



AQA AS and A Level

# Computer Science



PG ONLINE

PM Heathcote and  
RSU Heathcote

# Contents

## Section 1

<b>Fundamentals of programming</b>	<b>1</b>
<b>Chapter 1</b> Programming basics	<b>2</b>
<b>Chapter 2</b> Selection	<b>8</b>
<b>Chapter 3</b> Iteration	<b>13</b>
<b>Chapter 4</b> Arrays	<b>17</b>
<b>Chapter 5</b> Subroutines	<b>21</b>
<b>Chapter 6</b> Files and exception handling	<b>29</b>

## Section 2

<b>Problem solving and theory of computation</b>	<b>33</b>
<b>Chapter 7</b> Solving logic problems	<b>34</b>
<b>Chapter 8</b> Structured programming	<b>39</b>
<b>Chapter 9</b> Writing and interpreting algorithms	<b>42</b>
<b>Chapter 10</b> Testing and evaluation	<b>48</b>
<b>Chapter 11</b> Abstraction and automation	<b>52</b>
<b>Chapter 12</b> Finite state machines	<b>60</b>

## Section 3

<b>Data representation</b>	<b>67</b>
<b>Chapter 13</b> Number systems	<b>68</b>
<b>Chapter 14</b> Bits, bytes and binary	<b>72</b>
<b>Chapter 15</b> Binary arithmetic and the representation of fractions	<b>77</b>
<b>Chapter 16</b> Bitmapped graphics	<b>83</b>
<b>Chapter 17</b> Digital representation of sound	<b>88</b>
<b>Chapter 18</b> Data compression and encryption algorithms	<b>93</b>

## Section 4

<b>Hardware and software</b>	<b>99</b>
<b>Chapter 19</b> Hardware and software	<b>100</b>
<b>Chapter 20</b> Role of an operating system	<b>103</b>
<b>Chapter 21</b> Programming language classification	<b>106</b>
<b>Chapter 22</b> Programming language translators	<b>110</b>
<b>Chapter 23</b> Logic gates	<b>114</b>
<b>Chapter 24</b> Boolean algebra	<b>118</b>

## Section 5

<b>Computer organisation and architecture</b>	<b>125</b>
<b>Chapter 25</b> Internal computer hardware	<b>126</b>
<b>Chapter 26</b> The processor	<b>132</b>
<b>Chapter 27</b> The processor instruction set	<b>138</b>
<b>Chapter 28</b> Assembly language	<b>142</b>
<b>Chapter 29</b> Input-output devices	<b>148</b>
<b>Chapter 30</b> Secondary storage devices	<b>154</b>

## Section 6

<b>Communication: technology and consequences</b>	<b>158</b>
<b>Chapter 31</b> Communication methods	<b>159</b>
<b>Chapter 32</b> Network topology	<b>164</b>
<b>Chapter 33</b> Client-server and peer-to-peer	<b>168</b>
<b>Chapter 34</b> Wireless networking, CSMA and SSID	<b>171</b>
<b>Chapter 35</b> Communication and privacy	<b>176</b>
<b>Chapter 36</b> The challenges of the digital age	<b>179</b>

## Section 7

### Data structures

**187**

<b>Chapter 37</b>	Queues	188
<b>Chapter 38</b>	Lists	194
<b>Chapter 39</b>	Stacks	198
<b>Chapter 40</b>	Hash tables and dictionaries	202
<b>Chapter 41</b>	Graphs	207
<b>Chapter 42</b>	Trees	211
<b>Chapter 43</b>	Vectors	217

## Section 8

### Algorithms

**223**

<b>Chapter 44</b>	Recursive algorithms	224
<b>Chapter 45</b>	Big-O notation	229
<b>Chapter 46</b>	Searching and sorting	235
<b>Chapter 47</b>	Graph-traversal algorithms	243
<b>Chapter 48</b>	Optimisation algorithms	249
<b>Chapter 49</b>	Limits of computation	254

## Section 9

### Regular languages

**259**

<b>Chapter 50</b>	Mealy machines	260
<b>Chapter 51</b>	Sets	265
<b>Chapter 52</b>	Regular expressions	269
<b>Chapter 53</b>	The Turing machine	273
<b>Chapter 54</b>	Backus-Naur Form	278
<b>Chapter 55</b>	Reverse Polish notation	283

## Section 10

<b>The Internet</b>	<b>287</b>
<b>Chapter 56</b> Structure of the Internet	288
<b>Chapter 57</b> Packet switching and routers	292
<b>Chapter 58</b> Internet security	294
<b>Chapter 59</b> TCP/IP, standard application layer protocols	300
<b>Chapter 60</b> IP addresses	307
<b>Chapter 61</b> Client server model	313

## Section 11

<b>Databases and software development</b>	<b>318</b>
<b>Chapter 62</b> Entity relationship modelling	319
<b>Chapter 63</b> Relational databases and normalisation	323
<b>Chapter 64</b> Introduction to SQL	330
<b>Chapter 65</b> Defining and updating tables using SQL	336
<b>Chapter 66</b> Systematic approach to problem solving	342

## Section 12

<b>OOP and functional programming</b>	<b>346</b>
<b>Chapter 67</b> Basic concepts of object-oriented programming	347
<b>Chapter 68</b> Object-oriented design principles	353
<b>Chapter 69</b> Functional programming	360
<b>Chapter 70</b> Function application	367
<b>Chapter 71</b> Lists in functional programming	371
<b>Chapter 72</b> Big Data	374
<b>References</b>	379

## Appendices and Index

<b>Appendix A</b> Floating point form	380
<b>Appendix B</b> Adders and D-type flip-flops	387
<b>Index</b>	391

# Chapter 1 – Programming basics

## Objectives


- Define what is meant by an algorithm and pseudocode
- Learn how and when different data types are used
- Learn the basic arithmetic operations available in a typical programming language
- Become familiar with basic string handling operations
- Distinguish between variables and constants

## What is an algorithm?

An algorithm is a set of rules or a sequence of steps specifying how to solve a problem. A recipe for chocolate cake, a knitting pattern for a sweater or a set of directions to get from A to B, are all algorithms of a kind. Each of them has **input**, **processing** and **output**. We will be looking in more detail at properties of algorithms in Section 2 of this book.

**Q1:** What are the inputs and outputs in a recipe, a knitting pattern and a set of directions?

Ingredients	Method
100g plain flour	Put flour and salt into a large mixing bowl and make a well in the centre.
2 eggs	Crack the eggs into the middle.
300ml milk	Pour in about 50ml milk and the oil.
1tbsp oil	Start whisking from the centre, gradually drawing the flour into the eggs, milk and oil, etc.
Pinch salt	



In the context of programming, the series of steps has to be written in such a way that it can be translated into program code which is then translated into machine code and executed by the computer.

