CAMBRIDGE

Brighter Thinking

COMPUTER SCIENCE

A/AS Level for OCR

Teacher's Resource Component 1

Christine Swan and Ilia Avroutine

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

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

www.cambridge.org

Information on this title:
www.cambridge.org/ukschools/9781107496774 (Cambridge Elevate-enhanced Edition)

© Cambridge University Press 2015

First published 2015

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

ISBN 978-1-107-49677-4 Cambridge Elevate-enhanced Edition

Additional resources for this publication at www.cambridge.org/ukschools

Acknowledgements
Cover provided by Fabian Oefner © 2013 www.fabianoefner.com

How to use Cambridge Elevate resources

Cambridge Elevate includes a number of features that enables you to customise this resource:

- You can re-order the contents of this resource through Cambridge Elevate, to suit your teaching sequence.
- You can create your own notes (highlights, annotations, voice notes) to help build information within the resource.
- You can also add hyperlinks between sections within the resource, to sections within other resources within Cambridge Elevate, or to web pages.

If your centre is using the accompanying Cambridge Elevate-enhanced Editions for OCR A/AS Level Computer Science Component 1 and Component 2, you can use the following features to support your teaching:

- You can choose to share annotations, bookmarks, etc., with groups of students. So, for example, if you have different ability levels within your class, you can use groups to share links to more stretching activities with the higher-ability students only.
- You can share links within and between Cambridge Elevate resources with groups of students. For example, if you wanted students to link concepts or learning between Component 1 and Component 2. Please note, however, that the groups need to have access to the relevant titles in order for this to work. For example, you couldn't share a link between Component 1 and Teaching Programming as your students will not have access to Teaching Programming.
- Component 1 and Component 2 include 'Assess to Progress' features, which allow you to set students specific questions from the end-of-chapter tests. Once the student has submitted their work, this feature will allow you to mark their work against the assessment objectives and a set of levelling statements, and support you in monitoring and reporting your students' progress over the duration of the course.

You can find more information about the functionality available to you within Cambridge Elevate and how you can use it in your teaching within the Cambridge Elevate Student and Teacher User Guide, which is available from your Home screen.

Chapter 1: Structure and function of the processor and types of processor

Learning objectives

- Know the different parts of a computer's processor and their purpose.
- Know about different types of processor architecture.
- Understand the term 'machine code' and how it relates to software.
- Understand the FDE cycle and the special purpose registers involved.
- Be able to describe pipelining and explain how it improves efficiency.

What your students need to know

- A von Neumann processor consists of a single Arithmetic Logic Unit (ALU), a single Control Unit (CU) and a single Memory Unit (MU).
- The ALU is where arithmetic and logical operations are performed and the result stored in a special register called the Accumulator (ACC).
- The CU controls the timing of operations by generating a clock signal. Operations take place at regular intervals according to this signal.
- The MU is where programs are stored when they are ready to run. Each instruction and item of data in the program will have a memory address associated with it.
- All of the components inside the central processing unit (CPU) are connected by the bus system. The address bus carries memory addresses from the memory unit to registers in the CPU. The data bus carries data and the control bus carries control signals.
- There are different types of CPU. To increase efficiency, an additional co-processor may be installed to carry out specific operations (e.g. floating point). Parallel processors use multiple CPUs to complete more tasks per unit time. Array processors, or vector processors, all work together on the same task. Multicore processors can support multiple paths of execution (multithreading). An example would be a modern quad or dual core CPU.
- A Graphics Processing Unit (GPU) is a separate processor on the motherboard or separate graphics card. Its role is to handle graphics operations and improve user experience. It is specifically designed to process large amounts of data but on a specific task. A GPU can be thought of as a vector processor.
- Each CPU has an instruction set that it supports. Modern CPUs tend to have complex instructions (e.g. an Intel® or AMD® processor). This is Complex Instruction Set Computer (CISC) architecture. Each instruction may take more than one clock cycle to complete.
- Some CPUs have simpler instructions. This is Reduced Instruction Set Computer (RISC) architecture. Examples include the ARM® processor in many smartphones and the Raspberry Pi® computer. Each instruction is simple and takes only one clock cycle to complete.
- CPUs can be programmed in binary (machine code) or a low-level language called Assembly language. This uses mnemonics to represent the instructions. The instruction set, and therefore the Assembly language, will differ slightly between CPUs. The CPU will need to have a program called an assembler to convert the Assembly language code into machine code.
- The Fetch-Decode-Execute (FDE) cycle is the process by which programs are run. In the first part of the cycle, Fetch, an instruction is copied from the memory unit into a special register called the Current Instruction Register (CIR). The assembler needs to work out from the mnemonic which operation it represents. This is the Decode part. Finally, some data (operands) will be fetched from the memory unit and copied into the Memory Data Register (MDR). The instruction is then carried out on the operands in the ALU and the result stored in the ACC.

- In most cases, programs stored in the memory unit run sequentially, starting at the lowest address and using every memory address to the end of the program. A register called the Program Counter stores the address of the next instruction so the CPU knows where to look. The Memory Address Register stores a copy of the current memory address being worked on. When a program contains a branch or jump instruction, it will also specify the memory address to go to. This will be seen in selection and iteration constructs.
- Running programs sequentially in a von Neumann processor is relatively inefficient. Alongside developments in parallelism, the concept of pipelining developed. This allows resources to be used for maximum efficiency rather than having a single execution thread that must be completed before another can begin. Pipelining allows the execution of multiple threads as soon as the resources become available.

Common misconceptions

- Students can confuse the purposes of different registers and different processor architectures. It would be helpful for students to produce a glossary or their own wiki to work collaboratively on definitions of key terms.

Lesson activity suggestions

One useful activity is to present several physical CPUs for students to look at. If possible, try to find examples of different generations and from different types of devices. An email request to colleagues may produce some interesting specimens and none of the devices need to be functional. It is useful to include some examples of System on Chip (SoC) devices which combine CPU, random-access memory (RAM) and GPU into a single chip. These are commonly found in small systems such as a hand-held games console or mobile phone. These usually prompt some interesting questions such as: 'Where is the RAM?'

Students may find the microarchitecture of the von Neumann CPU difficult to grasp as there are a number of new acronyms and they need to appreciate how each component interacts with the others. An interactive and fun way to do this is to build a 'jigsaw' model of a von Neumann CPU. This encourages students to think about connections within the CPU and the Fetch-Decode-Execute (FDE) cycle.

If your students have access to Raspberry Pi® computers, these can be safely configured for overclocking. Performance improvements can then be measured.

The range of processor types can be investigated from looking at the range of tasks that students are expected to perform. For example, students will appreciate that an array processor will be more efficient at processing graphics in a game than a single-core von Neumann CPU.

The Little Man Computer (LMC) is a CPU simulator that can be downloaded or accessed online as a Java applet. This is a clear example for students of the link between a program written in low-level code and the operation in the CPU that produces the output. Once students have become familiar with the instruction set of the Little Man CPU, they can begin to appreciate the components required to build a working program. The card game described below, 'Whose Line of Code is it Anyway?', is a fun way to reinforce these concepts.

Students can assume the role of a CPU component and work collaboratively to 'execute' a simple program. This activity reinforces the sequential nature of processes. This also assists with understanding the name of specific registers and their role.

The final major concept in this section is that of pipelining. Simply making a processor faster will not have the biggest impact on performance. Students will appreciate that whole programs can be broken down into more simple processes and that some of these can take place at the same time as other processes. It is beneficial to use examples that will be familiar to students, such as doing the laundry or cooking a meal.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Processor components	**Starter:** Have a CPU 'show and tell', giving examples of processors in different devices (e.g. desktop PC, laptop, smartphone, games console, Raspberry Pi®). **Main activity:** Make a paper 'jigsaw' of components of a typical von Neumann processor. Identify the purpose of each component. Completed jigsaws could be wall-mounted as a poster. **Raspberry Pi® activity:** Investigate overclocking using the pre-set values available in the configuration panel. **Plenary:** Discuss how processors have evolved to increase system performance (e.g. multicore, multithreading and increased levels of cache memory).	Research a project on use of CPUs and microprocessors in mobile phones and games consoles. Research a project on use of CPUs and microprocessors in mobile phones and games consoles. Research the contribution of John von Neumann.
Processor architecture and low- level code	**Starter:** Discuss processor architecture and application 'walking wall', including von Neumann, vector, and parallel. **Main activity:** Conduct an LMC investigation involving basic programming, including challenges of iteration and selection. Play a card game involving LMC instructions: 'Whose Line of Code is it Anyway?' Students must lay cards in sequence taking turns to make a correctly functional program which can then be tested using the LMC simulator. **Plenary:** Discuss the difference between a CPU, a microprocessor (e.g. in a washing machine) and a SoC (e.g. in a Raspberry Pi®).	Practise examination questions. Set an LMC programming challenge of multiplication and division using the LMC.

Topic	Lesson suggestions	Follow-up ideas/Homework
The FDE cycle	**Starter:** Discuss what an instruction is and how we carry out a sequence of instructions. Illustrate using a task to make an origami paper model. **Main activity:** As a whole-class activity, simulate a CPU. Each student assumes the role of a component and runs a simple program to add two integers. **Plenary:** Discuss if it is better to have many simple instructions (RISC) or fewer, more complex instructions (CISC).	Investigate interactive resources to learn the FDE cycle. Produce a comparison of RISC and CISC architectures giving examples of computers that use each.
Pipelining	**Starter:** Discuss what happens in the CPU in one clock cycle. **Main activity:** Investigate increasing processor efficiency and perform a pipelining simulation. Explain using 'doing the laundry'. Students should simulate the efficiency of pipelining by making paper models assembled from pieces. **Plenary:** Discuss the increase in performance of systems that use pipelining compared to those that do not. How could the performance of CPUs be improved still further in the future?	Produce an example activity for pipelining – cooking a meal. Investigate supercomputers and their uses.

End-of-chapter answers

1
a. The control unit manages the fetching and executing of instructions. It synchronises actions using the built-in clock and sends control signals to other parts of the computer.
b. Executes all arithmetic instructions, such as adding, as well as logical operations such as testing for greater than. It also acts as a conduit, which all I/O passes.
c. The memory unit will store the operating system, data that is currently in use and any software currently in use.

2
a. Mnemonics are acronyms, which are used in assembly code, such as ADD. A single mnemonic represents a single machine code instruction in one-to-one mapping.
b. Machine code instructions are split into two main parts. The first part is the opcode, which specifies the type of instruction to be executed. The second part is the data, which the instruction requires.

3
a. Cache memory is fast memory located between, or on, the CPU and RAM. Instructions and data, which are currently in use are copied to the cache to help reduce the delay in fetching data from the CPU.
b. The CPU runs much faster than RAM, which means that when data or instructions are fetched a delay will occur and the CPU could potentially be idle. The von Neumann bottleneck is such that the CPU can only run as fast as the slowest device currently in use, meaning that the CPU would be throttled to the same speed as RAM. Cache memory, which is much faster than RAM, helps overcome the bottleneck by pre-empting what data needs to be fetched. Data fetched from cache helps reduce, but not eliminate, the idle time for the CPU thus improving overall performance.

4 Data is fetched into the CPU by first copying the PC into the MAR. The address from the MAR is used to fetch data into the MDR while the PC is incremented to point to the next instruction. The instruction is then copied from the MDR to the CIR (current instruction register) ready for execution. As the instruction is being decode and executed the next instruction can be fetched. Pipelining is where different stages of the FDE cycle are done in parallel, only on different instructions. As one instruction is fetched another could be decoded and executed. This helps reduce the delay caused by fetching. However, if a jump command occurs then fetching the next instruction will be a wasted operation as the value of the PC will change. This is why any form of branching is considered to be a computationally expensive operation.

5 Assembly code has one-to-one mapping with machine code meaning that each assembly mnemonic represents a single instruction. A single Python instruction will map too many machine code instructions. The mapping of Python instructions to machine code instructions is managed by the Python interpreter. Most user level programs, such as a mobile phone app, would be produced in a high-level language. It is much easier to code in high-level languages as a lot of the complexities of how the system works, such as how fonts are displayed on the screen, are managed for you through the use of library software. However, you have limited control over the hardware and have to rely on the interpreter to produce efficient machine code. For systems, which are much closer to the hardware, like an operating system, it could be advantageous to use assembly so you can take advantage of the hardware directly. It is unlikely that a whole operating system would be done in assembly, but certain parts might be.

6 RISC architectures use a small number of simple instructions that can be completed in a single clock cycle.

7 CISC architectures use a large number of complex instructions that usually take multiple clock cycles to complete.

A co-processor is a separate processor used for a specific task, for example, graphics processing (e.g. GPU).

8 In parallel processing two or more processors work together to complete a single task.

9 Financial modelling.

Chapter 2: Input, output and storage

Learning objectives

- Understand the main components of a computing system.
- Understand the key features of systems software and how they differ from application software.
- Have an overview of the key internal hardware that is essential in any computing system.
- Understand key input devices and how they could be used in a computing system.

What your students need to know

- All computer systems have components responsible for input, processing, storage and output.
- All computing systems are made up of a combination of **hardware** and **software**, where hardware is physical and could be made up of separate parts (e.g. a monitor and mouse), or be combined in a single unit (e.g. a mobile telephone). Software will make use of the hardware and perform useful tasks through a series of instructions that are executed by the microprocessor in the hardware.
- No system functions without systems software that brings hardware and software together. The two most common types of systems software are the **operating system** and system **start-up software**.
- Programs exist as zeros and ones in random-access memory (RAM), which is structured in such a way that various tasks could each inhabit a defined memory area, like rooms in a hotel can host very different people.
- **Read-only memory (ROM)** is used by all devices for configuration. ROM is usually loaded before RAM in a boot sequence.
- System start-up software is the other main type of systems software. This is normally found in the ROM, whereas the operating system is normally found on secondary storage devices, such as a hard disk. Start-up software has the job of getting the system ready to load up the main operating system.
- Start-up (or booting) follows a certain procedure for every computer system. Boot loaders are found on storage devices that have been set up as bootable, so a hard disk might contain a file with all the settings and preferences saved by a user.
- Before the data is read, computers perform **power-on self-test (POST)**. If a hardware part is missing, the computer will usually beep or display an error message and refuse to boot.
- All hardware connects directly to the motherboard which might have slots for the hardware to fit on or metal etchings on which to solder the parts. A common name for a **central processing unit (CPU)** slot in the motherboard is 'socket'. Sockets differ between brands and models of CPUs.
- Communication between the CPU and devices occurs over buses, which are split into three main parts, running at different speeds:
  1. address bus to specify where the data to be saved/loaded can be found
  2. data bus to send the data to be saved or transfer the data to be loaded into the CPU
  3. control bus to specify the operation to be performed and for other control signals.
- The CPU is the 'heart' of the computer.
- CPUs are measured by clock speed and architecture; for example, a 64-bit processor is more capable than a 32-bit processor.
- A fast CPU connected to a slow device will be idle most of the time. The slow device, for example an old hard drive, is known as a 'bottleneck', as the data slows down while passing through it.

- The number of transistors in a CPU has doubled every two years since the 1960s. This is known as 'Moore's law' (1965).
- All devices that contain a microprocessor have memory to draw data from.
- Input and output devices vary by function and computer type. A smartphone tends to have smaller devices for faster interaction with a user, while a company payroll computer will have a large cheque-reading device that can process thousands of documents with little user intervention. The most common industrial input devices are optical and magnetic readers that simplify batch input. In factories, sensors and actuators make robots functional by giving them the ability to 'see' and 'hear', as well as move themselves about.
- Sensors are becoming more popular as input devices as our computers become more personal – we carry them in our hands, we wear them on our wrists, we meet other people with similar devices – there is so much more to sense than in the old days of office computer boxes.
- Errors are common with input, so a check digit validation is used to increase reliability of data.

Common misconceptions

- Students can often confuse the types of bus. It might be beneficial to focus on the notion of the different bus speeds. Slower buses will obviously carry very different types (and volumes) of data than the faster ones. You might want to explain the buses like this: The address bus is like an address book in a phone – it might contain a maximum of a couple of hundred entries. The data bus is like messages – a typical user will probably have tens of thousands of text messages. The control bus is like a log of times of day (e.g. times of missed calls) or pressing 'Like' on a social network – a very small amount of data but crucial in triggering other events.
- Students can confuse RAM with ROM because of similar spelling. Take care when explaining that ROM is like a paper textbook – it's written on paper and can't be changed. It also contains data and instructions for the hardware. RAM, on the other hand, is like an interactive whiteboard – it can be written on and overwritten but must be cleared for the next day.

