	F	H	K	P	Q	S	T	V	W	Z
---	---	---	---	---	---	---	---	---	---	---	---	---



We first take the middle value (P). This is not the value that we are looking for and is smaller than Q alphabetically, so we can discard the bottom half of the list, up to and including P.

Q	S	T	V	W	Z
---	---	---	---	---	---

We now have a list of six values. As there is no middle value, we pick the value to the left of the middle, which is T. This is not the value that we are looking for and T is larger than Q alphabetically, so the top half of the list (including T) can be discarded.

Q	S
---	---

We now have a list of two values. Again, there is no middle value so we pick the value to the left of the middle which is Q. This is the value that we are searching for and so the algorithm stops.

Key point

We have covered how the binary search algorithm works. However, a full implementation of the binary search in OCR Exam Reference Language is given in the Appendix at the back of the book. In the exam YOU MAY BE ASKED ABOUT THIS ALGORITHM IN OCR EXAM REFERENCE LANGUAGE – so ensure you read and understand the Appendix.

Writing the binary search algorithm out in pseudocode could be done as follows:

REPEAT until (value found) or (list is of size 1 and value is not found)

Pick the middle value in a list (if odd number of values, pick left-of-middle value)

IF value matches what is being searched for:

Output value

Stop

ELIF value searched for > middle value:

Discard middle value AND bottom half of list

ELIF value searched for < middle value:

Discard middle value AND top half of list

END IF

Comparison of searching algorithms

A linear search and a binary search will both find a value in a list, but will do so in very different ways. A linear search will work with any list of values, but may be very slow to do as it checks every value in the list.

A binary search will be much more efficient, but requires the list to be sorted into order. A binary search halves the size of the list to be searched on every comparison.

Imagine searching for value in a list of one million numbers. In the worst-case scenario, a linear search will need to compare against each and every one of these million numbers. A binary search needs only to make 21 comparisons before it has completed its search.

Knowledge check



- 8 Explain how a linear search would find the value 18 in the list [1, 8, 6, 2, 18, 14, 7].
- 9 Which value would be the first to be compared in a binary search through the list [A, B, C, D, E]?
- 10 How does a linear search determine that a value does not appear in a list?

RECAP AND REVIEW

2.1 ALGORITHMS

Important words

You will need to know and understand the following for the exam:

Computational thinking
Decomposition
Abstraction
Algorithmic thinking
Algorithm
Structure diagram
Flowchart
Pseudocode
Trace table
Sorting algorithm – bubble sort, insertion sort, merge sort
Searching algorithm – linear search, binary search
Pre-requisite (for algorithm)

2.1.1 Computational thinking

Principles of computational thinking

Computational thinking is all about structuring solutions to problems in a way that computers can easily follow.

- **Decomposition** breaks a problem into smaller sub-problems that can each be tackled individually. Each stage can be further decomposed if needed.
- **Abstraction** means focusing on what is important in a problem and ignoring or hiding the irrelevant details.
- **Algorithmic thinking** involves thinking how the problem can be clearly defined as an **algorithm** – that is, what needs to be done and in what order.

For example, if a programmer wishes to create a system to sell T-shirts, this could be a relatively large and complex problem. By using these tools, a programmer can begin to work out how to tackle the problem.

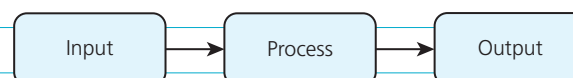
2.1.2 Designing, creating and refining algorithms

Identify the inputs, processes and outputs for a problem

Computer systems:

- accept input from a user (or from sensors or data files)
- process this data, and then
- output the result back to the user in some format.

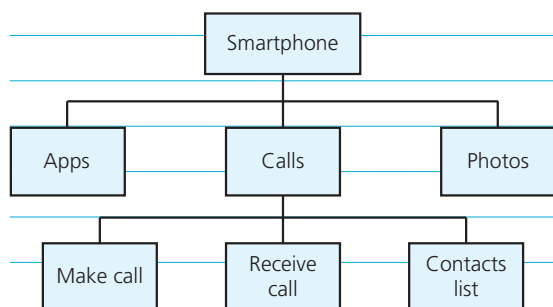
Identifying the inputs into a system and the outputs required enables us to decide how the two can be mapped; programmers can write programs to process the inputs in order to provide the required outputs.



Structure diagrams

A **structure diagram** is a graphical method of decomposing a problem, with each layer breaking down the layer above it into smaller and smaller sub-problems.

The example below shows a partially completed structure diagram for a smartphone. The phone software has been split up into Apps, Calls and Photos, with the Calls subsection further split up into Making Calls, Receiving Calls and a Contacts List. Of course, the Apps and Photos sections could also be broken down and the Calls subsections could even be broken down further.



Structure diagram of smartphone functions

Flowcharts

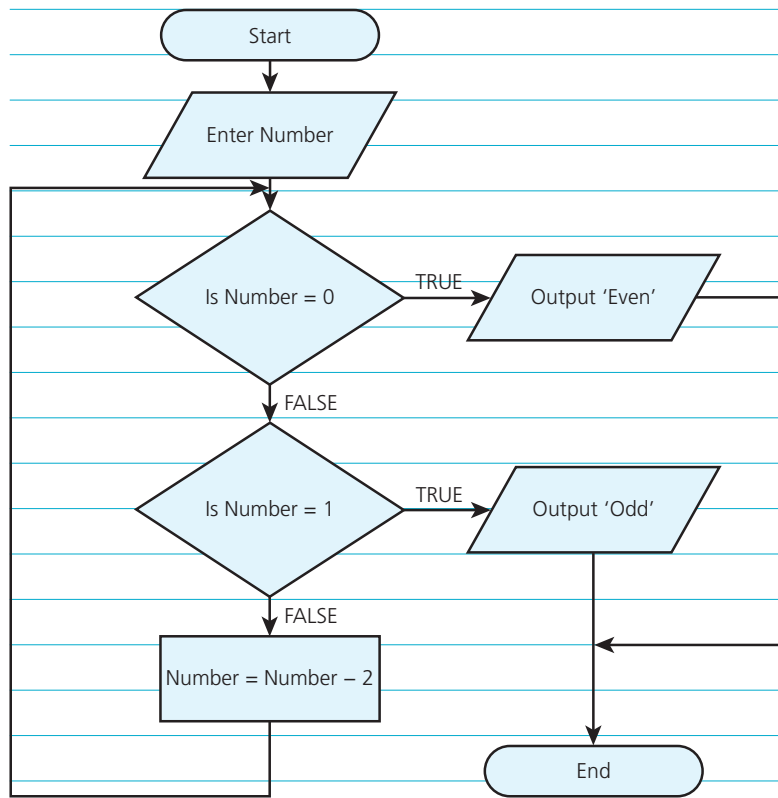
A **flowchart** is a graphical representation of an algorithm. It can be followed from top to bottom, making decisions as appropriate.

- Decisions are represented by diamond shapes and must have a True/False outcome.
- Inputs and outputs are represented by parallelograms.
- Processes are represented by rectangles.
- Start/Stop instructions are represented by rounded rectangles.

Each flowchart must begin and end with a Start and a Stop instruction.

As an example, the algorithm in the flow diagram on page 120 will decide whether a positive number entered is odd or even using repeated subtraction.

	Line	An arrow represents control passing between the connected shapes.
	Process	This shape represents something being performed or done.
	Subroutine	This shape represents a subroutine call that will relate to separate, non-linked flowchart.
	Input/output	This shape represents the input or output of something into or out of the flowchart.
	Decision	This shape represents a decision (Yes/No or True/False) that results in two lines representing the different possible outcomes.
	Terminal	This shape represents the 'Start' and 'End' of the process.



Flow diagram of odd/even number checking code

Pseudocode

- **Pseudocode** is a textual representation of an algorithm.
- It is very closely related to high-level programming code, although it does not have the precision in syntax required by languages such as Python or C#.
- Pseudocode enables programmers to communicate algorithms to other programmers, showing the steps used without worrying about which language they are using.

The pseudocode algorithm below carries out the same algorithm as the flowchart shown previously – finding whether a positive number is odd or even through repeated subtraction.

```

01 INPUT number
02 WHILE number >=1
03   IF number == 1 THEN
04     PRINT "odd"
05   ELIF number == 0
06     PRINT "even"
07   END IF
08   number = number - 2
09 END WHILE
  
```

Trace tables

Trace tables are used to follow an algorithm through from start to finish. At each point, the value of each variable and any outputs are recorded. A table such as the one shown below is used. More columns are added if more variables are used.

Line	Variable1	Variable2	Variable3	Output	Comments

If the program repeats certain lines, this is reflected in the trace table; each row shows one line that is executed and lines of code can be repeated as many times as required. The completed trace table below shows the algorithm running to check whether 5 is an odd or even number.

Line	Number	Output	Comments
01: INPUT number	5		Number inputted by user
02: WHILE number >= 1	5		Number is >= 1, continue
03: IF number == 1	5		Number is not equal to 1
05: ELIF number == 0	5		Number is not equal to 0
07: END IF	5		End of IF statement
08: number = number - 2	3		2 subtracted from number
09 END WHILE	3		End of loop, repeat again
02: WHILE number >= 1	3		Number is >= 1, continue
03: IF number == 1	3		Number is not equal to 1
05: EIF number == 0	3		Number is not equal to 0
07: END IF	3		End of IF statement
08: number = number - 2	1		2 subtracted from number
09 END WHILE	1		End of loop, repeat again
02: WHILE number >= 1	1		Number is >= 1, continue
03: IF number == 1	1		Number does equal 1
04: PRINT "odd"	1	'odd'	'odd' printed
07: END IF	1		End of IF statement
08: number = number - 2	-1		2 subtracted from number
09 END WHILE	-1		End of loop, repeat again
02: WHILE number >= 1	-1		Number is NOT >= 1, WHILE loop and program ends.

Bubble sort

The **bubble sort** algorithm uses the following steps:

REPEAT while a swap has taken place (always run at least once)

REPEAT for each pair of numbers in the list

Compare pairs of values

IF values are in the wrong order:

swap numbers over

record that a swap has taken place

- Bubble sort is a relatively simple algorithm to program.
- However, it is quite inefficient and may take much longer to complete than other sorting algorithms on very large lists.

Insertion sort

The **insertion sort** algorithm uses the following steps:

Split the list into a sorted section (containing the first item) and unsorted section (containing the rest of the values)

REPEAT for each item in the unsorted section

Insert each item in turn into the sorted section, placing it in the correct place.

- Insertion sort is more efficient than the bubble sort algorithm, but can be relatively tricky to implement in a high-level language.
- Moving values around without overwriting other values can be difficult for inexperienced programmers.

Merge sort

The **merge sort** is a divide and conquer algorithm. The divide stage uses the following steps:

REPEAT until each list is of size 1

Divide each list in half

When all numbers are split up into separate lists, the merge stage can begin. This uses the following steps:

REPEAT until all lists are merged

REPEAT for each pair of lists

REPEAT until all numbers in pairs of lists are merged

Compare the first value in both lists

Insert larger of two values into new merged list

Remove value from old list

If at any stage an odd number of lists are present, the final list can simply be ignored until the next iteration.

- The merge sort is much more efficient than both the bubble sort and insertion sort, especially on large lists.
- It will sort a large list of random values into order in a quicker time than both of the other algorithms.

Linear search

A **linear search** uses the following steps:

REPEAT until number is found or end of list is reached

Check one value (from the start)

IF value matches what is being searched for:

Output value

Stop

ELIF end of list reached:

Output "not found"

Stop

END IF

Move to next value

- The linear search is relatively inefficient, but it works on any list, regardless of whether it is in any particular order.
- Every single value in the list needs to be checked before you can be certain that a value is not present in a list.

Binary search

A **binary search** requires the list of values to be in order. It uses the following steps:

REPEAT until (value found) or (list is of size 1 and value is not found)

Pick the middle value in a list (if odd number of values, pick left-of-middle value)

IF value matches what is being searched for:

Output value

Stop

ELIF value searched for > middle value:

Discard middle value AND bottom half of list

ELIF value searched for < middle value:

Discard middle value AND top half of list

END IF

Binary search is highly efficient. If an ordered list of one million numbers is used, the binary search could find a number in the list with no more than 21 comparisons. The linear search by contrast could take up to one million comparisons.

However, the binary search will only work if the list of values is in order. Therefore, it cannot always be used.

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

PROGRAMMING FUNDAMENTALS

CHAPTER INTRODUCTION

In this chapter you will learn about:

2.2.1 Programming fundamentals

- The use of variables, constants, operators, inputs, outputs and assignments
- The use of sequence, selection and iteration to control the flow of a program
- The common arithmetic operators
- The common Boolean operators AND, OR and NOT

2.2.2 Data types

2.2.3 Additional programming techniques

- The use of basic string manipulation
- The use of basic file handling operations
- The use of records to store data
- The use of SQL to search for data
- The use of one-dimensional and two-dimensional arrays
- How to use subprograms (functions and procedures) to produce structured code
- Random number generation

2.2.1 Programming fundamentals

Key point

Paper 2 requires significant knowledge and understanding of programming. It is highly recommended that you complete as much practical programming practice as possible in a high-level language such as Python, Visual Basic or C#.

All questions in the examination will be presented in the **OCR Reference Language**. Full details of this can be found at the back of the OCR specification. However, your answers can be given in **either** this reference language **or in a high-level language of your choice**.

All examples given in this chapter are written in the OCR Reference Language.

The use of variables, constants, assignments, operators, inputs and outputs

Variables, constants and assignments

Variables are used in a computer program to store a single piece of data. For example, you may wish to store someone's score in a game. As the name suggests, the contents of a variable may be changed (or varied) during the running of a program.

When a variable is first defined, the programming language allocates a small area of memory to store this data.

A variable has an **identifier**, or name. This is the label given to the area of memory. Variable identifiers can be almost anything, but they must not contain a space, start with a number or be a **reserved keyword** (that is, a word that means something else in that programming language).

Tech terms

Identifier The name of a variable or constant.

Reserved keyword A word in a particular programming language that has some special purpose and cannot be used for a variable or constant identifier.

Examples of allowed variable names	Examples of disallowed variable names
<ul style="list-style-type: none"> • <code>score</code> • <code>max_points</code> • <code>item2</code> 	<ul style="list-style-type: none"> • <code>new score</code> (contains a space) • <code>2ndName</code> (starts with number) • <code>print</code> (reserved keyword)

Most programming languages are case sensitive; this means that `score` and `Score` are treated as different variables.

Key point

A common mistake is to describe a constant as a variable that doesn't change – variables and constants are similar, but they are not the same. Would you describe a car as a bike with four wheels?

Constants are used in computer programs to store a single piece of data that has a fixed value and cannot change during the running of the program. A constant also has an identifier that acts as a label for a memory location that stores the data.

Assignment means to give a value to a variable or constant. This is normally done using the '=' sign. (Some high-level languages may use ':=' or other symbols). The variable or constant identifier always goes on the left and the value to be assigned goes on the right.

A variable can be assigned a value multiple times, but when a new value is assigned the old value is overwritten and lost. A constant can only be assigned a value once, usually at the start of a program.

Worked example



The following program shows a variable and a constant being assigned values, using the OCR Reference Language. Note how the contents of the variable **score** is changed, but the constant **maxScore** is fixed and cannot change.

```
const maxScore = 100
score = 20
score = 30
score = score + 10
```

After this program is run, **maxScore** has the value 100 and **score** has the value 40.

Inputs and outputs

It is useful to be able to **input** data from a user and **output** data back to the user. The keywords **input** and **print** are used for these purposes in OCR Reference Language. The high-level language that you are using may have different keywords for the same purpose.

Worked example



The following program in the OCR Reference Language calculates the radius of a circle using pi and user input.

```
const pi = 3.14159
radius = input("enter the radius")
print ("the circumference is", 2*pi*radius)
```

Operators

A computer program uses **operators** to perform some sort of action; for instance the '+' sign is used to add two numbers together. Operators will be discussed in detail a little later in the chapter.

Knowledge check



- 1 What is a variable?
- 2 What is the difference between a variable and a constant?

The use of the three basic programming constructs to control the flow of a program

The building blocks for any computer program can be reduced down to only three constructs: **sequence**, **selection** and **iteration**.

Sequence

Sequence is the execution of statements one after the other, in order. The program runs from top to bottom and each instruction completes fully before the next one is executed.

<u>Program A</u>	<u>Program B</u>
<code>x = 10</code>	<code>x = 10</code>
<code>x = x * 3</code>	<code>x = x + 1</code>
<code>x = x + 1</code>	<code>print(x)</code>
<code>print(x)</code>	<code>x = x * 3</code>

Both programs have the same instructions but in a different order. Program A will print out 31 but program B will print out 11. However, in program B the final value of **x** equals 33 because there is another line of code after the `print(x)` statement. The **sequence** of instructions is important and changing this can change how the program works.

Key point

A **Boolean condition** is a statement that can be evaluated to be either True or False. 'What do I want for lunch today' is not a Boolean condition as the answer could be one of many things but 'Do I want pizza for lunch?' would be a Boolean condition as the answer could only be True (yes I want pizza) or False (no, I do not want pizza).

Selection

Selection is the construct used to make decisions in a program. These decisions are based on **Boolean conditions** and the program then takes one of two paths based on this condition.

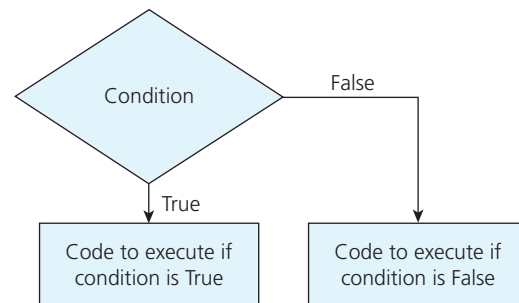


Figure 2.2.1 Selection based on a Boolean condition

The most common way of implementing selection is by using **IF** statements. The **IF** keyword is used to define the condition to be evaluated, with the code to be executed if true indented between the **IF** and **ENDIF** keywords.

Worked example

```
name = input("enter your name")
if name == "George" then
    print("Hello George")
endif
```

In this example, the condition that is evaluated is whether the inputted name matches the value given (George). If it does, the third line is executed. If not, the program skips over this line entirely.

To extend this program to do something else if the condition is false, the **ELSE** keyword can be used. The section of code indented under the **ELSE** statement is only executed when the **IF** condition is false.

Worked example

```
name = input("enter your name")
if name == "George" then
    print("Hello George")
else
    print("Hello stranger")
endif
```

If the entered name is George then only the indented line under the **IF** statement is executed, before the program moves on to the **endif** statement.

If the entered name is not George then only the indented line under the **ELSE** statement is executed before the program moves on to the **endif** statement.

But what if we want to check for multiple conditions, each with their own associated code to run if true? The **ELSEIF** keyword allows us to do this.

Worked example

```
name = input("enter your name")
if name == "George" then
    print("Hello George")
elseif name == "Lorne" then
    print("Great work Lorne")
elseif name == "Kirstie" then
    print("Nice to see you again")
else
    print("Hello stranger")
endif
```

In this example, each possible condition is evaluated in turn. First, the inputted name is checked against 'George'. If this is true, the message on line 3 is printed. If not, the second possible condition ('Lorne') is checked. If not true, the third and final condition ('Kirstie') is checked. If none of these are true, the 'Hello stranger' message is printed.

With the above example, the program will always have one path to follow. It is important to note that the conditions are checked in the sequence given.

Worked example

```
mark = input("enter mark out of 20")
if mark >5 then
    print("Could do better")
elseif mark >10 then
    print("Average mark")
elseif mark >15 then
    print("Excellent")
endif
```


What would happen if someone here got a mark of 19 out of 20? Unfortunately, the message 'Could do better' would be displayed as 17 is larger than 5 and so the first condition is true. The other conditions will not be checked.

The solution to this problem would be to change the sequence of the conditions so that the highest mark is checked first

```
mark = input("enter mark out of 20")
if mark >15 then
    print("Excellent")
elseif mark >10 then
    print("Average mark")
elseif mark >5 then
    print("Could do better")
endif
```

An alternative way of presenting selection is with a **SWITCH/CASE** statement. Not all high-level languages have these available (Python does not) but they are logically equivalent to **IF** statements where multiple choices for values are given.

Key point

Iteration can be **count-controlled** or **condition-controlled**. Count-controlled iteration repeats a set number of times (e.g. 'repeat this code 5 times') whereas **condition-controlled** iteration repeats based on a **Boolean condition**.

Tech term

Increment Add a value to; 'increment by one' adds 1 to the counter each time the loop repeats.

Worked example

```
name = input("enter your name")
switch name:
    case "George":
        print("Hello George")
    case "Lorne":
        print("Great work Lorne")
    default:
        print("Hello stranger")
endswitch
```

SWITCH/CASE statements are not suitable where ranges of values (e.g. 'larger than 10') are given as the programmer would need to list each possible value rather than give a range.

Iteration (count and condition-controlled loops)

Iteration is the construct used to repeat sections of code. Iteration is commonly called **looping**.

Count-controlled iteration uses the **FOR** and **NEXT** keywords. It uses a variable to act as a counter for the loop. By default, the program will assign this variable a starting value and then automatically **increment** the variable by one every time the code repeats. When this counter equals (or exceeds) the final value, the loop stops.

This code and flowchart are logically identical. Both print out the 8 times table from 1 to 10.

Key point

A **FOR** loop will automatically increment the value of the counter variable by 1 each time; there is no need to do this manually.

Worked example

```
for p=1 to 10
  print (p*8)
next p
```

Both the program code and flowchart will print out the values 8, 16, 24 and so on, up to and including the final value 80.

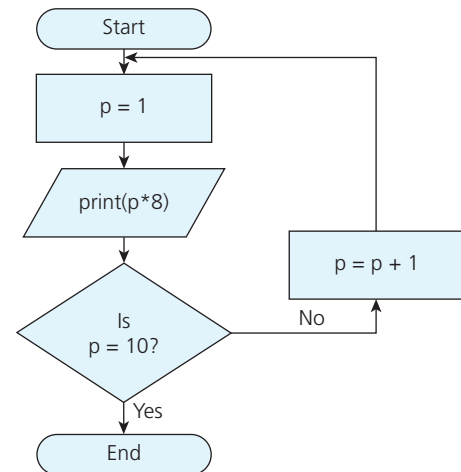


Figure 2.2.2 Flowchart representing this code

The **step** keyword can also be used with a **FOR** loop, in order to increment the counter variable by different values. This can be used to count up in steps, or even count backwards.

Worked example

This code will add 2 to the counter variable every time around the loop, printing out 1, 3, 5 and so on until the variable **k** exceeds 20. Note that it will never hit exactly 20.

```
for k=1 to 20 step 2
  print(k)
next k
```

This code will subtract 1 from the counter variable every time around the loop, printing out 10, 9, 8 and so on until the variable **y** equals 1.

```
for y=10 to 1 step -1
  print(y)
next y
```

Using the **FOR** loops shown so far, the programmer defines exactly how many times the given code will repeat.