## Using pseudocode

Whatever programming language you are using in your practical work, as your programs get more complicated you will need some way of working out what the steps are before you sit down at the computer to type in the program code. A useful tool for developing algorithms is **pseudocode**, which is a sort of halfway house between English and program statements. There are no concrete rules or syntax for how pseudocode has to be written, and there are different ways of writing most statements. We will use a standard way of writing pseudocode that translates easily into a programming language such as Python, Pascal or whatever procedural language you are learning.

This book does not teach you how to program in any particular programming language – you will learn how to write programs in your practical sessions – but it will help you to understand and develop your own algorithms to solve problems.

Example 3 is a classic logic problem, which has many different variations on the same theme.

**Q3:** A man has to get a fox, a chicken, and a sack of corn across a river. He has a rowing boat, which can carry only him and one other thing. If the fox and the chicken are left together, the fox will eat the chicken. If the chicken and the corn are left together, the chicken will eat the corn. How does the man do it?

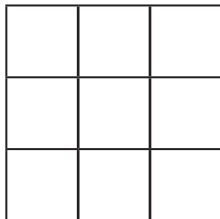


## Strategies for problem solving

There are some general strategies for designing algorithms which are useful for solving many problems in computer science. First of all it is useful to note that there are two types of algorithmic puzzle. Every puzzle has an **input**, which defines an **instance** of the puzzle. The instance can be either **specific** (e.g. fill a magic square with 3 rows and 3 columns), or **general** ( $n$  rows and  $n$  columns). Even when given a general instance of a problem, it is often helpful to solve a specific instance of it, which may give an insight into solving a more general case.

### Exhaustive search

For example, suppose you are asked to fill a ‘magic square’ with 3 rows and 3 columns with distinct integers 1-9 so that the sum of the numbers in each row, column and corner-to-corner diagonal is the same.



This is a **specific** instance of a more **general** problem in which there are  $n$  rows and  $n$  columns. Some problems can be solved by **exhaustive search** – in this example, by trying every possible combination of numbers. We can put any one of 9 integers in the first square, and any of the remaining 8 in the second square, giving  $9 \times 8 = 72$  possibilities for just the first two squares. There are  $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 = 362,880$  ways of filling the square. If you are a mathematician you will know that this is denoted by  $9!$ , spoken as “nine factorial”.

You might think a computer could do this in a fraction of a second. However, looking at the more general problem, where you have  $n \times n$  squares, you will find that even for a  $5 \times 5$  square, there are so many different combinations ( $25!$  or 25 factorial) that it would take a computer performing 10 trillion operations a second, about 49,000 years to find the answer!

So, to solve this problem we need to come up with a better algorithm. It turns out to be not very difficult to work out that for a  $3 \times 3$  square, each row, column and diagonal must add up to 15 and the middle number must be 5, which considerably reduces the size of the problem. (The details of the algorithm are not discussed here.)

**Q4:** Fill the magic square to solve the problem.

## Exercises

1. Figure 2 shows the state transition diagram of a finite state machine (FSM) used to control a vending machine.

The vending machine dispenses a drink when a customer has inserted exactly 50 pence.

A transaction is cancelled and coins returned to the customer if more than 50 pence is inserted or the reject button (R) is pressed. The vending machine accepts 10, 20 and 50 pence coins. Only one type of drink is available.

The only acceptable inputs for the FSM are 10, 20, 50 and R.

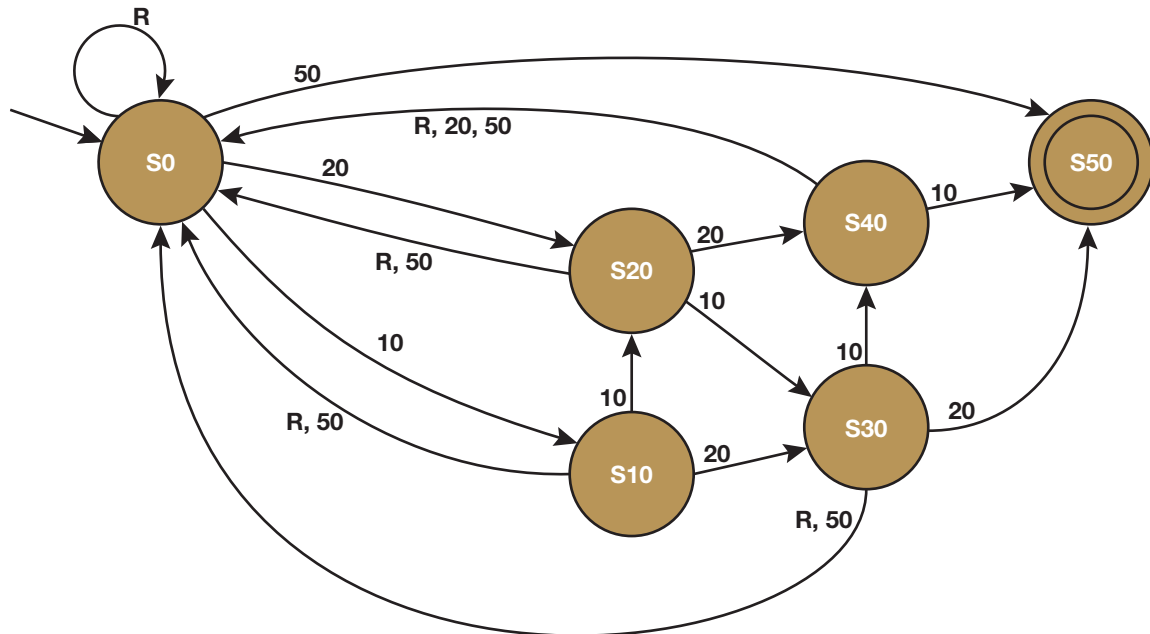


Figure 2

An FSM can be represented as a state transition diagram or as a state transition table. Table 2 is an incomplete state transition table for part of Figure 2.

- (a) Complete the missing sections of the four rows of Table 2.

Original state	Input	New state
S0	10	S10
S0		
S0		
S0		

Table 2

[3]

There are different ways that a customer can provide **exactly three** inputs that will result in the vending machine dispensing a drink. Three possible permutations are “20, 10, 20”, “10, R, 50” and “10, 50, 50”.

- (b) List **four** other possible permutations of **exactly three** inputs that will be accepted by the FSM shown in Figure 2.