Lesson activity suggestions

Systems software terminology lends itself to a bingo-style game – students are shown a word and must match it with a description on their bingo board. The same applies to hardware like a motherboard, etc. This can be either combined into the same bingo game, or run as a separate one.

For CPU and buses, it makes sense to show a video similar to this one.

This video concentrates on 32-bit versus 64-bit architecture.

Try explaining different buses using an analogy with motorways. The address bus is like the hard shoulder where high priority vehicles can overtake traffic. Not many vehicles are authorised or in fact will use this lane. The data bus is a multilane motorway that needs to carry the traffic. Each 8 bits are a lane. Doubling the number of lanes is better than doubling the speed limit (clock speed), which is why most modern processors are getting more and more cores to process their lanes of traffic. The reason having more lanes is better than increasing the speed is heat. Higher clock speeds pump more energy through circuits and some of that energy turns to heat which will melt the processors if not cooled with battery-draining fans. In the case of a motorway, a similar idea applies – driving at faster speeds decreases cars' efficiency – air becomes denser, so the engine needs to work harder, tyres heat up (that is why all tyres are speed rated), the number of accidents might increase, etc.

The control bus is like the little service entrances off the motorway where construction and maintenance crews come out to put up construction signs, etc.

For different types of CPU, it might be a good idea to obtain the logos of the companies that make them (e.g. ARM® or Intel), as well as the logos of companies/organisations that use their devices and match them. Various types of mobile devices can be assigned to different card suites and students can make a card deck where the device (e.g. Blackberry®) will be like a card rank, while the processor inside will be its suit.

Try explaining cache with this analogy: why do we carry money in our pockets/purses if we can keep it in a bank (assuming you're at a place where they don't take cards)? It is because it is faster to take the money out of the pocket than run and withdraw money every time you want to make a transaction. Cache is like this pocket for change that is used often – a very fast but low-capacity memory that speeds up common tasks by keeping them handy, without the CPU needing to go to the 'warehouse' of RAM to retrieve the data it needs for the next instruction. For devices that are attached to the motherboard, there could be a research activity where students have to rank all the devices by their speed (with hard disk/flash memory being the slowest; alternatively, DVD drive, if installed, is even slower).

For Moore's law, students could research the history of the microprocessor and should understand that this law mostly relates to the density of the transistors placed on a CPU. Smaller parts lead to a higher density. The limitations for Moore's law are heat, which increases as the CPU becomes denser, and the unpredictable behaviour of the materials used at the microscopic level.

For input/output devices, the key is to remember which devices work with which type of input data, and in what environments.

Students are unlikely to have access to industrial input and output devices, so use pictures to help. A starting activity would be to create a catalogue of input and output devices for sale. Students should make up a name and logo for a fictional company and create either a hard-copy board (through printing the images and gluing them on) or an electronic one, such as in HTML in the form of a website. They will be given a list of devices that customers have bought in the past. They will need to find the pictures and match them to the item description. They could also research the price from a supplier (optional).

Alternatively, you could use crossword puzzles or cue cards and animate this relatively dry activity of remembering which input devices work with what type of data.

You could also get students to act out, perform a pantomime or play charades with 'a worker' operating such a device. For example, a student could use Monopoly money or cut up pieces of paper to feed through a pretend scanner and could then be asked questions about what device it is.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Systems software	**Starter:** Play a bingo-style matching game. Students are given print-outs of names of various utility programs. The categories to be called up could be: anti-virus, register cleaner, download manager, etc. **Main activity:** Ideally, students will have access to a virtual machine player such as QEMU or VMware or an actual Raspberry Pi® on which they can run utilities without security restrictions (virtual machine players allow images of operating systems to be downloaded and run in a secure window). Identify and run at least one of each system utility described in a chapter (e.g. use the terminal to check disk space, run sudo to install software, etc.). **Plenary:** Discuss what can happen if systems software is not used properly or not used at all.	Research a project on the advantages and disadvantages of using different operating systems (some students can choose desktop systems: Windows versus Mac versus Linux, others might want to compare Android versus IOS). You could also have a mini debate on this topic. Another idea would be to let students create an easy manual (or a video that can be posted on YouTube and Vimeo) on usage of selected utilities.
CPU and buses	**Starter:** Students are given a mathematical activity to work out the number of combinations for 32-bit and 64-bit binary architectures (a formula to use is '2 to the power of number of bits'). Students are then asked to guess how this number would be relevant (it is the largest number of memory addresses that can be used by that CPU). **Main activity:** Use the Little Man Computer emulator (known to some through a GCSE task to create a program that asks users to input two numbers and returns the smaller of the two. Students are to comment how the buses are used for each instruction in the implemented program. **Plenary:** Discuss which of the buses should have the priority or wider bandwidth to speed the processor the most.	Practise past examination questions.

Topic	Lesson suggestions	Follow-up ideas/Homework
Types of CPUs Hardware components Moore's law	**Starter:** Present students with the a chart of increasing complexity of CPUs similar to this one here. Another useful graphic can be found at a blog here. Another interesting contrast of microchip density and the size of an atom can be found here. **Main activity:** Create a deck of cards with logos of CPU designers as suites, logos of devices (e.g. Nexus 7) as card ranks. **Plenary:** Discuss if the clock speed is the most important factor for a CPU's ability to process.	Research some of the famous CPUs from the past. Plot their speeds on the same axis. Research will include the concept of 'benchmarking' and the names of popular benchmarking programs. Here is a useful site for this. Create a price list for an online retailer of computer parts, from the cheapest/ weakest models to the best/ professional grade.
Input/output devices	**Starter:** Read out a list of devices and randomly point at a student who has to name the device as input, output or both. **Main activity:** Talk about case studies involving more interesting input/output devices such as seismic sensors or radio telescopes. An interesting case study could involve SETI and CETI projects that aim to find and communicate with aliens, respectively. What kind of input and output devices would they use? **Plenary:** Discuss what is more important: input or output devices? Can an input device be an output device at the same time? (Yes, it's a touchscreen.)	Create a crossword puzzle of devices and their definitions. Create a catalogue of input and output devices for sale in a small computer shop.

End-of-chapter answers

1 a. A data bus is used to transmit the data either from save or load operations.
b. The Address bus passes the location of the data to be loaded or saved.
c. The control bus states what operation should take place and sends a control signal to the recipient device.

2 RAM is volatile, which means that when the computer is switched off the data is lost. ROM is non-volatile so no data is lost when the power is switched off. RAM is used to store the currently running OS, programs and data. It is used temporarily and any data, which is changed must be saved to a storage device to prevent loss. ROM stores system start-up software and is key in setting up the initial BIOS and triggering the OS. If ROM was deleted then the computer would not be able to boot the OS.

3 Input devices:
Touch screen – To allow the user to use a menu based system to pick their beverage.
Chip and pin – To allow the user to pay by credit card.
Output devices:
Actuator – Used to dispense the coffee.
Printer – Used to print a receipt.

Chapter 3: Systems software

Learning objectives

- Understand the main functionality of operating systems (OS).
- Learn how memory is managed by the OS.
- Learn how scheduling is managed by the OS.
- Learn the main types of OS, including distributed, embedded, multitasking, multi-user and real-time OS.
- Understand the purpose of the Basic Input/Output System (BIOS).
- Understand how devices can be controlled through the use of device drivers.
- Learn how virtual machines work and how different operating systems can be hosted on the same machine.

What your students need to know

- Software is the name given to programs running on a computer.
- Software can be divided into three categories: systems software (the operating system and start-up programs), application software (programs that allow the user to perform tasks, e.g. use a word processor, create a database, browse webpages, etc.), and utility software (allows the user to maintain and manage their system, e.g. security software, disc defragmenter, backup utility and configuration programs).
- The operating system allows the computer system's hardware and software to run programs.
- The core programs form the **kernel** that runs all essential system processes. It allows the user to interact with the computer via a **user interface (UI)**, which may be **graphical (GUI)** or a **terminal** and **shell commands**.
- The kernel contains a memory manager, process scheduler and file manager.
- RAM holds currently running programs.
- Each memory location has a unique address.
- System management utilities allow the user to configure and maintain the system. An operating system will also protect the system from serious errors and inform the user when this has occurred.
- **Drivers** are programs that allow hardware resources to function with the operating system.
- Computer memory consists of primary memory (random-access memory [RAM]) for running programs and secondary memory (Hard Disk Drive [HDD] and other storage media).
- The memory manager allocates primary memory to running processes. Segments are blocks of memory that can vary in size. A segmentation table keeps track of the addresses of segments used by each process.
- **Pages** are fixed-size blocks of memory that are smaller in size than segments. Details of the mapping of logical addresses to physical addresses are stored in the page table.
- Pages can be swapped out to a virtual memory page file on the HDD. A flag in the page file indicates that this has happened.
- If too many pages are being swapped in and out of virtual memory, the system can slow down and become unresponsive, which is known as disc thrashing.
- The operating system also creates the file system on storage media allowing files and programs to be stored permanently.
- Scheduling of processes determines how programs access central processing unit (CPU) time. Running more than one process at a time is called multitasking.

A **scheduling algorithm** should:

  - minimise starvation
  - eliminate deadlock
  - ensure efficiency of process execution
  - ensure that all processes have an opportunity to execute.

- The simplest algorithm is **round robin** where each process gets a time slice.
- Other algorithms are:
  - **Shortest Job First (SJF)**
  - **First Come, First Served (FCFS)**
  - **Shortest Remaining Time (SRT).**
- Other algorithms base access on the required resources of the process and priority.
- Processes can be in one of three states: running, ready or blocked. Waiting processes are held in queues.
- **Multi-level Feedback Queues (MFQ)**, allow for three levels of priority to be managed in queues.
- **Interrupts** are used to halt the current process. These are signals to the CPU to request processor time. Interrupts can be generated by hardware, software or the processor.
- When an interrupt is received, a program called an **Interrupt Service Routine (ISR**) will run.
- The ISR is held in stack memory that uses a pointer to indicate the item in the stack to be executed next.
- Some processes involve a delay, for example printing a document: the computer can send the data faster than the printer can process it. A small area of memory called a buffer allows printing to appear continuous. As the buffer empties, an interrupt is sent for more data to be sent.
- **Flags** are used to indicate when an interrupt has occurred.
- Interrupts use priorities to indicate that a process is more critical so should be handled first.
- A multitasking operating system can run more than one process.
- Running processes can complete, become blocked or surrender resources.
- The contents of registers of a currently running process are stored in a **Process Control Block (PCB)**.
- Deadlock occurs when a process is waiting for a resource that never becomes available.
- Starvation occurs when a process cannot run because higher priority processes are being executed.
- Operating systems can be categorised as:
  - Single User
  - Multi-User or
  - Network.

- Embedded operating systems will have a smaller installation and simplified operation.
- Real-time operating systems respond instantaneously when an input is received. They are highly specialised and used on safety-critical systems.
- Distributed operating systems allow multiple computers to work together to use their combined processing power.
- The **BIOS** includes:
    - the POST program
    - a configuration utility including pointing to the location of boot devices in order of preference.
- Boot devices have a **boot sector** that points to the **boot loader** which loads the operating system.
- Virtualisation of operating systems allows hardware and software to be separated. Multiple operating systems can be installed to run within their own virtual machine.

Common misconceptions

- It is important that students build upon their knowledge of the CPU to consider why it is important to manage running processes. There are a number of key terms and concepts that need to be reinforced. For example, the terms 'multitasking' and 'multithreading' are similar. Other potential points of confusion are the roles of buffers and interrupts. Students may be familiar with the term 'buffering' from attempting to view videos online and have an awareness that a buffer provides a temporary store of data due to a mismatch between how fast the video can be processed by the computer (fast) and how fast it can be transmitted across the internet (slower). Students should strive to connect concepts to gain a holistic understanding. For example, if they understand the concept of multitasking and that input and output devices need access to the CPU, they will appreciate the need for and purpose of interrupts. They should also consider that some interrupts will have a higher priority than others. Interrupt handling can be drawn out as a flowchart to show the whole procedure. This will provide a helpful revision aid.
- Students can also become confused between segmentation and paging. A table will provide a useful comparison tool. The role of pages in allocation of virtual memory will also assist in separating the two methods.
- Queuing of processes and scheduling also need to be understood holistically. A larger group of students could role play the process (see lesson activity suggestions below).

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Systems software	**Starter:** Conduct a 'logo quiz' presentation of well-known and lesser-known operating systems. **Main activity:** Review the main functions of an operating system. Circus activity of different computer platforms (e.g. desktop, games console, Raspberry Pi®, mobile device [students can investigate their own], an embedded device such as a satellite navigation device). Factsheets with images could be used instead of devices (e.g. a real-time OS, distributed OS, server OS, etc.). **Plenary:** Complete a table showing type of OS versus typical use.	Students should investigate a single operating system of their choosing and present a 5–10 min seminar about it at the following lesson. It is helpful to provide suggestions (e.g. Android, MacOS®, MS Windows® and Raspberry Pi Raspbian®). Resources produced can be shared to develop a comprehensive overview of a range of operating systems and to keep the task current.
Memory management	**Starter:** Play a memory 'mix and match' using three sets of cards covering names of types of memory, approximate quantities and function. Registers, cache, RAM, secondary storage and ROM should be included. **Main activity:** Take an in-depth look at how operating systems manage memory. Students can produce a poster illustrating how different types of memory are utilised within a computer and the process of swapping blocks between RAM and virtual memory. **Plenary:** Discuss the limitations of memory management. Why is a reliance on virtual memory a performance disadvantage?	Practise past examination questions.

Topic	Lesson suggestions	Follow-up ideas/Homework
Process scheduling	**Starter:** Organise students into small teams. Give them a series of tasks to complete (e.g. sharpening a pencil, building a model using bricks, making a paper model, drawing a house, etc.). Initially, only one person (processor) should be used. The aim is to complete as many tasks as possible in two minutes. Then allow students to use multiple processors, or to reorder the sequence of tasks. This will naturally lead to 'multitasking' and possibly even 'multithreading' solutions as well as an appreciation of task queuing and resources required. **Main activity:** Discuss key scheduling concepts. Students to work collaboratively to produce a wiki. **Plenary:** Discuss how real-time processing and batch processing would differ from algorithms based on conventional scheduling.	Produce a Prezi explaining the basics of scheduling.
Buffers and interrupts	**Starter:** Discuss how the problem can be solved of sending data to a slower device from a faster one. Allow students to suggest possible solutions illustrated by a large leaky bucket being filled by a smaller tea cup. **Main activity:** The concepts of interrupts and interrupt handling build upon topics of previous lessons. Students can work in small teams to produce a set of quiz questions on the topic of buffers and interrupts. The quiz can then take place with groups asking their own questions to the rest of the group. **Plenary:** Hand out some examination questions. Allow students to talk through what they consider to be the key points to include in the answer. This could become a longer activity if small groups are given different questions.	Investigate the history of the development of interrupts. There are many excellent sources of information available on the internet. This activity will allow students to practise their research and writing skills.

Topic	Lesson suggestions	Follow-up ideas/Homework
How operating systems function	**Starter:** Provide the stages of booting a computer as separate cards. Students must arrange these in the correct order. **Main activity:** This lesson will focus on the relationship between the operating system, hardware and application software. Virtualisation is common practice on network servers. Students should consider the advantages of virtualisation in modern systems. If possible, a practical activity could be undertaken to install Oracle® VirtualBox and a virtualised OS (e.g. Ubuntu Linux). **Plenary:** Discuss the possible future direction of operating systems.	Investigate the differences and similarities between virtualisation and emulation. Students may have some experience of the latter and they could also prepare discussion points on the licensing implications of three technologies.

End-of-chapter answers

1 a. Allows memory to be partitioned, through either segmentation or paging, and allocated to processes. It ensures that processes cannot access each other's segments/pages for security and stability purposes. Memory management also allows for the use of virtual memory when RAM is almost full.

b. Scheduling will ensure that all processes get processed, taking into consideration any priorities that may exist. It will process the maximum number of processes in the least amount of time ensuring high throughput of processes. It ensures that all users of a system will get fast response times by making the most efficient use of the processor.