Condition-controlled iteration instead checks a condition each time around the loop to decide whether to repeat the code again or continue. For this type of iteration, the programmer will not know how many times the code will repeat.

Most programming languages provide two types of condition-controlled iteration – **WHILE** loops and **DO UNTIL** loops. These both perform a similar task but differ in when the condition is checked and whether the condition needs to be True or False to repeat again.

WHILE loop	DO UNTIL loop
<pre>total = 0 while total < 20 num = input("enter number") total = total + num endwhile print("done!")</pre>	<pre>total = 0 do num = input("enter number") total = total + num until total >= 20 print("done!")</pre>

Note that the **WHILE** loop will repeat **while** the total is less than 20 whereas the **DO UNTIL** loop will repeat **until** the total is larger than or equal to 20. They are both logically equivalent and produce exactly the same results but check different conditions at different times in the code.

Key point

A **WHILE** loop checks the condition before starting the loop whereas a **DO UNTIL** loop checks the condition after the loop has completed. In some circumstances, this could mean that the **WHILE** loop will never start but the **DO UNTIL** loop will always execute at least once. Both types of loop in the example above will run infinitely if the user simply repeats typing in negative values.

Key point

Note that Python only provides a **WHILE** loop – there is no **DO...UNTIL** loop option in Python.

Knowledge check



- 3 State the three basic program constructs and describe each one.
- 4 What is the difference between a While loop and a Do loop?

The common arithmetic operators

The **arithmetic operators** are used to carry out basic mathematical operations on values in computer programs. The common operators are:

Operator	Name	Example	Comment
+	Addition	<code>a = b + c</code>	Adds the two values given.
-	Subtraction	<code>x = y - 1</code>	Subtracts the second value from the first.
*	Multiplication	<code>score = score * 10</code>	Multiplies the two values together.
/	Division	<code>value = num1/num2</code>	Divides the first value by the second.
MOD	Modulus	<code>r = score MOD 2</code>	Returns the remainder after dividing the first value by the second value.
DIV	Quotient	<code>q = score DIV 2</code>	Returns the whole number part after dividing the first value by the second value, ignoring any remainder.
^	Exponent	<code>square = num ^ 2</code>	Raises the first value to the power of the second value. In this case, raises num to the power of 2. (Raising num to the power of 2 means num * num)

Key point

The **MOD** operator can be used to decide if a number is odd or even. If we hold a value in the variable **num**, then **num MOD 2** will give 0 if the value is an even number and 1 if the value is an odd number. It can also be used in the same way to decide if a number is an exact multiple of another smaller number.

Operator precedence is the same as in GCSE Mathematics, with **BIDMAS** being important. Any operators in brackets are applied first, with indices next, then division and multiplication, then addition and subtraction last.

Worked example



For example, $(3+6) + 7 * 2$ means that the $(3+6)$ is completed first, with the multiplication $(7*2)$ completed next and then the results of these added together:

$$\begin{aligned}
 &(3+6) + 7 * 2 \\
 &= 9 + 7 * 2 \\
 &= 9 + 14 \\
 &= 23
 \end{aligned}$$

This gives the answer of 23. If the calculation was (incorrectly) carried out sequentially then this would give the wrong answer of 32.

The comparison operators

The **comparison operators** all evaluate to a Boolean **True** or **False** outcome.

Operator	Name	Examples	Comment
<code>==</code>	Equal to	<code>7 == 7 (true)</code> <code>8 == 2 (false)</code>	Some languages may use a single <code>=</code> sign. Do not confuse this with assignment!
<code>!=</code>	Not equal to	<code>7 != 9 (true)</code> <code>3 != (1+2) (false)</code>	<code>!=</code> gives the opposite outcome to <code>==</code> . (Some languages may use <code><></code> .)
<code><</code>	Less than	<code>4 < 7 (true)</code> <code>4 < 4 (false)</code>	This gives a False output if the values are equal.
<code><=</code>	Less than or equal to	<code>7 <= 7 (true)</code> <code>6 <= 4 (false)</code>	This gives a True output if the values are equal.
<code>></code>	Greater than	<code>2 > 1 (true)</code> <code>3 > 5 (false)</code>	This gives a False output if the values are equal.
<code>>=</code>	Greater than or equal to	<code>9 >= 9 (true)</code> <code>6 >= 9 (false)</code>	This gives a True output if the values are equal.

The common Boolean operators AND, OR and NOT

If multiple conditions need to be evaluated, the **Boolean operators** **AND** and **OR** can be used. A condition can also be reversed using the **NOT** operator. These function in the same way as the Boolean logic gates with the same names in section 2.4.1.

The **AND** operator requires **both** conditions to be True for the overall condition to be True. If either or both of the conditions are False, the overall condition will be False.

Worked example

```
username = input("enter your username")
password = input("enter your password")
if username == "admin" and password == "changeme123" then
    print("Correct details. Logged in")
else:
    print("Incorrect details")

endif
```

In this example, the **AND** keyword means that both conditions (the username and password both matching the correct ones) need to be True for the user to be logged in.

The **OR** operator requires **one or the other (or both)** of the conditions to be True for the overall condition to be True. If both of the conditions are False, the overall condition will be False.

Worked example



```
num1 = input("enter a value over 10")
num2 = input("enter another value over 10")
if num1 >10 or num2 >10 then
    print ("Accepted")
else:
    print ("Rejected")
endif
```

In this example, the **OR** keyword means that if either of the entries is over ten then the 'Accepted' message will be displayed. If both are over ten then the accepted message is still displayed. The 'Rejected' message is only displayed if neither of the two entries are over ten.

Key point

The two conditions given to an **AND** or **OR** operator must be full conditions. A very common mistake is to write code like:

```
if num1 or num2 >10 then
```

This is incorrect; the first condition (**num1**) has nothing to be compared against and so will not work as intended. The correct way, as shown in the **OR** example would be:

```
if num1 >10 or num2 >10 then
```

The **NOT** operator reverses the True or False outcome from a comparison.

Worked example



```
num1 = input("enter a number")
num2 = input("enter a number")
if NOT(num1 == num2):
    print("the numbers are not the same")
endif
```

In this example, the equivalence operator (**==**) would give a True outcome if the input values were the same. However, the **NOT** operator reverses this, so that False is returned if the numbers are the same and True if they are not the same.

Of course, this is logically the same as using the **!=** operator.

Knowledge check



- 5 What is the value assigned to the variable **x** in each of the following?
- | | |
|---------------------------|---------------------------|
| (a) x = 23 - 4*3 | (e) x = 19 MOD 5 |
| (b) x = (12 - 5)*3 | (f) x = 22 MOD 4*2 |
| (c) x = 6*2/3 | (g) x = 28 DIV 6 |
| (d) x = 8/(5-1) | (h) x = 23 DIV 2*4 |
- 6 What will be returned by the following comparisons?
- | |
|---|
| (a) a != b if a = 8 and b = 5 |
| (b) a >= b if a = 9 and b = 5 |
| (c) a > b OR c < d if a = 5, b = 5, c = 3, d = 2 |
| (d) a <= b AND c != d if a = 6, b = 6, c = 2, d = 4 |

2.2.2 Data types

The use of data types

We have previously seen that variables and constants are allocated space in memory to store their data. The type of data to be stored determines how much memory needs to be allocated to that variable or constant.

There are five main data types that GCSE Computer Scientists need to be aware of.

Integer

Integers are whole numbers, positive or negative, that have no decimal or fractional part. Integers are used for counting or storing quantities. For example:

```
score = 25
highScore = 100
numOfAttempts = 0
```

All variables used in this example are integers.

Tech term

Floating point numbers

Another name for real numbers – the floating point refers to the position of the decimal point, which can be different (or 'float') for different numbers.

Real

The real data type is used for numbers, positive or negative, that have (or may have) a decimal or fractional part. **Real numbers** are sometimes called **floating point numbers**. For example:

```
price = 19.99
fastestTime = 9.983
score = 17.0
```

All variables used in this example are real numbers.

Boolean

Boolean variables only ever store True or False values. They are often used as flags or to indicate the result of a condition. For example:

```
sorted = False
LoggedIn = True
```

Both variables used in this example are Boolean values.

Character

A **character** is a single item from the character set used on the computer, such as H, r, 7 or &. Uppercase and lowercase are different characters and space is also a character.

When assigning a character to a variable, quotation marks are required to indicate that the value to be assigned is a character. For example:

```
a = "t"
b = "4"
c = "%"
d = " "
```

All variables used in this example are characters.

String

A **string** data type stores a collection of characters, typically used for names, addresses or other textual information. Note that numbers and symbols can also be characters and so can be included in a string.

Just like characters, when assigning a string to a variable, quotation marks are required:

```
a = "the colour blue"
b = "47 times"
c = "95% increase"
d = "p@55w0rd"
```

All variables used in this example are strings.

Key point

Strings and characters require quotation marks around the values to be assigned. This is to differentiate them from variables. Compare the following lines of code:

```
colour = "blue"
colour = blue
```

The first line assigns the string value of **"blue"** to the variable **colour**. However, the second line treats both **colour** and **blue** as variables and assigns the contents of variable **blue** to the variable **colour**. If the variable **blue** does not exist, this would cause an error in the program.

Casting

Casting means to convert data from one data type to another. To do this, the following keywords are used:

Tech term

Concatenate Joining multiple strings together.

Keyword	Converts to ...	Examples	Comment
str()	String	a = str(123) b = str(True)	Converts any other data type into a string. Some languages require this to be able to concatenate values.
int()	Integer	c = int("123") d = int(87.0)	Converts real numbers to integers by removing the decimal part and returning the integer part (it does not round the number). Strings containing only numeric values can be cast as integers.
real()	Real	e = real("112.9") f = real(46)	Integers and strings containing only numeric values can be cast as real numbers. Some languages use float() to cast.
bool()	Boolean	g = bool("True")	Some languages will cast integers 1 and 0 to be True and False.

Not all data types can be cast as another data type. For example, if a programmer attempted to cast the string 'hello' as an integer value using `newval = int("hello")`, an error would be raised. There is no sensible way of deciding what numeric value 'hello' should be converted to.

```
>>> newval = int("hello")
ValueError: invalid literal for int() with base 10: 'hello'

>>> newval = int("hello")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    newval = int("hello")
ValueError: invalid literal for int() with base 10: 'hello'
```

Figure 2.2.3 Example of an error in Python when casting a string to an integer

Knowledge check



- 7 State whether the following data are real numbers, integers, characters, Boolean or strings:
- (a) 35
 - (b) &
 - (c) 3!=2
 - (d) twenty
 - (e) 35.0
 - (f) 6.63

2.2.3 Additional programming techniques

The use of basic string manipulation

Just as integers and real numbers can be manipulated using arithmetic operators, strings can be manipulated using basic string handling keywords.

Keyword	Use	Examples
<code>.length</code>	To count how many characters are contained in a string.	<code>name="Seth Bottomley"</code> <code>name.length</code> gives 14
<code>.substring(x,y)</code>	To extract characters from the middle of a string where x is the starting point (beginning at 0) and y is the number of characters required. This is known as string slicing.	<code>name.substring(2,5)</code> gives 'th Bo' <code>name.substring(0,3)</code> gives 'Set'
<code>.left(x)</code>	To extract characters from the left of a string, where x is the number of characters required.	<code>name.left(7)</code> gives 'Seth Bo'

Keyword	Use	Examples
.right(x)	To extract characters from the right of a string, where x is the number of characters required.	name.right(5) gives 'omley'
.upper	To convert a string to UPPERCASE.	name.upper gives 'SETH BOTTOMLEY'
.lower	To convert a string to lowercase.	name.lower gives 'seth bottomley'
ASC()	To find the ASCII value of a character.	ASC("D") gives 68
CHR()	To find the character that relates to the ASCII value given.	CHR(68) gives 'D'

String slicing

String slicing is the term used to refer to extracting characters from a string. In the OCR Reference Language, the **substring()** command can be used for this, although **left()** and **right()** can also be used if the characters to be extracted are at the start or end of the string.

Concatenation

Concatenation of strings means to join multiple strings together. This is done using the **+** operator. When strings are concatenated, they are joined together in the order given.

Worked example

```
texta = "this is a message"
textb = "great string"
new = "GCSE " + texta.substring(5,2) + " " + textb.left(5) + "
print(new)
```

The above code would join together extracts from the two strings to print out 'GCSE is great'.

Note that some languages such as Python also allow programmers to treat a string as an array of characters and extract individual characters this way. This would be an acceptable alternative in examination questions.

Key point

The **+** operator is 'overloaded', which means that it does different things depending on what data is given to it. Compare the result of **2+7** (9) with the result of **"hello"+"world"** (helloworld). The **+** operator concatenates strings but also adds numeric values.

Knowledge check

- 8 If **text = "Computing is fun"** what is returned by:
- text.len**
 - text.left(2)**
 - text.right(8)**
 - What command returns the string 'fun'?

The use of basic file handing operations

Variables and arrays are temporary in their nature. When the program ends, all values are cleared and not remembered when the program is run again. However, **text files** can be used to permanently store data.

Opening and closing

Before a text file can be used, it must be **opened**. The OCR Reference Language uses the **open()** keyword to do this, with the filename passed in as a parameter in brackets. This is assigned to a variable with a specified identifier, for example:

```
names = open("classnames.txt")
```

where the text file **classnames.txt** has been opened and assigned to the variable called **names**.

Once the file has been used, it must be **closed** to ensure that all data is consistent. The **.close()** keyword is used in OCR Reference Language to do this.

Read

To **read** a line of text from an open text file, the **.readLine()** keyword is used. This will return the next line in the file. If the **.readLine()** keyword is used multiple times, it will return the next line each time.

The **.endOfFile()** keyword returns a Boolean True or False value to indicate whether the end of the file has been reached. This can commonly be used with a **WHILE** or **DO UNTIL** loop to keep on reading lines until the end of the file is reached.

Write

To **write** a line of text to an open file, the **.writeLine(x)** keyword is used, where x is the string to be written to the text file. The line of text will always be written to the bottom of the text file, underneath any other lines of text that are already present.

Worked example

```
colourFile = open("colours.txt")
while NOT colourFile.endOfFile()
    data = colourFile.readLine()
    print(data)
endwhile
colourFile.close()
```

In the example above, the code prints out all of the data contained in the "**colours.txt**" file. It:

- opens the text file "**colours.txt**" using the identifier **colourFile**
- sets up a **WHILE** loop that runs until the end of the file is reached
- reads one line of data from **colourFile** into a variable called **data**
- prints out the contents of the variable **data**
- this loop repeats for each line in the text file, and prints out the contents of "**colours.txt**" one by one
- once the end of the file is reached the loop stops
- **colourFile** is closed.





Worked example

```
colourFile = open("colours.txt")
do
    data = input("enter a colour")
    if data != "end" then
        colourFile.writeline(data)
    endif
until data == "end"
colourFile.close()
```

The example above writes colours to a text file until 'end' is entered. It:

- opens the text file "**colours.txt**" using the identifier **colourFile**
- sets up a **DO UNTIL** loop
- asks the user to input a colour and stores this in the variable **data**
- using an **IF THEN** statement, checks that the input is not the word 'end' and if not writes the contents of the variable **data** to the end of **colourFile** – this prevents the word 'end' from being written to **colourFile**
- the **DO UNTIL** loop checks if the input was the word 'end' and repeats the loop if not but ends the loop if it was
- **colourFile** is closed. The file "**colours.txt**" now contains the extra data that the user inputted.

The use of records to store data

A **record** is a data structure that allows multiple data items to be stored, using **field names** to identify each item of data. To create a record, we must first define the field names that will make up each record. For a record about a student, these might be:

- FirstName
- YearGroup
- Surname
- Email

We can then store data under these field names in a database management system using a **table**.

Table 2.2.1 Table called 'Student' showing three records

FirstName	Surname	YearGroup	Email
Bradley	Jenkins	9	bjenkins@notreal.co.uk
Jamie	Cable	10	jcable@notreal.co.uk
Charlotte	Pegg	9	cpegg@notreal.co.uk

Key point

Some special symbols called wildcards can be used in searches. The * wildcard can be used with a **SELECT** keyword as a shortcut to indicate that all fields from the table will be returned.

The use of SQL to search for data

Structured Query Language (SQL) is a language used to access data stored in a database. There are three main keywords to be aware of:

- **SELECT** identifies the fields to return from the database
- **FROM** identifies which table(s) the data will be returned from
- **WHERE** allows the programmer to include criteria, with only matching records being returned.

SELECT and **FROM** are compulsory to use in an SQL query; the **WHERE** clause is optional; if it is not included then all records from the table specified will be returned.

Worked example



Using the Student table from the previous section as a guide, the FirstName and Surname fields for all students can be shown by using this SQL query:

```
SELECT FirstName, Surname
FROM Student
```

FirstName	Surname
Bradley	Jenkins
Jamie	Cable
Charlotte	Pegg

To show all fields, but only those students in Year 10, a **WHERE** clause would be used

```
SELECT *
FROM Student
WHERE YearGroup = 10
```

FirstName	Surname	YearGroup	Email
Jamie	Cable	10	jcable@notreal.co.uk

To show the FirstName, YearGroup and Email for all students whose first name was either Jamie or Charlotte, this SQL query would be used :

```
SELECT FirstName, YearGroup, Email
FROM Student
WHERE FirstName = "Jamie" or FirstName = "Charlotte"
```

FirstName	YearGroup	Email
Jamie	10	jcable@notreal.co.uk
Charlotte	9	cpegg@notreal.co.uk

Knowledge check



- 9 A junior football club stores details for their members including their surname, telephone number, membership fee, membership paid or not and number of goals scored.

Identify the most appropriate field type for each of these fields.

- 10 The table **Address_book** contains the following data:

First_name	Last_Name	Telephone	Email
Bill	Wilson	02223334445	bw@notreal.cod
Graham	Mills	02232232232	gm@notexist.cot
Harry	Smith	01223123123	harry@home.vid
Sheila	Jones	01212121212	SJ@home.vid

- What is returned by the following query? **SELECT First_name, Telephone FROM Address_book**
- Write a query to return the First_name and Email for the Last-name 'Mills'.
- Write a query to return all the information for the entry with the Email 'SJ@home.vid'.

The use of arrays (or equivalent), including 1D and 2D arrays

We have seen that a variable can be used to store a single item of data in a computer program. An **array** allows a programmer to store multiple items of data under a single identifier. This would be useful, for example, in a computer game to store the names of multiple players.

Arrays have a fixed number of items that they can store; this number is defined when the array is created. Each item in the array must also be the same data type.

Tech term

Index value A number that corresponds to the location of an item of data in an array.

One-dimensional arrays

A one-dimensional array is accessed via a single numeric **index value**.

Array index	0	1	2	3
Data	"Fletcher"	"Imogen"	"Tia"	"Muhammad"

The above 1D array can be created using the following code in OCR Reference Language:

```
array students[4]
students[0] = "Fletcher"
students[1] = "Imogen"
students[2] = "Tia"
students[3] = "Muhammad"
```

Note that the array index starts at 0, so an array of ten values would have indexes from 0 to 9.

Key point

Python does not have simple arrays by default. Instead, lists can be used which are similar in operation but have two key differences. First, lists can contain a mix of different data types. Second, lists are not of a fixed size and can be added to or reduced in size during the running of the program.

Arrays are commonly used with **FOR** loops to access each item in the array in turn. The code below uses a **FOR** loop to add up each element in an array of scores, where the array has eight items.

Array index	0	1	2	3	4	5	6	7
Data	5	7	0	10	8	3	7	3

```
total = 0
for i = 0 to 7
    total = total + scores[i]
next i
print(total)
```

This code would return a total of 43.

Two-dimensional arrays

A two-dimensional array is accessed using two index numbers. It can be represented using a table as shown in Table 2.2.2

Table 2.2.2 2D array showing scores for a game

	0	1	2
0	19	123	85
1	27	99	75
2	84	35	20
3	102	33	7

Tech term

Nested loop One loop that sits within another loop.

To access each value, two index numbers are required separated by a comma, for example `scores[0,2]`.

Key point

There is no one 'correct' way to represent a 2D array; it can be thought of as being accessed via [row, column] or [column, row]. A table is simply an abstract representation of how the data is stored in a 2D array. Any exam question using a table for this will tell you how to access the array.

Worked example

To populate this 2D array, the following **nested FOR** loops could be used.

```
array scores[3,4]
for x = 0 to 2
  for y = 0 to 3
    scores[x,y] = input("Enter a score")
  next y
next x
```

This code populates the [0,0] entry first, then the [0,1] entry, then [0,2], then [0,3]; then [1,0], [1,1], [1,2] and so on up until the final [2,3] entry.

Knowledge check

11 The contents of the array `fruit` is displayed below, where `fruit[1,3]` is 'Apple'.

	0	1	2	3	4
0	Pear	Grape	Banana	Damson	Orange
1	Raspberry	Blueberry	Blackcurrant	Apple	Grapefruit
2	Strawberry	Greengage	Lemon	Lime	Kiwi

- What is the value of `fruit[1,1]`?
- What is the value of `fruit[2,3]`?
- What is the array element for 'Blackcurrant'?

How to use subprograms to produce structured code

When programs grow in size, they can become hard to manage. Ideally, larger programs should be broken down into **subprograms** (sometimes called subroutines).

The advantages of using subprograms include:

- They reduce the overall size of the program as code does not need to be repeated in multiple places.
- They make the code much easier to maintain as it is easier to read and understand the purpose of each subprogram.
- They reduce development and testing time as code is much easier to write and debug.
- They allow reuse of code between programs, especially where pre-written and pre-tested subprograms can be used.

When a subprogram is called, control passes from the main program to the subprogram. Once the subprogram has completed, control is passed back to the main program.

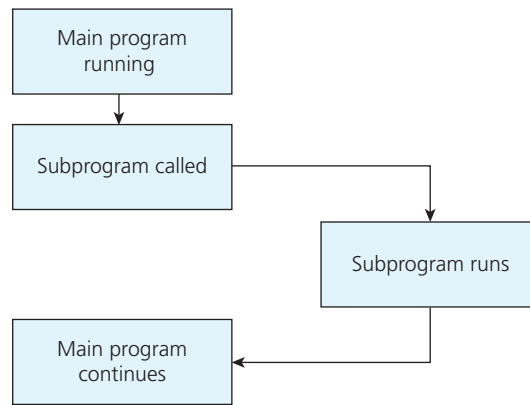


Figure 2.2.4 A subprogram being called from a main program

Subprograms can be divided into two types, **procedures** and **functions**.

Procedures

A procedure is a section of code that is defined outside the main body of the program. A procedure is given its own identifier, which is then used to **call** the procedure as many times as required.

Worked example



```
//procedure definition
procedure timestableuser()
    num = input("enter times table")
    for x = 1 to 10
        print(num*x)
    next x
endprocedure

//call the procedure to run
timestableuser()
```

Here, a procedure to produce a printed times table is defined once and given the identifier **timestableuser**. Once defined, the procedure can be called from within a program, as many times as required.