[4]

AQA Comp1 Qu 4 June 2012



# Chapter 17 – Digital representation of sound

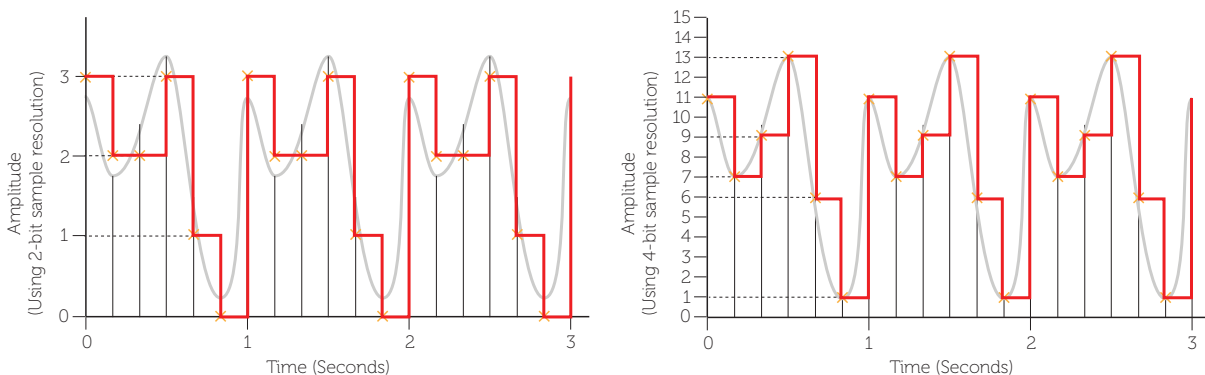
## Objectives

- Describe the digital representation of sound in terms of sampling rate and resolution
- Describe the principles of operation of an analogue to digital converter and a digital to analogue converter
- Understand and apply the Nyquist theorem
- Calculate sound sample sizes in bytes
- Describe the purpose of MIDI and the use of event messages
- Describe the advantages of using MIDI files for representing music

## Sound sampling and resolution

Sound waves are naturally in a continuous, analogue form. To represent sound in a computer, the (**continuous**) analogue sound waves have to be converted to a (**discrete**) digital format. This can be done by measuring and recording the amplitude of the sound wave at given time intervals (several thousand times per second). The more frequently the samples are taken, the more accurately the sound will be represented. The frequency at which samples are taken is measured in **hertz (Hz)**, a unit of frequency equal to one cycle per second.

In addition, in the same way that an image's quality is improved with a more precise representation of colour enabled by a greater colour depth, the accuracy of a sound recording increases with a greater audio bit depth. Increasing the number of points of amplitude (represented on the y axis below) increases the accuracy at which you can record a sound's amplitude (or wave height) at a given point in time.



**Q1:** Which of the graphs above represents a more accurate recording? Why?

## Sample rate

The **sampling rate** is the frequency with which you record the amplitude of the sound. The more often you take a sample, the smoother the playback will sound. The disadvantage of this, is that every time you take a sample, at a resolution of say 16 bits, you need to store another 2 bytes of data. A typical CD recording is made at 44,100Hz, or 44,100 times per second. This means that for every second of sound,  $2 \text{ bytes} \times 44,100 = 88,200$  bytes is required and for every minute, approximately 5.3MB is required. For stereo sound, this is doubled to provide samples for left and right channels.

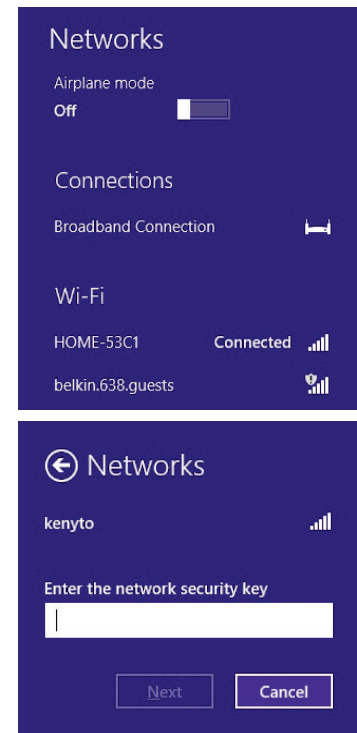
## Securing a wireless network

**Wi-Fi Protected Access (WPA)** and **Wi-Fi Protected Access II (WPA2)**, which has replaced it, are two security protocols and security certification programs used to secure wireless networks. WPA2 is built into wireless network interface cards, and provides strong encryption of data transmissions, with a new 128-bit key being generated for each packet sent.

Each wireless network has a Service Set Identification (SSID) which is the informal name of the local network – for example, HOME-53C1. The purpose of the SSID is to identify the network, and if, for example, you visit someone else’s house with a laptop and wish to connect to their Wi-Fi network in order to use the Internet, when you try to log on to the Internet the computer will ask you to enter the name of the network.

Your computer may be within the range of several networks, so having chosen the correct SSID you will then be asked for the password or security key - an identifier of up to 32 bytes, usually a human-readable string. SSIDs must be locally unique.

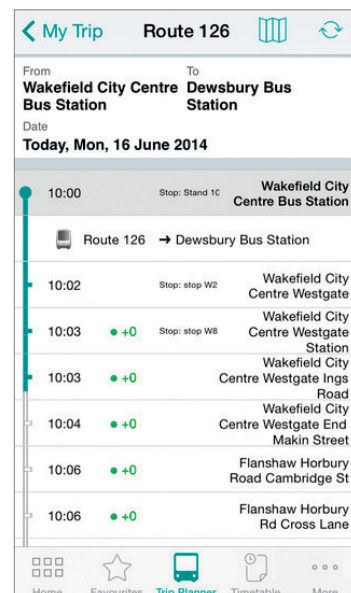
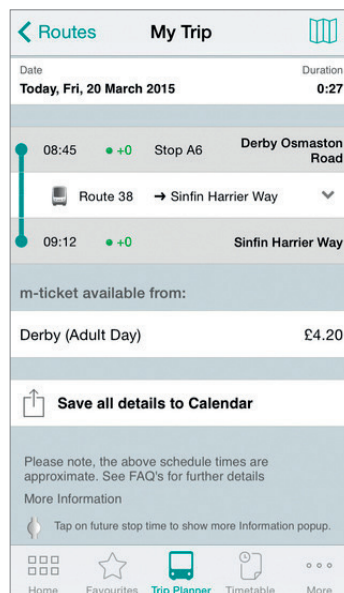
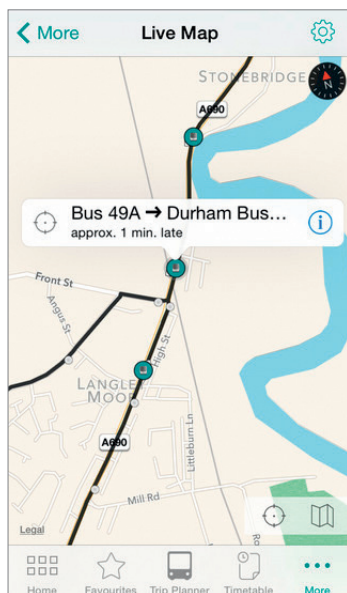
It is possible to disable the broadcast of your SSID to hide your network from others looking to connect to a named local network. However, this will not hide your network completely.