2 Memory is allocated to a process and recorded using a page table. Each process will have its own page table which will say where in RAM the page resides. A process will not be aware of what pages it uses, but rather will use a virtual address space. On memory access, a virtual address is converted into a real address using the page table. A process sees its memory pages as a continuous block, but each page may reside in different parts of memory.

3 When a process is suspended it will have its process control block saved into virtual memory. This will include the stack and the contents of registers (PC, ACC etc.). A process may need to be suspended if there is a limited amount of RAM remaining and a new process has been started. The memory manager will try and choose a process to suspend that is not currently being used. Once a process has been suspended it must be loaded back into RAM in order to continue processing.

4 Data will be sent to the hard drive and stored into the hard drive's buffer. As the buffer works much faster than the hard drive itself, the CPU will be free to run other processes while the data is saved and the process which is saving the file becomes blocked. Once the buffer is full, the hard drive will slowly save that data. When the buffer is empty and the data saved, an interrupt is sent back to the CPU. On the next FDE cycle, the interrupt register will be checked and depending on priorities, the blocked process will be brought back into the ready to run queue and more data sent to the buffer. This algorithm repeats until the file is fully saved.

5 A distributed OS will have a number of computers, known as nodes, connected over a network. Each node will have a micro kernel, which has a small set of key features such as device management, memory management and network communication. Scheduling is done on top of the micro kernel as a process could be run on any node. Scheduling itself would be done by a single node, while other distributed functions like file management may be done by another node.

Chapter 4: Applications generation

Learning objectives

- Understand the nature and variety of application software.
- Describe and understand the purpose of utility software.
- Know the difference between open and closed source software.
- Understand the differences between the three main types of translator: compiler, interpreter and assembler.
- Understand the different stages of compilation, including lexical analysis, syntax analysis, code generation and optimisation.
- Describe the roles that linkers and loaders play with regard to translated software.

What your students need to know

- Application software allows the user to perform a specific task.
- Examples of application software are:
  - desktop publishing
  - word processing
  - e-mail
  - spreadsheets
  - databases.
- **Knowledge-based systems** are sophisticated programs used to solve problems (e.g. medical diagnosis). They consist of four parts:
  - user interface
  - knowledge base
  - rule base (facts)
  - inference engine.
- **Computer-aided design (CAD) / Computer-aided manufacture (CAM)** is software that is used in the design and manufacture of products.
- Utility software performs system tasks.
- Examples include:
  - file management
  - compression
  - anti-virus and security
  - backup
  - task management.
- Application software and utilities for mobile devices are often termed as an 'app'.
- **Open-source software** is distributed with the **source code**. It can be shared and modified.
- **Closed-source software** cannot be decompiled but may be free.
- **Proprietary software** is protected by copyright and is licensed. There is usually a cost associated with acquiring and using the software and it cannot be decompiled.
- **Translators** change high-level code into machine code that the central processing unit (CPU) can execute.

- **Compilers** check the entire program, translate it and produce an executable which the CPU can run.
- **Interpreters** convert high-level code into machine code line by line. If an error is encountered, the program will halt at the line before the error.
- Some translators produce intermediate code (between high-level and machine code). Java is an example of a program that produces code within a virtual machine that allows it to achieve platform independence.
- **Assemblers** translate Assembly language code into machine code.
- **Opcodes** represent the operation part of an instruction. Instructions usually have a memory address too.
- **Mnemonics** are simple strings that represent an operation, for example ADD (addition), SUB (subtraction), MUL (multiply), DIV (divide), etc.
- Variables use labels that are stored in a symbol table. Each label will be associated with a memory address.
- The **compilation process** occurs in the following stages:
    - lexical analysis (parsing) – identifies keywords and generates tokens
    - syntax analysis
    - checking types
    - machine code generation
    - sequencing code blocks
    - register allocation
    - code optimisation
    - linking libraries.
- How **Backus-Naur Form (BNF)** is used to define the syntax of a language.

Common misconceptions

- The range of application and utility software is generally well understood although some students may be less familiar with some (e.g. knowledge-based systems). It is worth spending time looking at different uses. Ensure that keywords are frequently reinforced.
- Students need to be clear on the differences between compilers, interpreters and assemblers. They also need to understand that some programming languages compile the program to an intermediate stage and produce code that is then further translated into machine code. Java is a good example of such a language and students should appreciate the concept of a virtual machine. Students should have some experience of using the Little Man Computer simulator. This will help to reinforce understanding of the Fetch-Decode-Execute cycle but also, in the context of this chapter, to understand assemblers.
- A detailed understanding of the stages of compilation is expected. Students need to record the steps carefully and ensure that they understand each one and key terms. Much of this section will be unfamiliar and students sometimes confuse stages and terms.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Nature and variety of application software	**Starter:** Conduct a ‘mix and match’ activity of common applications and their uses. For the starter, stick to software that students will have had experience of using (i.e. word processor, desktop publishing, graphics package, e-mail client, etc.). **Main activity:** Ask students to consider the facts and rules about a scenario (e.g. diagnosing computer faults or identifying animals). They can design a user interface and consider how the system will function (i.e. what queries [goals] the user may wish to solve). Provide examples of different sectors; students need to consider what software applications would be the most appropriate software for these. There are a number of online videos showing how large organisations use software applications. Investigate current most popular mobile apps and describe priorities when developing software for mobile devices. **Raspberry Pi® activity**: Install the LibreOffice software package and evaluate its features. **Plenary:** Use sticky notes to record new facts that students have learned in the lesson and one question that they wish to have answered.	Practise examination questions.
Utility software and software licensing	**Starter:** Define utility software. **Main activity:** Investigate different examples of utility software. Ideally, this would be carried out as a practical activity. For example, students could perform a backup, run an anti-virus scan/ spyware scan, manage directory structures and file attributes. Produce a summary table of software licensing options. **Plenary:** Evaluate the open-source software model as a verbal discussion.	Write a fictional magazine article about three useful open-source utilities.

Topic	Lesson suggestions	Follow-up ideas/Homework
Translators – compilers and interpreters	**Starter:** Provide students with short passages written in different foreign languages. Without using the internet, they should try to translate the text into English. As a discussion point, they should consider what they looked for and how they attempted to make meaning of the words. **Main activity:** Ask students to consider the advantages and disadvantages of compiling versus interpreting. They should consider this from the developer's perspective and the program user's perspective. It would be helpful if students have some software development experience and they have a chance to witness the operation of a compiled language (e.g. C++ or C, compiled and interpreted software e.g. Java [compiled from high-level code to intermediate code] or Python). **Plenary:** Discuss the translation process. What are the advantages of producing an executable by compilation? Compiled versus interpreted? Provide students with a list of programming languages.	They are required to research which are compiled and which are interpreted. They will complete a table with two columns; one for compiled languages and one for interpreted.
Translators – assemblers	**Starter:** Review students' understanding of the FDE cycle and the difference between high-level and low-level code. **Main activity:** Investigate the LMC simulator, produce programs and interpret evidence of assembler output. The LMC uses a numeric representation of the opcode. Discuss with the students how machine code would appear. **Plenary:** Review the syntax of some simple programming constructs using the LMC and how instructions are handled.	Investigate Assembly language for the ARM processor as used in the Raspberry Pi®. What differences would you see between a RISC instruction set and a CISC one?

Topic	Lesson suggestions	Follow-up ideas/Homework
The compilation process	**Starter:** Return to the activity in the first lesson on translators. Compare the translation of written language to a computer program. Introduce the term ‘syntax’. **Main activity:** Provide students with sets of cards each representing a single step of the compilation process. These should be placed in order and a flowchart produced of them. The flowchart should show what happens if a step fails due to an error. Students will also produce a glossary of all key terms. The role of libraries and how these are used by programs should be included. **Plenary:** Discuss why old computer programs won’t run on a modern computer system.	The role of libraries and how these are used by programs should be included.

These answers relate to Questions asked in OCR Component 1. Further information on this element of the material is available at www.cambridge.org/ukschools/OCRcomp1

End-of-chapter answers

1 a. A collection of facts, which are implicitly true based on expert knowledge.
b. Using the knowledge base and rule base it will infer answers. If no one answer could be chosen yet, the inference engine will pick a new question to ask the user.
c. Rule base connects facts together to create links, which are useable by the inference engine.

2 a. Compression software will reduce the size of a file and may also archive a group of files into a single file. When a file is to be accessed it must be decompressed first.
b. Anti-virus software will use heuristic methods to match known virus signatures or behaviours to what is happening on the computer. If a virus is found, it will be placed into quarantine. Each file on the system will have to be scanned.

3 Open source and free software are related but are not the same. Free software, where the user does not have to pay, can be downloaded as an executable file. It may be the file is financially supported through advertisements or micro transactions, which is very common for mobile apps. There is no guarantee that the source code will be included in free software. Conversely, open source software does include the source code meaning you can compile, tweak and improve the code. Depending on the licence, those updates can be shared with the wider public. Some software, such as MySQL, has a free open source version and a paid closed source. The open source version gets some of the improvements from the paid version and is a way to support the growth of the software.

4 Languages that are compiled and interpreted will make use of intermediate code. The source code will be compiled into intermediate code, which uses a generic instruction set, which no real computer can run. In order to run the program a virtual machine (VM) must be used. The VM will understand the generic instruction set and will provide a sandboxed environment for the program to run. The VM will interpret and translate the generic instructions into real machine code. This has the advantage that programs can be compiled into intermediate code and then run on any architecture, which has a virtual machine.

5 a. Lexical analysis will tokenise the source code, removing white space and grouping characters into more meaningful tokens. For example “myVariable” would be turned into a single variable token. This makes syntax analysis simpler. It will produce a token stream and a symbol table.
b. Syntax analysis will check the order of tokens against the rules of the language’s grammar. If a series of tokens fails to match a rule then a syntax error is produced. An abstract syntax tree is produced at the end or translator diagnostics are reported back.
c. Common patterns of tokens are turned into sections of machine code. Memory addresses are allocated as well as registers. Optimisation will also occur and will either be optimising for running speed or final size. This will produce object code.
d. Any libraries used will need to be connected to the final executable. A loader is used to allow a library to be linked and the memory addresses translated

Chapter 5: Software development

Learning objectives

- Understand the systems lifecycle and what the main phases are.
- Understand different methodologies that can apply to the systems lifecycle, including the waterfall cycle, Extreme Programming, the spiral model and Rapid Application Development.
- Learn about the different merits and drawbacks of different methodologies and when they might be used.

What your students need to know

- Reasons why new software may be needed.
- Who the stakeholders of the system are.
- The role of the **systems analyst**.
- The stages of the **software development lifecycle**:
  - Problem definition – The formal description of the problem to be investigated.
  - Feasibility study – An investigation into the viability of the project. This part of the process aims to find out whether a solution is technically and financially possible.
  - System investigation – Assuming that the project is viable, existing solutions are thoroughly investigated to discover limitations of the current system. The current inputs, processes and outputs are documented.
  - Requirements analysis – The client will provide details of the functional and non-functional requirements of the proposed new system. The analyst must interpret exactly what it is the client needs.
  - Design – The developer must interpret the client's requirements and design the system input, processes and output.
  - Development – In this phase, the solution is developed. Agile Development hinges on developing working software in close collaboration with the client.
- Testing:
  - White box – This is the systematic testing of each software component to check that it functions correctly. It is also known as alpha testing.
  - Black box – Once the individual components have been tested, the whole system can be tested to ensure that the inputs produce the expected outputs. The intended end users will be involved in black box testing.
  - Acceptance – The client needs to ensure that the software meets their requirements. This is usually the final phase of testing and will result in the client 'signing off' the software.
- Documentation:
  - User – This will take the form of a manual and will describe normal use, advanced options and troubleshooting.
  - Technical – Documentation is also required for technicians who will need to install, configure and maintain the software. A technical manual includes detailed information about the software product.

- Implementation:
  - Direct – The existing system is decommissioned and the new one is put into service. This may represent a drastic change for users and there is always a risk that the new solution may not work and then it will be very difficult to roll back to the old system.
  - Phased – The software is rolled out across the organisation more gradually, by department, floor or building, until the whole system has been migrated. A key advantage with this method is that problems can be spotted early and the roll-out halted until these have been rectified.
  - Pilot – A select group or department will trial the software in a pilot before it is fully rolled out across the whole organisation.
  - Parallel – The existing system is run alongside the new. This is a failsafe method as 'teething problems' with the new system can be dealt with without downtime while the old system is still functioning. After successful implementation, the old system is switched off. A disadvantage is that two sets of hardware will be required to run the old and the new and users might get confused between the two.
- Maintenance:
  - Adaptive – Adaptive maintenance is that which keeps a system up to date.
  - Corrective – Corrective maintenance resolves errors in the software.
  - Perfective – Perfective maintenance improves or 'tweaks' the system.
- Evaluation – A new piece of software represents a massive investment for an organisation. It is very important that a new system is evaluated fully so that further improvements can be made.
- **Development methodologies:**
  - Incremental – This is stepwise development of a system.
  - Iterative – Iterative development repeats sections of the system's development lifecycle. For example, a software product may be refined through consultation with a client.
  - Prototyping – This follows on from the iterative model and results in the production of prototypes before the final system design is developed. Each prototype will be discussed with the client.
- **Development models:**
  - Linear – This is a very traditional methodology for development. Initial meetings with an analyst lead to the design and development of a product that is worked upon from beginning to end.
  - Waterfall – The waterfall model allows minor revisions to be made to each stage prior to testing and evaluation.
  - Spiral – Spiral development allows the entire system to be redesigned and refined with each revolution of the lifecycle.
- **Modern methodologies:**
  - Rapid Application Development – Rapid Application Development (RAD) results in several prototypes before the final product is produced. The client is heavily involved in development. This methodology was popular during the 1990s at the height of the popularity of visual, event-driven software development. RAD is also known as prototyping.

  - Agile development – The standard for Agile Development was developed in 2001. It has working software at its heart. The client is also consulted over developments as with RAD.
- **Common misconceptions**
  - The concepts associated with systems analysis and development are usually well assimilated; however, students should be careful with the accuracy of their answers. They should focus on learning the purpose of each stage of the systems development lifecycle, the models and methodologies. Accurate use of key terms is important. Students also need to be able to understand how diagrams (e.g. Data Flow Diagrams) are used to analyse an existing system and to design the new version.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Phases of the systems lifecycle	**Starter:** Play 'What's my line?' One student is given a description of a business. Other students ask questions about the work to find out what the first student actually does. This is a fact-finding warm-up activity. **Main activity:** Discuss the phases of the systems lifecycle. Students produce notes on each stage that will be assembled into a poster in the next lesson. **Plenary:** Discuss the roles of the analyst, programmer, manager and end user. What are their priorities? Where are the potential constraints and areas for discussion?	Conduct a case study investigation of a fictional business that wants to improve its ordering system. Students assume the role of the analyst. They should document the existing system and suggest an improved design.
Development methodologies	**Starter:** Discuss different ways of working with clients to develop a new system and the advantages of each. **Main activity:** Ask students to use the stage descriptors produced in the last lesson to create posters illustrating the different lifecycle models. **Plenary:** In small groups, evaluate different methodologies of system development.	Produce an investigative report into the Agile Methodology and Extreme Programming for a fictional news report.

Topic	Lesson suggestions	Follow-up ideas/Homework
Developing algorithms	**Starter:** Discuss what an algorithm is. **Main activity:** Provide students with examples of completed, but faulty, system design documentation and Requirements Specification. Students will identify the errors in the design and identify where it does not meet the user requirements. Through this activity, students will appreciate the link between designing a solution and the requirements that it must meet. Secondly, a test log can be provided along with the Requirements Specification to evaluate the system against what was required. Students will be able to appreciate the links between the stages of analysis, design and development and the importance of ensuring accuracy early on. **Plenary:** Conduct an 'exit pass' activity. Write down one thing that you have learned in the lesson that you didn't know before; and then something that you are not quite sure about and have one question ready to ask.	Play a web-based quiz on terminology associated with systems analysis and design.