Note the use of brackets after the procedure's identifier. In the last example, these brackets were empty but they can be used to pass **parameters** into the procedure.

A parameter is a value that the procedure will use. We can rewrite the previous procedure to use parameters to control which times table will be produced and how many numbers to print out.

Worked example



```
//procedure definition
procedure timestable(tt, nums)
    for x = 1 to nums
        print(tt*x)
    next x
endprocedure

//call the procedure to run
timestable(8, 10)
timestable(9, 12)
```

Another procedure is defined, called **timestable**. This time it expects two parameters to be passed into the procedure in the form of variables **tt** and **nums**. When the procedure is called, the parameter values also have to be defined.

The first time the procedure is called, the values 8 and 10 are passed into the procedure. The second time the procedure is called, the values 9 and 12 are passed into the procedure. The first call will print out the 8 times table from 8×1 to 8×10 whereas the second call will print out the 9 times table from 9×1 to 9×12 .

Using this procedure is far more flexible and efficient than writing new code for each different times table.

Functions

A **function** is a type of subprogram that differs from a procedure in one key way: it **returns a value** back to the main program when it returns control. This value can then be stored, printed or otherwise used in the main program.

In every other way, a function is identical to a procedure; it is defined outside the main body of the program and can be called as many times as required.

To define a function in the OCR Reference Language, the **function** keyword is used.

Worked example



```
//function definition
function circle_area(radius)
    const pi = 3.14159
    area = pi*(radius^2)
    return area
endfunction

//calls to the function
new = circle_area(10)
print(new)
```

In this example a new function is defined called **circle_area(radius)**, which calculates the area of a circle. It requires a value for the circle's radius to be passed to it as a parameter.

Note that in the function definition the **return** keyword is used to define what is passed back to the main program – in this case the value of the circle's area. When the function is called, this value is then stored in a variable **new**. Note that the programmer could have chosen to do something else with the returned value for the area, such as print it out or use in a further calculation. This choice is made when the function is called.

Key point

Procedures and functions are both examples of subprograms. Both can accept parameters passed into them. Functions return a value to the main program whereas a procedure does not.

Knowledge check



- 12 Describe one difference between a function and a procedure.
- 13 Describe two benefits of using subprograms.

Random number generation

Random numbers can be generated by a programming language. For example, a game may require that a random score between one and ten is given when a certain action is completed.

A very common built-in function that you may have used in a high-level language is the **random()** function. Typically, an upper and lower limit to the number are passed in as parameters and a random value between these two limits is returned by the function.

In the OCR Reference Language, the **random(...)** function allows us to do this. Programmers can pass in the lowest and highest number required as arguments (in brackets):

```
x = random(1,5) //chooses a random integer between 1 and 5
```

```
y = random(20,30) //chooses a random integer between 20 and 30
```

Worked example



```
r = random(1, 10)
print(random(20, 50))
```

Here, the built-in random function is used in two different ways. The first use shows how the returned value can be assigned to a variable, to be used later on in the program. The second shows another random value being printed for the user. Note how this choice is made by the programmer after the value has been returned; it is not decided within the function itself.

RECAP AND REVIEW

2.2 PROGRAMMING FUNDAMENTALS

Important words

You will need to know and understand the following for the exam:

Variables
Constants
Assignment
Input
Output
Sequence
Selection
Boolean conditions
Iteration
Looping
Count-controlled iteration/loops
Condition-controlled iteration/loops
Arithmetic operators
Modulus
Quotient
Exponent
Operator precedence
Comparison operators
Boolean operators
Integers
Real numbers
Character
String
Casting
Slicing
Concatenation
Opening (files)
Reading (files)
Closing (files)
Writing (files)
Record
Field names
SQL commands: SELECT, FROM, WHERE
Array
Subprograms
Procedures
Functions
Parameters
Random numbers

2.2.1 Programming fundamentals

The use of variables, constants, assignments, inputs and outputs

A **variable** stores a single piece of data. It is a label for an allocated area of memory. The value of a variable can be changed during the execution of the program.

A **constant** is also a label for an allocated area of memory. Unlike a variable, the value of a constant cannot change during the execution of the program.

Variables and constant are given an identifier (or name). Their identifiers can be almost anything but must:

- not contain spaces
- not start with a number
- not be particular words reserved for use in the programming language.

Variables and constants are **assigned** values using the = operator. (Your chosen language might use a different symbol.)

Variables can be assigned new values throughout a program, which overwrites the previous value.

Constants can only be assigned a value once.

Variable examples:

```
x = 7
```

```
name = "Wilf"
```

Constant examples:

```
const maxScore = 100
```

```
const minimum = 0
```

Data can be **input** from the user and **output** back to the user. For example:

```
name = input("Enter your name")
```

```
print(name)
```

print is the output command used in the OCR Reference Language but this may be different in your chosen language.

The use of the three basic programming constructs to control the flow of a program

Sequence is the execution of statements one after the other, in order. A program runs from top to bottom and each instruction completes fully before the next one is executed.

Selection is the construct used to make decisions in a program. These decisions are based on **Boolean conditions** and the program then takes one of two paths based on this condition. Selection can be implemented using **IF** statements:

```
if x > 100 then
    print("Number is over 100")
endif
```

Selection can also be implemented using **SWITCH/CASE** statements:

```
switch colour:
    case "blue":
        print("blue is the colour of the sky")
    case "green":
        print("green is the colour of the grass")
    default:
        print("you picked a different colour")
endswitch
```

Note that not every language has **SWITCH/CASE** statements – for instance Python does not.

Iteration is the construct used to repeat sections of code. Iteration is commonly called **looping**.

Count-controlled iteration repeats code a defined number of times. **FOR** loops can be used to implement count-controlled iteration. A step can also be defined:

```
for y=1 to 100 step 5
    print y
next y
```

Condition-controlled iteration checks a condition each time around the loop and decides whether to repeat the code again or continue. **WHILE** loops and **DO UNTIL** loops can be used to implement condition-controlled iteration.

```
total = 0
while total < 100:
    x = input("enter a value")
    total = total + x
endwhile
```

Operators

A computer program uses operators to perform some sort of action.

Arithmetic operators can be used to carry out basic mathematical operations on numeric values.

Operator	Name	Example
+	Addition	$3 + 5 = 8$
-	Subtraction	$6 - 2 = 4$
*	Multiplication	$4 * 3 = 12$
/	Division	$6 / 3 = 2$
MOD	Modulus – returns the remainder after division	$r = 5 \text{ MOD } 2$ would give the result that $r = 1$
DIV	Quotient – returns the whole number after division	$q = 5 \text{ DIV } 2$ would give the result that $q = 2$
^	Exponent	$x = 2 ^ 4$ Gives the result that $x = 16$ (because $2 \times 2 \times 2 \times 2 = 16$)

Comparison operators are used to evaluate expressions to a Boolean True or False outcome.

Operator	Name
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Boolean operators allow multiple conditions to be evaluated.

The **AND** operator requires **both** conditions to be True for the overall condition to be True.

The **OR** operator requires **one or the other (or both)** of the conditions to be True for the overall condition to be True.

The **NOT** operator reverses the True or False outcome from a comparison.

2.2.2 Data types

The use of data types

Integers are **whole numbers**, positive or negative, that have no decimal or fractional part. For example, 99 is an integer.

The **real number** data type is used for numbers, positive or negative, that have (or may have) a **decimal or fractional** part. For example, 18.779 is a real number.

Boolean variables only ever store **True** or **False** values. For example, `True` is a Boolean value.

A **character** is a single item from the character set used on the computer. For example, `'@'` is a character. When assigning a character to a variable, quotation marks are required.

A **string** data type stores a collection of characters. For example, `'Hello world'` is a string. When assigning a string to a variable, quotation marks are required.

Casting is the conversion of one data type to another. Not all data can be cast to another data type.

OCR Reference Language keyword	Converts to ...
<code>str()</code>	String
<code>int()</code>	Integer
<code>real()</code>	Real
<code>bool()</code>	Boolean

2.2.3 Additional programming techniques

Basic string manipulation

String manipulation tools allow strings to be sliced, concatenated or modified.

- **String slicing** means to extract individual characters from strings.
- **String concatenation** means to join strings together.

Keyword	Use
<code>.length</code>	To count how many characters are contained in a string.
<code>.substring(x,y)</code>	To extract characters from the middle of a string where x is the starting point (beginning at 0) and y is the number of characters required.
<code>.left(x)</code>	To extract characters from the left of a string, where x is the number of characters required.
<code>.right(x)</code>	To extract characters from the right of a string, where x is the number of characters required.
<code>.upper</code>	To convert a string to UPPERCASE.
<code>.lower</code>	To convert a string to lowercase.
<code>ASC()</code>	To find the ASCII value of a character.
<code>CHR()</code>	To find the character that relates to the ASCII value given.

Basic file handling operations

Files can be written to and read from by a program.

They must be **opened** before they can be used with the keyword `open()`.

The contents of the file can be **read** line by line using the keyword `.readLine()`.

To write a line at the end of the file we use the keyword `.writeLine()`.

The `.endOfFile()` keyword returns a Boolean value for whether the file is at the end or not. Files must be **closed** at the end of their use with the keyword `.close()`.

Keyword	Use
<code>open()</code>	<code>contents = open("textfile.txt")</code> Opens textfile.txt into the variable contents
<code>.readLine()</code>	<code>data = contents.readLine()</code> Reads a single line of data from contents
<code>.writeLine()</code>	<code>contents.writeLine("Add a new line of data")</code> Writes a single line of data at the end of the file
<code>.endOfFile()</code>	<code>contents.endOfFile()</code> Returns TRUE if the last line of contents has been reached and FALSE if not.
<code>.close()</code>	<code>contents.close()</code> Closes the file

The use of records to store data

A **record** is a data structure that allows multiple data items to be stored, using **field names** to identify each item of data.

Data is organised using field names and stored in a table.

The use of SQL to search for data

SQL (Structured Query Language) is a language used to access data stored in a database.

- **SELECT** identifies the fields to return from the database
- **FROM** identifies which table(s) the data will be returned from
- **WHERE** is an **optional command** that allows the programmer to include criteria, with only matching records being returned.

For example:

```
SELECT names, points, DOB
```

```
FROM games
```

```
WHERE points > 100
```

The use of 1D and 2D arrays

A **one-dimensional array** allows a programmer to store **multiple items of data** under a single identifier.

```
array words[5]
```

```
words[0] = "hello"
```

```
words[1] = "world"
```

Array index	0	1	2	3	4
Data	"hello"	"world"			

A **two-dimensional array** allows a programmer to store **multiple items of data** using two identifiers.

```
array words[5,4]
```

```
words[0,4] = "hello"
```

```
words[3,2] = "world"
```

	0	1	2	3	4
0					"hello"
1					
2					
3			"world"		

Any exam question using a table for this will tell you whether you access the array as [row, column] or [column, row].

How to use subprograms to produce structured code

Subprograms allow programs to be split up into multiple sections. This makes the code easier to read and maintain, reduces the size of the code and promotes the reuse of code without copying and pasting.

Procedures are a type of subprogram that do not return a value to the main program.

Functions are a type of subprogram that return a single value to the main program.

Both procedures and functions need to be defined:

```
procedure add(x, y) // definition of procedure called add
```

```
    total = x + y
```

```
    print(total)
```

```
endprocedure
```

```
function addition(x, y) // definition of function called addition
```

```
    total = x + y
```

```
    return total
```

```
endfunction
```

The **return** keyword must always be used when defining a function, in order to state what value the function will send back to the main program.

Once defined, procedures and functions can be called at any point in the main program and **parameters** passed into them in brackets:

```
call add(1,2) // call the procedure add using the numbers 1 and 2.
```

```
call addition(2,3) // call the function addition using the numbers 2 and 3
```

Random number generation

Random numbers can be generated using the **random()** keyword.

```
print(random(1, 100)) // prints a random integer between 1 and 100
```

PRODUCING ROBUST PROGRAMS

CHAPTER INTRODUCTION

In this chapter you will learn about:

2.3.1 Defensive design

- Defensive design considerations
- Input validation
- Maintainability

2.3.2 Testing

- The purpose of testing
- Types of testing
- Identifying syntax and logic errors
- Selecting and using suitable test data
- Refining algorithms

Tech term

Robust program A program that functions correctly under less than ideal conditions.

It is not enough for a programmer to just ensure that their programs work correctly for them; what happens when the software is used by a different user, with different input data or on different hardware? A **robust program** will handle all of these changes and still function correctly.

In order to be as robust as possible, it is important that programs incorporate **defensive design** and also undergo thorough **testing** before being released.

All examples given in this chapter are written in the OCR Reference Language.

2.3.1 Defensive design

Defensive design considerations

Defensive design involves thinking about problems that could potentially occur and nullifying them before they happen. For example, if a user is asked for a number between one and ten, how do we know that they will actually enter a number at all, let alone one of the correct magnitude? A well-designed program that applies defensive design will check both of these things.

Anticipating misuse

Misuse of a program could be deliberate (users looking for ways to hack into a system by making it crash) or accidental (users not understanding what they are supposed to enter or accidentally entering the wrong data). However, we must deal with these as if they were the same.

When creating a program, think about the actions, either deliberate or accidental, that would cause the program to fail. By **anticipating misuse** in this way, we can then work out how to deal with the issues and produce a robust program.

Worked example

The following is a program for a banking application that allows the user to withdraw money from their account.

```
balance = 100
withdraw = input("enter amount to withdraw in £")
balance = balance - withdraw
print ("your balance is now", balance)
```

We could check this program works if the balance is £100 by withdrawing £20 – we would expect to see a balance of £80 remaining.

However, if we anticipate misuse, some potential issues become clear:

- Should we be allowed to withdraw more than our balance?
- What if we withdraw a negative balance? This will effectively add money to our account.
- What if the user enters an amount in words, such as 'TEN POUNDS'? Will this cause the program to crash?
- What if the user enters numbers with the preceding '£' sign? Will this cause the program to crash?

All of these issues would need to be dealt with by the programmer before the program could be said to be robust.

Key point

Authentication can be typically reduced to three areas: what you **know** (passwords), what you **have** (devices and keys) and what you **are** (biometrics).

Beyond the spec

Passwords should never be stored by computer systems as this would be very dangerous – if a hacker was able to get access to the database then they would know everyone's passwords. Instead, a hash of each password is stored; this is a mathematical one-way process that returns a very long unique number for each password. When a user enters a password, this is hashed and compared with the hash stored in the database; if the two match, the user can be authenticated without ever having their password stored.

Authentication

Some computer systems, such as online shopping websites, are available to all users on an anonymous basis. Users can search for products without giving any personal details. However, in order to do any more than this, users are required to be authenticated. Some computer systems require authentication to use the system at all.

Authentication is the process of establishing a user's identity. How can the computer system be sure that the user is who they say they are? Users can be authenticated in many ways:

- **Username and passwords** are perhaps the most common method. A username and a secret password are chosen by the user and when these are entered a computer system can check that they match those of a known user.
- **Possession of an electronic key, device or account** is used as authentication since only one person will have access to that particular device. Some computer systems will check that an email address or phone number belongs to you by sending an email or text containing a secret code.
- **Biometrics** is the use of measurements relating to biological traits. If your school uses fingerprint scanners to identify you, you will have experience of this. Banks are increasingly using voice recognition to authenticate users who use telephone services.

Two-factor authentication is where two of the above are checked simultaneously. For example, you may log in to a system with a username and password (1st method of authentication) and are immediately then sent a text message or email to respond to (2nd method of authentication).

Knowledge check



- 1 What is meant by the term 'defensive design'?
- 2 Explain two defensive design considerations.
- 3 What is meant by 'two-factor authentication'?

Tech term

Presence check A data validation method that prevents users from leaving a field blank.

Key point

Validation cannot check that data is correct, only that it follows certain rules. For example, a programmer could validate that a phone number entered is made up entirely of numbers and starts with a 0. If a phone number of 123ABC was entered, this would be rejected as invalid. However, if a user entered a valid phone number but mistakenly entered the wrong digits, this would not be identified.

Input validation

A major source of potential problems in a computer program is invalid data entered by the user. **Input validation** is the process of checking input data against rules defined by the programmer to ensure that the data is sensible and as expected.

Data can be validated to ensure that is:

- of the correct **type**, such as integer or string
- within a sensible **range**, such as between 1 and 100
- of the correct **length**, such as 8 characters or more
- **present**, to stop users leaving certain information empty; this is known as a **presence check**.

Worked example



The following program shows the previous banking application program, but this time the input has been partially validated. The user cannot enter a value below 0 and cannot withdraw more than their current balance. The **WHILE** loop ensures that they are continually asked for a withdrawal amount until the value entered is validated successfully.

```
balance = 100
withdraw = 0
while (withdraw <= 0) or (withdraw > balance)
    withdraw = input("enter amount to withdraw in £")
    if withdraw < 0 then
        print("You must withdraw a positive amount")
    elseif balance < 0 then
        print("You are overdrawn!")
    elseif withdraw > balance then
        print("You cannot withdraw more than your balance")
    endif
endwhile
balance = balance - withdraw
print("your balance is now", balance)
```

Of course, there would be more to do; the user can still leave the input amount blank or enter a non-numeric value.

Knowledge check



- 4 State what is meant by validation.
- 5 Suggest two rules that could be used to validate a date of birth entered by a user.
- 6 Explain why someone's name could not easily be validated.

Maintainability

Over time, programs may need to be modified as the requirements of users change. Errors may also be spotted that require fixing by programmers who did not write the original code. A **maintainable** program is one that the original programmer has deliberately made straightforward to understand and modify. This can be achieved in the following ways.

Use of subprograms

By splitting programs down into multiple **subprograms**, code repetition can be reduced and the overall number of lines of code can be smaller. This also means that important values and processes can be modified in one place in the code without having to find every time that process is used.

Worked example



Example A

```
price1 = 100 * 1.2
print ("the price is £",price1)
price2 = 87 * 1.2
print ("the price is £",price2)
price3 = 35 * 1.2
print ("the price is £",price3)
price4 = 17 * 1.2
print ("the price is £",price4)
price5 = 99 * 1.2
print ("the price is £",price5)
price6 = 400 * 1.2
print ("the price is £",price6)
```

Example B

```
procedure calculate(price)
    newprice = price * 1.2
    print ("the price is
    £",newprice)
endprocedure

calculate(100)
calculate(87)
calculate(35)
calculate(17)
calculate(99)
calculate(400)
```

In the examples above, both examples take six prices and add 20% in tax before printing the value. However, example A achieves this through copying and pasting the code whereas example B uses a subprogram.

Example B is not only shorter but is much easier to follow through. Crucially, if the tax rate changes from 20%, in example B it only needs to be changed in one place. This makes the program much more maintainable.

Beyond the spec

Variable and subprogram identifiers with multiple words have always been problematic, as spaces are not allowed. One solution is to use **camelCase**, a naming convention where no spaces are used but new words are indicated by capital letters. Therefore, 'current pay rate' would be **currentPayRate** and 'total sales' would be **totalSales**. Note that the first letter is always in lowercase. An alternative would be **snake_case**, where spaces are replaced by underscores, giving **current_pay_rate** and **total_sales**.

Naming conventions

Using meaningful variable and subprogram identifiers helps to identify their purpose. Suitable **naming conventions** will allow future programmers to see much more clearly how a program works.

A programmer will be unsure what a variable called **x**, **temp** or **abc** should contain – but variables called **username**, **pointsScored** or **loggedIn** give a very clear indication of what data they store.

Indentation

Indentation is a way of formatting a program to highlight the structure of the code. This makes it easier for a programmer to read through the code and follow the process. Indentation is used to show that code belongs inside a particular block.

The TAB key or a number of spaces (usually three or four) can be used to indent lines of code, but this should be done consistently.

Key point

Note that Python uses indentation differently to most other languages. Python insists that code is indented to function correctly, whereas most other languages, such as Visual Basic or C#, use optional indentation.

Worked example

The following program would work but is hard to follow:

```
score = 100
while score > 0
new = input("enter a value")
if new > 10 then
score = score - new
endif
endwhile
print("complete")
```

Which lines of code will be repeated inside the **WHILE** loop? If the value entered is greater than ten, which lines of code will be executed as a result?

The following program is identical but includes indentation to make it much easier to follow:

```
score = 100
while score > 0
    new = input("enter a value")
    if new > 10 then
        score = score - new
    endif
endwhile
print("complete")
```

Commenting

Comments are lines of code that are ignored by the translator when the program is run. They are extremely useful for programmers to identify the purpose of each section and how the code is intended to work.

OCR Reference Language uses `//` to indicate a comment. Other high-level languages may use other symbols such as `#` or `'`.

Worked example

The code shown in green below is ignored by the translator. It is added solely to allow programmers to understand how the program works. This makes the code much more maintainable.

```
function volume(height,width,depth)
    //function to calculate the volume of a cuboid
    return (height * width * depth)
endfunction
```

Knowledge check

- 7 Describe why maintainability is so important to computer programmers.
- 8 Give two rules to be followed when sensibly naming a variable.
- 9 Give two other ways that a program can be made to be more maintainable.
- 10 Which of the following lines of code would be most helpful to a programmer trying to work out how a program works? Explain why.

```
w = (y*t) + b
totalWages = (hoursWorked * payRate) + bonus
```


2.3.2 Testing

No matter how well written a program is, there is always the chance that errors have crept in. **Testing** allows us to systematically check that a program functions as it should in all circumstances.

Tech term

Destructive testing

Instead of simply checking whether a program works as intended, destructive testing actively tries to find ways to break the program.

The purpose of testing

The purpose of testing is to ensure that the program functions as expected and meets all requirements. However, testing should also be **destructive**; that is, we should not simply aim to prove that the program works, but we should also try to do all we can to break it. Only by knowing that it cannot easily be broken can we be satisfied that it works.

For instance, if we create a program for a hot drinks machine that should give us a selection of drinks for £1 each, is it enough to test it by inserting £1 and pressing the corresponding drink button? This is a starting point, but if we only relied on this test we may not realise that someone is able to get a drink without inserting any money. Only by testing it destructively and trying to see if there is any other way of getting a drink for less than £1 can we be sure that it works fully.

Types of testing

There are many types of testing, but only two are covered in the OCR GCSE Computer Science specification: iterative testing and final/terminal testing. Both are important for different reasons.

Tech term

Modular Subdividing into small independent sections.

Iterative

Iterative testing is the process of testing each part of the program as it is developed, focusing exclusively on how that section works. Testing on a **modular** basis like this allows a programmer to be sure that any new code functions correctly by itself, interacts properly with other modules and does not introduce new errors.

For example, if a 2D game were to be developed, the first **module** to be written may be the code to display the user's character and move it around on screen. This code would then be tested thoroughly to ensure that it works before moving on to develop the next module.

Iterative testing is repeated for each new module that is developed.