## Whitelists

Some network administrators set up **MAC address whitelists** (the opposite of blacklists) to control who is allowed on their networks. (The MAC address is a unique identifier assigned to a network interface card by the manufacturer: see page 167.)

**Q1:** Research some of the applications of “location-based services” such as *Presence Orb*. What are some of the benefits and some of the drawbacks to individuals of tracking software?



Arriva's Bus App

# Chapter 41 – Graphs

## Objectives

- Be aware of a graph as a data structure used to represent complex relationships
- Be familiar with typical uses for graphs
- Be able to explain the terms: graph, weighted graph, vertex/node, edge/arc, undirected graph, directed graph
- Know how an adjacency matrix and an adjacency list may be used to represent a graph
- Be able to compare the use of adjacency matrices and adjacency lists

## Definition of a graph

A graph is a set of **vertices** or **nodes** connected by **edges** or **arcs**. The edges may be one-way or two way. In an **undirected graph**, all edges are bidirectional. If the edges in a graph are all one-way, the graph is said to be a **directed graph** or **digraph**.

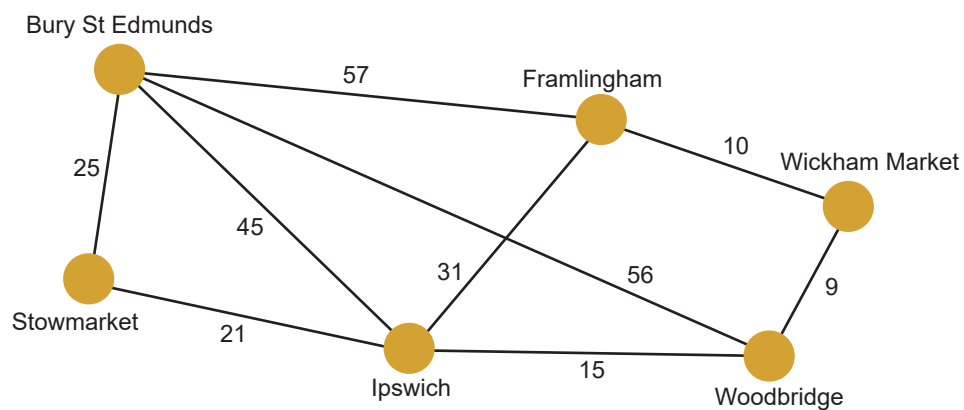


Figure 41.1: An undirected graph with weighted edges

The edges may be **weighted** to show there is a cost to go from one vertex to another as in Figure 41.1. The weights in this example represent distances between towns. A human driver can find their way from one town to another by following a map, but a computer needs to represent the information about distances and connections in a structured, numerical representation.

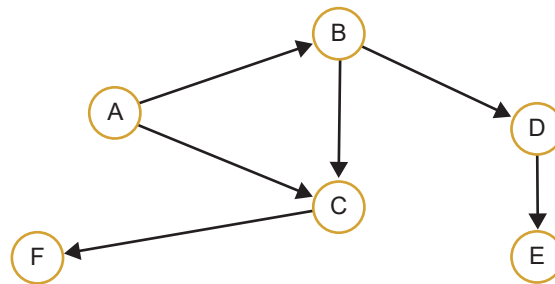
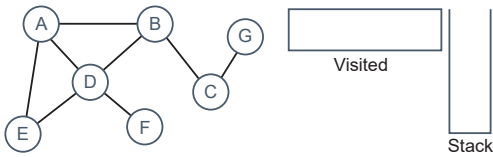
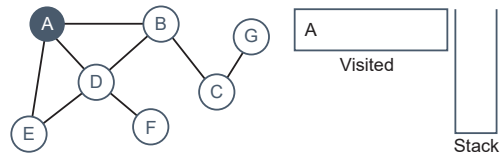


Figure 41.2: A directed, unweighted graph

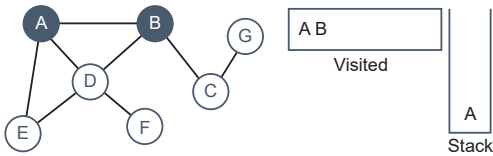
It is easiest to understand how this works by looking at the graphs below. This shows the state of the **stack** (here it just shows the current node when a recursive call is made), and the contents of the **visited** list. Each visited node is coloured dark blue.



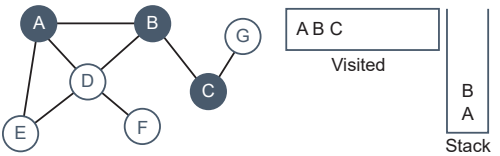
1. Start the routine with an empty stack and an empty list of visited nodes.



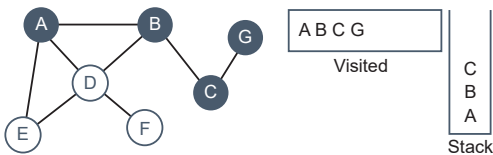
2. Visit A, add it to the visited list. Colour it to show it has been visited.



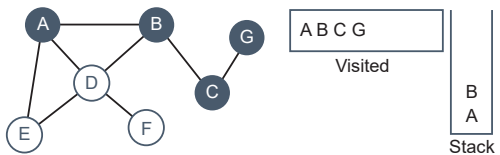
3. Push A onto the stack to keep track of where we have come from and visit A's first neighbour, B. Add it to the visited list. Colour it to show it has been visited.



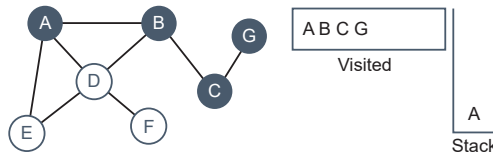
4. Push B onto the stack and from B, visit the next unvisited node, C. Add it to the visited list. Colour it to show it has been visited.



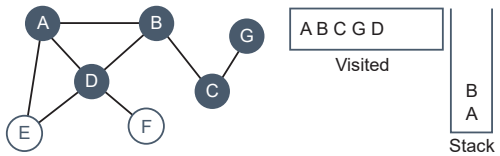
5. Push C onto the stack and from C, visit the next unvisited node, G. Add it to the visited list. Colour it to show it has been visited.



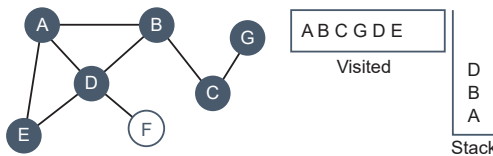
6. At G, there are no unvisited nodes so we backtrack. Pop the previous node C off the stack and return to C.