These answers relate to Questions asked in OCR Component 1. Further information on this element of the material is available at www.cambridge.org/ukschools/OCRcomp1

End-of-chapter answers

1 During analysis, information about the current system and what problems are to be solved must be gathered. Feasibility must also be done to ensure that the project is worthwhile. This will include checking to see if the project can be done on time and on budget. Also, is the project technically feasible and will it conform to current law? If the project is not feasible, then it will end here. Fact-finding is done through a combination of methods including interview, document collection, observation and questionnaires. Once all of the information is gathered it is then analysed by the analyst and a requirements specification produced. The requirements specification will include requirements related to the input and output, what processing is required and what hardware will be needed.

2 a. A prototype is a version of the full product but has a limited or restricted amount of features.
b. In a single iteration of the spiral model, a prototype will be produced.
c. This is then evaluated and tested by the customer and the feedback used to inform the next iteration of the prototype.

3 Both RAD and spiral models use prototypes. A prototype has a restricted or limited set of the final features. When using the spiral, developers work together on a single prototype, which is then tested by the end user to get feedback. RAD on the other hand will split the problem down into smaller modules and each module will be turned into a prototype. This allows for prototypes to be done in parallel allowing for faster development. Once a prototype is done and tested it can be linked into the final product.

4 In the agile methodology there is close contact with the client meaning that communication is much faster and therefore is valued more highly. This allows the developer to respond much faster to changing requirements as the requests would be picked up faster. There is regular feedback included in the methodology so if the client is not happy or notices any issues, agile programmers can respond quicker. Prototypes are produced, which have limited functionality, in small, set time frames. As these prototypes are small they can be shown to the client on a regular basis, again ensuring more opportunity for feedback and the ability to respond to change faster.

5 The application will be fairly small and unlikely to be built by a large team of developers. As such, using RAD or agile would not be suitable. The spiral model would be the best choice as the students could be shown different versions of the prototype and feedback gathered allowing for an app that is going to meet the needs of the users much more closely.

6 A deliverable (for example, the Requirements Analysis document at the end of the Analysis phase).

7 The waterfall model is not suitable for projects where the requirements may not be fully understood at the start.

8 An unfinished version of the product that can be refined or built on to create the final solution.

9 Collaboration.

10 Methodologies.

Chapter 6: Types of programming language

Learning objectives

- Understand what is meant by the term 'procedural languages'.
- Be able to read and understand object-oriented pseudocode, based on Java, including:
  - defining classes
  - instantiating objects
  - methods and attributes
  - inheritance
  - encapsulation
  - polymorphism.
- Learn and write basic Assembly code, using the Little Man Computer (LMC), as well as understanding the basics of how real Assembly code works.
- Learn the different memory addressing modes, including immediate, direct, indirect and indexed.

What your students need to know

- The characteristics of **procedural programming**. These include programs being built around modules called procedures. Procedures contain instructions in a sequence that define exactly how the program should run.

- The concept of **variable scope**. Global variables are declared at the start of the program and are accessible throughout the program including within subprocedures and functions. Local variables are defined inside procedures and functions and are only visible within them.

- The characteristics of **object-oriented (OO)** languages, including encapsulation, inheritance and polymorphism. Encapsulation is the principle of keeping data private within a class and only allowing it to be accessed by public methods. Inheritance is another core principle of OO programming which allows a subclass, or derived class, to inherit all of the attributes (properties) and operations (methods) of its parent or superclass. Polymorphism allows methods to be adapted for objects from derived classes but methods from the superclass will always give the same results.

- The advantages of defining classes and code reuse. Classes define all of the attributes and operations that objects of that class will support. Encapsulation allows classes to be portable between programs. Once a class has been imported, it means that all of its methods are available for use in the program. A key advantage of OO programming is that classes can be reused in many programs.

- What classes, objects, attributes and methods are.

- The advantages of encapsulation, inheritance and polymorphism.

- Stacks are **Last-In, First-Out (LIFO)** data structures. Stacks are used in function calls and parameter passing. The purpose of Assembly language as a form of low-level programming Assembly language is CPU specific and programs compile and run very quickly. However, even simple programs will require many lines of code as each instruction only performs a simple operation.

- Direct, immediate, indirect, relative and indexed modes of addressing. In Assembly language programming, different modes of addressing are supported. Reduced Instruction Set Computer (RISC) architectures tend to support fewer addressing modes and Complex Instruction Set Computers (CISC) more. Direct addressing uses the value at a given memory location as the operand. Immediate addressing uses a 'hard coded' operand. Therefore, no memory address is required. Indirect addressing uses a vector table to store memory address locations rather than data. It is these that give the location of operands. Relative addressing uses an offset that is used with the current memory location to locate the memory address location of the operand. Indexed addressing uses an array to store addresses. Each element is identified by an index which provides the offset to locate the required address.

- How programming constructs are implemented in Assembly language (e.g. iteration and selection). Branch instructions and labels allow complex Assembly language programs to be written that manage the flow of control.
- How the LMC simulator can be used to understand Assembly language programming. This is a useful activity to allow students to visualise the execution of low-level programs in terms of CPU components.

Common misconceptions

- Students can confuse key features of different programming paradigms. A glossary or wiki can be of benefit. Addressing modes can also be the cause of misunderstandings. Clear definitions and examples are helpful but students should be mindful of the similarities of terms (e.g. indexed and indirect).

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Procedural languages	**Starter:** Practise producing sequencing instructions to complete a simple task (e.g. drawing a square on a whiteboard with a marker pen or similar activity). Students plan the algorithm and then pass it to another to 'act out' their instructions. Explain that procedural languages execute instructions in order and that the program code describes how it will run. **Main activity:** Investigate a procedural language (e.g. Python). Students should write simple procedural code such as that required to calculate the number of rolls of wallpaper needed to paper a room or to calculate the area of a circle. **Plenary:** Produce a procedural programming factsheet	Perform simple programming exercises involving mathematical calculations (e.g. averaging numbers, calculating a discount, multiplication tables or finding areas or volumes).
The object-oriented paradigm	**Starter:** Begin the lesson with a brief introduction to the terms 'class', 'object', 'attributes' and 'methods'. **Main activity:** Define the key characteristics of OO programming. Students can investigate pseudocode or, preferably, an environment such as Eclipse Java. **Plenary:** Compare OO programming to procedural and discuss the strengths, limitations and typical uses of each.	Write a fictional magazine article about the history and uses of Java.

Topic	Lesson suggestions	Follow-up ideas/Homework
Assembly language programming	**Starter:** Discuss simple devices where computers might be installed (e.g. in a car). In each case, what is the computer's purpose? **Main activity:** Use a board with an 8 × 8 grid to represent RAM and individual cards containing opcodes, memory addresses and data. Put together a simple program that adds two numbers. Allocate memory to instructions by placing the card in a grid space. Take cards out in sequence to illustrate the instruction being executed. This 'board game' could be played as such and could be extended to include storage of instructions, data and memory addresses in registers on the central processing unit. Introduction to the LMC. **Plenary:** Discuss the uses, advantages and disadvantages of Assembly language.	Attempt some LMC challenges beginning with simple activities and progressing to harder ones.
Memory addressing modes	**Starter:** Provide students with a list of mnemonics. They can fill in the second column with what they believe the instruction to be. Students should also answer 'memory address' or 'no memory address' next to each to indicate which need to use a memory address (e.g. HLT [halt] should be 'no memory address' and STA [store] should be 'memory address'). **Main activity:** Investigate different addressing modes. Students to produce interactive posters or presentations to illustrate how they work. **Plenary:** Compare OO programming to procedural and discuss the strengths, limitations and typical uses of each.	Play a web-based quiz on the characteristics of different programming paradigms.

End-of-chapter answers

1
a. A set of related items in programming which share common attributes and methods.
b. A specific instance of a class representing a single element which has its attributes populated.
c. When a class inherits the actions and attributes of another class. This class can then specialise further by adding more methods or overwriting existing ones.
d. Where the details of how a class works are hidden from the developer and only the public attributes and methods are exposed.
e. When two classes which share common hierarchies from inheritance can be referred to as their superclass. A subclass can be referred to by its superclass, but not visa versa.

2
a. `Stack myStack = new Stack()`
b.
```
myStack.push("football")
myStack.push("Drawing")
```
c. `print myStack.pop() will output "Drawing"`

3
a. Sprites in a platform game to not just move up and down but will jump instead. This requires use to overwrite the functionality of the sprite class, but still retain the rest of the functionality.
b. If every sprite had a drawing method then this can be invoked on any class inheriting from Sprite. If you had subclasses of sprites, for example a jumping sprite, then you could have an array of all sprites which you could iterate over and invoke draw() on all of them. How they draw will be encapsulated by the class itself.

4
a. indexed addressing mode uses an index register and is used for arrays. To find the address you need to use the formula "index register * size of item + base address" where the base address is the start of the array.
b. Indirect addressing is where the data in the instruction points to a memory address which, in turn, points to the actual data item.
c. Relative addressing uses the current address and then adds or subtracts a number of addresses to find the final destination.

5 There are no IF statements in assembly, so you have to use jumps and labels. Values are loaded into the accumulator for comparison and then a compare command run. You can then use a conditional jump, like jump if not equal JNE, to jump based on the result of the comparison. If the comparison is true, the jump occurs. If not, the next line of code is run. This means that the else part of an if statement will be located immediately after the jump while the true part will be located using a label. At the end of the false section an unconditional jump will be needed to jump to the code after the IF statement.

Chapter 7: Compression, encryption and hashing

Learning objectives

- To understand lossy and lossless compression.
- To understand run-length encoding and dictionary coding for lossless compression.
- To understand symmetric and asymmetric encryption.
- To appreciate the different uses of hashing.

What your students need to know

- Computer storage and network bandwidth are both limited. If we can reduce the amount of file bytes we get more storage space and faster transmission speeds by removing unnecessary bits of data.
- The level of compression is measured by its **compression ratio**: size of compressed file/ size of the original, the smaller the better.
- A number of compression packages exist, such as: zip, rar, tgz and 7-zip, and their compression performance is slightly different.
- In particular, 7-zip is Open Source and uses multiple algorithms to achieve very high compression ratios, as well as 256-bit encryption.
- There are two ways to reduce file volume: **lossy** and **lossless**.
    - Lossy removes more data during compression but some of this data is lost and doesn't get restored.
    - Lossless creates a perfect copy of the original after decompression.
- The choice between them is not always clear, it depends on particular needs.
- Run-length encoding is popular in lossless compression. Any repeated characters in a file are replaced with a flag character followed by the character replaced and how many times it was repeated. It will not compress sequences of characters that are shorter than four, as that four would include itself, a flag, and a digit as additional data. Often used to compress media like graphics or music where a similar pattern can repeat every frame (portion of a second).
- Another lossless approach is dictionary encoding where commonly occurring sequences in a type of file are stored in a dictionary as the file is scanned to search for such sequences. All the multiple instances of the sequences are replaced with a link to the sequence in the dictionary – thus saving space.
- **Huffman encoding** is one of the easiest dictionary encoding methods, where most characters used in a file are placed in a dictionary and then their occurrences in a file are replaced with numerical codes that link to the dictionary. The more common the character, the shorter the code.
- Compression works better on longer files as more similarities and patterns exist in such files.
- **Encryption** scrambles plain text into cipher text which can't be understood. **Decryption** is the process of unscrambling the encrypted message to bring back the original. The piece of information needed to decrypt the cipher text (and kept secure) is called the key.
- Ciphers have been around for a long time. Caesar's cipher is simple and famous. Every letter in an original message is replaced with the one either a few positions ahead or behind it in an alphabet. This can easily be broken by modern computers by trying all possible combinations.
- Encryption is now used in almost all communication and the power of computers is making older encryption methods unsafe.

- Alan Turing was the first to use a computer to break enemy codes, which he did successfully during WWII.
- In symmetric encryption, decryption is simply the opposite of the encryption process and the same key is used for both. It is easy to do but is also easily cracked by the brute force of modern computers.
- In asymmetric encryption, otherwise known as 'public key encryption', the encryption and decryption keys are different, so the party that encrypted the message can't decrypt it and the party that can decrypt the message can't encrypt it. It is much more secure but was unknown until recently. In this method, the encryption key is public and can be freely shared by the party that needs to receive an encrypted communication. The decryption key is private and secret, though. Anybody can find a public key of another person and use it to send them an encrypted message which only the recipient can decrypt.
- This is achieved via a mathematical one-way function that can only be reversed under a single circumstance. Such functions are called RSA ciphers (from the three inventors' initials). The function requires a person to choose three large prime numbers, of which one and the product of the other two are published collectively as a public key – for everybody to encrypt messages to this person. An exponential function with modular division is applied to every ASCII character in the message. The original recipient is the only one who knows the three secret numbers and there are too many combinations of the products of these numbers for somebody to guess them.
- While secure, public key encryption is vulnerable to corruption and interference with message. Hashing helps with that. A hash is a unique string of data through which the original message is put. Unlike a key, which is unique to a person but is the same for all the messages from/to this person, a hash is unique for every message and is sent along with the encrypted message to the recipient. Any smallest change in the original message would generate a completely new hash. Upon receiving the message and the hash, the recipient decrypts the message, generates a hash and if whatever has been generated doesn't match the hash that came along with a message, the message has been tampered with! Hashes are secure; they can't be reversed without trying an almost infinite number of possible combinations. So, a hash is a bit like a unique signature.
- One of the most common signatures is the **Digital Signature Algorithm (DSA)**. It is possible to apply a signature to a hash, creating two layers of authentication.

Common misconceptions

- Students often misunderstand that the compression ratio is better when it is smaller.
- Students also confuse lossless and lossy compression or think that lossless is always better.
- They also tend to forget the names of encoding schemes.
- In practical work, students may not be able to do encryption reliably, and in programming it, they may not be familiar with string manipulation and nested loops.
- Students also confuse public and private keys (which one is used for encryption, and which for decryption).
- They may also not know what modular division or exponential functions are, which may need clarification in class.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
Identifying patterns and	**Starter:** Target a random poster or school rules banner around the classroom and ask students to identify any repeating patterns. **Main activity:** Ask students, working in teams, to then try to 'compress' the text of this poster with pen and paper and calculate the compression ratio. **Plenary:** Consider what type of data can yield the highest compression ratios.	Ask students to try alternative methods of lossless compression – for example a dictionary approach – and compare the result with their first attempt. To illustrate the concept of longer files yielding better compression ratios, students can first try compressing a few separate texts and then combine them into one text and see if the compression ratio will go down.
Practical use of open-source archival package	**Starter:** Show students the 7-zip documentation and then ask which features might be the most popular. **Main activity:** Obtain copies of a couple of freeware compression packages and instruct students to run a 'lab' – compressing a variety of files (e.g. a text document, an image [bmp, not gif or jpg], a sound recording [including an mp3 which is already compressed to the max]). **Plenary:** If students did try to compress mp3 or jpg files, ask them what they can say about compression ratios.	Research how jpg compression differs from gif compression. Prepare a debate on which one is better. Ask students to use a language of their choice to create a program that can compress a text file from the Gutenberg project on the internet.
Practical use of decryption	**Starter:** Demonstrate encryption using one of the tools mentioned in the chapter. **Main activity:** Designate students as 'spies' and give them encrypted portions of a text and their own keys. They have to decrypt their part, find other students and their portions of the text and put all the text together into an original document. **Plenary:** Assume that, given enough time and computer power, most encryption can be broken. Are longer messages or shorter messages easier to crack?	An extension of this could be trying to do a simple example on public key encryption, similar to the one given in this chapter. It might be fun to get one of the students to encrypt a short message using that, and tell others that the offset (the number of alphabet positions all letters were shifted by) was within a certain range. Then the others, working separately, will try the brute-force approach of trying all combinations until the message makes sense. Create a cipher encryption/decryption program in their choice of programming language (even Scratch if need be). Create a program to crack a simple cipher by trying all combinations until a certain word is recognised.

These answers relate to Questions asked in OCR Component 1. Further information on this element of the material is available at www.cambridge.org/ukschools/OCRcomp1

End-of-chapter answers

1 Lossy compression achieves a higher compression ratio but some data is lost. Lossless compression retains all the original data but compression ratios are lower.

2 Data type of encryption where the same key is used to encrypt and decrypt the data.

3 The substitution or Caesar cipher.

4 The encryption and decryption keys are different. For example, the RSA cipher.

5 Hash functions can be used to create digital signatures.

Chapter 8: Databases

Learning objectives

- Understand relational database, flat file, primary key, foreign key, secondary key, normalisation and indexing.
- Understand entity relationship modelling.
- Understand methods of capturing data.
- Understand normalisation to third normal form (3NF).
- Understand SQL.
- Understand referential integrity.
- Understand transaction processing, ACID (Atomicity, Consistency, Isolation, Durability), record locking and redundancy.