Final/terminal

Final (or terminal) testing is completed near to the end of development; the whole program is tested for functionality and the individual detail of each module is ignored. The point of final testing is to check whether the system as a whole works correctly.

For example, final testing of a game would involve playing each level to try and ensure that the game contains as few errors as possible.

Beyond the spec

Final testing is sometimes referred to as 'black box testing' because it treats a program as a 'black box' with no interest in what happens inside the box or how it works. The only concern is whether the inputs and outputs work as expected.

'White box testing' is the opposite of black box testing – the only interest is what is happening 'inside the box', that is, how each of the modules work. White box testing is an alternative name for iterative testing.

Knowledge check



- 11 What is meant by 'destructive testing'?
- 12 State the purpose of testing.
- 13 Explain one difference between iterative testing and final testing.

Identifying syntax and logic errors

If errors are found, they could either be **syntax errors** or **logic errors**. A syntax error breaks the **grammatical rules** of the programming language in some way, such as missing off a quotation mark, misspelling a keyword or using assignment incorrectly. A syntax error will cause the program to stop running (or not run in the first place) because the translator does not understand the instruction given.

Worked example



The code below contains a number of **syntax errors**.

```
num = input("enter a number
10 = x
pritrn(num + x)
```

First, the input statement on the first line is missing the closing quotation mark and closing bracket. Next, the assignment statement on the second line is the wrong way around (it should be `x = 10`). Last, the keyword print has been misspelled as **pritrn**. Any one of these errors would stop the program from running and would produce a syntax error.

A **logic error** by comparison is an error in the algorithm that does not stop the program from running but does not produce the correct output. This is usually caused by the programmer writing instructions that have the correct syntax but do not do what was intended.

Worked example



The code below contains a logic error.

```
function addup(a, b)
    return a * b
endfunction
```

The intention of the function is to add up the two numbers passed in as parameters. However, the function instead multiplies the two numbers. This is a valid instruction and would not cause the program to stop, but it is certainly not what the programmer intended.

Key point

Syntax errors are errors relating to the rules of the programming language. A program containing a syntax error will not run and hence it is clear that there is an issue with it. Programs containing **logic errors** do run but do not produce the desired output. Logic errors are harder to spot because it is not always immediately clear that there is an issue with the program.

Knowledge check



- 14 Give one difference between a logic error and a syntax error.
- 15 A programmer writes the line `IF x > FOR`. Explain whether this would cause a logic or syntax error.
- 16 A programmer finds that a function to calculate values does not work properly when negative values are passed in as parameters. Explain whether this would be caused by a logic or syntax error.

Selecting and using suitable test data

To test a program effectively, a **test plan** is needed. This plan lists all of the tests that will be carried out, the **test data** to be used and the expected outcome. Test data should cover as many of the following situations as possible.

Normal

Normal test data is data of the correct type that would typically be expected from a user who is correctly using the system. This should be accepted by the program without causing errors.

Boundary

Boundary test data is data that is of the correct type but is on the very edge of being valid. Boundary test data should be accepted by the program without causing errors.

Invalid

Invalid test data is data that is of the correct type but outside the accepted limits. Invalid test data should be rejected by the program.

Erroneous

Erroneous test data is data that is of the incorrect type and should be rejected by the system. For example, if a program expected numeric input, a string would be erroneous input.

Consider this example: A system allows a user to enter a value between 0 and 100, with the number being rounded up or down to the nearest ten. Any numbers outside the range 0 to 100 should be rejected.

There are many possible tests but a typical **test plan** could be as shown in this table.

Test data	Type of test data	Reason	Expected result
47	Normal	To check if values round up	50
32	Normal	To check if values round down	30
0	Boundary	To check the low boundary	0
100	Boundary	To check the high boundary	100
-1	Invalid	To check if numbers below 0 are rejected	Rejected
101	Invalid	To check if numbers above 100 are rejected	Rejected
'Twelve'	Erroneous	To check data of the wrong type is rejected	Rejected

Key point

Test data should be listed on a test plan using the actual data that would be entered. A very common mistake is to simply describe the test data (such as 'a number larger than 100'). This is not specific enough; instead an actual value (e.g. 101) should be included in the test plan.

Knowledge check



- 17 State what is meant by normal test data.
- 18 Describe how boundary test data is different from invalid test data.
- 19 A system should allow passwords that are between eight and 15 characters in length. Suggest one suitable item of invalid test data for this system.
- 20 Complete the following test plan for a system that checks if users are 18 years of age or older. Anyone younger than 18 should be rejected. Anyone 18 or over should be accepted.

Test data	Type of test data	Expected result
	Normal	
	Boundary	
	Invalid	
	Erroneous	

Refining algorithms

Refining an algorithm means to improve it. If testing has picked up any errors, an obvious improvement would be to fix the problem.

Worked example

The code below should allow values between one and ten.

```
num = input("enter a value between 1 and 10")
if num > 1 and num < 10 then
    print("Allowed")
else
    print("Not allowed")
endif
```

When this code is tested thoroughly, a number of errors are discovered. These are shown in the test plan below.

Test data	Type of test data	Reason	Expected result	Actual result
5	Normal	Check valid data	Allowed	Allowed
1	Boundary	Check low extreme of range	Allowed	Not Allowed
10	Boundary	Check high extreme of range	Allowed	Not Allowed
0	Invalid	Check low outside range	Not allowed	Not allowed
11	Invalid	Check high outside range	Not allowed	Not allowed

The code can then be refined to ensure that it works on these boundaries and then tested again.

```
num = input("enter a value between 1 and 10")
if num >= 1 and num <= 10 then
    print ("Allowed")
else
    print ("Not allowed")
endif
```

Another way that code could be refined would be to make it more **efficient**. Code that repeats may benefit from being split into subprograms or by using iteration to reduce this repetition. Comments, indentation and naming of variables can all be used to refine a program and make it more maintainable. Subprograms (and functions in particular) can also be used to make the code far more reusable in the future.

Key point

Refining an algorithm means to improve it. This may be an improvement that makes it function correctly or just to be more efficient or maintainable.

Tech term

Efficient A more efficient program is one that produces exactly the same results with fewer lines of code.

Knowledge check

- 21 State what is meant by the term 'refine' in relation to an algorithm.
- 22 Describe two issues that mean an algorithm would need to be refined.

RECAP AND REVIEW

2.3 PRODUCING ROBUST PROGRAMS

Important words

You will need to know and understand the following for the exam:

- Defensive design
- Anticipating misuse
- Authentication
- Two-factor authentication
- Input validation
- Maintainability
- Subprograms
- Naming conventions
- Indentation
- Commenting
- Testing
- Iterative testing
- Module
- Final testing
- Terminal testing
- Syntax errors
- Grammatical rules
- Logic errors
- Test data
- Normal test data
- Boundary test data
- Invalid test data
- Erroneous test data
- Test plan
- Refining algorithms

A **robust program** is one that functions correctly even if the input data, intended use or situation changes. **Defensive design** and **testing** help to make programs robust.

2.3.1 Defensive design

Defensive design means thinking ahead about problems that could occur. It involves the following areas.

Defensive design considerations

- **Anticipating misuse** is thinking about ways that users could cause the program to fail. This could be either deliberate or accidental on the user's behalf. These potential problems can then be dealt with.
- **Authentication** is checking the identity of a user. This could be done by using a **username and password**, through possession of a unique **electronic key** or through **biometrics** (biological measurements).

Input validation

Input validation is checking whether data matches certain rules as it is input. These rules can check that data is sensible but can never show that the data is correct. For example, **ST13 7TY** would be a sensible postcode and meets the defined rules, but it is not possible for a computer to easily decide whether it is actually that user's postcode.

Maintainability

Maintainability is ensuring that a program is as easy to understand and modify as possible for other future programmers. It involves the following:

- Use of **subprograms** is covered more in Chapter 2.2 (Programming Fundamentals). Splitting programs down into

multiple subprograms reduces the need to copy and paste code, therefore making the overall code shorter and easier to follow.

- Variables and subprograms should use sensible **naming conventions**. Variables and subprograms should be given identifiers that reflect their purpose. Identifiers cannot contain spaces and cannot start with a numeric value.
- **Indentation** is the use of the tab or space key to align code in a program. Indenting is used to highlight the structure of code and show which code belongs inside a particular block. Code after **IF** statements and iteration in particular should be indented.
- **Python requires correct indentation for a program to function correctly. It is optional but highly recommended in other high-level languages.**
- **Commenting** allows programmers to add notes to their program to explain what sections of code do or how they work. These comments are ignored by the translator when the program is executed.

2.3.2 Testing

The purpose of testing

The purpose of **testing** is to ensure that a program functions as expected and meets all requirements. Testing should be destructive and aim to find errors, not just show that the program works in one situation.

The types of testing

Iterative testing is testing during the development of a program. Each **module** is thoroughly tested as it is completed. This type of testing is repeated for future modules.

Final/terminal testing is completed near to the end of program development. It is testing the program as a whole for functionality.

Identify errors

A **syntax error** is one that breaks the **grammatical rules** of the programming language. Examples include misspelling a keyword, missing a bracket or using a keyword in the wrong way. Syntax errors will stop the program from running when encountered.

A **logic error** is one that causes the program to produce an unexpected or incorrect output but will not stop the program from running.

Selecting and using suitable test data

Test data should be chosen so that the system as a whole can be tested destructively, checking for errors wherever they may occur. Test data should be chosen to include as many of the following as possible:

- **Normal test data** is data of the correct type that would typically be expected from a user who is correctly using the system. This should be accepted by the program without causing errors.
 - **Boundary test data** is test data that is of the correct type but is on the very edge of being valid. Boundary test data should be accepted by the program without causing errors.
 - **Invalid test data** is test data that is of the correct type but outside the accepted limits. Invalid test data should be rejected by the program.
 - **Erroneous test data** is test data that is of the incorrect type and should be rejected by the system. For example, if a program expected numeric input, a string would be erroneous input.
- A **test plan** lists all of the tests that will be carried out, the expected result and the actual result in each case. For example:

Test data	Type of test data	Reason	Expected result	Actual result
	Normal			
	Boundary			
	Invalid			
	Erroneous			

Refining algorithms

Refining algorithms means improving them, either to fix errors found by testing or to make them more efficient and maintainable using the methods described above.

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

BOOLEAN LOGIC

CHAPTER INTRODUCTION

In this chapter you will learn about:

2.4.1 Boolean logic

- Truth tables
- Simple logic diagrams using the operators AND, OR and NOT
- Combining Boolean operators using AND, OR and NOT
- Applying logical operators in truth tables to solve problems

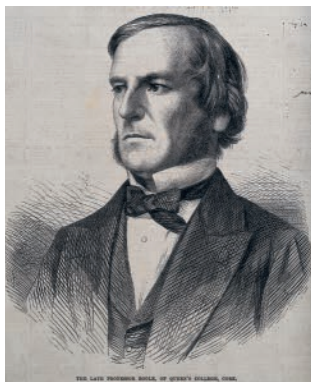


Figure 2.4.1 George Boole the inventor of Boolean logic

Tech term

Boolean data type

Data that can only have the values TRUE (1) or FALSE (0).

2.4.1 Boolean logic

George Boole (1815–1864) was an English mathematician who identified that all logical decisions could be reduced down to simple True and False values. In computer science, the **Boolean data type**, which can only be True or False, is named after him.

Computers store everything using electronic switches known as transistors, with on and off represented by the binary values 1 and 0. These same 1s and 0s can also be used to represent the True and False values of Boolean logic.

Key point

When discussing Boolean logic, 1 and 0 are usually used but you may see True and False, T and F or even On and Off used. We will use 1 and 0, which is also how any questions in your examination will be presented.

Truth tables

A **truth table** is a table used to display all possible inputs and associated outputs from a logic system. Inputs are usually labelled with the letters from the beginning of the alphabet and the output labelled using the letter P or Q.

As an example, consider the situation of learning to drive, where both a practical driving test and a theory test are needed in order to be given a driving licence. We can assign the input A to 'passes the practical test', input B to 'passes the theory test' and output P to 'can be given a driving licence'.

The table below shows a truth table for this logic system:

Practical test (A)	Theory test (B)	Driving licence (P)
0	0	0
0	1	0
1	0	0
1	1	1

Note how every possible combination of inputs is listed and the associated output for those inputs given. We can see that if someone fails both tests, they cannot be given a driving licence. If they pass one of tests but fail the other, they are still unable to be given a driving licence. However, if both tests are passed then a driving licence can be given.

Some logic systems have more than two inputs. For example, a logic system with four inputs would be labelled with inputs A, B, C and D. The more inputs, the more rows are needed in the truth table; a 4-input truth table would have 16 possible permutations of input values.

Simple logic diagrams using the operators AND, OR and NOT

Logic gates are circuits within a computer that produce a Boolean (i.e. 1 or 0) output when given Boolean inputs. There are three logic gates that we need to know about for GCSE: **NOT**, **AND** and **OR**.

NOT gate

The **NOT gate** is sometimes known as an inverter. It only has one input and it outputs the opposite of the value input. If a 0 is supplied as input then a 1 will be produced as output. Conversely, if a 1 is supplied then a 0 will be output. This is shown in the truth table below.

A	P
0	1
1	0

The diagram used to denote a NOT gate is shown in Figure 2.4.2.

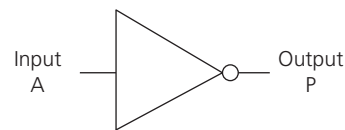


Figure 2.4.2 A NOT gate

AND gate

An **AND gate** has two inputs and only produces a 1 as output if **both** of the inputs are 1s. If any other combination of inputs is given then the output will be 0.

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

The diagram used to denote an AND gate is shown in Figure 2.4.3.

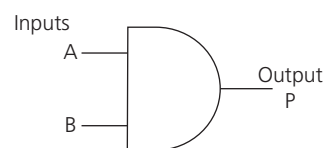


Figure 2.4.3 An AND gate

OR gate

An **OR gate** has two inputs and produces a 1 as output if **either or both** of the inputs are 1s. If both inputs are 0 then the output will be 0.

A	B	P
0	0	0
0	1	1
1	0	1
1	1	1

The diagram used to denote an OR gate is shown in Figure 2.4.4.

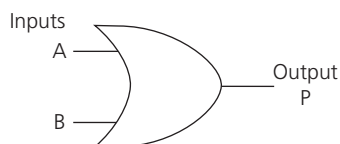


Figure 2.4.4 An OR gate

An easy way to remember the purpose of the three logic gates is to think about what inputs produce a 1 output.

- NOT requires the input **NOT** to be 1
- OR requires inputs A **OR** B (or both!) to be 1
- AND requires inputs A **AND** B to be 1.

Combining Boolean operators using AND, OR and NOT

AND and NOT gates

The three logic gates shown can be combined into more complex logic systems. For example, we could connect a NOT gate to the output of an AND gate.

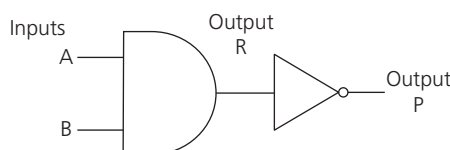


Figure 2.4.5 An AND gate connected to a NOT gate

The output for this can be worked out in stages. It is useful to think about the output of each gate in turn and we can even add this to our truth table. Here, R is the output from the first part of the logic diagram and P is the final output.

A	B	R = (A AND B)	P = (NOT R)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Another way of describing this system is: **NOT (A AND B)**.

AND and OR gates

This logic system has three inputs, with the output of the AND gate feeding into one input of an OR gate. An additional input C is also given.

Another way of describing this system is: **(A AND B) OR C**.

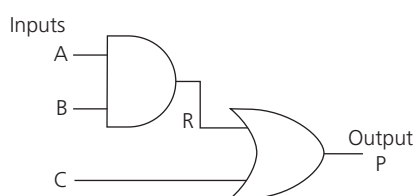


Figure 2.4.6 An AND and OR gate

The truth table now requires more rows to cater for all of the possible input values.
The first thing you need to do is fill in all of the different permutations of the inputs A, B and C.

A	B	C	R = (A AND B)	P = (R OR C)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Then you need to fill in the values for R for all permutations of A and B.

A	B	C	R = (A AND B)	P = (R OR C)
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	1	

Once the R column is completed you can then fill in the values for P for all the permutations of R or C.

A	B	C	R = (A AND B)	P = (R OR C)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Key point

To make sure that you don't miss out any permutations of input values, try counting up in binary, starting at 000 (if you have three inputs), then 001, 010, 011 and so on.

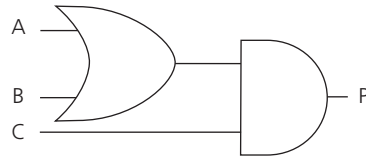
Another check is that with **n** inputs, there should always be **2ⁿ** rows in the truth table (not counting the heading). This means that for 3 inputs, we need $2^3 = 2 \times 2 \times 2 = 8$ rows.

Knowledge check

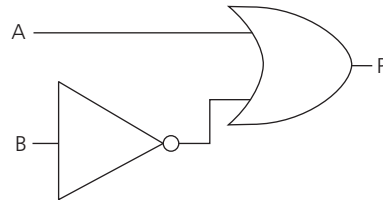


Create truth tables for the following expressions.

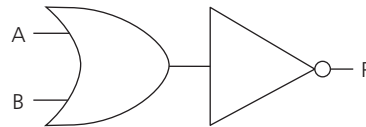
1 $P = C \text{ AND } (A \text{ OR } B)$



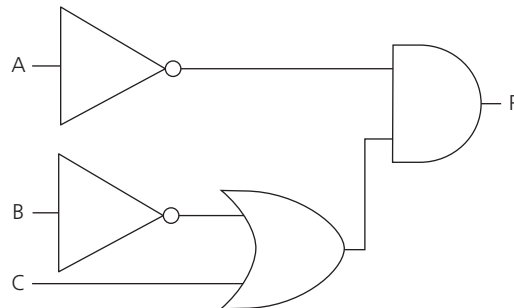
2 $P = A \text{ OR } (\text{NOT } B)$



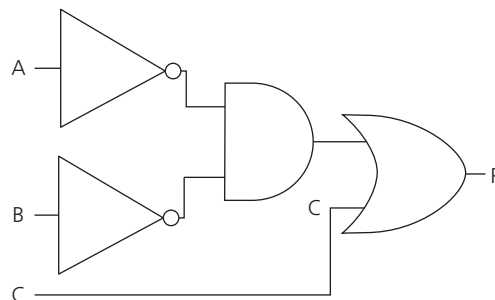
3 $P = \text{NOT } (A \text{ OR } B)$



4 $P = \text{NOT } A \text{ AND } (\text{NOT } B \text{ OR } C)$



5 $(\text{NOT } A \text{ AND } \text{NOT } B) \text{ OR } C$



Creating logic diagrams from expressions

So far, we have been given logic diagrams to work from. However, it is important that we can draw logic diagrams when given the equivalent expression.

Take the example $P = \text{NOT } (A \text{ AND } B) \text{ AND } C$.

As with mathematics, use of brackets help us to decide what has priority. In this case, **A AND B** is an expression in brackets and therefore should be looked at first. The logic diagram for this is shown in Figure 2.4.7.

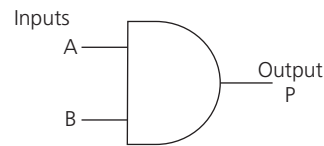


Figure 2.4.7 A AND B

Next, the **NOT** gate should be applied to this expression to give us **NOT (A AND B)**. The logic diagram for this is shown in Figure 2.4.8.

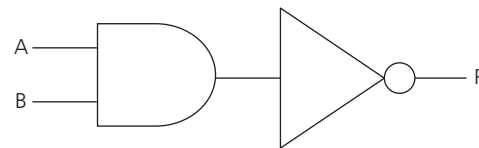


Figure 2.4.8 NOT (A AND B)

Note that the NOT gate is applied to the output of the expression **A AND B**, not on either of the inputs.

Finally, the expression is complete by joining the output of **NOT (A AND B)** in the previous stage to **AND C**. This gives us **NOT (A AND B) AND C** as shown in the logic diagram in Figure 2.4.9.

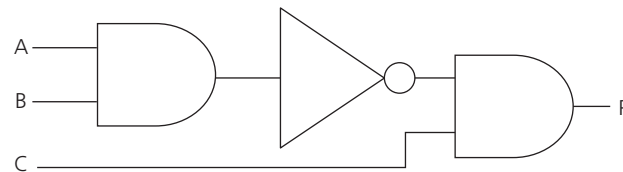


Figure 2.4.9 NOT (A AND B) AND C

The truth table for this logic diagram is:

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

Knowledge check



Create logic diagrams for the following expressions:

- 6 $P = A \text{ OR } (\text{NOT } B)$
- 7 $P = A \text{ OR } (B \text{ OR } C)$
- 8 $P = A \text{ AND } (\text{NOT } B)$
- 9 $P = (A \text{ OR } B) \text{ OR } (C \text{ OR } D)$

Applying logical operators in truth tables to solve problems

Reducing real-life problems down to Boolean logic statements can help us to decide what inputs are needed to produce certain outputs.

Imagine a family looking to book a holiday. They have a budget of £2000 for the holiday and they would like this to be somewhere with a pool. However, if a holiday in the USA was available at this price, with or without a pool, they would be happy.

We can construct this situation as a logic statement.

Let **P** be the outcome of being happy with the holiday. The inputs can also be defined :

- **A** = costs £2000 or less
- **B** = has a pool
- **C** = is in the USA

A must be met; the family only have this in their budget to spend. However, either **B** or **C** can be true for the family to be happy.

This situation is equivalent to the logic statement **P = A AND (B OR C)**.

The truth table for this logic statement is therefore:

A	B	C	B OR C	P = A AND (B OR C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

This then allows us to apply these logical rules to check if certain holidays are appropriate.

Knowledge check



- 10 Find the row in the above truth table that relates to each of these holidays:
 - (a) Costs £1800 in Spain with a pool
 - (b) Costs £2500 in the USA with no pool
 - (c) Costs £1996 in France with no pool
- 11 Give the details of one further holiday that would give a TRUE (1) output from this logic system.
- 12 Draw the logic gates that match the logic statement **P = A AND (B OR C)**
- 13 The family go to the cinema. They would be happy if the film is NOT an 18 certificate and lasts less than 2 hours. Create a truth table for this situation where A = the film is an 18 certificate and B = lasts less than 2 hours.

RECAP AND REVIEW

2.4 BOOLEAN LOGIC

Important words

You will need to know and understand the following for the exam:

Truth table

Boolean operators – NOT, AND, OR

NOT gate

AND gate

OR gate

2.4.1 Boolean logic

Boolean logic uses two values – True and False. These are represented in computer systems using binary 1 and 0 values.

Truth tables

Truth tables show all possible input permutations and the corresponding outputs for a logic system.

If a logic system has n inputs, it will have 2^n possible input permutations. This equals the number of rows in the truth table, for example with three inputs it will have $2^3 = 2 \times 2 \times 2 = 8$ rows.