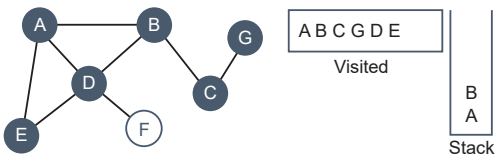
7. At C, all adjacent nodes have been visited, so backtrack again. Pop B off the stack and return to B.



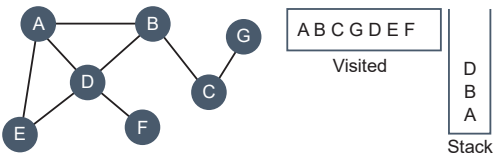
8. Push B back onto the stack to keep track of where we have come from and visit D. Add it to the visited list. Colour it to show it has been visited.



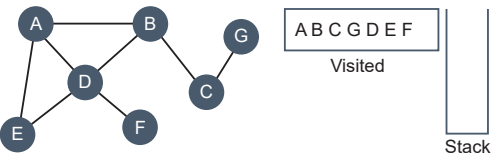
9. Push D onto the stack and visit E. Add it to the visited list. Colour it to show it has been visited.



10. From E, A and D have already been visited so pop D off the stack and return to D.



11. Push D back onto the stack and visit F. Add it to the visited list. Colour it to show it has been visited.



12. At F, there are no unvisited nodes so we pop D, then B, then A, whose neighbours have all been visited. The stack is now empty which means every node has been visited and the algorithm has completed.

## Regular language

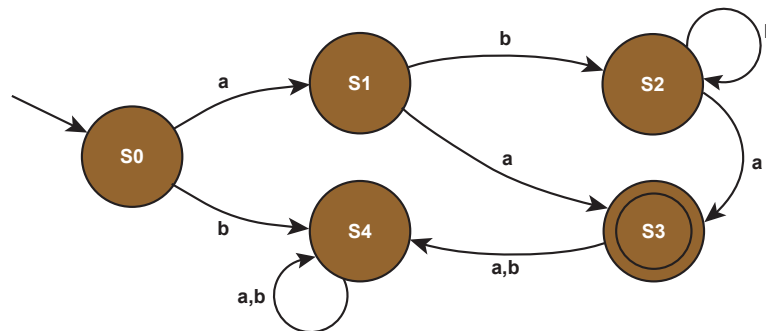
A language is called **regular** if it can be represented by a regular expression. A regular language can also be defined as any language that a **finite state machine** will accept. Any finite language (one containing only a finite number of words) is a regular language, since a regular expression can be created that is the union of every word in the language.

### Example 1

A regular language consists of all words beginning and ending in *a*, with zero or more instances of *b* in between, e.g. *aa*, *aba*, *abba*, *abbba*.

Write a regular expression that describes this language, and draw the corresponding finite state machine (FSM).

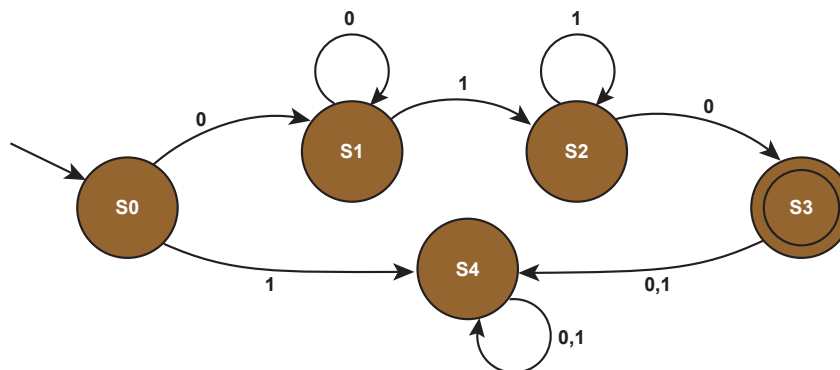
Answer:  $R = ab^*a$ . Note that the FSM is drawn with an outgoing transition from every state for every possible input symbol.



### Example 2

Describe the set of strings found by  $0^+1^+0$  and draw the FSM.

Answer: It would find all strings with one or more zeros followed by one or more ones followed by one zero. e.g. *010*, *0010*, *00010*, *0010*, *00110*



**Q1:** Write a regular expression to find all the occurrences of “color” or “colour” in a document.

**Q2:** Write a regular expression that matches any non-empty string that starts with zero or more “a”s, followed by one or more “b”s.

**Q3:** Which of the following strings is matched by the regular expression  $Sc(o^+)(b|d)^*y$ ?

Scooby   Scoby   Scddy   Scooby   Scoobdbbdy

Draw an FSM that recognises the same language.

# Chapter 53 – The Turing machine

## Objectives

- Know that a Turing machine can be viewed as a computer with a single fixed program, expressed using
  - a finite set of states in a state transition diagram
  - a finite alphabet of symbols
  - an infinite tape with marked off squares
  - a sensing read-write head that can travel along the tape, one square at a time
- Understand the equivalence between a transition function and a state transition diagram
- Be able to:
  - represent transition rules using a transition function
  - represent transition rules using a state transition diagram
  - hand-trace simple Turing machines
- Explain the importance of Turing machines and the Universal Turing machine to the subject of computation

## Alan Turing

Alan Turing (1912–1954) was a British computer scientist and mathematician, best known for his work at Bletchley Park during the Second World War. While working there, he devised an early computer for breaking German ciphers, work which probably shortened the war by two or more years and saved countless lives.

Turing was interested in the question of **computability**, and the answer to the question “Is every mathematical task computable?” In 1936 he invented a theoretical machine, which became known as the **Turing machine**, to answer this question.

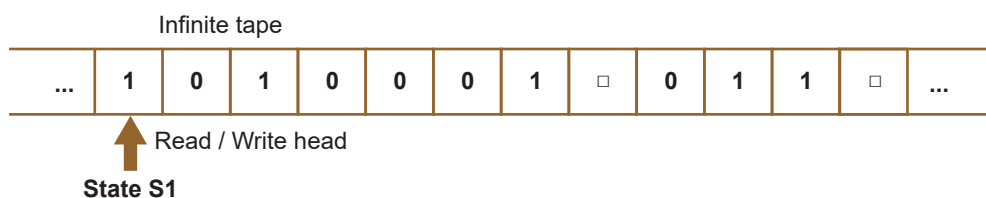


9-53

## The Turing machine

The Turing machine consists of an infinitely long strip of tape divided into squares. It has a read/write head that can read symbols from the tape and make decisions about what to do based on the contents of the cell and its current state.

Essentially, this is a finite state machine with the addition of an infinite memory on tape. The FSM specifies the task to be performed; it can erase or write a different symbol in the current cell, and it can move the read/write head either left or right.