What your students need to know

- Modern computers are synonymous with data.
- **Big Data** is a buzzword for (usually internet-based) incredibly high-volume databases such as those run by the leaders of online industries, for example, eBay, Facebook, Google, etc.
- Big Data aims to understand the needs of consumers or other trends and target them with advertising, which is controversial due to invasion of privacy and vulnerability to data theft by hackers and governments overstepping boundaries.
- Prior to the widespread use of computers, data was held in paper form, which took a lot of storage space and was quite an effort to copy and search.
- **Databases** are best for keeping the data accessible and searchable.
- Many apps and websites are just interfaces to databases.
- The flat file database is a simple type of a database where all data is held in a single table. It is not suitable for large projects due to data redundancy.
- Data redundancy is a problem that not only wastes space but also results in inconsistent data and failed searches (when databases are updated, we might miss some copies of the same data, for example the customer's name, and entries will look like they belong to different customers).
- **Relational databases** overcome the problem of redundancy through the use of multiple tables, creating another dimension in addition to the two dimensions of a flat file database. Primary and foreign keys are used to link the tables, so that a query will pull the data from multiple tables into one flat file report.
- **Primary keys** must be unique and every table has one. A primary key of one table might be linked to another table, where it appears as a foreign key.
- **Secondary keys** are used for faster searching, and are similar to tags/hashtags used in social networks and blogs (e.g. '#LOL'). They don't have to be unique.
- Indexing saves searching by copying out the most searched-for database attributes, for example surnames, into a separate file/table. An index table is like a table of contents in a book, except instead of topics and page numbers it would have records of surnames and their location on a hard disk.
- **Normalisation** is the process of splitting up a busy redundant flat file database into multiple tables that form a relational database. There are various degrees of normalisation, with the 'un-normalised' being the lowest and third normal form being the highest, where all the data that can be sensibly isolated in their own table have been separated out.
- The first normal form is where no cells in a table contain multiple entries/bits of data and all the columns can be connected to a primary key.
- The second normal form has no partial dependencies. This means that if a table contains columns that depend on each other more than on the primary key, we can remove them to

a separate table and just leave one column as a foreign key. The rule of thumb is, if multiple instances of a foreign key relate to many instances of a primary key (known as a many-to-many relationship), then the table is not in a second normal form.

- The third normal form is about removing 'transitive dependencies'. These are indirect links, like a 'friend of a friend', or a 'student's form tutor's department' where students and the form tutor's department are linked only indirectly or 'transitively'. We would create a separate table for our 'friend' and their friends will be stored there, but not in our main table. Transitive dependency occurs when two non-keys attributes of the same entity are related (e.g. student's form and form tutor's name).
- Third normal form databases usually feature transaction tables; for example, rather than students and courses, we will have students, enrolments (which is a transaction table as it is likely to contain more records than other tables) and courses.
- **Structured Query Language (SQL)** is used to manipulate most common databases. Any manipulation of data (including just retrieving it) is known as a 'query'. We know select queries (retrieve data), insert queries (add data to a database), update queries (change values in a database) and delete queries (delete unneeded records). Additionally, we can use SQL to create, modify and delete entire tables and their relationships.
- Queries can combine data from multiple tables, perform calculations and use criteria to work either with all records or with just the ones fitting the criteria.
- SQL is written in plain text, is very close to English and was created for non-computer specialists to understand. It is understood by many otherwise incompatible databases and can be combined with other programming languages such as Python, Visual Basic or **Hypertext Preprocessor (PHP)**.
- Referential integrity is important for relational databases. If a related table is updated it might not match the related data in another table, especially when foreign keys are involved. This results in 'orphan records'. This is similar to ending with broken links on the internet if a site linked to has changed its URL.
- Databases hold large volumes of data while most queries work with a small portion of a database, in small bursts of activity called transactions.
- Successful transactions have four properties, known together as ACID:

  1. atomicity (every single objective has been achieved, every part worked)
  2. consistency (data is not corrupted, database rules followed)
  3. isolation (if multiple transactions are executed in parallel the result is identical to that obtained if they were executed one after the other – so, they can't affect each other's operation, which might involve 'locking' the data for one query that a parallel query is working on)
  4. durability (changes are committed and saved to disk).

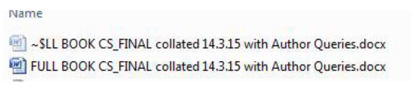
Common misconceptions

- Students can often confuse the different normal forms.
- You might also want to point out that some businesses still use paper-based records as some students may not know this.
- They also might not know that databases are not something that most people would use, spreadsheets or text documents being the usual choice for keeping records.
- You should also emphasise that databases don't always look the same.
- Student may often confuse primary and foreign keys.
- Some students may not be sure what indexing is for; for example, that it is used for speeding up searching.

- Students often confuse partial with transitive dependencies, and don't really understand what a dependency is.
- They might also not understand the downsides of having many-to-many relationships.
- You should remind students about SQL syntax and how it combines with underlying languages like Python or Visual Basic.
- Students may also not know or understand the difference between entities and queries, which both look like spreadsheets on the screen.

Topic	Lesson suggestions	Follow-up ideas/Homework
Big Data and issues	**Starter:** Discuss internet history. A good start for Big Data and the application of databases could be visiting a site called The Wayback Machine. This site keeps a copy of most webpages going back to the 1990s. Is it possible to see what your school's website looked like ten years ago? **Main activity:** Discuss Snowden's revelations – this is a good attention-getter and is related to many current political issues. Students can be tasked with researching five famous data breaches (e.g. Google's accounts, WikiLeaks, iCloud theft of celebrity photos, etc.). **Plenary:** Talk about the iOS and Google mobile devices. It is known that if your mobile were stolen, you wouldn't have to pay for your apps again; Apple or Google would restore these apps to your new device. What kind of data needs to be stored on your device by these companies to allow this?	Give students a selection of popular sites and ask them to find the oldest and the least cool version of these sites in terms of design and content (e.g. advertising iPhone 1). This could be followed up with a discussion of issues around copyright infringement, uncovering things that are best left forgotten and the techniques used to grab whole websites with pictures and data and copy them into this database.

Topic	Lesson suggestions	Follow-up ideas/Homework
Databases	**Starter:** For paper-based storage, ask students to research pictures of old-fashioned law offices/medical surgeries with boxes of records stored. The Early Office Museum website has some good illustrations. To discuss the underlying database nature of modern apps and websites, you could ask the students to brainstorm how a games company would keep track of in-app purchases of lives, ammunition, etc. What data would need to be recorded? **Main activity:** Give students an MS Excel workbook and ask them to create a corresponding MS Access database. Use MS Excel and MS Access to normalise the flat file database. In an MS Access database, use a query wizard to create a report or a query, then switch to the SQL view to see what MS Access did behind the scenes. **Plenary:** Show students how various database packages will implement the same data model.	Create a database for a doctor's surgery appointments system. How many tables will be needed?
Normalisation, 3NF and SQL	**Starter:** Flat file databases are best illustrated with Excel (or other spreadsheets). Every sheet is a flat file database. Instruct students to construct an un-normalised, or back-of-a-napkin, database by recording this chapter's examples in Excel. **Main activity:** As the process of normalisation is explained, students should move the data to other sheets and name them accordingly. Excel will allow labelling of columns and, through the use of '3D' formulas, we can implement primary and foreign keys, too! Using MS Access will be a logical step from there. If there is a capacity to use PHP or MySQL with HTML/web hosting, it might be a good exercise. Python comes with the SQLITE package that allows students to implement all the queries mentioned in the chapter text. **Plenary:** With the same data structure implemented in MS Access and Python, compare the SQL code that these packages generate/use and comment on the differences.	Given an MS Access database, create a Python SQLITE database with identical functionality.

Topic	Lesson suggestions	Follow-up ideas/Homework
Referential integrity and transaction processing	**Starter:** Create a copy of the MS Excel/MS Access/Python database and remove some data. Demonstrate the errors that result. **Main activity:** Ask students why a MS Office document opened by a person, creates a strange copy of itself with '~' in front of its name?  Name ~$LL BOOK CS_FINAL collated 14.3.15 with Author Queries.docx FULL BOOK CS_FINAL collated 14.3.15 with Author Queries.docx Why, if two people open the same copy of a Word document, will one of them see the message 'This file is in use by another user. Open read-only?' Get the students (once they are comfortable with creating MS Access queries) to try to create the one that will break these principles. **Plenary:** Discuss if ACID is always possible and/or desired.	Research how referential integrity of a database can be restored (MS Access has a database repair utility, while in Python we might be able to program a routine that updates the links, restoring the integrity).

These answers relate to Questions asked in OCR Component 1. Further information on this element of the material is available at www.cambridge.org/ukschools/OCRcomp1

End-of-chapter answers

1 All the data is stored in a single table.

2 The primary key is a unique identifier, usually created by the system e.g. an ID number.

3 A database made up from lots of smaller tables, linked together by primary and foreign keys.

4 Everything from the Students table will be returned.

5 To ensure that every foreign key relates to a primary key from another table.

Chapter 9: Networks

Learning objectives

- Understand how data is transmitted over networks and other wired connections.
- Understand how protocols work and the difference between physical and logical protocols.
- Learn about protocol stacks and why they are structured in this manner.
- Learn about the key networking hardware, including hardware needed for a wireless network.
- Understand and describe the key differences between circuit and packet switching.

What your students need to know

- This topic can be delivered entirely through theory, but if it is possible to incorporate practical activities, this can be very beneficial to learning.
- Raspberry Pi® computers running Raspbian Linux can be used as network nodes in peer-to-peer (P2P) or client server networks.
- Students need to understand the characteristics of a **Local Area Network (LAN)** and a **Wide Area Network (WAN)**. LANs generally operate at higher bandwidths than WANs, typically between 100 to 1000 Megabits per second (Mbps). WAN bandwidths are usually less than 100 Mbps. LAN connections may be wired, using either copper or fibre optic media, or wireless, using radio frequency communication.
- The data-carrying capacity of a network medium is termed the 'bandwidth'. In a wireless network, this is the range of data-carrying frequencies that are supported. In reality, the amount of data carried is less and this is termed the throughput.
- WAN technologies are based on the telephone system, which originally used analogue signals.
- LANs, on the other hand, have always been digital networks.
- A **modem** (modulator/demodulator) was used to connect a digital computer to the analogue telephone network. Modern WAN connections such as **Asymmetric Digital Subscriber Line (ADSL)** still require the use of a modem but operate at higher connection speeds. The telephone system exchanges were designed to connect circuits using relays. Once the circuit was connected, all data was sent down the same path until the connection was terminated. This is circuit switching.
- Data networks required faster switching and the flexibility of being able to send small chunks of data, called packets, through the least congested pathways. This is termed packet switching.
- **Routers** connect networks and analyse which is the best path for a packet to be sent along to the destination IP address. On a LAN, switches are used to connect nodes and to forward data based on a physical address (MAC address).
- Wireless networks use **Wireless Access Points** to generate the radio frequency signal. These may connect directly to the rest of the network via a cable or via a wireless connection.
- **Protocols** define the rules of communication between two devices. TCP/IP is a well-known protocol suite that was originally developed for the internet. It is the most widely implemented protocol on LANs. The **Internet Protocol (IP)** delivers packets and defines a logical addressing system. The **Transmission Control Protocol (TCP)** ensures reliable delivery of packets and incorporates error correction and flow control.
- Other protocols perform more specific functions in the communication process: **Dynamic Host Configuration Protocol (DHCP)** assigns IP addresses to hosts; **Domain Name System (DNS)** maps human-friendly resource and computer names to IP addresses; **Hypertext Transfer Protocol (HTTP)** delivers webpages from a server to a client browser.

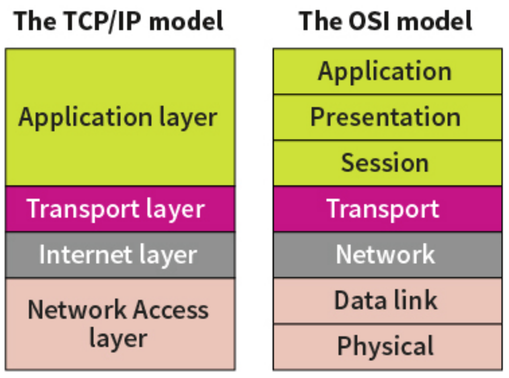
- Other protocols handle e-mail (**Simple Mail Transport Protocol [SMTP]**), files (**File Transfer Protocol [FTP]**) or routing information (**Routing Information Protocol**). The protocols of the TCP/IP suite are organised into a stack, with the most basic protocols at the bottom and those performing specific functions at the top.

Common misconceptions

- Students often get confused between the terms 'physical' and 'logical'. These relate to protocols but also to topologies. It is helpful to have some unshielded twisted pair UTP cable to show students and to talk about the purpose of standards. Physical protocols specify the network medium (cables, connectors or wireless signal, electrical characteristics, signal encoding, timing, character set and error correction. Logical protocols are concerned with how the message is transmitted and may include: handshaking, addressing and port numbers, message initiation and teardown, error correction and flow control.

- Students may also confuse circuit switching and packet switching. They should be aware of the differences between these two methods of data transfer. It can help to role play the two methods.

- Students and teachers may confuse the TCP/IP and Open Systems Interconnection OSI networking models. Many protocols are associated with these models but only significant ones should be covered. It is also helpful to teach students the history of the two models. The four-layer TCP/IP model was developed based on internet communication. The OSI model was developed for networks of all sizes. The top three layers of the OSI model equate to the upper layer of the TCP/IP model, and the bottom two layers of the OSI model map to the bottom layer of the TCP/IP model.



Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
How data is transmitted	**Starter:** Ask students, using mini whiteboards or paper, to consider the steps involved in sending an e-mail. They may not know the precise process or names of protocols at this stage but they may be able to pick up on certain features such as delivery receipts and delay notifications. Use Q&A to elicit as much detail as students know at the start of the lesson. **Main activity:** Ask students to investigate LANs and WANs and the role of clients and servers. They should consider how digital signals can be modulated to carry data on copper media as electrical signals, on fibre optic cable as light pulses and in wireless networks as radio frequency waves. They can use this information to produce a factsheet about networks to be given to householders by a telephone company / ISP (Internet Service Provider). **Plenary:** Discuss the measurement of speed of a network. Students will have encountered the term 'Megabits per second' (Mbps) and this should be explained to them. Consider how fast network signals can travel. What are the limitations with data transfer rates? This will permit the definition of two important terms: 'bandwidth' and 'throughput'.	Ask students to investigate P2P networks both as LANs and on the internet. P2P networks led to illegal file sharing and prosecution; however, there is also a host of legal sites. What are the advantages of internet file sharing? The mobile phone network has developed alongside internet access via the 'Plain Old Telephone System/ Service'. What are the differences between these services? Do we still need 'landlines'? What is a VPN (Virtual Private Network) and how can this allow businesses to work securely using the internet?

Topic	Lesson suggestions	Follow-up ideas/Home-work
Protocols	**Starter:** What is a protocol? Ask students to work in teams to develop a communication method that must be secure. Students should be encouraged to think about what they will do if the transmitter is sending data too fast? What will they do if an error occurs? They must ensure that a ‘hacker’ will not be able to work out what the message is by eavesdropping on the conversation. **Main activity:** Identify protocols that relate to physical standards, for example Ethernet, and those that are entirely implemented in software (logical). Investigate a number of key protocols: IP (Internet Protocol), TCP (Transmission Control Protocol), RIP (Routing Information Protocol), ICMP (Internet Control Message Protocol – PING), DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol), HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transport Protocol), IMAP (Internet Mail Access Protocol). Each student could produce a wiki page about their protocol. (If you use the VLE Moodle, wiki is a built-in tool.) Alternatively, Google Slides or other presentation software could be used. **Practical activity:** If resources are available, create a small LAN using two clients and use Wireshark to capture live protocol packets in real time. This activity can also be carried out on a Raspberry Pi® although use of Wireshark is not recommended for this. Instead use the terminal-based Tshark or tcpdump and export to a file which can be read using Wireshark on a laptop. **Plenary:** Play an online quiz with questions relating to protocols (e.g. using Socrative).	Conduct an investigation into the changes between iPhone 4 and iPhone 6 and the challenges that migration will present. Produce a mind map of protocol concepts. This can be carried out using online software such as Popplet.