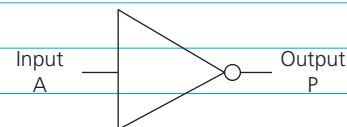
Inputs are labelled with letters from the start of the alphabet; outputs are typically labelled as P, Q and later letters from the alphabet.

Logic diagrams

NOT gate

A **NOT gate** reverses the input given to it. If a 0 is input, a 1 will be output and vice versa. It is sometimes known as an **inverter** for this reason.

A	P = (NOT A)
0	1
1	0

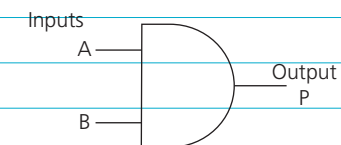


A NOT gate

AND gate

An **AND gate** gives a 1 output only if both inputs are 1s. Any other inputs (0,0 / 0,1 / 1,0) give a 0 output.

A	B	P = (A AND B)
0	0	0
0	1	0
1	0	0
1	1	1

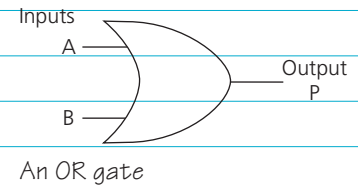


An AND gate

OR gate

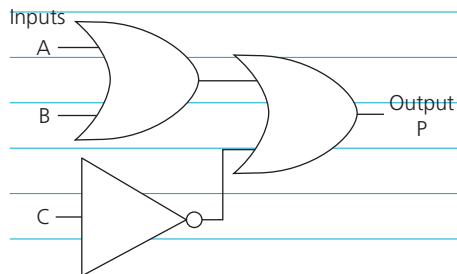
An **OR gate** gives a 1 output if either (or both) inputs are 1s. If both inputs are 0s, the output will be 0.

A	B	P (A OR B)
0	0	0
0	1	1
1	0	1
1	1	1



Combining Boolean operators

Logic gates can be **combined** by joining the output of one gate to the inputs of another gate. To decide on the output for a particular set of inputs, trace the system through from left to right.



If the inputs 1, 0, 1 were input as A, B and C into this logic system, we can work out the output of each logic gate individually. Firstly, the first **OR** gate takes input of 1 and 0, giving a 1 output. The **NOT** gate will take input of 1, giving a 0 output. This means that the final **OR** gate will have inputs of 1 and 0, giving a 1 output.

The final truth table for this system would be:

A	B	C	A OR B	NOT C	P = (A OR B) OR NOT C
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	0	1

We can therefore see from this truth table that 0, 0, 1 is the only permutation of inputs that produces a 0 output. Every other set of inputs gives a 1 output.

Applying logical operators in truth tables to solve problems

Real-world problems can be solved by applying logic to them and using truth tables to decide on the output for each set of inputs.

For example, 'I will go and watch Port Vale play football only if they are at home and it is not raining' can be written as $P = A \text{ AND NOT } B$, where P is 'I will watch Port Vale', A is 'they are at home' and B is 'it is raining'. The truth table for this is:

A (At home)	B (Raining)	NOT B	$P = A \text{ AND NOT } B$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

PROGRAMMING LANGUAGES AND INTEGRATED DEVELOPMENT ENVIRONMENTS

CHAPTER INTRODUCTION

In this chapter you will learn about:

2.5.1 Languages

- Characteristics and purpose of different levels of programming language
- The purpose of translators
- The characteristics of a compiler and an interpreter

2.5.2 The integrated development environment (IDE)

- Common tools and facilities available in an IDE

2.5.1 Languages

As we have previously seen, every computer contains a processor that fetches, decodes and executes instructions. These instructions are given to the computer by programmers, but the **language** that the instructions are given in could take many forms.

Characteristics and purposes of different levels of programming language

You may be able to name many **programming languages**; as part of the GCSE Computer Science course you are required to have practical experience of programming in at least one textual language. You may have heard of Python, Java, Visual Basic and many more, but the surprising fact is that a processor cannot execute instructions in any of these languages – they all have to be translated into **machine code** before they can be executed.

As covered in Chapter 1.1, processors can only execute instructions in machine code, which is in binary form. For example, a machine code program executed by the processor would have to look something like this:

```
10110101 10010010
01101101 10110001
00100111 11100101
01101101 10110001
```

In fact, early computers were programmed in this way. Binary 1s and 0s could be input into the computer by physically moving switches or connecting wires, as with ENIAC, the Electronic Numerical Integrator And Computer, which was first used in 1945 – see Figure 2.5.1.

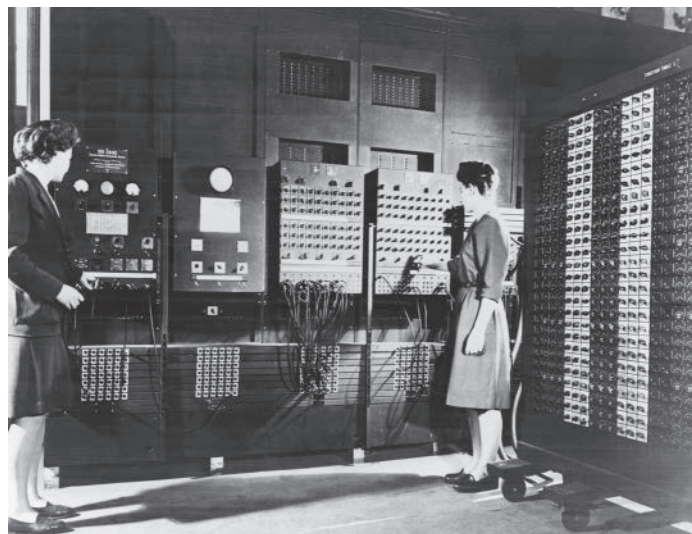


Figure 2.5.1 The early computer ENIAC (Electronic Numerical Integrator And Computer)

Tech terms

Machine code

Instructions in binary that a processor can execute directly.

Abstraction Removing unnecessary detail; the programmer of a high-level language does not need to know exactly how a processor performs a particular task; high-level language instructions are an abstract representation of what is actually happening in the processor itself.

Mnemonics In assembly language, a text code that represents a machine code operation.

Assembly language A low-level language – but not as low-level as machine code.

Key point

Java and JavaScript are two very different languages. They may share the same first part of their name, but then again so do ham and hamster or car and carpet.

Other computers allowed programs to be input using punched paper, tape or cards to represent the binary 1s and 0s. Entering instructions in binary was obviously very time-consuming and mistakes were hard to spot. Every different type of computer also had their own version of machine code and so programs written on one computer would not necessarily run on another computer.

Low-level languages

Machine code is a **low-level language**. This means that it can be run directly by the processor with no translation needed. It also has little or no **abstraction**; each instruction deals directly with the computer's hardware and so a machine code programmer would need to understand the inner workings and hardware structure of the processor. Something as simple as adding two numbers up would take numerous steps to load data into registers, perform the addition and then store the answer in memory. The programmer would need to specify each of these steps.

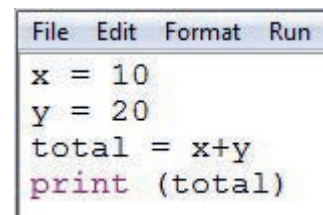
Low-level programs are also hardware dependent – they will not run on a different type of computer as the machine code instructions for that computer are likely to be different.

One simple way to make low-level programming slightly easier is to replace the binary machine code instructions with short **mnemonics** (such as replacing **00100111** with **ADD**). This is known as **assembly language** and a program called an **assembler** would convert the mnemonics back into their binary equivalents before the processor executes it. Assembly language is another low-level language. The **Little Man Computer** is an excellent introduction to programming using assembly language.

One advantage of low-level programming is sheer speed of program execution. Using machine code allows a programmer to make their program run as fast as possible with no unnecessary routines. Programmers can also be very efficient in their use of primary and secondary storage. Low-level languages such as machine code were once the only way to program a computer. Nowadays, it is only used when direct access to the hardware is needed (such as for device drivers) or where speed on relatively low-powered hardware is important (such as in embedded systems).

High-level languages

Languages like Python and JavaScript are examples of **high-level languages**. There are many different high-level languages, each with their own specific use. For example, PHP and JavaScript are used to create web applications, whereas Visual Basic is commonly used to create desktop applications for Windows computers.

A screenshot of a text editor window with a menu bar containing 'File', 'Edit', 'Format', and 'Run'. The editor contains four lines of Python code: 'x = 10', 'y = 20', 'total = x+y', and 'print (total)'. The code is displayed in a monospaced font with syntax highlighting: 'x' and 'y' are in blue, 'total' is in black, and 'print' is in red. The parentheses in 'print' are in black, and the space before the opening parenthesis is also in black.

```
File Edit Format Run
x = 10
y = 20
total = x+y
print (total)
```

Figure 2.5.2 A simple program written in Python

Programs written in high-level languages use **English-like syntax**, with keywords such as **WHILE** or **IF**. This means that they are much easier for programmers to use and errors can be spotted much more easily.

High-level programming languages also use **abstraction** to hide away details of the underlying processor operations that are needed to execute a line of code. For example, the line

```
x = 1 + 2
```

Key point

High and **low** in this context refers to the level of abstraction away from the processor. Low-level languages are so-called because they are very closely related to how the processor works. High-level languages are much further away.

in a high-level language actually requires the processor to do three things:

- load the first value into the accumulator (1)
- add the second value to it (+2)
- store the result at the memory address referenced by the label.

By hiding this underlying detail, programming becomes much easier. It also means that high-level languages are hardware independent – which means they can run on many different types of computer.

However, programs written in high-level languages cannot be executed directly by the processor in a computer. They must first be translated into machine code, a process that takes time to carry out.

The purpose of translators

A **translator** is a piece of software that converts high-level code into low-level machine code that can then be executed by the processor. Without translators, high-level programming languages like Python would not exist.

Beyond the spec

Grace Hopper (1906–1992) was an American Computer Scientist who created the first ever translator in 1952. By 1956, she had programmed the UNIVAC (Universal Automatic Computer) to translate 20 English-like statements into machine code.



Figure 2.5.3 Grace Hopper and colleagues programming the UNIVAC

The characteristics of a compiler and an interpreter

Translators come in two types: compilers and interpreters.

A **compiler** works through the high-level code, translating every line into machine code. After the compilation process has translated the whole file, an executable file is produced that the processor can run directly. The executable file can be saved and run in the future without needing to compile it again.

Most applications that a user would buy or download will have been compiled; the executable file is distributed so that users can run it. This also has the advantage of not allowing users to view or modify the high-level program code.

In contrast, an **interpreter** translates one line of high-level code and then immediately runs this before moving on to translate and run the next line of code. No executable file is produced and when the program is run again in the future the interpreter must retranslate every line of code again. If code that is run using an interpreter is distributed, all of the source code for the program must be shared.

Key point

Both compilers and interpreters are hardware dependent. Different processors or computers require different translators, even if the high-level language that they translate is the same.

Another important difference between compilers and interpreters is the **speed of execution** of programs. Because compilers translate everything first, the program itself runs quickly. Conversely, because interpreters only translate one line at a time, this translation is happening while the program is running which slows it down.

Code that is run using an interpreter is fully portable – as long as the computer you are using has an interpreter you can run the code regardless of operating system. Compiled code, by contrast, is specific to a particular operating system and therefore cannot be shared across different operating systems.



Figure 2.5.4 Python interpreter available for PC (Windows, Linux) and Mac (OSX)

Knowledge check



- 1 Name three high-level languages.
- 2 Give two advantages that high-level languages have over low-level languages.
- 3 Give one advantage that low-level languages have over high-level languages.
- 4 Give two ways that code can be translated from high-level code into machine code.

2.5.2 The integrated development environment (IDE)

An **integrated development environment (IDE)** is a piece of software that provides all of the tools required to develop computer programs.

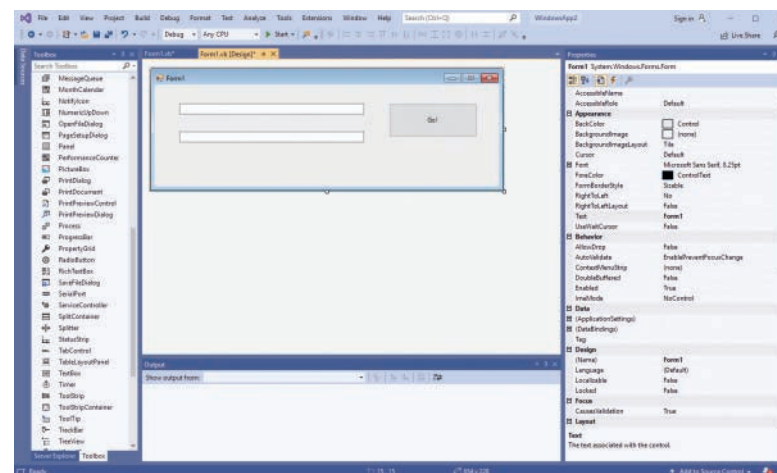


Figure 2.5.5 An IDE

Common tools and facilities available in an IDE

IDEs include everything a programmer needs in one place. By opening up one piece of software, the programmer can enter their code, check it for errors and then run the code. It is possible to have separate pieces of software to do each of these tasks, but an IDE is much more convenient.

A standard IDE will include four tools as a minimum.

Tech term

Pretty printing

Automatic colour formatting in an editor that makes the code easier to read.

Editors

The **editor** is where the programmer can enter their code and make any changes that they require. This can be relatively simple and just allow text to be entered and saved, for example Notepad in Windows or Vi/Nano in Unix.

Text editors do not support formatting like underline or bold. However, some editors do include **pretty printing**, which formats code to make it easier to understand, such as displaying all variables or keywords in a certain colour or automatically indenting code.

```
total = 0
for x in range(10):
    score = int(input("enter score : "))
    total += score

print(total)
```

Figure 2.5.6 Pretty printing in Python

Some editors include automatic completion of code, so that when the programmer types **WHI** (for example), the **WHILE** keyword will be suggested.

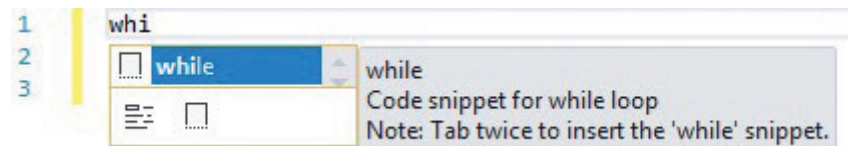


Figure 2.5.7 Visual Studio automatically completing code

Tech term

Breakpoints A line in the code where an executing program is instructed to stop.

Error diagnostics

All IDEs have features which help programmers find and fix errors in their code. These are known as **debuggers** or **error diagnostics**. As part of this, **breakpoints** can be set, which is a marked line in the program where execution will stop.

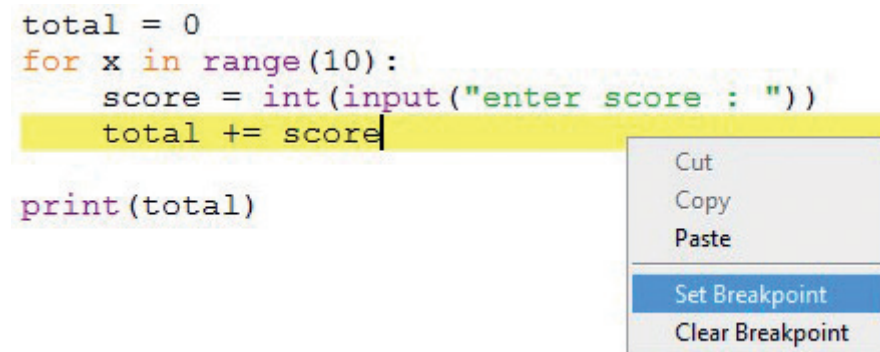


Figure 2.5.8 A breakpoint being set in Python

Once execution has stopped at a breakpoint, the programmer can **step through** the program line by line to investigate how the program runs and check the contents of variables at that point.

Runtime environment

When a programmer has written a program, the **runtime environment** is the part of the IDE that allows it to run. In most IDEs, this is accessed through a 'run' menu option, or sometimes a 'play' button.

A runtime environment provides a quick way for the programmer to test what they have written without needing to open any other programs. For some programming languages such as Java, this may involve using a **virtual machine** to run the code. For web-based languages such as PHP, this may involve software that acts as a web server to allow the code to run.

Tech term

Virtual machine An emulator of a particular computer system.

Translators

As we have covered previously, a translator converts high-level code into low-level machine code so that the processor can execute it. Most IDEs contain a compiler or an interpreter, or both.

Knowledge check



- 5 State what is meant by the term 'IDE'.
- 6 Explain the difference between an editor in an IDE and a Word Processor.
- 7 Give two methods of debugging code that an IDE provides.
- 8 Apart from an editor and error diagnostics, give one other tool that an IDE provides.

RECAP AND REVIEW

2.5 PROGRAMMING LANGUAGES AND INTEGRATED DEVELOPMENT ENVIRONMENTS

Important words

You will need to know and understand the following for the exam:

Language (programming)
Low-level language
High-level language
Translator
Compiler
Interpreter
Integrated development environment (IDE)
Editor
Debuggers
Error diagnostics
Runtime environment

2.5.1 Languages

Characteristics and purposes of different levels of programming language

Machine code and assembly language are **low-level languages**.

Python, C# and Visual Basic are examples of **high-level languages**.

Low-level languages	High-level languages
Use binary (machine code) or mnemonics (assembly language) to represent instructions.	Use English-like keywords like PRINT and WHILE
Hardware dependent – run on one specific type of computer only.	Hardware independent – will run on many different types of computer.
Refer directly to the computer's hardware. Programmers need to understand how the processor works.	Abstract (hide away) the details of the processor. Programmers can concentrate on what the program needs to do.
Can be run directly by the processor.	Must be translated into machine code before they can be run.

The purpose of translators

Translators convert high-level programming code into machine code so that the processor can execute it. High-level code **cannot** be executed directly.

The characteristics of a compiler and an interpreter

Compilers	Interpreters
Translate every line of code in a program into machine code and run it afterwards.	Translate one line of code and run that line, repeating this process.
Produce an executable file.	Do not produce an executable file.
Program can be run again without re-compiling ; simply run the executable file again.	Running the program again needs the interpreter to retranslate every line of code .
Executable file can be distributed meaning that users will not see the source code.	No executable file to distribute. Would need to share the source code to distribute the program.
Compiled code runs quickly .	Interpreted code runs more slowly than compiled code.

2.5.2 The integrated development environment (IDE)

Common tools and facilities available in an IDE

An **integrated development environment (IDE)** provides all of the tools that a programmer needs to write and test programs in one place.

IDEs contain four main tools:

Tool	Description
Editor	<ul style="list-style-type: none">• A text editor to allow the programmer to enter or modify code in their chosen language.• May include auto-suggestion of keywords.• May include pretty printing to colour code keywords and automatically indent code.
Error diagnostics	<ul style="list-style-type: none">• Tools to allow the programmer to find and fix errors.• Breakpoints stop the program at a specific point.• Stepping allows the programmer to run the code from this point one line at a time.• Variable contents can be checked.
Runtime environment	<ul style="list-style-type: none">• Allows the programmer to run the code from within the IDE.• The program output can be seen without opening additional programs.• May involve the use of a virtual machine.
Translators	<ul style="list-style-type: none">• Converts the high-level code into machine code to allow execution by the processor.• IDEs include interpreters or compilers (or both).

Extra resources

A free set of practice questions accompanies this section and is available online at:
www.hoddereducation.co.uk/OCRGCSEComputerScience

These practice questions have not been produced either by OCR or by the OCR Principal Examiner. They are also not endorsed by OCR and have not been subject to any OCR quality assurance processes.

APPENDIX

Bubble sort

```
01 function BubbleSort(sortList)

    //Set a flag to false to show array is not sorted

    02 sorted = false

    03 while sorted == false

        //Set the flag back to true (assume pass will sort array)

        04 sorted = true

        //Step through the array, index by index

        05 for sortCount = 0 to len(sortList) - 2

            //Check if the element is larger than the one next door

            06 if sortList[sortCount] > sortList[sortCount + 1] then

                //If so – swap elements around using a '3-way' swap

                07 temp = sortList [sortCount]

                08 sortList[sortCount] = sortList[sortCount + 1]

                09 sortList[sortCount + 1] = temp

            //The array was not sorted, so set the flag back to false

            10 sorted = false

        11 endif

        //Move along the array index and check next element

        12 next sortCount

    13 endwhile

    //Once sorted, return the array back

    14 return sortList

    15 endfunction
```

Explanation

In this example, a function is created that takes one parameter:

- **sortList**: An array of items to be sorted.

How does it work?

The algorithm works by using a FOR loop, nested within a WHILE loop.

- The WHILE loop keeps running until the array is sorted.
- The FOR loop is used to sort the elements within the array.
- Line 01: The function is defined with one parameter.
- Line 02: We create a 'flag' and set it to **false** – assuming the array is not sorted at the start of the algorithm.
- Line 03: We start the WHILE loop, which keeps running whilst the array is not sorted (**sorted == false**).
- Line 04: We assume that this next iteration will sort the array.
- Line 05: We use a FOR loop to iterate through the array.
- Lines 06–09: If the current element is greater than the one to the right, then it swaps these elements around.
- Line 10: Because a swap was made, the array is 'unsorted' and so the flag is set back to **false**.
- Line 12: The index is increased by 1, i.e. we move to the next index and Lines 06–09 are repeated.
- Line 13: Once the FOR loop exits, the WHILE loop iterates.
- Line 03: The WHILE loop checks to see if **sorted** is **false** and executes again if so.
If **sorted** is **true**, we know the array is sorted.
- Line 14: Once **sorted == true**, we return the sorted list.

Insertion sort

01 function InsertionSort(sortList)

```
//Start with the 2nd element in the list and carry on until the last
//element in the list
```

```
02 for sortCount = 1 to len(sortList) - 1
```

```
03   listIndex = sortCount
```

```
//While not at the start of the list AND the value in the index position
//is less than the one to left of it
```

```
04   while listIndex > 0 and sortList[listIndex] < sortList[listIndex - 1]
```

```
//Swap the elements around
```

```
05     temp = sortList [listIndex]
```

```
06     sortList[listIndex] = sortList[listIndex + 1]
```

```
07     sortList[listIndex + 1] = temp
```

```
//Move back to the previous element (to the left) and test again
```

```
08     listIndex = listIndex - 1
```

```
09   endwhile
```

```
//Once the element is in the correct place, move to the next list index
```

```
10   next sortCount
```

```
//Return the sorted list
```

```
11 return sortList
```

```
12 endfunction
```


Explanation

In this example, a function is created that takes one parameter:

- **sortList**: An array of items to be sorted.

How does it work?

The algorithm works by using a FOR loop with a WHILE loop nested inside.