The Turing machine is an early precursor of the modern computer, with input, output and a program which describes its behaviour. Any alphabet may be defined for the Turing machine; for example a binary alphabet of 0, 1 and □ (representing a blank), as shown in the diagram above.

A computer sending data across a network will use a **subnet mask** and the destination IP address to determine from the network ID whether or not the destination computer is on the same subnetwork. This is done by performing the same AND operation between the computer's own IP address and the subnet mask; if the two network IDs produced are the same then the computers are on the same network so data can be sent directly between them. Otherwise the sending computer must send the data to a router for forwarding to the network that the destination computer is on.

	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1
	140								.	24								.	112								.	57							
<b>IP Address:</b>	1	0	0	0	1	1	0	0	.	0	0	0	1	1	0	0	0	.	0	1	1	1	0	0	0	0	.	0	0	1	1	1	0	0	1
<b>Subnet mask:</b>	1	1	1	1	1	1	1	1	.	1	1	1	1	1	1	1	1	.	1	1	1	1	1	1	1	1	.	0	0	0	0	0	0	0	0
<b>Network ID:</b>	1	0	0	0	1	1	0	0	.	0	0	0	1	1	0	0	0	.	0	1	1	1	0	0	0	0	.	0	0	0	0	0	0	0	0

## Subnetting

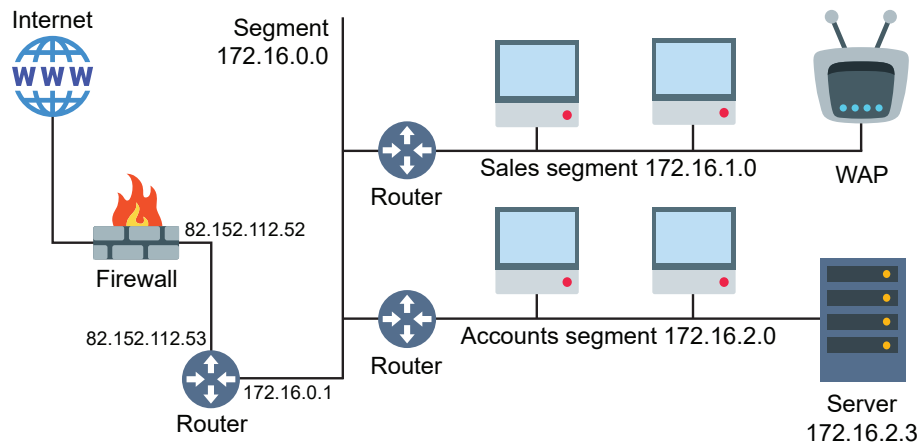
A network administrator of a large organisation using an **IP address** with a 16-bit network ID may wish to create **subnetwork** segments within their own larger IP network in order to ease management and improve efficiency by routing data through one segment only. Using a bus network, this would allow two computers in subnetwork A to communicate at the same time as two computers in subnetwork B avoiding any collisions. **Subnetting** reduces the size of the broadcast domain which can improve security, speed and reliability.

A **subnet ID** is created by using the most significant bits from the host ID section of the IP addresses. In the example below, the eight most significant bits of the 16-bit host ID have been used as a subnet ID leaving 8 bits or 254 (28 = 254-2 to exclude all-zero and all-one) unique host addresses in each of 256 (28) new subnetworks. The term Subnet ID is often used to cover the Network ID and Subnet ID together. For example, if you configure a computer or home router no distinction is made between the two.

10-60

	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1	.	128	64	32	16	8	4	2	1	
	172								.	16								.	1								.	5								
<b>IP Address:</b>	1	0	1	0	1	1	0	0	.	0	0	0	1	0	0	0	0	.	0	0	0	0	0	0	0	1	.	0	0	0	0	0	0	1	0	1
<b>Subnet mask:</b>	1	1	1	1	1	1	1	1	.	1	1	1	1	1	1	1	1	.	1	1	1	1	1	1	1	1	.	0	0	0	0	0	0	0	0	
	1	0	1	0	1	1	0	0	.	0	0	0	1	0	0	0	0	.	0	0	0	0	0	0	0	1	.	0	0	0	0	0	0	0	0	
	<b>Network ID</b>																<b>Subnet ID</b>								<b>Host ID</b>											

A network diagram showing subnetwork segments might look like this:



**Q2:** Suggest a suitable IP address for the Wireless Access Point in the diagram above.

# Chapter 70 – Function application

## Objectives

- Understand what is meant by partial function application
- Know that a function takes only one argument which may itself be a function
- Define and use higher-order functions, including map, filter and fold

## Higher-order functions

A **higher-order function** is one which either takes a function as an argument or returns a function as a result, or both. Later in this chapter we will be looking at the higher-order functions **map**, **filter** and **fold**, in which the first argument is a function and the second argument is a list on which the function operates, returning a list as a result.

*Every function in Haskell takes only one argument.* This may seem like a contradiction because we have seen many functions, such as the one below which adds three integers,

```
add3Integers x y z = x + y + z
```

which appear to take several arguments. So how can this be true?

### Any function takes only one parameter at a time

Taken at face value and assuming the function takes three integer parameters and returns an integer result, the type declaration for this function would normally be written

```
add3Integers :: integer -> integer -> integer -> integer
```

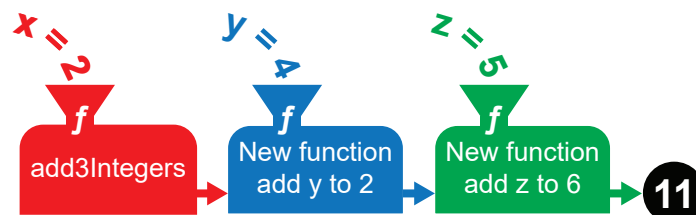
It could also be written

```
add3Integers :: integer -> (integer -> (integer -> integer))
```

### How the function is evaluated

What happens when you write `add3Integers 2 4 5`?

The function `add3Integers` is applied to the arguments. It takes the first argument `2` and produces a new function (shown in blue above) which will add `2` to its arguments, `4` and `5`.



This function (shown in blue) produces a new function (shown in green) that takes the argument `5` and adds it to `6`, returning the result, `11`.