Topic	Lesson suggestions	Follow-up ideas/Homework
Network models	**Starter:** Recap physical and logical protocols. **Main activity:** Run through the two key networking models (TCP/IP and OSI) with reference to their age and purpose. Students should complete a 'mix and match' activity of layers and protocols at each layer. For this activity, use the TCP/IP model as it's simpler. **Plenary:** Discuss the importance of standard interfaces. Changes in physical protocols do not affect logical ones, and vice versa.	Produce an animated presentation or a poster to illustrate example communications between a client and a server. Students could select one application layer protocol (e.g. HTTP, DNS, DHCP).
Network hardware	**Starter:** Discuss the hardware that students have at home to connect to the internet. Introduce the terms 'router' and 'wireless access point'. **Main activity:** Build a network from cut-out and laminated images of network hardware. Additional cards can be used to 'configure' devices with IP addresses. **Raspberry Pi® Activity (Model B):** Set up a working LAN using Raspberry Pi®s, a switch or hub and UTP cables. Students can make video blogs of their practical activity. A wireless network can also be set up using USB wifi adapters and a wireless access point. **Group activity (optional):** Enlist an IT technician to conduct students on a guided tour of your school or college network. Usually, servers, switches and routers are hidden from view and it can be a very valuable opportunity to see a large LAN and to talk about the issues of maintaining it. **Plenary:** Play an online quiz on the topic of network hardware (e.g. using Socrative).	Ask students to produce a blog post on the topic: 'Securing your home network.'

Topic	Lesson suggestions	Follow-up ideas/Homework
Circuit and packet switching	**Starter:** Present students with a maze game in which they must move four counters from the 'source' to the 'destination' in the smallest number of moves using any route. They must then repeat the activity but can only move the counters along the same pathway once they have decided which they think is the shortest. This will help students consider the advantages of packet switching over circuit switching. **Main activity:** List different types of data that would be sent across a network that must be streamed (video and audio) and that can be split up into packets that can be sent separately (e-mail, webpage, file transfer). Explain the historical basis of internet traffic and how it used the telephone network before the transition to packet-switched technologies. Divide the class in two: one to represent circuit-switched networking, the other packet switching. Each half must prepare a 'pitch' to promote their technology. **Plenary:** Complete example examination questions and peer mark.	Ask students to research VoIP (Voice over IP) and the convergence of data and voice networks.

End-of-chapter answers

1
a. A hub will allow multiple networked devices to be connected together. Packets sent to the hub will be broadcast to every other connected device.
b. A switch will allow multiple networked devices to be connected together. Packets sent to the switch will be routed to the correct port using their MAC addresses.
c. A router behaves in a similar way to a switch. However, routing is done using IP addresses and allows a LAN to be connected to a WAN, like the internet.

2 If there is a low bit rate then smaller amounts of data can be transmitted per second. When streaming video, if the bit rate falls below the amount needed to show a single second of footage then the video will stutter or start buffering. When downloading files, the bit rate directly impacts the length of the download for large files. For small files, the bit rate is not as important.

3 If there is a low bit rate then smaller amounts of data can be transmitted per second. When streaming video, if the bit rate falls below the amount needed to show a single second of footage then the video will stutter or start buffering. When downloading files, the bit rate directly impacts the length of the download for large files. For small files, the bit rate is not as important.

4
a. Packets can be sent along a cable in one direction only at any time.
b. Duplex allows packets to travel in both directions at the same time.
c. Bits sent in parallel are sent at the same time, rather than one after another. The duplex mode can be set on top of this.

5 Each row and column has a parity bit added to help locate when a bit is in error. Row 3 has an odd number of bits set to 1, which means an error happened on this row. The forth column also has an odd number of bits which means that the bit in row 3 column 4 has an error. Because we know where that bit is we can swap it from 0 to 1 in order to correct the error.

6 On the top layer is the application layer, which is where applications will format their packets before they are sent. Web browsers will use the HTTP protocol (hypertext transfer protocol) while file transfer uses FTP (file transfer protocol). In order for the packet to be understood, both sender and receiver must be using the same protocol. When the packet has been correctly formatted for the destination software, it must then either use the TCP or UDP protocol. TCP allows error detection and packets to be ordered correctly. UDP, on the other hand, does not have any verification checks. Finally the packets have routing information added by the IP protocol before being paced onto the network hardware

Chapter 10: Web technologies

Learning objectives

- Appreciate the difference between the world wide web and the internet.
- Have a basic understanding of HTML, CSS and JavaScript.
- Understand search engine indexing and the PageRank algorithm.
- Understand server-side and client-side processing.

What your students need to know

- This topic will provide students with an in-depth understanding of the underlying technologies associated with the internet. Websites consist of a collection of pages stored on a web server to be accessed by clients and viewed in a browser. Each server will have a unique IP address but, as these are more challenging for humans to remember, the Domain Name System contains a mapping between IP address and a more memorable domain name.

- A **Uniform Resource Locator (URL)** contains not only the domain name but the server name (often called 'www') and the file path and name on the server, for example: http://www.ietf.org/rfc.html

- HTTP is used to deliver pages to a client's browser when a request is made. HTTP is part of the TCP/IP protocol suite. All of the web servers on the internet form what is known as the world wide web (WWW), which is a term first coined by its developer, Sir Tim Berners-Lee.

- Webpages consist of text, images and other media encoded within a page of **Hypertext Markup Language (HTML)** code. This instructs the browser how to display the webpage. Webpages can also contain code in the form of a script.

- **JavaScript** is the most common language and is run on the client, which aids efficiency. JavaScript features make webpages interactive to include sound, animation and streamed video. The current version of HTML is 5. This version supports interactive content using scripting and the HTML canvas module. Server-side scripts allow data entered by a user into a webform to be validated to check correctness. They can also carry out database updates and queries and perform security scans. An example of a server-side scripting language is PHP.

- **Cascading Style Sheets** are small files associated with a webpage that define how it will appear to the user: the fonts, font colour, background colour or graphics can all be defined. They can also be used to increase the accessibility of a webpage.

- Students can experiment with scripting languages, HTML5 and CSS3 here.

- Search engine technologies aim to provide results quickly that are highly relevant to the search criteria. Algorithms are used extensively to ensure that the most popular and significant websites appear most frequently in search results. Google developed the PageRank algorithm, which uses the number of links on a webpage as a measure of its importance. Indexing searches for keywords to ensure relevance of results based on search criteria that a user might enter. It should be impressed upon students that the marketplace for many organisations exists only online (e.g. Amazon, eBay, Etsy and Asos).

- **Search Engine Optimisation** is the process of structuring a website to ensure that it attains a high ranking in search engine results. The topic of search engine operation is new but investigation of the underlying algorithms is beneficial. Google search algorithms can be learned about at the following URL.

Common misconceptions

- The terms 'server' and 'client' can cause students some confusion. They need to be aware that clients are the recipients of network services and that servers provide them.

- This topic also has a number of acronyms (e.g. HTML, HTTP, DNS, CSS). A glossary can help students remember what each protocol and technology achieves.

- Students may also describe HTML as a programming language rather than a mark-up language. Its purpose is to provide the necessary formatting information for the browser to be able to display the webpage correctly.
- JavaScript is a scripting language that is interpreted inside the HTML code of a webpage. Java is a programming language that can be used to develop standalone programs.

Lesson activity suggestions

Topic	Lesson suggestions	Follow-up ideas/Homework
The WWW and the internet	**Starter:** Ask students to complete an activity chart of their internet use within the last 24 hours. This could take the form of a clock upon which they indicate the time spent and their activity. **Main activity:** Produce a timeline of the historical development of the internet. **Plenary:** Discuss the possible future direction of the internet. What kinds of devices will we be using to access the internet in the future? What types of information and data will we be able to access?	Write an essay on the work of Sir Tim Berners-Lee.
The role of HTML	**Starter:** Create a basic HTML 'scratch pad' in a text editor and build a simple webpage. **Main activity:** Investigate HTML5 and its capability. Students should produce a webpage with interactive features including the use of CSS and JavaScript. **Plenary:** Ask students to find their five top HTML5 websites and demonstrate these to the rest of the class.	Produce a quick start guide to HTML5. Here is an excellent resource for students to learn HTML5, JavaScript and CSS3.
Search engines	**Starter:** Produce a paper timeline from 1990 to the present day. Students to use sticky notes to add on key developments in search engine history. **Main activity:** Ask students to produce a diagram and summary of how search engine indexing and the PageRank algorithm works. **Plenary:** Conduct a search engine simulation activity using different newspaper articles and search criteria.	Investigate Search Engine Optimisation: What techniques can an organisation use to ensure that its pages always attain a high ranking in search engine results?

Topic	Lesson suggestions	Follow-up ideas/Homework
Server and client technologies	**Starter:** Recap the definition of server and client. Review network services. **Main activity:** Ask students to use their knowledge of HTML5 to produce an interactive webpage about the interaction between servers and clients for different protocols: FTP, HTTP, DNS, DHCP and SMTP. **Raspberry Pi® activity:** Set up an apache web server on a Raspberry Pi® to host simple webpages. A Raspberry Pi® client can then access the pages across a simple LAN. **Plenary:** Play an online ‘quick fire’ quiz on network services and server-client interaction.	Produce a mind map of network services and their roles. This could be done using an online utility such as this one.

End-of-chapter answers

1 A collection of all the webpages on the internet.

2 Hypertext markup language. It is used to create webpages.

3 Cascading style sheet. Contains information on how a webpage should be formatted.

4 In server-side processing operations are carried out on the server and the results sent to the user. In client-side processing all processing is carried out by the user’s computer.

5 The algorithm Google’s search engine uses to ensure that users only receive the most relevant results.

Chapter 11: Data types

Learning objectives

- To be able to recognise primitive data types including: integer, real/floating point, character, string and Boolean.
- Know how to represent positive integers in binary.
- Understand the use of sign and magnitude and two's complement to represent negative numbers in binary.
- Be able to add and subtract binary integers.
- Be able to represent positive integers in hexadecimal and convert between binary, hexadecimal and denary.
- Understand the representation and normalisation of floating point numbers in binary.
- Understand floating point arithmetic, positive and negative numbers, addition and subtraction.
- Understand bitwise manipulation and masks: shifts, combining with AND, OR and XOR.
- Understand how character sets (ASCII and UNICODE) are used to represent text.

What your students need to know

- **Electrical current** can either flow through an electronic circuit or be stopped, which a computer represents with the digits 1 for 'ON' and 0 for 'OFF'; these form the binary number system.
- Each 0 or 1 is known as a **binary digit (bit)** and it takes large groups of these digits to represent any significant amount of data; 4 bits are called a nibble, and 8 bits are called a byte. Other units are kilobytes, megabytes, gigabytes and terabytes.
- Students need to know the base numbers representing various degrees of 2: 1, 2, 4, 8, 16, etc. Any decimal number is converted to a binary by subtracting these base numbers until we are left with 0. Converting binary to decimal involves adding these numbers together (only if there is a binary 1 corresponding to their position in a byte).
- They also need to know that the number of bits allocated to storing a particular decimal number will influence the maximum size of the number that can be stored in that memory.
- There are a few ways to represent negative numbers in binary: some of the more common are 'sign and magnitude' and 'two's complement'.
- Sign and magnitude is the easier one of the two; the leftmost digit (otherwise known as **Most Significant Digit [MSB]**) becomes an indicator for a sign: 1 for negative and 0 for positive. By dedicating a whole bit to a sign, we halve our range of numbers that can be stored in the same amount of memory, so two's complement is a more economical solution, which, however, requires more processing.
- To convert a negative decimal number to a two's complement binary one, we first write it out as if it was positive. Then, starting on the right, leave every bit up to and including the first 1 alone, but flip/invert the others from 0 to 1 and from 1 to 0. This is known as complementing, as 0 and 1 together complete the binary set, so they complete each other.
- Binary addition isn't that different from decimal addition of long numbers. One has to remember to carry, which is quite common.
- Binary subtraction is the same as binary addition but the number to be subtracted is converted into a negative one using two's complement.
- **Hexadecimal numbers** have base 16, rather than base 2 of binary numbers. Therefore, each hex number is equivalent to a nibble. Two hex numbers make up an equivalent of a byte. To convert a decimal to a hex number, you might need to convert this number to an 8-bit binary and then split that into two nibbles, 4 bits each, and convert them individually to hex.

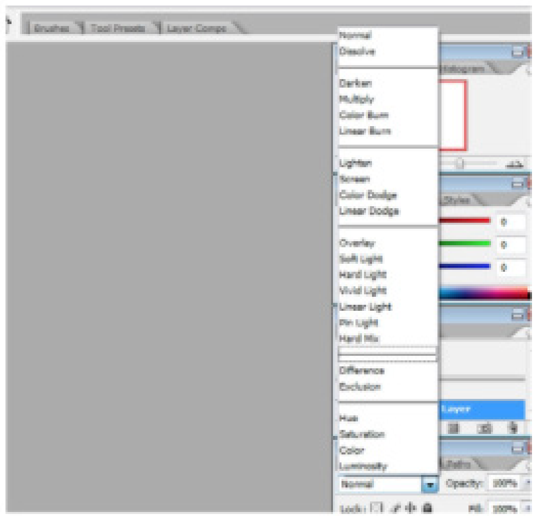
- The **floating point** is a binary version of scientific notation you can get in your calculator, made up of the **mantissa** and the **exponent**. The number of digits is fixed but the decimal point can float around. In binary, both the mantissa and the exponent are in binary, obviously. The mantissa is 'normalised', that is, always written with 01 as the two leftmost digits with a floating point in between the 0 and the 1. The exponent, once converted to decimal, tells us how many times to shift the floating point left or right (depending on whether the exponent is positive or negative) to get the actual number we have originally stored.
- The number of bits that represent the mantissa determine its precision – how small a fraction we can store without truncating the really small ones. The number of bits in the exponent determines range – the max and the min numbers we can represent in a given amount of memory.
- Both negative and positive fractional numbers can be represented.
- When adding two floating point numbers, their exponent must be the same and move the floating point in the mantissa accordingly by adding zeros from the left if needed. For subtraction, we change the mantissa of the number that's being subtracted to the negative.
- Bitwise manipulation and masks involve applying logical operators AND, OR, NOT and XOR to every digit in a binary number. To apply these to a number, we need another number, called a **mask**, so that each digit of the mask number will interact with a corresponding bit of the original number, depending on the logical operator applied (e.g. 1 in original number AND 1 in the masking number will produce 1, 1 and 0 will produce 0). Students need to know the effect of all of the logical operators.
- Internally, all characters are represented as binary. The two most common character sets are **ASCII** and **UNICODE**. ASCII is a 7-bit code with a smaller set of characters, only really good enough for English characters. UNICODE uses up to 32 bits to represent many times more characters, including those used in the writing of all living and dead languages.

Common misconceptions

- Hex is not used by computers but by humans working with them, to reduce errors reading long binary numbers.
- Students often forget to carry numbers to the next column when doing binary addition and subtraction.
- They can also forget which bits to flip and which ones not to flip when applying two's complement.
- Students can mix up mantissa and exponent, as well as forgetting to normalise mantissa.
- They can forget what logical operators do, especially confusing OR and XOR, as well as forgetting that the NOT operator only needs one number, not two.
- Students need to remember that ASCII only handles English letters and can often forget how many bits are in each character set.

Lesson activity suggestions

For bitwise numbers and masking, a good exercise could be to use either Photoshop or its free alternative, GIMP, to create two layers of graphics and then cycle the blending mode between the layers, which represent variable masking operations. There is also a masking feature which applies brightness or other adjustments to only masked portions of the image, as shown on the pictures, using Photoshop CS2.