- The FOR loop is used to move element by element through the **sortList** from the second element to end.
- The WHILE loop checks to see if an element in **sortList** is greater than the one to the left of it. It then repeats this working backwards through the list, until the element is in the correct place.
- Line 02/03: The FOR loop is created to run through the array. Each time the **sortCount** is assigned to the **listIndex** variable. This means that later we can decrement the **listIndex** variable without affecting the FOR loop pointer.
- Line 04: The WHILE loop checks to see if the value in the current index is larger than the one to the left.
 - Line 05/06/07: If so, we swap the values.
 - Line 08: We now need to check that the swapped value is in the correct place in the sorted sublist, so we reverse down the array, checking and swapping if needed.
- Line 10: Once one element is sorted, we increase the FOR loop counter to check the next element in **sortList**.
- Line 11: Finally, we return the sorted array.

Linear search

```
//Function is defined called LinearSearch

01 function LinearSearch(searchList, searchItem)

02     searchIndex = 0

//Loop through each index in the array searchList

03     while searchIndex < len(searchList)

//If the item in the searchList equals searchItem

//then we return true

04         if searchList[searchIndex] == searchItem then
05             return true
06         endif

//If a match is not made, increment the searchIndex

07         searchIndex = searchIndex + 1

08     endwhile

//If no match is found, then we return false

09     return false

10 endfunction
```

Explanation

In this example, a function is created that takes two parameters:

- **searchList**: An array of items to be searched through.
- **searchItem**: The item to be searched for.

How does it work?

- Line 01: The function is defined with two parameters.
- Line 02: A loop counter **searchIndex** is set to 0.
- Line 03: The WHILE loop starts at index 0, and repeats until the end of the list.
Using **len(searchList)** allows any array size to be used.
- Line 04 compares the element in the array with the item we are searching for.
If the element matches the **searchItem** then **true** is returned.
Because a **return** is used here, the function then terminates by default.
- Line 07: If there is no match, **searchIndex** increments by 1.
The WHILE loop iterates (back to Line 03) using the new **searchIndex** value.
The WHILE loop ceases to run if the end of the array is reached.
- Line 09: If the WHILE loop exits due to reaching the end of array then it returns **false**.

Binary search

01 function BinarySearch(searchList, searchItem)

//Define the initial start and end points in the array

02 start = 0

03 end = len(searchList)

//Keep looping until only one element left

04 while start != end

//Find the mid-point between start and end

05 midPoint = (start + end) DIV 2

//If mid-point is greater, then value is to the left

06 if searchList[midPoint] > searchItem then

07 start = midPoint + 1

//If mid-point is less, then value is to the right

08 elseif searchList[midPoint] < searchItem then

09 end = midPoint - 1

//If we find a match, return true

10 elseif searchList[midPoint] == searchItem then

11 return true

12 endif

13 endwhile

//If no match is found then return false

14 return false

15 endfunction

Explanation

In this example, a function is created that takes two parameters:

- **searchList**: An array of items to be searched through.
- **searchItem**: The item to be searched for.

How does it work?

The algorithm works by continually **splitting** the list in half until it finds the value it is looking for or has eliminated all the options.

- Line 01: The function is defined with two parameters.
- Line 02/03: An initial value for **start** and **end** are set.
On the first pass, the **start** index is 0 and the **end** index is the final index in the array.
- Line 05 finds the middle of the array.
- Line 06 checks to see if the element in the array at the midpoint is greater than the item we are looking for.
 - If so, then Line 07 sets the index, one to the right of this position, as the new **start**.
- Line 08 checks to see if the element in the array at the midpoint is less than the item we are looking for.
 - If so, then Line 09 sets the index, one to the left of this position, as the new **end**.
- Line 10 checks if the item in the array and the item we are looking for are the same.
 - If so, then Line 11 returns **true** and the function exits.
- If either Line 06 or Line 08 are true, the while LOOP repeats with either the new **start** or **end** values.
- Line 14: By default, if the value is not found, **false** is returned.

GLOSSARY

Abstraction Removing unnecessary or irrelevant detail from a problem.

Access rights The permissions that a user on a computer system has to view, write, modify and delete files, change configurations, and add or remove applications. Different types of users may have different permissions.

Accumulator (ACC) Stores the results of calculations carried out by the ALU.

Algorithm A sequence of instructions.

Algorithmic thinking Breaking the solution to a problem into a series of decisions and instructions that a computer can implement.

Analogue Continuously changing signals.

AND gate An AND gate output is 1 only if the two inputs are also both 1.

Anticipating misuse Considering what might happen if users behave in unexpected ways.

Anti-malware software Software installed on a computer system to protect it against malware and unauthorised access.

Application software The programs used to carry out different tasks on a computer, such as word processors, media players and browsers.

Arithmetic Logic Unit (ALU) The component of a CPU that performs arithmetic and logical operations.

Arithmetic operators Such as +, -, *, / etc. – used to perform mathematical operations.

Array A series of items of data that are grouped together under one identifying name or label. 1D arrays label each item of data using one index number. 2D arrays label each item of data using two index numbers.

ASCII A 7-bit code to represent a set of characters available to a computer.

Assignment Variables and constants are assigned values using the '=' operator.

Authentication Establishing a user's identity.

Bandwidth The amount of data that can be transferred through a specific connection in a given amount of time, measured in bits per second.

Binary A number system based on 2, using the digits 0 and 1 only.

Binary code A coding system using the binary digits 0 and 1 to represent a letter, digit, or other character in a computer or other electronic device.

Binary shift Moving the binary digits one place to the left multiplies the number by 2, one place to the right divides the number by 2.

Bit (b) A binary digit, 0 or 1, symbol b.

Bit depth The number of bits used to store each sound sample.

Boolean conditions Conditions that can evaluate to either True or False.

Boolean operators Such as AND, OR, NOT, used to compare multiple conditions.

Boundary test data Values that sit at the very end of the expected range.

Brute force attack An attempt to gain access to a system using repeated login attempts.

Buffering The use of a reserved area of memory to hold data until it is needed; for example, when streaming audio from the internet a certain amount of data is downloaded before the music starts to play.

Byte (B) A group of 8 bits, symbol B

Cache memory A small area of memory that can be accessed quickly. Used to store commonly used instructions or data.

Casting Converting one data type to another.

Central Processing Unit (CPU) A collection of billions of electronic switches that process data, execute instructions and control the operation of the computer.

Character A single item from the computer character set, for example 'h', '?'.

Character set The complete set of characters available to a computer.

Client-server networking A network model where a server provides resources and services to one or more client computers.

Clock speed The number of cycles per second measured in Hz.

Closing (files) External files need to be closed by a computer program once they are finished with, using the close() command in OCR Exam Reference Language.

Cloud Storage, services and applications that exist on the internet.

Colour depth The number of bits used for each pixel.

Command line interface (CLI) An interface in which the user has to type in all of the commands via a keyboard.

Commenting Lines of code written for programmers to understand. They are ignored by the computer when the program is run but can make code much easier to understand.

Comparison operators Such as >, <, ==, etc., used to compare values.

Compiler A translator that decodes all of the lines of code in a high-level language to produce an executable file which can then be run by the computer. Programs only need compiling once.

Concatenate Joining multiple strings together.

Condition-controlled iteration/loops Loop until a particular condition is met, for example WHILE loops.

Constants A label for an area of memory that stores a value that does not change during the execution of a program.

Control unit (CU) A component of a CPU that coordinates the activity of the CPU.

Count-controlled iteration/loops Loop for a specified number of times, for example FOR loops.

Culture The ideas, customs and social behaviour of a particular group or society.

Data interception and theft The unauthorised taking of computer-based information. This can be achieved by routing the data to a different destination, or using packet sniffing software to intercept wireless transmissions.

Debuggers A feature of IDEs that help programmers to work through lines of code to check how their program works and find errors.

Decomposition Breaking a problem into sub-problems.

Defensive design Writing a program that foresees potential problems and accounts for them accordingly.

Defragmentation The process of grouping together fragments of the same file and free space on a hard disk drive.

Denary A number system based on 10, using digits 0–9.

Denial of service (DoS) An attempt to bring down a website or computer system by overloading the server so that it cannot respond to genuine requests.

Device drivers Software that enables an operating system to communicate with peripheral devices.

Domain name Part of a network address that identifies it as belonging to a specific domain, e.g. gov.uk.

Domain Name Server (DNS) A worldwide network of servers that translate memorable domain names for websites into IP addresses.

Editor The part of an IDE where code can be entered.

Embedded system A computer system that forms part of an electronic device.

Encrypting The process of encoding a message or information so that only authorised persons can access and understand it.

Environmental Effects relating to the natural world and the impact of human activity on it.

Erroneous test data Values that are of the wrong type, for example a string when the value should have been an integer.

Error diagnostics A feature of IDEs that will pinpoint where certain errors occur.

Ethernet A protocol used to connect devices in a LAN over a wired connection.

Ethics Moral principles that govern a person's behaviour.

Exponent An operator (such as \wedge) that raises one number to the power of another, for example $2^2 = 4$.

Fetch-Execute cycle The basic operation of the CPU. It continually fetches, decodes and executes instructions stored in memory.

Field names The label given to each data field in a record.

File management The process by which the OS keeps track of the physical location of files and folders in secondary storage, whilst allowing the user to organise and find data using a folder and file structure.

File permissions Permissions that control the ability of a user to view, write, modify and delete files.

File Transfer Protocol (FTP) A protocol for transferring files over the internet.

Final testing Testing the whole complete program.

Firewall Software and/or hardware that inspects and controls all inbound and outbound network traffic.

Flowchart A visual way of representing inputs, processes and outputs of an algorithm.

Functions A subprogram that returns a value to the main program.

Grammatical rules Another name for a syntax error.

Graphical user interface (GUI) An interface where small icons that represent the files and devices on a computer can be used by clicking, dragging or touching them.

Hard disk drives (HDDs) A non-volatile storage device.

Hexadecimal A number system based on 16, using digits 0–9 and A–F to represent the denary values 0–15.

High-level language A programming language which is written using English-like statements, that are easier for programmers to work with. High level languages must be translated into machine code before a computer can run them, such as Python, Java, C and Visual Basic.

Hypertext Transfer Protocol (HTTP) A protocol used to transfer data between a web browser and web servers.

Hypertext Transfer Protocol Secure (HTTPS) A secure version of HTTP that adds Secure Socket Layer (SSL) encryption to the data.

Identifier Another word for the name of a variable or constant.

Indentation Used to make code structure clearer; an essential part of the syntax in Python.

Input validation Limiting the kind of data a user can input, to help minimise errors.

Input–Process–Output Problems and algorithms can be broken down into Input–Process–Output.

Integers Whole numbers, for example 432.

Integrated development environment (IDE) A piece of software that provides all the tools necessary to develop programs in a particular language.

Internet A global network of networks that connects computer systems together across the world.

Internet Message Access Protocol (IMAP) A protocol for accessing email messages on a mail server without having to download them to your device.

Internet Protocol (IP) A set of rules for sending and receiving data across the internet.

Internet Service Provider (ISP) A company that provides access to the internet.

Interpreter A translator that decodes one line of code of high-level language and then runs it before moving on to the next line. Programs need to be interpreted each time they are run.

Invalid test data Values that are outside the expected range.

IP address A unique address that identifies a device on the internet, or on a local network.

Iteration Repetition of sections of code.

Iterative testing Testing each standalone part of a program separately.

Language (programming) A set of instructions, written according to particular rules, that a computer can execute to perform a particular task.

Layers A way of separating the different activities involved in communication over the internet.

Least significant bit (LSB) The smallest place value in a number.

Legal Actions permitted or denied by force of law.

Licence An official document giving permission to use something.

Local area network (LAN) A network of computers in a small geographic area, such as a single building.

Logic errors Something that causes the program to behave in an unexpected way.

Looping Another name for iteration.

Lossless compression Compression technique that does not lose any of the original data and the original file can be recovered.

Lossy compression Compression technique that removes some of the original data. The original cannot be recovered.

Low-level language A programming language whose lines of code directly correspond to the CPU's hardware processes. This means it has little or no abstraction, for example machine code, assembly language.

Magnetic storage Storage medium using magnetism to store data.

Maintainability Making code easier to understand and modify.

Malware Software programs designed to cause damage to a computer system or steal information. It is short for 'malicious software'.

Media Access Control (MAC) address A number programmed into the network interface controller (NIC) that uniquely identifies each device on a network.

Memory address register (MAR) Stores the address of the location in memory for data to be fetched from or sent to.

Memory data register (MDR) Stores data fetched from or to be sent to memory.

Memory management The process in which the operating system assigns blocks of memory to programs that are running in order to optimise system performance.

Mesh network topology A network arrangement where all devices are connected, either directly or indirectly, without a central switch.

Metadata Additional information stored with data to enable the computer to recreate the original image/sound/document from the data.

Module A standalone section of a program.

Modulus (MOD) An operator that returns the remainder after a division, for example $11 \text{ MOD } 2 = 1$.

Most significant bit (MSB) The largest place value in a number.

Multitasking The way in which an OS enables several programs to run side by side.

Naming conventions The way in which variable and constant identifiers are chosen.

Network interface controller/card (NIC) A component that enables a device to connect to a network using a wired or wireless connection.

Network performance How quickly data is transmitted through a network.

Nibble A group of 4 bits, half a byte.

Normal test data Values that sit within the expected range.

NOT gate A NOT gate output reverses the input.

Opening (files) External files need to be opened by a computer program first, using the open() command in OCR Exam Reference Language.

Open-source software Software whose source code is available for anyone to use or modify.

Operating systems Software that communicates with the hardware and allows other programs to run.

Operator precedence The order in which operations are carried out – BIDMAS from mathematics.

Operators Symbols or words in a program that are reserved for particular actions.

Optical storage Storage devices that use laser light to read and write data.

OR gate An OR gate output is 1 if one or the other, or both, of the two inputs are 1.

Overflow error When a number becomes too large to store in the number of bits allocated, it is said to 'overflow' and some bits are lost.

Parameters Values that a main program sends to subprograms for them to use.

Passwords A memorised set of characters used to confirm the identity of a user.

Peer-to-peer networking A network model where all computers are connected to each other and files can be shared without the need for a central server.

Penetration testing Testing a computer network for vulnerabilities that an attacker could exploit. It is an authorised activity also known as ethical hacking.

Peripheral management The process in which the OS controls peripheral devices such as disk drives and printers using programs called drivers.

Physical security Security measures to protect computer systems from physical actions and events such as fire, flood, natural disasters, theft and vandalism.

Pixel The smallest element of an image. Pixels are the dots that make the image on the screen.

Post Office Protocol (POP) A protocol for downloading email messages from a mail server to the user's device.

Prefixes: kilobyte (KB), megabyte (MB), gigabyte (GB), petabyte (PB) Naming convention based on multiples of 1000.

Pre-requisite (for algorithm) Some search algorithms may require the data to be already sorted – this is known as a pre-requisite for that algorithm.

Primary storage Storage for data that the CPU needs to access quickly.

Procedures A subprogram that does not return a value to the main program.

Processor cores Multiple processor cores used to process instructions simultaneously.

Program counter (PC) Stores the address of the next instruction to be processed.

Proprietary software Software whose source code is owned and protected by the developer. It is illegal to modify such code and a licence is required to use it.

Protocols A set of rules for transmitting data between electronic devices.

Pseudocode Similar to a high-level programming language but without strict syntax rules.

Quotient (DIV) An operator that returns the whole number after a division, for example $11 \text{ DIV } 2 = 5$.

Random access memory (RAM) The main memory of a computer that stores data, applications and the operating system while in use. When the power is turned off, RAM loses its data.

Random numbers A number returned by a program that cannot be predicted in advance. A series of random numbers would not display any patterns at all.

Read and write Data can be both read and written by the computer.

Reading (files) External files can read into a computer program once they have been opened, using the `readLine()` command in OCR Exam Reference Language.

Read-only Data cannot be written by the computer.

Real numbers Numbers with a decimal point, for example 4.0, 302.81. Also known as float numbers.

Record A series of data fields.

Refining algorithms Making changes to programs to correct issues discovered through testing.

Registers Very small memory locations within the CPU that temporarily store data and can be accessed very quickly.

Resolution The number of pixels of dots per unit used in an image, for example DPI (dots per inch).

Read-only memory (ROM) Used to store data and programs to initialise a computer system.

Run-time environment The part of an IDE where code can be run.

Sample rate The number of times a sound is sampled per second, measured in Hz.

Searching algorithm: linear search, binary search Different algorithms that can search a list of data.

Secondary storage Non-volatile storage for files and programs.

Secure socket layer A protocol for securing connections between web clients.

Selection The decision-making process in a program.

Sequence Execution of statements in a program one after another.

Service Set Identifier (SSID) The name of the wireless network broadcast by the access point.

Simple Mail Transfer Protocol (SMTP) A protocol used to send emails from an email client, such as Outlook, to a mail server.

Slicing Extracting characters or sub-strings from a string.

Social engineering Methods used to trick people into divulging sensitive or confidential information.

Solid-state drives (SSDs) A non-volatile storage device using solid state storage.

Solid-state storage Storage device using electrical transistors to store data.

Sorting algorithm: bubble sort, insertion sort, merge sort Different algorithms that can sort a list of unordered data.

SQL commands: SELECT, FROM, WHERE Used to interrogate databases.

SQL injection A method of attacking databases by inserting malicious SQL statements into entry fields in order to access resources or make changes to sensitive data.

Standards A set of agreed rules and definitions that govern all aspects of computing.

Star network topology A network arrangement where each computer or client is connected to a central point, usually a switch or hub.

String A collection of items from the computer character set, for example 'hello'.

Structure diagram A visual way of representing sub-problems.

Structured Query Language (SQL) A language used to access data stored in a database.

Subprograms General name for smaller standalone sections of code that can be called by the main program.

Syntax errors Something that breaks the grammatical rules of the programming language.

Systems software The files and programs that make up the operating system of a computer.

TCP/IP A set of protocols for sending data over the internet that specifies how data should be broken into packets, addressed, transmitted, routed and received at its destination.

Terminal testing Testing the whole complete program – the same as final testing.

Test data Values that are used to check that a program behaves correctly.

Test plan A list of test data, expected and actual results.

Testing Checking that a program does what it should for every possible input and output.

Trace table Records the values of variables and outputs for each line of code, used when checking a program to make sure it is correct.

Translator A program that decodes a high-level language into machine code. There are two types of translators: compilers and interpreters.

Transmission media The media used to transmit data on a network, cables or radio waves.

Truth table A table listing all possible inputs and outputs for a logic circuit.

Two-factor authentication Establishing a user's identity using two different methods, such as a password and biometrics.

Unicode A character set using code pages and 16/32 bits to represent a range of language symbols. There are several billion possible character codes available to Unicode.

Uniform Resource Locator (URL) The address of a specific webpage on the internet. It includes the protocol used to access the location and the IP address.

User interface The way in which a person controls a software application or hardware device.

User management The use of separate login accounts for different users, whose access to various resources such as applications, devices, storage and networks can be managed by an administrator.

Variables A label for an area of memory that stores a value that can change during execution of a program.

Virtual memory A section of the hard drive used as if it were RAM to supplement the amount of main memory available to the computer.

Volatile and non-volatile Volatile means data is lost when the power is removed. Non-volatile memory retains data even when the power is turned off.

Von Neumann architecture The most common organisation of computer components, where instructions and data are stored in the same place.

Web server A computer system that hosts websites.

Wide area network (WAN) A network of computers that spans a large geographic area, often between cities or even across continents.

Wi-Fi A set of rules defining how network devices can communicate using radio waves.

Windows, Icons, Menus and Pointers (WIMP) An interface where applications are shown in windows, programs and files are represented by icons, menus allow the user to access features and options, and interaction is provided by moving a pointer.

Wireless access point (WAP) A device that allows devices to connect to a network using Wi-Fi.

Writing (files) External files can be written to by a computer program once they have been opened, using the writeLine(x) command in OCR Exam Reference Language.

ANSWERS

Computer systems

1.1 System architecture

- 1 To process data, carry out instructions and control the components of the computer.
- 2 The processor fetches instructions from memory.
 - The address of the next instruction is copied from the program counter (PC) and placed in the memory address register (MAR).
 - Data is fetched from this location stored in MAR and placed in the memory data register (MDR).
 - PC is incremented to point at the next instruction.The processor decodes the instruction in MDR to see what to do.
The processor then executes them:
 - it performs a calculation and stores the result in the Arithmetic Logic Unit (ALU) OR
 - it changes the value in the PC to point to another instruction to be fetched next.
- 3 The location in memory where the MDR needs to fetch data from or send data to.
- 4 To store any data or instructions fetched from memory or any data that is to be transferred to and stored in memory.
- 5 It stores the address of the next instruction to be processed.
- 6 Any two from arithmetic operations (add, subtract, divide, multiply), logical operations (AND, OR, NOT) or binary shift.
- 7 A processor with four processor cores (able to deal with four simultaneous processes).
- 8 The clock speed is 2.3 GHz. (It is able to carry out 2.3 billion processes per second.)
- 9
 - It is used to hold data that needs to be accessed very quickly.
 - It sits between the CPU and main memory.
 - It is faster than accessing main memory.
 - CPU looks to cache for necessary data or instruction.
 - If the data is not in cache, it is found in main memory then transferred to cache.

- 10
 - Clock speed determines how many operations per second.
 - Cache memory holds frequently required data so more cache less time accessing main memory.
 - More cores allow more processes to complete simultaneously, more cores more speed.
- 11 ROM is non-volatile and does not require power to maintain its contents. It holds data and instructions operate the device. RAM is required to store user selections or any output generated by the device.
- 12
 - Input: User selection for time, power level, program.
 - Output: Display of user selections, timer countdown, 'ping'.
- 13 There are many examples including: washing machines, dishwashers, microwaves, set top boxes, telephones, televisions, home security, water meters, energy smart meters, home security or heating monitoring systems, missile guidance, vehicle management, CAM, digital cameras and portable entertainment devices. (There are several other examples.)
Justification is based on the device selected but can include:
 - power requirements, for example battery operated devices
 - size, for example in portable devices
 - rugged, for example in missiles or car engines
 - low cost for domestic devices
 - dedicated software – limited need for user input and output, limited range of programs/options.

1.2 Memory and storage

- 1
 - RAM is **volatile**, meaning it needs electrical power to operate.
 - Any data stored in RAM is lost when the power is turned off.
 - ROM is **non-volatile** memory which means it does not require power to maintain its contents.
 - ROM is **read-only**. This means that the data stored in ROM is fixed and cannot be overwritten once it is created.
- 2 The operating system, applications that are running and any associated data while the computer is on and in use.