Our function `add3Integers` takes an integer argument (`2`) and returns a function of type

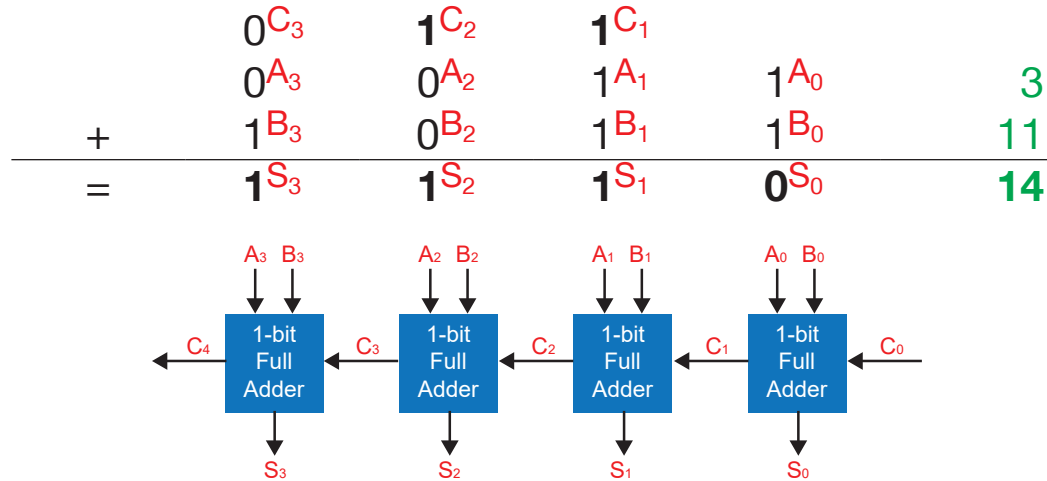
```
(integer -> (integer -> integer))
```

12-70



### Concatenating full adders

Multiple full adders can be connected together. Using this construct, n full adders can be connected together in order to input the carry bit into a subsequent adder along with two new inputs to create a concatenated adder capable of adding a binary number of n bits.



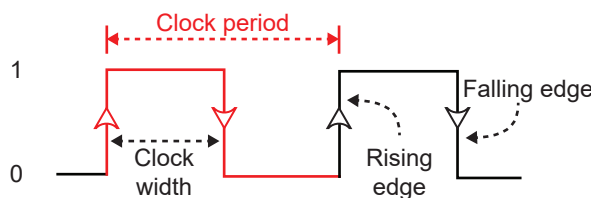
**Q1:** What would be the output  $S_4$  from a fifth adder connected to the diagram above if the inputs for  $A_4$  and  $B_4$  were 0 and 1? What would be the output  $C_5$ ?

B

### D-type flip-flops

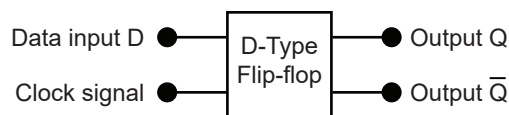
A flip-flop is an elemental **sequential logic circuit** that can store one bit and flip between two states, 0 and 1. It has two inputs, a control input labelled D and a clock signal.

The **clock** or **oscillator** is another type of sequential circuit that changes state at regular time intervals. Clocks are needed to synchronise the change of state of flip-flop circuits.



The **D-type flip-flop** (D stands for Data or Delay) is a positive **edge-triggered flip-flop**, meaning that it can only change the output value from 1 to 0 or vice versa when the clock is at a rising or positive edge, i.e. at the beginning of a clock period.

When the clock is not at a positive edge, the input value is held and does not change. **The flip-flop circuit is important because it can be used as a memory cell to store the state of a bit.**

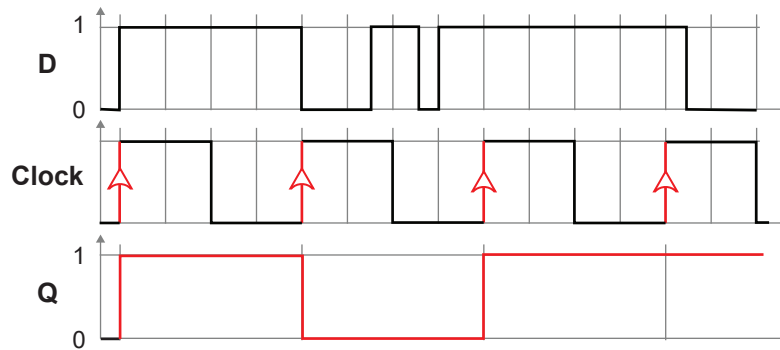


Output Q only takes on a new value if the value at D has changed at the point of a clock pulse. This means that the clock pulse will freeze or ‘store’ the input value at D until the next clock pulse. If D remains the same on the next clock pulse, the flip-flop will hold the same value.

### The use of a D-type flip-flop as a memory unit

A flip-flop comprises several NAND (or AND and OR) gates and is effectively 1-bit memory. To store eight bits, eight flip-flops are required. **Register memories** are constructed by connecting a series of flip-flops in a row and are typically used for the intermediate storage needed during arithmetic operations. Static RAM is also created using D-type flip-flops. Imagine trying to assemble 16GB of memory in this way!

The graph below illustrates how the output Q only changes to match the input D in response to the rising edge on the clock signal. Q therefore delays, or ‘stores’ the value of D by up to one clock cycle.



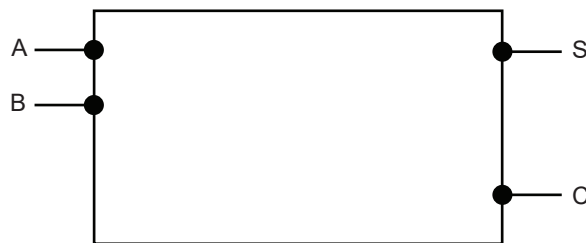
### Exercises

B

1. A half-adder is used to find the sum of the addition of two binary digits.

(a) Complete the diagram below to construct a half adder circuit.

[1]



(b) Complete the following truth table for a half adder's outputs S and C.

A	B	S	C

[2]

(c) How does a full adder differ from a half adder in terms of its inputs?

[2]

## Index

### A

absolute error, 385  
 abstract data types, 188  
 abstraction, 52, 108  
   data, 57  
   functional, 56  
   problem, 57  
   procedural, 55  
 accumulator, 132, 138  
 active tags, 152  
 ADC, 90  
 adders  
   concatenating, 387  
 address bus, 127, 128, 135  
 addressing mode  
   direct, 139  
   immediate, 139  
 adjacency  
   list, 208  
   matrix, 208  
 ADT, 188  
 aggregation, 353  
 agile modelling, 342  
 Alan Turing, 273  
 algorithm, 2  
 ALU, 132  
 Amazon, 179  
 analogue  
   data, 89  
   to digital conversion, 90  
 analysis, 34, 342  
 AND, 10, 144  
 AND gate, 115  
 API, 313  
 appending, 372  
 application layer, 300, 301  
 Application Programming  
   Interface, 103, 313  
 application software, 102  
 arithmetic logic unit, 127, 132  
 arithmetic operations, 3, 127, 143  
 ARPANET, 288  
 array, 17, 19, 190  
 ASCII, 73  
 assembler, 110  
 assembly language, 108,  
   109, 140, 142  
 association, 353  
 asymmetric encryption, 296

asynchronous transmission, 162  
 attributes, 319, 347  
 audio bit depth, 88  
 automation, 58  
 automaton, 61