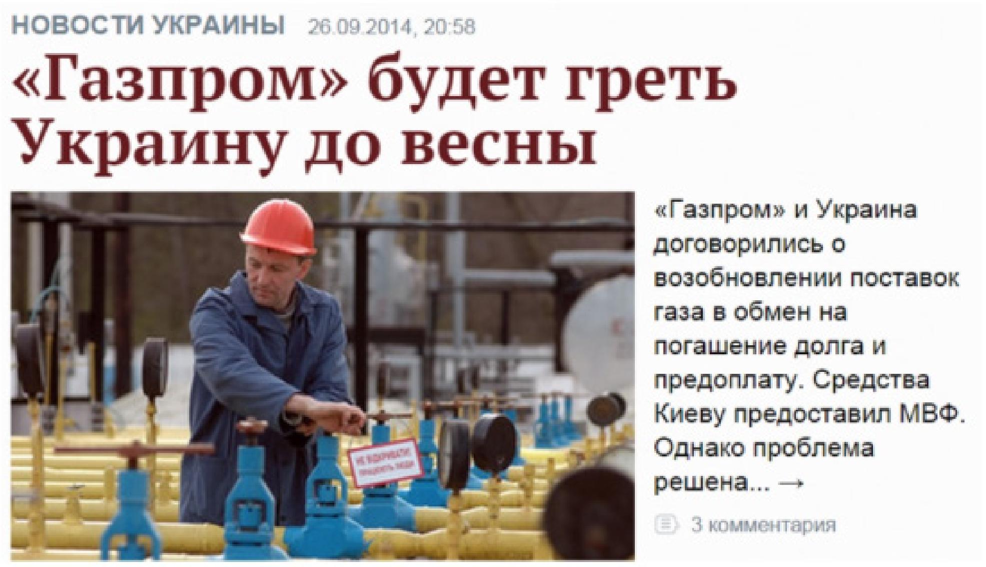
Photoshop layers allow different graphics and colours to interact when placed in layers. Layers can be combined in a number of ways, as shown on a picture. Students may be asked to research (or brainstorm) how some of these effects are achieved in the software (e.g. how would 'Difference' mode work?).

Doing a lot of past questions and also modelling operations via programming seems to be the best way to approach these very hands-on topics.

For character sets, it is entertaining to explore various code page settings in a browser.

A sample activity (could be any non-English alphabet site):

Here is a Russian news site www.gazeta.ru



НОВОСТИ УКРАИНЫ 26.09.2014, 20:58

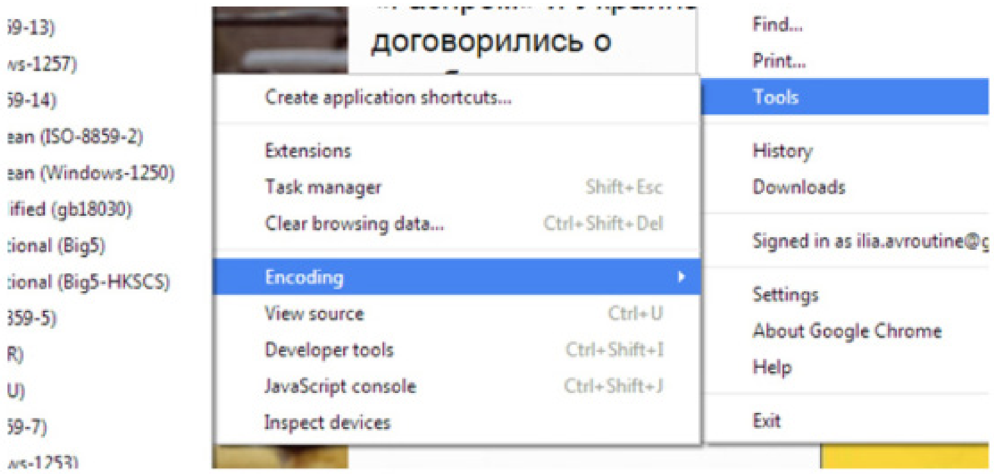
«Газпром» будет греть Украину до весны

«Газпром» и Украина договорились о возобновлении поставок газа в обмен на погашение долга и предоплату. Средства Киеву предоставил МВФ. Однако проблема решена... →

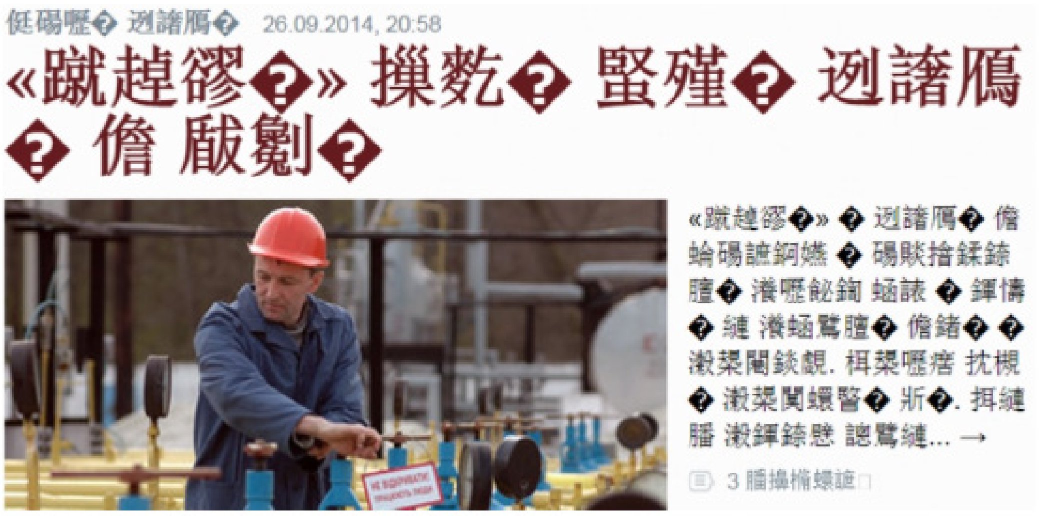
3 комментария

The text is in Cyrillic, the name of the alphabet used by a number of East European countries. Your internet browser recognised the fact that the text is not in English and automatically utilised the Cyrillic 'code page' of characters. However, we can override it and force Chrome (could be any browser) to use a 'wrong' character set.

Use Google Chrome's encoding change option:



Using the Encoding options, we forced the character set to be Simplified Chinese.



We just changed the text to nonsense Chinese that no Chinese speaker will understand because the letters are still put together using Russian vocabulary, so changing the characters is not the same as translating the text into a different language.

Topic	Lesson suggestions	Follow-up ideas/Homework
Binary	**Starter:** Have a 'dictation' where a decimal number is read out, students write down the binary (or hex) version, and a few seconds later the teacher moves to the next number. Student's papers are then swapped and peer marked. **Main activity:** Do past exam questions (can use questions from different exam boards; they are not very different on this topic). **Plenary:** Ask students what is the cause of most mistakes when working in binary (e.g. forgetting to carry, etc.).	Create another dictation for the teacher to read out in the next class. Use a programming language like Python that supports binary-to-decimal-and-back conversion and shows floating point binary numbers to create a dictation for a teacher to read out next. Complete and extend programs at home. Create a program that creates sample questions on binary addition/subtraction and other binary operations and supplies correct answers – ideally, writing all this into a text file.
Bitwise manipulation and masks	**Starter:** Give a demonstration of two Photoshop/GIMP layers interacting via different blending modes. Teacher provides a mathematical explanation of how AND can be used to mask layers. **Main activity:** Give explanations and set practical activities involving shifts and masks. **Plenary:** Give a summary of bitwise operations and discuss their practical uses.	Research masking here. Use tutorials such as this one off blog or videos such as this one here. Students might undertake writing a program in their choice of a programming language that can randomly modify a media file (to 'glitch it') and then explain why the glitch works the way it does.
Character sets	**Starter:** Open a foreign language site that uses non-Latin characters (e.g. Bulgarian or Chinese), then force change the character encoding. Bulgarian Cyrillic characters can immediately become Chinese, but it's not Chinese text that makes any sense! **Main activity:** Explain the history of character sets and the advanced features of UNICODE (e.g. its use of ancient Egyptian hieroglyphics). **Plenary:** Having looked at the UNICODE with its premise of supplying every known character, ask students what they could recommend for additional inclusions into UNICODE.	Have a treasure hunt – find UNICODE characters for some obscure symbols, for example Egyptian hieroglyphics or a particular Chinese character. Reference for UNICODE is here.

End-of-chapter answers

1 A series of 1s and 0s used by computers to represent information.

2 00010001

3 00001110
00100101
00111011

4 To represent negative numbers using binary.

5 0001 1000 = 18

6 6.5

Chapter 12: Data structures

Learning objectives

- Understand the principles of using arrays of up to three dimensions.
- Understand the principles of using records, lists and tuples.
- Understand the principles of using linked lists, graphs (directed and undirected), stacks, queues, trees, binary search trees and hash tables.
- Know how to create, traverse, add data to and remove data from the data structures mentioned above.

What your students need to know

- **Data structures** bring together related bits of **data** in a variety of forms, the use of which depends on application and memory requirements of a particular task. They can be either static (the size of the structure doesn't change throughout the program run) or **dynamic** (the size might change through the run of the program). While more economical with space, dynamic structures are harder to program and are less predictable (again, a challenge for a beginning programmer).
- The most common structures are one- or two-dimensional (2D) arrays, linked lists, stacks, queues, binary trees, directed/undirected graphs and hash tables.
- Complex data structures are often just nested simple data structures arranged one inside the other. Binary trees are implemented with linked lists, while stacks can be implemented in an array (or natively supported by a central processing unit on a hardware level).
- Most data structures share the same operations – inserting, deleting, sorting and searching for the elements inside these structures. The efficiency of these operators depends on the good match between the structure and the task.
- One-dimensional arrays (or lists in Python) were covered at GCSE, with each variable inside an array called an element and each element's position in the array known as its index. Arrays make it easy to change or assess the data inside them. Each element can be directly referred to by its index (e.g. Names[6]).
- Two-dimensional arrays are like tables, with rows and columns, and they have a double index, one for row, the other for column (e.g. Names[6][3]).
- In most languages, arrays are static; their size is predefined at the beginning of the program. So, adding new data is accomplished via finding an empty index and putting data there – which involves searching. Deletion is likewise accomplished via searching for the required element and then setting its value to null. Of course, in Python, lists/arrays are actually collections of objects and are dynamic, and data can be either inserted at any index, or appended at the end of the list.
- Students need to know either **pseudocode** or actual code snippets for inserting new data into an array, both of one and two dimensions.
- **Big O notation** describes the changes in the performance (speed) of an algorithm, like sorting or searching, as the volume of data to process increases.
- Big O notation assumes the worst case scenario, for example, that the search item is located at the last list element or that elements are not pre-sorted, so searching can't use any of the shortcuts that are available when material is somewhat sorted.
- A process that takes the same time to complete for any data size is said to be O(1), such as reading the first element of an array. If the length of time to complete this process/algorithm is directly proportional to the data size, it is said to be in O(n), such as a simple serial search that looks at every element. Sometimes, the length of time the process takes is proportional to the square of the amount of data – O(n^2) – which is common for nested loops.

- All algorithms and data structures have a known Big O efficiency rating, just like all housing in the UK is rated for electrical efficiency. The Big O rating is needed to choose the best matching data structure and algorithm for the job.
- Linked lists are very common and very dynamic, acquiring new 'nodes', which is how we call its elements. Each node contains data and is linked to both the previously added node and the node added after this one via pointers. Like clues during a treasure hunt, pointers indicate which node is next and which is the previous node. Students need to know the algorithm for adding and removing data to a linked list.
- Stacks are Last-In, First-Out (LIFO) – the last added (pushed) element is the first to be taken off (popped). They can be implemented either with arrays or with linked lists.
- Queues are First-In, First-Out (FIFO) structures. Elements can only be processed in the order that they were added to the queue. They are easily implemented using an array and two variables for the front and back of the queue (which happens to be the next free space). Students need to know both algorithms and should be able to actually code a solution that implements a queue.
- Binary trees are composed of nodes (called leaves) and links between them (called branches). In a binary tree, each node has two child nodes. Binary search trees are a special case of binary trees where the data on the left branch of a node must be less than the data in the node, while the data on the right must be greater – so, that they are sorted. Binary trees are multidimensional, so there are more ways to iterate or traverse through a tree like that. The type of a traversal (running through a tree) will determine the order of the results.
- A subtree is a tree that is actually a part of a larger tree
    - Pre-order traversal (depth-first search): Start at the root node, traverse the left subtree, and then traverse the right subtree.
    - In-order traversal: Traverse the left subtree, then visit the root node, and then traverse the right subtree.
    - Post-order traversal: Traverse the left subtree, then traverse the right subtree, and then visit the root node.
- They are implemented using an array or a linked list (better as it is dynamic and node-based, too).
- In addition to traversals, students also need to know how to implement a tree with a 2D array, as well as the code for adding and removing nodes to and from a binary tree. Special cases with a single branch and single leaf should also be considered.
- Graphs consist of two sets of data: vertices (corners or data points) and edges (sides/connectors). The edge set is all links between vertices that are in a set. If two vertices are connected by an edge, we say they are adjacent.
- In undirected graphs, the edges go both ways, usually drawn using straight lines without arrow heads. In directed graphs, the edges have a direction, shown by arrows on the graph. Two vertices are said to be adjacent only if their edge is declared. The order of the edges indicates the direction of the link.
- Graphs are implemented with linked lists or arrays. Students need to know a way of implementing graphs using a 2D array, including removing vertices and edges.

- **Hash tables** contain a table with data and a mapping function which decides which data is moved into which 'cell'. The hashing function tries to take an attribute of a data element, such as the length of a string entered, and perform a modular division on it by the size of the table, with the remainder generating this string's index number. While this allows for faster placement/storage of elements, the hashing function is not very aware of the nature of the data it stores, so it is possible that multiple elements will get assigned the same address/place; this is called 'data collision'. Separate chaining is used to overcome that at the expense of slowing down; it is where linked lists are implemented inside the hash table to add nodes that were double-booked.

Common misconceptions

- Students often confuse stacks with queues, expecting both to behave like arrays, which they can't.
- Students can forget how to calculate the Big O rating of an algorithm.
- They will often mix up the traversal algorithms, forgetting how binary search trees are sorted.
- You should remind students that hash tables are not indices.

Lesson activity suggestions

This is a very hands-on topic, so practical exercises are the most important way to support the learning. They will mostly revolve around solving exam-type questions with pen and paper, and programming data structures on a computer.

When programming on a computer, rather than try to abstract data for the structures, it makes sense to make them relevant. For example, the program implementing a queue could be a system that handles customers at IKEA waiting to return products – students will relate to this experience; it involves entering a new customer into a queue and then removing them when they are served. Similar ideas are foreign passport appointments, landing patterns for aeroplanes, etc.

A stack can be made topical by storing a list of actions that are undone; for example, repairing a laptop requires a lot of disassembly and it is hard to remember the sequence of actions. A program could be made that keeps track of all the actions; for example, remove power supply, undo the screws holding the keyboard ribbon cable, etc. Then, when the laptop is being put back together, the program will return all actions but in the opposite order.

A linked list program could revolve around a secret society the members of which started to disappear (or turned into zombies) and for which the links need to be restored – the link that pointed previously at the node 'treasurer' will have to point at somebody else now.

A binary tree could have data about a Dungeons and Dragons-type role playing game where, depending on your previous decision (e.g. to enter a room or to challenge a dragon), you get two additional choices.

Last but not least, an array is a shopping list – items are added and removed in no particular order as a person moves around the store and runs into the items on the list.

Topic	Lesson suggestions	Follow-up ideas/Homework
All topics	Look at some past paper questions. Implement data structures in actual programs in students' choice of programming language. Practise more past paper questions.	Extensions of programs can be assigned as homework – better interface, storage to CSV files.
Big O notation	**Starter:** Present students with this animation which shows multiple sorting algorithms. Ask students why a slower algorithm would ever be used. **Main activity:** Implement a bubble sort of students by height, first with only half of the class taking part and the sorting timed, then with the whole class and the sorting timed. If the time increases proportionally more than one to one, we can calculate O(n). **Plenary:** Discuss whether or not Moore's law is making Big O ranking of algorithms less relevant.	Using a timestamp feature in a programming language (time.time() in Python, for example), implement a sorting algorithm, saving timestamps to beginning and end variables, respectively. This way, we can find out how long a procedure took with different sets of data and even get its Big O rating.
Stacks and queues	**Starter:** Present students with the following paired scenarios: A waiting list versus a queue at the canteen. A stack of unwashed dishes versus re-parking three cars on a narrow driveway that is only one car wide. Ask students why they think these scenarios were paired; what have they got in common? **Main activity:** Continuing work with the definitions of stacks and queues, get students to brainstorm how they could act out the algorithms for adding a new item to a stack and queue using themselves as data elements (queue is obviously the easiest and the most common sense to implement). **Plenary:** Discuss which of these data structures requires less processing power and why.	For queues, write a program that could run in a self-help kiosk at a custom service centre. This program would keep track of people in a queue by allowing users to receive waiting list numbers; the program will process these numbers at random time intervals and then retire the numbers that have been 'served'. For stacks, write a program that can accept the user's input of a formula involving brackets and parse this input to execute the formula, for example, (4+6)*3.