- 3 The instructions and data needed to get the system up and running and ready to load the operating system from secondary storage.
- 4 An area of secondary storage used to temporarily store data from RAM.
If a computer is processing large amounts of data, there may be insufficient RAM to hold them all. In this case, the computer can allocate a section of secondary storage to temporarily act like RAM.
- 5 There is a delay when transferring data from secondary storage back into RAM. A computer with more RAM will need less virtual memory, reducing the number of data transfers between RAM and secondary storage and therefore delivering improved performance.
- 6 To store various other files on our computers so that they are available the next time we switch on the computer.
- 7 The operating system, data, images, programs, documents.
- 8
 - Fast data access times due to lower latency times because there are no moving parts.
 - Lower power requirements and do not generate any heat or noise.
 - SSDs do not fragment data.
 - Light and small with low power requirements making them useful for portable devices.
 - Robust because they have no moving parts.
- 9
 - Lower cost per GB of storage.
 - Higher capacities available for large storage requirements.
- 10 Blu-Ray
- 11
 - Capacity: How much data does it need to store?
 - Speed: How quickly does the data need to be accessed?
 - Portability: Does the device or media need to be transported?
 - If so, the size and weight are important.
 - Durability: Will the device or media be used in a hostile environment?
 - If so, the media must be resistant to external shocks or extreme conditions.
 - Reliability: Does it need to be used repeatedly without failing?
 - Cost: What is the cost per unit of storage related to the value of the data?
- 12 4 500 000 KB
- 13 32 000 GB
- 14 $200 * 6 = 1200 \text{ MB}$
 $20 * 6 = 120 \text{ MB}$
Total 1320 MB or 1.32 GB
- 15 $40 * 100 = 4000 \text{ KB}$ or 4 MB
 $15 * 6 = 90 \text{ MB}$
Total 94 MB which will fit comfortably on a CD but most devices do not contain a CD drive and the most appropriate device would be a USB flash drive.
- 16 (a) 9
(b) 29
(c) 49
(d) 140
(e) 219
(f) 252
- 17 (a) 10100
(b) 101110
(c) 1001011
(d) 1100010
(e) 10010011
(f) 11010101
- 18 10010
- 19 10100
- 20 1011010
- 21 1010011
- 22 10111101
- 23 11101011
- 24 101110001 – we get a ninth bit so the number has overflowed
- 25 11 denary 3 1100 denary 12 divided by 4
- 26 1101100 denary 108 11010 denary 54 multiplied by 2
- 27 101000 denary 40 101 denary 5 multiplied by 8
- 28 110 denary 6 110000 denary 48 divided by 8
- 29 1110000 denary 112 111 denary 7 multiplied by 16
- 30 1000 denary 8 10000000 denary 128 divided by 16
- 31 1100110000 more than 8 bits, we have overflow
- 32 101 denary 5 10111 denary 23 divided by 4, but we have lost accuracy
- 33 (a) 34
(b) 43
(c) A5
(d) BF
(e) C9
- 34 (a) 18
(b) 88
(c) 93
(d) 174
(e) 202
- 35 (a) 9C
(b) 33
(c) FF

- (d) 39
(e) 4E
- 36 (a) 10010101
(b) 10101011
(c) 11101
(d) 10100011
(e) 1010110
- 37 The higher the resolution, the larger the file needed to store the image.
- 38 The higher the resolution, the better the image quality.
- 39 The number of bits per pixel.
- 40 It reduces the number of colours that can be displayed and reduces the quality of the image.
- 41 2^4 or 16
- 42 The number of bits used to store each sample of music.
- 43 The higher the sample rate, the closer the sound is to the original but the larger the file needed to store the data.
- 44 Reducing the size of a file.
- 45 With lossy compression some data is lost and the original file cannot be recovered. With lossless compression no data is lost and the original data is available.
- 46 With a computer program all the data must be available otherwise the program will not work.
- 47 Image and video files can be very large and would take a significant amount of time to send if not compressed. The data in a video or large high-resolution image may exceed the file size for an email attachment.

1.3 Computer networks, connections and protocols

- 1 A LAN is a network in a small geographic area, such as a home, a school, an office or a group of buildings on a single site. The hardware is usually owned and maintained by the organisation that uses it. It will often use both wired and wireless connections.
- 2 A LAN is a network covering a small geographic area; a WAN covers a wide geographic area.
The networking hardware in a LAN is usually owned and maintained by the organisation that uses it. The connections used in a WAN are usually hired or leased from a telecommunications company.
- 3 Bandwidth is the amount of data that can be transmitted over a network in a given period of time. It is usually measured in bits per second.

- 4
- The number of users. A large number of users can cause network congestion and slow the network.
 - The type of transmission media used. Wired connections are generally faster and more reliable than wireless connections.
 - Error rate. There may be transmission errors due to interference from other wireless networks nearby, or other electronic devices, or due to a weak Wi-Fi signal. Errors mean that packets have to be re-sent which increases data traffic.
- 5 In a client-server network, the computers, known as clients, are connected to a central server. The clients request the services and resources that they require from the server. The server processes this request and sends a response back to the client.
- 6 Advantages:
- Users files can be stored on the server so they can be accessed from any connected client.
 - Backups can be managed centrally, ensuring that all files are included.
 - Software and security updates can be managed centrally, without the need to update each computer individually.
 - User accounts can be managed centrally and access levels can be controlled for different categories of user, which helps to keep the network secure.
- Disadvantages:
- If the server becomes unavailable then users will not be able to access their files.
 - Servers can become overwhelmed by too many requests, preventing clients from accessing their services.
 - Server hardware is more expensive than ordinary computers, so there is an increased up-front cost of setting up the network.
 - A cyber-attack only needs to focus on the server, not on each individual client.
- 7 All of the computers, called peers, have equal status and are connected directly without the use of a central server. Each computer stores its own files. Peers can be configured so that specified files and folders can be accessed by other peers on the network.
- 8 Advantages:
- P2P networks are easy to set-up as they do not require expensive or dedicated hardware.
 - If one device fails, the rest of the network will still continue to operate normally.
- Disadvantages:
- There is no central management or maintenance, so software and security updates have to be carried out individually on each peer device.

- There is no centralised backup of files, so each peer needs to be backed up separately.
 - Files are duplicated when they are transferred between devices, which can lead to multiple versions of the same document existing.
- 9** The NIC enables a device to connect to a LAN. It formats the data to be sent using the correct protocol, and received incoming signals.
- 10** A desktop computer may be connected to a home network using ethernet cables or wirelessly using W-Fi.
- 11** Individual devices are uniquely identified on a LAN using their MAC address.
- 12** Wireless access point (WAP)
- 13** The router connects the home network to the internet.
- 14** The IP address is used to uniquely identify devices on the internet so that data can be sent from one device to another.
- 15** A user types the domain name for a website into their browser. The browser sends a request to a DNS server to ask for the matching IP address. The DNS server returns the appropriate IP address to the browser. The browser then sends a request for the webpage to the IP address.
- 16** Cloud computing refers to the use of storage, services and applications that are accessed via the internet rather than being stored locally on a device.
- 17**
- File storage, such as DropBox, iCloud Drive.
 - Applications, such as Office 365, Google Docs.
- 18**
- An internet connection is required to access the data.
 - There is little control over the security of the data or where it is stored.
 - Terms and process of data storage can be changed with little notice.
 - Fees may become expensive in the long term.
- 19** The way in which devices are arranged and connected together.
- 20** Each computer or client is connected individually to a central point, usually a switch or hub.
- 21** Advantages:
- There is no single point of failure.
 - Data can be transmitted from several devices simultaneously, allowing for high volumes of data traffic.
- Disadvantages:
- Wired mesh networks tend to be too expensive and impractical to set up due to the many connections required.
 - They require a lot of maintenance due to the many connections.
- 22** A set of rules for how devices communicate and how data is transmitted across a network.
- 23** Ethernet
- 24** 2.4GHz and 5 GHz
- 25** Bluetooth
- The data only needs to travel over a short distance.
 - It is quick to pair devices. Data communication is more secure than open Wi-Fi.
- 26** An IP address is unique and identifies the location of a device on a network.
- 27** 32 bits (4 × 8 bits)
- 28** Eight sets of four hexadecimal numbers separated by colons. This requires 128 bits.
- 29** A MAC address uniquely identifies each device that is connected to a network.
- 30** Standards allow different hardware and software manufacturers to make components and programs that are compatible and will work with each other. Without the use of standards, only hardware and software made by the company could be used together.
- 31** File Transfer Protocol (FTP)
- 32** Hypertext Transfer Protocol Secure (HTTPS)
- 33** It is used to send email from a device to the email server, and to send emails between servers.
- 34**
- TCP stands for Transmission Control Protocol. It splits data in packets and adds the header, which includes the packet number, number of packets and checksum.
 - IP stands for Internet Protocol. This adds the source and destination IP addresses to the packet.
- 35** The different activities involved in sending data are organised into layers, with each layer being concerned with a different task. The relevant protocols are assigned to each layer.
- 36** It breaks down the process into smaller, more manageable parts. Each layer can be developed or changed without affecting the other layers. Software and hardware developers only have to understand how one layer works when developing new products. It is helpful when trying to identify and correct networking errors and problems as the issue can be narrowed down to one layer in the process.

1.4 Network security

- 1** Virus = Malware that is spread through infected files
 Spyware = Malware that comes packaged with other software
 Worm = Malware that is self-replicating and spreads via email
 Trojan = Malware disguised as legitimate software

- 2 Pharming is where users are directed to a fake website in order to obtain their login details.
- 3 Pretexting involves a criminal inventing a scenario to persuade a victim to divulge information that they would normally keep secret.
- 4 This is where a hacker attempts to crack a password by systematically trying different combinations of letters and numbers until the correct one is found.
- 5
 - Packet sniffing may be used to read the contents of data packets. This may be done by manipulating the network switch to route all data to a sniffing device which will then send them on to their intended destination once the data has been captured. Alternatively, it may be done by intercepting data being sent on wireless networks.
 - A man-in-the-middle attack may be used by setting up a fake Wi-Fi hotspot and luring people to use it. All data packets can then be captured and sniffed.
- 6 The details entered into the login fields on an online form includes SQL code, which enables hackers to circumvent the requirement to enter legitimate login details and allows them to gain access to the database.
- 7
 - It performs real-time scans of incoming network traffic to detect if they are infected with a virus.
 - It performs periodic scans of the whole system looking for malicious applications.
 - If it detected a virus or other malware, it is quarantined to prevent it from running and allows users to attempt to clean or remove it.
- 8
 - Malware
 - Denial of Service (DoS) attacks
- 9 Wireless networks are vulnerable to data packets being intercepted and read. Using encryption means that the data cannot be read by unauthorised persons, even if they manage to access the data.
- 3 Command line interface (CLI)
- 4 The memory manager controls where a program and its data will be stored in RAM. When a program is finished or the data is no longer needed, it frees up the space for re-use.
- 5 These are programs which act as a translator to allow the CPU and devices to communicate correctly.
- 6 It makes it easier for users to organise and find data in a systematic way. Files can be created, moved, re-named, saved or deleted, and the file manager keeps track of where they are physically located in secondary storage.
- 7 Programs that are not essential to the operation of the computer, but which are involved in maintaining a computer system.
- 8
 - Encryption software
 - Defragmentation
 - Data compression
- 9 To protect data from unauthorised access. The data is scrambled into a form that cannot be understood if it is accessed by unauthorised users.
- 10 The performance of a system is slowed as the disk needs to be accessed more frequently to read all of the data.
- 11
 - To reduce the size of files so that they take up less storage space.
 - To reduce the size of files so that they can be transmitted more quickly over the internet.

1.5 Systems software

- 1
 - Provides a user interface.
 - Controls the use of the RAM.
 - Shares processor time between different programs and processes.
 - Controls peripheral devices.
 - Controls who can access the computer and what resources they can use.
 - File management to allow users to organise their work into folders and subfolders.
- 2 It uses small icons to represent the files, devices and applications and allows the user to interact by clicking, dragging or touching them.

1.6 Ethical, legal, cultural and environmental impacts of digital technology

- 1 For example:
 - CCTV on the streets and in public places, private homes with CCTV, corporate buildings with CCTV, and the workplace
 - mobile phones can be tracked and are tracked by various apps
 - online activity in the workplace and by various websites, for example to monitor searches to target advertising
 - online monitoring of social media activity to provide a profile for organisations.
- 2 For example:
 - Social media posts are viewable by a wide audience often well beyond friends and acquaintances and may influence how the individual is seen by potential employers or members of specific groups or the general public.

- Families may see posts intended for close friends; employers may see unguarded moments from social activities.

3 For example:

- sharing recent activities with friends
- keeping friends up to date with what you like and are doing
- unguarded moments available to employers
- comments may solicit abuse, trolling.

4 For example:

- targeting advertising more effectively
- promoting special offers.

For example, searches for shoes may solicit social media advertising for various shoe brands or online retailers can provide better targeted promotions.

5 For example:

- use of electricity by data centres
- use of rare substances within the technology depleting resources
- energy used to manufacture devices
- toxic materials used and their disposal.

6 For example:

- work rate and work patterns
- time spent on breaks
- online activities
- work quality.

7 Yes:

- The activity may reflect on the company.
- It may identify opinions and activities of the individual that are incompatible with the company.

No:

- They are private posts from private computers.
- The individual has the right to their own opinions and the right to free speech.

- 8**
- Lawfulness, fairness and transparency: There must be valid reasons for collecting and using personal data.
 - Purpose limitation: The purpose for processing the data must be clear from the start.
 - Data minimisation: Data being processed must be adequate, relevant and limited to what is necessary.
 - Accuracy: Steps must be taken to ensure data is accurate, up to date and not misleading.
 - Storage limitation: Data must not be kept for longer than necessary.
 - Security: There must be adequate security measures in place to protect the data held.
 - Accountability: The data holder must take responsibility for how the data is used and for compliance.

- 9** This act makes it a criminal offence to access or modify computer material and includes hacking and the distribution of malware.

10 Proprietary software:

- Written by organisations trying to make a profit.
- The source code is kept securely and versions of the software are distributed as executable programs so that the user is not able to access the source code or modify it.
- The software is copyright protected, making it illegal to modify or distribute it.
- It is usually licensed for a fixed number of computer systems.
- The software is fully tested and supported by the organisation.

Open source software:

- Uses a community of developers.
- Software developed under open standards makes the source code available to everyone.
- Developed and updated by a community of programmers.
- Can be installed on as many computers as necessary.
- Others can modify the code and distribute it.
- Versions are made available at no or very little cost.
- Relies upon the community for testing and support, modified versions may not be supported or fully tested.

- 11** An organisation that issues licences allows a user to modify or distribute parts of the software under certain conditions. Also known as 'some rights reserved'.

Computational thinking, algorithms and programming

2.1 Algorithms

- 1**
- (a) Decomposition is breaking a problem down into smaller sub-problems. (Once each sub-problem is small and simple enough, it can be tackled individually.)
 - (b) Abstraction is removing or hiding unnecessary details from a problem (so that the important details can be focussed on or more easily understood).
 - (c) Algorithmic thinking is deciding on the order that instructions are carried out (in order to identify decisions that need to be made by the computer).

- 2 Steps may include:
 - generating the text for the message
 - encryption method
 - generating or selecting the key
 - encrypting the text
 - sending the encrypted text
 - sharing the key
 - decrypting the text.
- 3
 - The details of the game are removed.
 - Players' details are limited to name.
 - Winner and loser are identified without further details.
 - No ranking involved.

4

Line	NumOne	NumTwo	Output	Comment
01	8			
02		5		
04			5	Error prints smaller number, should print NumOne Or the condition in line 03 should be <=

- 5 It compares pairs of values starting at the beginning of the list.
If they are out of order, they are swapped and a flag is set to say a swap has taken place.
It repeats checking the list and swapping values that are out of order until the list is in order and no swaps are made.
Pass 1: 6, 2, 5, 8, 9 swap made
Pass 2: 2, 5, 6, 8, 9 swap made
Pass 3: 2, 5, 6, 8, 9 no swap list is sorted.
- 6 Dog, Cat, Mouse, Ant
Cat, Dog, Mouse, Ant

Cat, Dog
Mouse, Ant

 Sorted Unsorted
 Cat, Dog, Mouse, Ant

Cat, Dog
Mouse, Ant

 Sorted Unsorted
 Ant, Cat, Dog, Mouse
- 7 The list of values is split into individual lists of size 1.
- 8 It compares each value with 18 in turn until it finds a match or reaches the end of the list.
- 9 C
- 10 It searches through all of the list. If the item is not found, it declares that it is not in the list.

2.2 Programming fundamentals

- 1 A variable is (a memory location) used to store a single piece of data. This can be changed during the running of a program.
- 2 A variable can be changed by the program; a constant cannot be changed by the program.
- 3
 - Sequence is the execution of statements one after the other, in order.
 - Selection is the construct used to make decisions in a program.
 - Iteration is the construct used to repeat sections of code. Iteration is commonly called looping.
- 4 A While loop does something while a condition is met. A Do loop does something until a condition is met. A do loop will always run at least once.
- 5 (a) 11
(b) 21
(c) 4
(d) 2
(e) 4
(f) 4
(g) 4
(h) 44
- 6 (a) TRUE
(b) TRUE
(c) FALSE
(d) TRUE
- 7 (a) Integer
(b) Character
(c) Boolean
(d) String
(e) Real
(f) Real
- 8 (a) 16
(b) 'Co'
(c) 'g is fun'
(d) **text.right(3)**
- 9
 - Surname: string.
 - Telephone number: string.
 - Membership fee: real.
 - Membership paid: Boolean.
 - Number of goals scored: integer.
- 10 (a) Bill 02223334445
Graham 02232232232
Harry 01223123123
Sheila 01212121212

- (b) `SELECT First_name, Email FROM Address_book WHERE Surname = "Mills"`
- (c) `SELECT * FROM Address_book WHERE Email = SJ@home.vid`

- 11 (a) Blueberry
(b) Lime
(c) `fruit[1,2]`
- 12 Functions return values to the program; procedures do not return value to a program.
- 13 Subprograms
- allow programs to be split up into multiple sections
 - make the code easier to read and maintain
 - reduce the size of the code
 - reuse code without copying and pasting.

2.3 Producing robust programs

- Defensive design is where a programmer tries to anticipate ways in which their program could be misused or fail, and then puts measures in place to stop that happening.
- Using input validation to ensure that only expected types of input are accepted by the program, such as only allowing letters for a name, or suitable values for someone's age.
 - Including authentication systems to establish a user's identity before they are allowed to access the program, such as requiring a username and password.
- Two factor authentication requires the use of two different types of authentication. These are usually selected from something you know (such as a password), something you have (such as a code sent to a mobile phone in your possession), or biometrics (such as fingerprint scanners). For example, you may be asked to enter your password and then enter a code sent to you in a text message.
- Validation involves checking values against a set of rules to see if the data is sensible and as expected.
- A type check, to make sure that numbers are entered.
 - A range check, to make sure that the person is not over 100.
 - The use of drop-down lists so that users can only select from a pre-set range of values.
- It is not easy to validate a name because, apart from checking for any unusual characters, such as %, the length of someone's name can vary greatly, and there is no set range that it would fall into.

- Maintainability is important as it enables programs to be read easily, and therefore for errors to be identified and fixed, or updates and modifications to be made easily.
- The name should relate to the contents being stored in the variable to help programmers identify their purpose.
 - Variable names should start with a lowercase letter.
- Use of indentation.
 - Use of sub-programs.
 - Use of comments.
- The second line would be more useful because the variable names make it clear what data is being used in the calculation.
- Destructive testing involves trying to find ways to break a program and make it fail to ensure that it is fully robust.
- Testing is used to ensure that a program works as expected and meets all of the requirements.
- Iterative testing involves checking each part of the program as it is developed, whereas final testing is carried out once the whole program has been written.
- A logic error will still allow the program to run but will produce an unexpected result, whereas a syntax error will stop a program from running in the first place.
- This would cause a syntax error because FOR is a keyword and so the program does not make sense and will not run.
- This would cause a logic error. The program would still run, but the result would be incorrect.
- Normal test data is data that is of the correct type that you would expect a user to input. It should be accepted and not cause any errors.
- Boundary test data is of the correct type and is at the very edge of what should be accepted, allowing the program to run without causing errors. Invalid test data is of the correct type but should not be accepted as it is not within the expected range.
- Any appropriate password that is less than eight characters or more than 15 characters long.

Test data	Type of test data	Expected result
For example, 25 (Any value above 18)	Normal	Accepted
18	Boundary	Accepted
For example, 17 (Any value below 18)	Invalid	Rejected
For example, twenty (Any value that is not an integer)	Erroneous	Rejected

- 21** Refining algorithms means improving them, for example fixing any errors or making the code more efficient.
- 22**
- Errors identified when testing would need to be fixed.
 - Sections of code that are repeated should be made into sub-programs, or use iteration.
 - Poorly named variables would need to be renamed to something more meaningful

2.4 Boolean logic

1

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

2

A	B	P
0	0	1
0	1	0
1	0	1
1	1	1

3

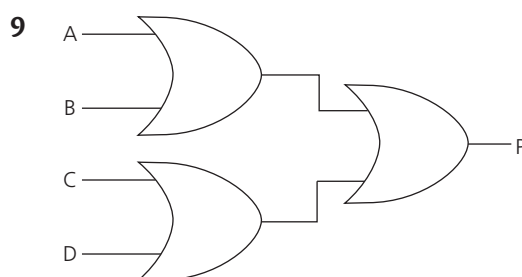
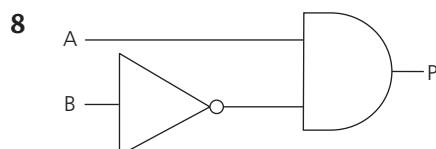
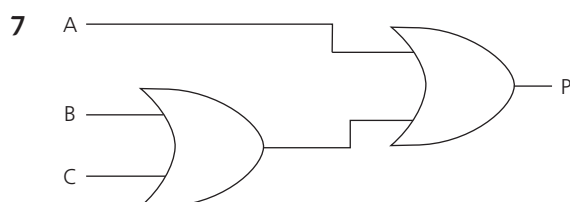
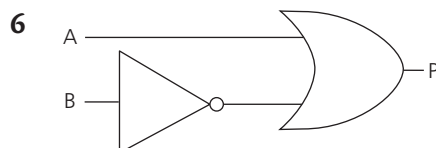
A	B	P
0	0	1
0	1	0
1	0	0
1	1	0

4

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

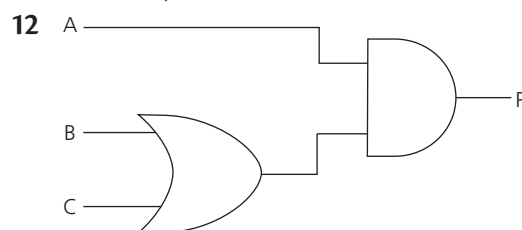
5

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



- 10**
- (a) (1 1 0 1 1)
- (b) (0 0 1 1 0)
- (c) (1 0 0 0 0)

11 For example: Cost £1900, no Pool in USA



13

A	B	P
0	0	0
0	1	1
1	0	0
1	1	0

2.5 Programming languages and integrated development environments

- 1** Any three from Python / C# / C++ / Visual Basic / Ruby / Pascal / Fortran / Java / JavaScript or other suitable alternatives.
- 2**
 - They use English-like syntax which makes them easier for programmers to use.
 - They use abstraction to hide the details of the underlying instructions that need to be completed by the processor.
- 3** They enable programs to be run very quickly.
- 4**
 - Using an interpreter. This translates and runs the code one line at a time.
 - Using a compiler. This translates the whole program into machine code and produces an executable file.
- 5** IDE stands for integrated development environment. It is software that provides all the tools needed to develop computer programs.
- 6** An editor in a word processor allows text to be entered and saved, and some formatting such as bold or coloured text can be added manually. An IDE often supports pretty printing, where variables and key words are displayed in certain colours, or code is automatically indented, to make it easier to read and understand.
- 7**
 - An inbuilt debugger program will identify syntax errors within the code when it is run.
 - A programmer can step through the program line by line, to check how the program is running and check the contents of variables. Breakpoints can be set which halt the program at a particular point.
- 8** Either of the following:
 - translator
 - run-time environment.