End-of-chapter answers

1 Big O notation.

2 A collection of variables, all of the same type stored under a single identifier.

3 Data and a pointer to the next element in the list.

4 To represent negative numbers using binary.

5 Queue

6 Edges and vertices.

Chapter 13: Boolean algebra

Learning objectives

- Understand how to read and write Boolean algebra.
- Learn how to define problems using Boolean algebra.
- Understand propositional logic and the related key terms.
- Derive or simplify Boolean algebra using the following rules:
  - de Morgan's laws
  - distribution
  - association
  - commutation
  - double negation.

What your students need to know

- **Propositional logic** is defined in terms of true and false and follows mathematical rules. Most of the time a proposition could state that A > B (can be either True or False), while another could state B > C (again, can be either True of False). If both of them are true, we can say that A > C must be True.
- We label propositions with capital letters, like Q or P, and use the connective symbols for AND (conjugation ∧), OR (disjunction ∨), NOT (negation ¬), IF (implication →) and equality (biconditional equivalence ↔). Using this we evaluate usually two input propositions (one for NOT) and can generate an output proposition. Each of these comes with a table of possibilities that will need to be practised to perfection.
- Conjugation (AND) requires that both input propositions are True. If either one is false, the output is False. It is equivalent to a simple multiplication; for example, True AND False = False is the same as 1 × 0 = 0, if we hold True to be 1 and False to be 0.
- Disjunction (OR) requires that at least one of the input propositions is true and is equivalent to a simple addition; for example, True OR False = True is the same as 1 + 0 = 1, if we hold True to be 1 and False to be 0.
- Negation is a unary operator that works only on one proposition.
- A **tautological proposition** is the one that outputs True regardless of what truth values of input propositions are, for example X AND NOT(X) (x∨¬x).
- Biconditional equivalence means 'if and only if' both sides (input propositions) are True the output is True.
- Using biconditional equivalence we can prove that some propositions are tautological and can be omitted and the logical chains can be shortened as a result.
- Logical laws of deduction include commutation – the order of propositions for both conjunction and disjunction do not matter and they can be interchanged and are biconditional; association – the statements in brackets are evaluated first; distribution – we can move letters inside the brackets and outside the brackets, provided we do it the right way (the order of brackets matters), for example, T AND (P OR Q) = (T AND P) OR (T AND Q).
- Double negation is the same (tautological) as no negation. With NOT, just as -(-2) is 2, NOT(NOT P) ↔ P.
- **de Morgan's law** states that the negation of disjunctions is the conjunction of the negations (NOT(P OR Q) = NOT(P) AND NOT(Q)) and that the negation of conjunction is the disjunction of the negations (NOT(P AND Q) = NOT(P) OR NOT(Q)). This is significant because it allows us to transform AND into OR and vice versa for the purpose of simplifying long logical chains.
- Plain English propositions can be converted into the language of Boolean algebra by paying attention to 'command words' (AND, OR, must, could, all, some, etc.) inside these propositions.

- When you get a situation where you can reverse the implication and still have a true statement, we have a bidirectional condition.
- You can use Boolean algebra to simplify complex conditions for control statements such as WHILE loops or IF statements.

Common misconceptions

- Students often think that algebra must be about numbers.
- Students often think that de Morgan's laws are difficult – they are not; they are just two concepts similar to a basic maths distribution principle that students would already know.
- Students can think that logic is not for everybody and requires a certain mind – anybody can do logic chains, knowing the rules of Boolean operations explained in this chapter.
- Students will often confuse the terminology from Boolean algebra – true, these are not common terms but with practice they make sense.

Lesson activity suggestions

A good lesson starter is to have a look at the famous book *The Lady, or the Tiger?* by Frank R. Stockton. Students could also look for the puzzle known as 'the hardest logic puzzle ever'.

Get your students to research a crime short story; for example, a Sherlock Holmes story or one by Agatha Christie. They only need to look at the last few pages where the inspector/detective is explaining what happened and then rephrase their words into propositional logic.

Students should also work on solving lots of problems and answering past paper questions. This is a very hands-on topic that requires practice, precision and instant remembering of relevant symbols and terms.

Topic	Lesson suggestions	Follow-up ideas/Homework
How to define problems using Boolean algebra	**Starter:** A good lesson starter is to have a look at the famous book *The Lady, or the Tiger?* by Frank R. Stockton. **Main activity:** In the court of law, it is important to establish the guilt through finding evidence of motive and means to commit a crime. Ask students to come up with a list of cases where neither, both or either of these conditions is met to illustrate AND and NOT. Ask students to follow a similar line of reasoning to illustrate OR. What life situation would help to describe OR? **Plenary:** Discuss the following: What is common between Boolean OR and arithmetic addition? Boolean AND and arithmetic multiplication?	The students can research a very important person to the development of popular logic thinking: Raymond Merrill Smullyan and his collections of logical puzzles, as well as his connection to Alan Turing (he studied under Alonzo Church who was working in the same direction as Turing).

Topic	Lesson suggestions	Follow-up ideas/Homework
To derive or simplify Boolean algebra	**Starter:** Ask students to come up with the most difficult Boolean expression under 20 characters. Then the class can identify the three most difficult of those submitted. **Main activity:** Ask students to do exercises that illustrate de Morgan's laws. **Plenary:** Discuss what the practical application of de Morgan's laws is.	Practise past examination questions.

End-of-chapter answers

1 De Morgan's law states that the negation of a conjunction is the disjunction of the negated items. ¬ (P∧Q) is the negation of a conjunction. So if P and Q were both true, then the final result would be false. This is equivalent to ¬ P ∨ ¬ Q as ¬ P is false and ¬ Q is also false. The disjunction of two false values is also false. The truth table below shows their equivalence.

P	Q	*¬ (P∧Q)*	*¬ P ∨ ¬ Q*
0	0	1	1
1	0	1	1
0	1	1	1
1	1	0	0

2
a. Conjunction uses the ∧ symbol and both elements must be true in order for the final value to be true.
b. Disjunction uses the ∨ symbol and either elements can be true for the whole statement to be true.
c. Implication uses the → symbol and means that if the equation on the left is true then the equation on the right must also be true.
d. Negation uses the ¬ symbol and will swap a truth value around.

3 ¬ (A∨¬ B)≡ ¬ A ∧ ¬ ¬ B de Morgan's law.
¬ A ∧ ¬ ¬ B≡¬ A ∧ B double negation.
¬ A ∧ B≡ B ∧¬ A law of commutation.

4
a. ¬ P→¬ T
b. P∧Q→T
c. Q∧P∧R→¬ T

5

P	Q	*P→Q*	*P↔Q*
0	0	1	1
1	0	0	0
0	1	1	0
1	1	1	1

Chapter 14: Legal and ethical issues

Learning objectives

- To learn about computer-related laws.
- To gain an awareness of the ethical issues raised by the use of computers.

What your students need to know

- The use of computers raises a number of ethical issues, including their use in the workplace, artificial intelligence and automated decision making, environmental effects and censorship.
- A lot of modern crime, such as **software piracy** or **phishing** attacks, didn't exist even decades ago.
- Combating **cybercrime** requires a lot of very specific knowledge, made difficult by development of computers and the communities that use them. Computers are changing faster than laws can be drafted.
- Modern technology allows enormous amounts of data on third parties to be stored and transferred around the world, waiting to be stolen or used in a way that erodes other parties' privacy. This is especially true with cloud computing, where more data is stored online, rather than on disks locally. The Data Protection Act (1998) gives individuals the right to know what data is stored about them and who is storing it, and the right to see any data a company is storing about them and have it amended if necessary. This Act also defines two information categories: personal data (widely available information on a person, such as their address) and sensitive data that by default you wouldn't want to be available (e.g. medical history, ethnic origin).
- The Act also sets forward eight principles of data protection that apply to any data stored. These include legitimate grounds for collective information (so, 'for blackmail' doesn't count), and it can't be used to hurt you. Whoever collects the data must explain in advance how they will use the data and give individuals notice that their data is being collected. There is also a responsibility to safeguard data, so it can't be used by somebody else (e.g. a hacker or a hostile government) to do anything unlawful.
- Collecting more data than necessary and claimed in advance is not lawful under the Act, it needs to be accurate, kept up to date, safe from early deletion, and discarded securely when it is no longer needed for the purpose for which it was originally collected.
- The person has a right to access a copy of information stored on them, and to object if it will cause them damage or distress, or if it's going to be used for direct marketing. Otherwise, if this right is denied, the collector of data will have to pay compensation set by the Act.
- Last but not least, personal data can't be sent to a country outside the European Economic Area unless that country or territory ensures an adequate level of protection for the rights and freedoms of data subjects in relation to the processing of personal data. As recent cases in the United States demonstrate, this is not always easy to enforce, so by default, companies should avoid sending data to countries not in the European Economic Area. Foreign governments can enact laws that override the Data Protection Act commitments, and this is a problem, considering how much of the world's data is stored on US servers, in particular.
- The Information Commissioner is the person whom parliament has empowered to enforce the Act. The Act establishes the following terms: the 'data controller' – a person or company that collects and keeps data about data subjects (people); and the 'data subject' – someone who has data about them stored by a third party.
- The Computer Misuse Act (1990) prohibits using a computer to secure access to any program or data held electronically if the access is unauthorised and this is known by the perpetuator at the time, especially if this is with intent to commit or facilitate commission of further offences.

- It is also illegal to create, adapt, supply or offer to supply any materials with the intention that they be used to commit a crime, including doing that with a view to passing them on for somebody else to commit the crime.
- The Copyright, Designs and Patents Act (1988) states that creators of artistic and creative works have the right to choose how their material is used. This includes use in application software, games, films, music, dramatic productions, sculptures and books.
- **Copyright** is an automatic right from the moment of creation of a work. This means that there is no need to put a copyright symbol on your work and it is valid for 70 years after the creator dies for literary, dramatic, musical and artistic works and films and 50 years from when the work was first released for sound recordings.
- There are exceptions to the Copyright, Designs and Patents Act (1988) that allow using copyrighted material without permission. This is for educational purposes where no profit is being made, to criticise or report on them, so that libraries can copy and lend with a special licence, for recording a broadcast for viewing/listening to later, for creating backups for personal use, to play recordings in a club or society and to copy a program to study and test to learn how it works.
- Breaches of this Act are punishable by up to two years in prison and a fine, including any damages caused.
- The Regulation of Investigatory Powers Act (2000) is a controversial 'Big Brother' Act that particularly focuses on electronic surveillance, the legal decryption of encrypted data and the interception of electronic communications by the security services, the Secret Intelligence Service and the Government Communications Headquarters (GCHQ). Under the Act, they can intercept e-mails, access private communications and plant surveillance devices. There are reasons to believe that the Act is being used by the government agencies more often than originally intended (with a very low level of convictions made based on the information gathered).
- Students need to know specific examples for every principle of every Act mentioned in this chapter.
- Computer technology raises ethical issues. While they helped to cut costs, the introduction of computers resulted in higher unemployment and the movement of jobs overseas. However, many dangerous and repetitive jobs have been taken over by computers. Widespread production of computing technology and its fast obsolescence and replacement result in a very pollution-creating industry and one which uses up limited resources. It gives rise to exploitation of developing countries and even finances civil wars. Disposal of old computer equipment is also a big environmental issue that seems to hurt poor countries with inadequate environmental protection.
- **Artificial intelligence (AI)** aims to replace human decision making, such as with Google's driverless cars, but at the current state of technology it can cause problems, such as herd behaviour by stockmarket computers all following an instruction to sell a stock that hits a certain price – causing the whole stockmarket to crash as the volume of trade increases exponentially. It can also cause overreliance, such as cases of cars falling off cliffs or driving the wrong way on motorways following wrong or misinterpreted satellite navigation directions. The free and wide nature of the internet brings up issues of **censorship**. Schools censor the internet to avoid distractions to students. China censors the internet to restrict criticism of the government. Many countries restrict access to sites promoting software piracy and pornography. The optimal amount of censorship and its place in a society, as well as the ethics of enforcing it, are widely discussed issues and it is likely our opinions on what is or is not acceptable on the internet will change in the future.

Common misconceptions

- Students may think that cybercrime is punished lightly being non-violent crime.
- To avoid being cyberbullied, students shouldn't give out any information – but giving information out is not an agreement to having it used unlawfully.
- Students can stop third parties collecting information about them and they don't need to communicate the reasons why they don't want that data collected – in reality, they have to prove this will cause them objective distress or damage.
- Some may think that helping others to commit cybercrime will not make them liable, as long as it is on someone else's computer.
- Students may think that copyright needs to be registered – this is not true, it is automatic. They may also think that copyright is 25 years, like patents – again this is not true, it is much longer.
- Students may also have concerns over artificial intelligence from its mixed portrayal in popular culture.

Lesson activity suggestions

Lessons on the Acts could revolve around discussions of legal precedents and how much the students themselves are aware of the scope of some of their everyday activities. Students can be asked if it is acceptable for schools to photocopy textbook pages, or if taking notes from a textbook is breaching the author's copyright. Is it different for electronic textbooks?

Students can be asked to look up the difference between these two copyrights in music: 'c in the circle' versus 'p in the circle' (author's versus performer's copyright, respectively). How much is a school paying a copyright holder company to stage a school play like Guys and Dolls?

Talking of the jobs that have changed or disappeared/appeared because of the computing revolution is a good way to get started on the Computers and the workplace topic.

Topic	Lesson suggestions	Follow-up ideas/Homework
The Data Protection Act (1998)	**Starter:** Look at the wording of the Patriot Act (2001) and then look at the case of WikiLeaks. **Main activity:** Present the Data Protection Act's main articles backed up by case studies. **Plenary:** Ask students which of these Acts are easier to convict somebody on.	Research Snowden's revelations and write up a cause for suspending information sharing between the UK and US unless specifically authorised by the government. Alternatively, identify three other countries you would not want to send information to.
The Computer Misuse Act (1990)	**Starter:** Brainstorm a list of activities/scenarios that would breach this Act. **Main activity:** Present the Computer Misuse Act's main articles backed up by case studies. **Plenary:** Ask students which of these articles are easier to convict somebody on.	Research three cases of people prosecuted under this Act.

Topic	Lesson suggestions	Follow-up ideas/Homework
Computers and the workplace and Copyright Law	**Starter:** Discuss with the class how computers have changed the job of a teacher over the past decade? Has anything changed since students have been in the school? **Main activity:** Conduct a class discussion resulting in the compiling of a list of ten jobs that have been replaced by computers and ten new jobs that have been created over the last 20 years as a result of the introduction of computers. **Plenary:** Discuss the rationale behind 'Project Gutenberg'.	Research a list of books that have recently gone out of copyright.
Artificial intelligence	**Starter:** Discuss Isaac Asimov's laws for AI – can they prevent computers harming people? **Main activity:** As a class, brainstorm modifications needed for our roads before we can have driverless cars. **Plenary:** Discuss if there will be things that will never be replaced by artificial intelligence. Look at excerpts of Asimov's I, Robot.	Research modifications needed for our roads before we can have driverless cars.

End-of-chapter answers

1 Personal data shall be processed fairly and lawfully.
Personal data shall be obtained only for lawful purposes.
Personal data shall be adequate, relevant and not excessive.
Personal data shall be accurate and, where necessary, kept up-to-date.
Personal data shall not be kept for longer than is necessary.
Personal data shall be processed in accordance with the rights of data subjects.
Data must be kept secure.
Personal data shall not be transferred outside the EEA.

2 To unlawfully access computer systems or modify computer material.

3 Regulation of Investigatory Powers Act 2000.

4 The Copyright, Designs and Patents Act 1988.

5 The Information Commissioner.