INDEX

A

- abstraction 106–7, 118, 175
- accumulator (ACC) 4, 10
- adding binary numbers 24–6, 46–7
- algorithmic thinking 106, 118
- algorithms 107
 - flowcharts 109–10, 119–20
 - Input–Process–Output 108, 118
 - pseudocode 111, 113, 120
 - refining 160, 163
 - searching 116–17, 124
 - sorting 112–16, 122–3
 - structure diagrams 108–9, 119
 - trace tables 111, 121
- analogue data 36, 50
- AND gates 165, 166, 171
- AND and NOT gates 166
- AND and OR gates 166–7
- AND operator 132, 148
- anticipating misuse 152–3, 161
- anti-malware software 77–8, 82
- applications software 83
- Arithmetic Logic Unit (ALU) 3, 10
- arithmetic operators 131, 148
- arrays 150–1
 - one-dimensional 141
 - two-dimensional 141–2
- ASCII (American Standard Code for Information Interchange) 31
 - extended set 32
- assembly language 175
- assignment 126, 146
- authentication 153, 161
- automated decision-making 95
- automatic number plate recognition (ANPR) 97

B

- bandwidth 54, 69
- batch files 84

- binary codes 31
- binary numbers 20, 22, 44–5
 - adding 24–6, 46
 - conversion to and from denary numbers 22–4, 45–6
 - conversion to and from hexadecimal 30–1, 48–9
 - overflow errors 26
- binary search 116–17, 124
- binary shifts 27–8, 47
- biometric authentication 79, 153
- bit depth
- images 35
- sound sampling 37, 50
- bits 20, 45
- black box testing 157
- Bluetooth 58, 64, 71
- Blu-Ray 18, 43, 44
- Boolean conditions 127, 147
- Boolean data type 164
- Boolean logic 171–2
 - combining operators 166–7, 172
 - logic diagrams 165–9
 - solving real-life problems 170, 173
 - truth tables 164
- Boolean operators 132–3, 148
- Boolean variables 134, 149
- botnets 75
- boundary test data 159, 163
- breakpoints 178–9
- brute force attacks 75, 78, 79, 82
- bubble sort algorithm 112–13, 116, 122
- buffering 85, 89
- buses 5
- bytes 20, 45

C

- cables 57
- cache memory 3, 6–7, 10, 11
- casting 135–6, 149
- CCTV cameras 92, 97
- CDs 18, 43, 44

- censorship 94–5
- central processing unit (CPU) 9
 - components and their function 3–4, 10
 - performance 6–7, 10–11
 - purpose of 2
 - Von Neumann architecture 4–5, 10
- channels 63
- character sets 31, 49
- characters 49, 134–5, 149
 - ASCII 31–2
 - binary codes 31
 - Unicode 32
- ciphertext 86
- client computers 61
- client–server networks 55–6, 69
- clients 55
- clock speed 6, 10
- closing files 138
- cloud computing 60, 70
- colour depth 35, 50
- colours 33–4
- command line interfaces (CLIs) 84, 89
- comments 156, 162
- comparison operators 132, 148
- compilers 176–7, 180
- compression 39–40, 51, 87–8, 90
 - lossless 40
 - lossy 39
- computational thinking 106–7, 118
- Computer Misuse Act 1990 99, 103
- computer networks *see* networks
- computer systems 2
- concatenation 137, 149
- condition-controlled iteration 130–1, 147
- constants 126, 146
- control unit (CU) 3, 10
- copper wire cables 57
- Copyright, Designs and Patents Act (CDPA) 1988 99, 103
- cores, number of 7, 11
- count-controlled iteration 129–30, 147
- Creative Commons 100, 104
- cultural issues 93–5, 101

D

- data capacity calculations 20–1, 45
- data collection 94, 97
- data compression 39–40, 51, 87–8, 90
 - lossless 40
 - lossy 39
- data interception and theft 76, 79, 82
- data packets 54, 76
- Data Protection Act 2018 97–9, 102–3
- data searches 139–40, 150
- data storage
 - characters 31–2, 49
 - compression 39–40, 51
 - images 33–5, 50
 - metadata 34
 - numbers 22–30, 45–9
 - records 139, 150
 - sound 36–8, 50–1
 - units of 19–20, 44–5
 - see also* memory; secondary storage
- data transfer speeds 13
- data types 134–5, 148–9
 - Boolean data 164
 - casting 135–6
- debuggers (error diagnostics) 178–9, 181
- decision-making, automated 95
- decomposition 106, 118
 - structure diagrams 108–9
- defensive design 161–2
 - anticipating misuse 152–3
 - authentication 153
 - input validation 154
 - maintainability 155–6
 - testing 157–60
- defragmentation 86–7, 90
- denary (decimal) numbers 22
 - conversion to and from binary numbers 22–4, 45–6
 - conversion to and from hexadecimal 30, 48
- denial of service (DoS) attacks 75, 79, 82
- destructive testing 157
- device drivers 85, 90
- discrete data 36
- distributed denial of service (DDoS) attacks 75
- DIV operator 131, 148
- DNS servers 74
- DO UNTIL loops 130–1
- Domain Name Server (DNS) 59
- domain names 59, 70
- drivers 85, 90
- DVDs 18, 43, 44

E

- editors 178, 181
- efficient programs 160
- electromagnetic spectrum 58
- electronic tagging 97
- ELSE statements 128–9
- embedded systems 7–8, 11
- encapsulation 67
- encryption 64, 66, 79, 82, 86, 90
- energy use 95
- engine management systems 8
- ENIAC (Electronic Numerical Integrator And Computer) 174
- environmental issues 95–6, 102
- erroneous test data 159, 163
- error diagnostics (debuggers) 178–9, 181
- errors 158, 162
- ethernet 63, 70
 - advantages and disadvantages 64
- ethics 91–3, 101
- exponent (^) operator 131, 148
- extended ASCII 32, 49

F

- Fetch–Execute cycle 2, 5, 9
- fibre-optic cables 57
- field names 139, 150
- File Allocation Tables (FATs) 85
- file handling operations 138–9, 149–50
- file management 83, 85, 90
- file permissions 85
- final (terminal) testing 157, 162
- firewalls 78, 82
- flash memory 17, 43
- floating point numbers 134
- flow diagram symbols 110
- flowcharts 109–10, 119–20
- FOR loops 129–30
- fragmentation 17, 86–7, 90
- frequency bands 63
- FTP (File Transfer Protocol) 66, 71
- functions 144, 151
 - random number generation 145

G

- General Data Protection Regulation (GDPR) 98
- graphical user interfaces (GUIs) 83–4, 89

H

- hacktivists 75
- hard disk drives (HDDs) 15–16, 42, 44
- hardware 2, 57–8, 70
- headers 66
- heuristic analysis 78
- hexadecimal (hex) 28, 48–9
 - conversion to and from binary numbers 30–1
 - conversion to and from denary numbers 28–9
- high-level languages 175–6, 180
- Hopper, Grace 176
- hosts, internet 60
- HTTP (Hypertext Transfer Protocol) 66, 71
- HTTPS (Hypertext Transfer Protocol Secure) 66, 71
- hubs 58

I

- identifiers 125
- IF statements 127–9
- images 33–4, 50
 - colour depth and resolution 35
 - metadata 34
- IMAP (Internet Message Access Protocol) 66, 72
- impacts of digital technology 91
 - cultural issues 93–5, 101
 - environmental issues 95–6, 102
 - ethical issues 91–3, 101
 - legal issues 93, 101
 - privacy issues 96–7, 102
- increments 129
- indentation 155–6, 162
- index values 141
- input validation 154, 161
- inputs 108, 118, 126, 146
- insertion sort algorithm 113–14, 116, 122
- integers 134, 148
- integrated development environments (IDEs) 177–9, 181
- internet 59–61, 70
 - copyright 94–5
 - policing of 93
 - tracking 97
- Internet Protocol (IP) 59, 66, 72
- interpreters 176–7, 180
- invalid test data 159, 163
- IP addresses 59, 64–5, 70, 71, 78
- iteration (looping) 129–31, 147
 - nested loops 141–2
- iterative testing 157, 162

K

keys 79, 86

L

lands 18

latency 17, 49, 54

layers 67–8, 72

least significant bit (LSB) 22

legislation 93, 101, 102–4

Computer Misuse Act 1990 99

Copyright, Designs and Patents Act 1988 99

Data Protection Act 2018 97–9

software licences 99–100, 103–4

linear search 116, 117, 123

local area networks (LANs) 52–3, 69

client–server networks 55–6

logic diagrams 165–7, 171–2

creation from expressions 168–9

logic errors 158, 162

London Underground map 107

looping (iteration) 129–31, 147

nested loops 141–2

lossless compression 40, 51, 87

lossy compression 39, 51, 87

low-level languages 174–5, 180

M

MAC (Media Access Control) addresses 57, 65, 70, 71, 78

machine code 4, 174–5

magnetic storage 15–16, 42, 44

magnetic tape 16

maintainable programs 155–6, 161

malware 73–4, 79, 81–2

anti-malware software 77–8

man-in-the-middle (MITM) attacks 76

megabits (Mb) 54

megabits per second (Mbps) 54

memory 41–2

cache 3, 6–7

comparison of RAM and ROM 13

in embedded systems 7

random access (RAM) 12–13

read-only (ROM) 13

virtual 14

see also secondary storage

memory address register (MAR) 4, 10

memory data register (MDR) 4, 10

memory management 83, 84–5, 89

merge sort algorithm 114–15, 122–3

mesh network topology 62, 70

metadata 21, 34, 50

microcontrollers 7

mnemonics 175

mobile phone signal tracking 92, 97

modules 157, 162

modulus (MOD) operator 131, 148

most significant bit (MSB) 22

multitasking 85, 89

N

naming conventions 155, 162

natural language interfaces 84

nested loops 141–2

network diagram 58

network interface controller/card (NIC) 57, 70

network performance 69

influencing factors 54

network protocols 66, 70

network topologies 61–2, 70

networks

addresses 64–5, 71

advantages and disadvantages 52

client–server 55–6, 69

encryption 64

hardware 57–8, 70

internet 59–61, 70

layers 67–8, 72

modes of connection 63–4, 70–1

peer-to-peer 56, 69

types of 52–3, 69

nibbles 20, 45

non-volatile memory 41

ROM 13, 41–2

normal test data 159, 162

NOT gates (inverters) 165, 166, 171

NOT operator 133, 148

numbers

binary 20, 22–8

denary (decimal) 22

hexadecimal (hex) 28–30

O

one-dimensional arrays 141, 150

opening files 138, 149

open-source software 99–100, 103–4

operating systems 83, 89–90

file management 85

memory management and

multitasking 84–5

peripheral device management

and drivers 85

user interface 83–4

user management 85

operator precedence 131

operators 126, 148

arithmetic 131

Boolean 132–3

comparison 132

file handling 138–9, 149–50

string manipulation 136–7, 149

optical storage 18, 43

OR gates 165–6, 172

OR operator 132–3, 148

outputs 108, 118, 126, 146

overflow errors 26, 47

P

packet sniffing 76

parameters 143, 151

passwords 78, 153

payloads 66

peer-to-peer (P2P) networks 56, 69

penetration testing 77, 82

peripheral device management 83, 85, 90

pharming 74, 81

phishing 74–5

physical security 79, 82

PINs (personal identification numbers) 75

pits 18

pixelation 35

pixels 33, 50

POP (Post Office Protocol) 66, 71

presence checks 154

pretexting (blagging) 75

pretty printing 178

primary storage 12, 41–2

random access memory (RAM)

12–13

read-only memory (ROM) 13

virtual memory 14

privacy issues 96–7, 102

procedures 143–4, 151

processes 108, 118

processor cores 7, 11

program counter (PC) 4, 10

programming

arrays 141–2

data searches 139–40

data storage 139

data types 134–6

defensive design 152–6, 161–2

file handling 138–9

inputs and outputs 126

- integrated development environments (IDEs) 177–9, 181
- iteration (looping) 129–31
- operators 126, 131–3
- random number generation 145
- selection 127–9
- sequence of instructions 127
- string manipulation 136–7
- subprograms 142–4
- testing 162–3
- variables, constants and assignments 125–6
 - see also* algorithms
- programming languages 180
 - levels of 174–6
 - translators 176–7
- proprietary software 99–100, 103–4
- protocols 63, 66, 70, 71
 - Internet Protocol (IP) 63
- pseudocode 111, 113, 120

Q

quotient (DIV) operator 131, 148

R

- radio waves 58, 70
- random access memory (RAM) 12–13, 41–2
- random number generation 145, 151
- ransomware 74, 81
- reading files 138, 149
- read-only memory (ROM) 13, 41–2
- real numbers 134, 148
- records 139, 150
- refining algorithms 163
- registers 4, 10
- reserved keywords 125
- resolution 35, 50
- resource use and recycling 95–6
- robust programs 152
- routers 58, 70
- runtime environment 179, 181

S

- sample rate 37, 50
- sandboxes 78
- searching algorithms
 - binary search 116–17, 124
 - comparison of 115–17
 - linear search 116, 123

- searching for data 139–40, 150
- secondary storage 15, 42–3
 - capacity and cost 18
 - choice of 18–19, 43–4
 - magnetic 15–16
 - optical 18
 - portability, durability and reliability 19
 - solid-state 17
 - speeds 19
- security measures 77–9, 82
- security threats 73–6, 81–2
- selection 127–9, 147
- sequence of instructions 127, 147
- servers 55, 74
 - web servers 60
- Service Set Identifier (SSID) 58
- shell scripts 84
- shouldering (shoulder surfing) 75
- SMTP (Simple Mail Transfer Protocol) 66, 72
- social engineering 74–5, 79, 81
- social media 93
 - privacy issues 96–7
- software 2
 - anti-malware 77–8, 82
- software licences 99–100, 103–4
- solid-state drives (SSDs) 17, 43, 44, 87
- solid-state storage 17, 43
- sorting algorithms
 - bubble sort 112–13, 122
 - comparison of 115
 - insertion sort 113–14, 122
 - merge sort 114–15, 122–3
- sound sampling and storage 36–8, 50–1
- spyware 74, 81
- SQL injection 76, 79, 82
- SSL (Secure Socket Layer) encryption 66
- stakeholders 92
- standards 65, 71
- star network topology 61, 70
- storage *see* data storage; memory;
 - secondary storage
- string manipulation 136–7, 149
- string slicing 137, 149
- strings 135, 149
- structure diagrams 108–9, 119
- Structured Query Language (SQL) 76, 139–40, 150
- subprograms (subroutines) 110, 142–3, 151, 155, 161
 - functions 144
 - procedures 143–4
- SWITCH/CASE statements 129
- switches 19
 - networks 58, 70, 76
- syntax errors 158, 162

- systems software 83
 - operating systems 83–5, 89–90
 - utility software 86–8, 90

T

- TCP (Transport Control Protocol) 66, 72
- ten commandments for computer ethics 92
- terminal (final) testing 157, 162
- test data 159, 162–3
- test plans 159, 163
- testing 162–3
 - purpose of 157
 - refining algorithms 160
 - syntax and logic errors 158
 - types of 157
- trace tables 111, 121
- translators 176–7, 179, 180, 181
- transmission errors 54
- transmission media 54, 57–8, 70
- Trojans 73, 81
- truth tables 164–7, 171
- two-dimensional arrays 141–2, 151
- two-factor authentication (2FA) 79, 153

U

- Unicode 32, 49
- Uniform Resource Locators (URLs) 59
- USB connectors 65
- user access rights 78, 85, 90
- user interface 83–4, 89
- user management 83
- usernames 153
- utility software 86, 90
 - data compression 87–8
 - defragmentation 86–7
 - encryption 86

V

- validation 154, 161
- variables 125–6, 146
- virtual machines 179
- virtual memory 14, 42
- viruses 73, 81
 - anti-malware software 77–8
- voice-controlled systems 84
- volatile memory 41
 - RAM 13, 41–2
- Von Neumann architecture 4–5, 10

W

web servers 60, 70

WHILE loops 130–1

white box testing 157

wide area networks (WANs) 53, 69

Wi-Fi 58, 63, 71

advantages and disadvantages 64

bandwidth 54

encryption 64

Windows, Icons, Menus and Pointers

(WIMP) interface 84

wireless access points (WAPs) 58, 70

workplace

computer use 94

logging systems 92–3, 97

worms 73, 81

writing to files 138–9, 149

ACKNOWLEDGEMENTS

The Publishers would like to thank the following for permission to reproduce copyright material.

Adobe is either a registered trademark or trademark of Adobe in the United States and/or other countries.

Apple product screenshot(s) reprinted with permission from Apple.

Google and the Google logo are registered trademarks of Google LLC, used with permission.

Libre Office product screenshot(s) reprinted under the Creative Commons Attribution-Share Alike 3.0 License (<https://creativecommons.org/licenses/by-sa/3.0/>).

Microsoft product screenshot(s) used with permission from Microsoft.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Python is a registered trademark of the Python Software Foundation.

Photo credits

p. 1 © Raia/stock.adobe.com; **p. 2** © Ymgerman/Shutterstock.com; **p. 7** © Sergey Jarochkin/123RF.com; **p. 8** © samunella/stock.adobe.com; **p. 16** *t* © mingis/istock/thinkstock, *b* © Gianni Furlan/Getty Images/Hemera/Thinkstock; **p. 17** © sergojpg /stock.adobe.com; **pp. 34, 35** and **39** © George Rouse; **p. 84** *screenshot background image* © Mike Berenson/Colorado Captures/Moment/Getty Images; **p. 94** © Andrey Rudakov/Bloomberg/Getty Images; **p. 96** © Peter Essick/Aurora Photos/Cavan/Alamy Stock Photo; **p. 105** © Casimiro /stock.adobe.com; **p. 107** © TfL from the London Transport Museum collection; **p. 164** Wellcome Collection. Attribution 4.0 International (CC BY 4.0); **p. 174** © Bettmann/Getty Images; **p. 176** © Division of Medicine and Science, National Museum of American History, Smithsonian Institution.

Every effort has been made to trace all copyright holders, but if any have been inadvertently overlooked the Publishers will be pleased to make the necessary arrangements at the first opportunity.

CODE-IT in Python

Discover a new way to help students learn how to code with Code-IT in Python.

Help your GCSE students progress beyond simple programming skills and removing any fear they might have transitioning from a block-based language to a text-based language with our coding resource, Code-IT for Python.

Code-IT in Python consists of 14 stand-alone modules, each focused on a different programming content required at GCSE – allowing you to pick and choose which modules your students need. Using our responsive, online environment, students are encouraged to write and test their own code in order to solve Coding Challenges.

It's not your average 'how-to code' product

A digital resource that provides your students with a learning journey through essential programming skills required at GCSE.

It will fill students' coding skills gap

Designed to focus on a range of programming skills, Code-IT in Python will equip your students with the necessary tools needed to complete any GCSE programming task effectively and efficiently.

It will save you time!

Auto-marked Coding Challenges require students to write, test and de-bug their code. Feedback is given immediately, so students can understand the areas they need to amend and learn from.

It's packed full of resources

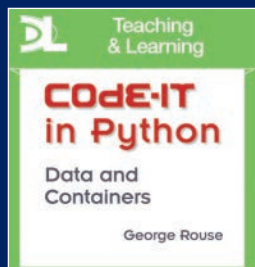
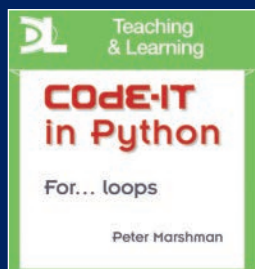
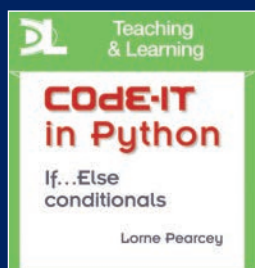
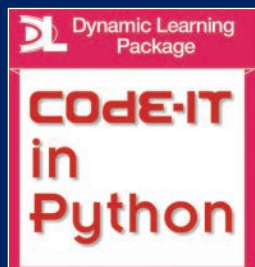
As well as detailed progress reports on students' activity, you will find guidance sheets, lesson ideas and starter presentations to help reinforce learning and cut down on the time you spend creating new resources.

Pick and choose the modules you want

£30 + VAT per module for one-year access. Save up to 20% by exploring our bundle offers.

Visit our website or contact your local Sales Representative to find out more about Code-IT in Python and to register for a free trial.

www.hoddereducation.co.uk/code-it computing@hoddereducation.co.uk



Please note,
none of the
Code-IT in
Python modules
are endorsed by
OCR

FOR THE
J277
SPECIFICATION

OCR GCSE
(9–1)

COMPUTER SCIENCE

SECOND EDITION

Written by leading Computer Science teachers and endorsed by OCR, this brand-new textbook will guide students through the updated **OCR GCSE Computer Science** specification topic by topic, and provide them with standalone recap and review sections, worked examples and clear explanations of complex topics.

This Student Book:

- develops computational thinking skills in line with the new Practical Programming element of Component 02
- provides differentiated material with the 'beyond the spec' feature
- includes standalone recap and review sections at the end of each chapter
- provides definitions of technical terms, along with a glossary of words that will be needed for assessment.

George Rouse, Lorne Pearcey and **Gavin Craddock** are highly respected and widely published authors of resources.

Dynamic Learning

This book is fully supported by Dynamic Learning – the online subscription service that helps make teaching and learning easier. Dynamic Learning provides unique tools and content for:

- front-of-class teaching
- streamlining planning and sharing lessons
- focused and flexible assessment preparation
- independent, flexible student study



Sign up for a free trial – visit: www.hoddereducation.co.uk/dynamiclearning

Endorsed by OCR

- ✓ This title fully supports the specification
- ✓ It has passed OCR's rigorous quality assurance programme
- ✓ It is written by curriculum experts

HODDER EDUCATION

t: 01235 827827

e: education@bookpoint.co.uk

w: hoddereducation.co.uk

ISBN 978-1-5104-8416-0