### B

backing store management, 104  
 Backus-Naur form, 278  
 bandwidth, 161  
 barcode reader, 149  
 barcodes  
   2-D, 148  
   linear, 148  
 base case, 224  
 baud rate, 161  
 behaviours, 347  
 Big Data, 374  
 Big-O notation, 229, 231  
 binary  
   addition, 77  
   converting to and from decimal, 69  
   file, 31  
   fixed point, 80  
   floating point, 81  
   multiplication, 78  
   negative numbers, 79  
   number system, 69  
   subtraction, 80  
 binary expression tree, 286  
 binary search, 236  
   recursive algorithm, 237  
   tree, 212  
 binary search tree, 215  
 binary tree search, 238  
 bit, 72  
   depth, 88  
   rate, 161  
 bitmap image, 83  
 block-structured languages, 39  
 Blu-Ray, 155  
 BNF, 278  
 Boolean algebra, 120  
   Absorption rules, 120  
   Associative rules, 120  
   Commutative rules, 120  
   Distributive rules, 120  
 Boolean operators, 10

breadth-first  
   search, 248  
   traversal, 245, 246  
 bridges of Königsberg, 54  
 browser, 305  
 bubble sort, 44, 238  
 bus, 127  
   address, 128  
   control, 128  
   data, 128  
 byte, 72  
 bytecode, 112

### C

cache memory, 135  
 Caesar cipher, 96  
 call stack, 200, 225  
 camera-based readers, 150  
 cardinality, 265  
 carry, 78  
 Cartesian product, 266  
 CASE, 10  
 CCD reader, 150  
 CD-ROM, 155  
 Central Processing Unit, 126  
 check digit, 75  
 checksum, 75, 292  
 ciphertext, 96, 295  
 CIR, 133  
 circular queue, 190  
 class, 348  
 classful addressing, 308  
 classless addressing, 308  
 client-server  
   database, 339  
   model, 313  
   network, 168  
 clock speed, 135  
 CMOS, 151  
 co-domain, 360  
 collision, 202  
   resolution, 204  
 Colossus computer, 106  
 colour depth, 83  
 comments, 3  
 commitment ordering, 340  
 compact representation, 266  
 compare and branch  
   instructions, 143

compiler, 110, 112  
 composite data types, 188  
 composition, 57, 353  
 compression  
     dictionary-based, 95  
     lossless, 93  
     lossy, 93  
 computability, 273  
 computable problems, 256  
 computational thinking, 35, 52  
 Computer Misuse Act, 183  
 constant, 6  
 constructor, 348  
 control bus, 127, 128  
 control unit, 127, 132  
 convex combination, 220  
 Copyright, Designs and  
     Patents Act (1988), 183  
 CPU, 126  
 CRC, 292  
 CRUD, 314  
 cryptanalysis, 96, 97  
 CSMA/CA, 173  
 CSS Object Model, 305  
 CSSOM, 305  
 current instruction register, 133  
 cyber-attack, 177  
 cyber-bullying, 181  
 cyclical redundancy check, 292

**D**

DAC, 90  
 data  
     analogue, 89  
     boundary, 48  
     bus, 127, 128, 135  
     communication, 159  
     digital, 89  
     erroneous, 48  
     normal, 48  
     structures, 17  
     transfer operations, 143  
     types, 3  
     user-defined type, 29  
 data abstraction, 188  
 data packets, 292  
 Data Protection Act (1998), 183  
 database

    defining a table, 336  
     locking, 340  
     normalisation, 324  
     relational, 323  
 De Morgan's laws, 118  
 decomposition, 57  
 denary, 80  
 depth-first  
     traversal, 243  
 design, 34, 343  
 destruction of jobs, 180  
 dictionary, 205  
 dictionary based compression, 95  
 digital  
     camera, 151  
     certificate, 297  
     data, 89  
     signature, 296  
     to analogue conversion, 90  
 digraph, 207  
 Dijkstra's algorithm, 249, 293  
 directed graph, 207  
 disk defragmenter, 101  
 divide and conquer, 43  
 DNS, 290  
 Document Object Model, 305  
 DOM, 305  
 domain, 360  
 domain name, 289, 290  
     fully qualified, 291  
 Domain Name System, 290  
 dot product, 220  
 DPI, 83  
 driverless cars, 182  
 dry run, 49  
 D-type flip-flop, 388, 389  
 dual-core processor, 134  
 dynamic data structure, 190  
 dynamic filtering, 295

**E**

EAN, 76  
 early computers, 106  
 eBay, 179  
 edge, 207  
 elementary data types, 17, 188  
 embedded systems, 130  
 encapsulating what varies, 357  
 encapsulation, 188, 350

encryption, 96, 295  
     asymmetric, 296  
     private key, 296  
     public key, 296  
     symmetric, 296  
 Enigma code, 106  
 entity, 319  
     identifier, 319  
     relationship diagram, 320, 321  
 error checking, 74  
 ethics, 182  
 evaluating a program, 46  
 evaluation, 50, 344  
 event messages, 91  
 exbi, 72  
 exponent, 381  
 exponential function, 230

**F**

fact-based model, 377  
 fetch-execute cycle, 134  
 field, 29  
 FIFO, 188  
 file, 29  
     binary, 31  
     server, 168  
     text, 29  
 File Transfer Protocol, 303  
 filter, 370  
 finite set, 265  
 finite state  
     automaton, 61, 260  
     machine, 60, 260  
 firewall, 294  
 first generation language, 53  
 First In First Out, 188  
 First normal form, 324  
 first-class object, 362  
 fixed point, 385  
 floating point, 385  
     binary numbers, 381  
 fold (reduce), 370  
 folding method, 203  
 FOR ... ENDFOR, 15  
 foreign key, 320, 324  
 FQDN, 291  
 frequency of a sound, 90  
 FSM, 260  
 FTP, 303

full adder, 387  
 Fully Qualified Domain Names, 291  
 function, 360  
   application, 362  
   higher-order, 367  
 functional  
   composition, 364  
   programming, 360  
 functions, 5, 21, 230  
   string-handling, 5

**G**

gate  
   NOT, AND, OR, 114  
   XOR, NAND, NOR, 116  
 gateway, 293  
 general purpose registers, 132  
 getter messages, 349  
 gibi, 72  
 Google, 179  
   Street View, 178  
 graph, 207  
   schema, 377  
   theory, 55  
   traversals, 243

**H**

half-adder, 387  
 Halting problem, 257  
 hard disk, 154  
 hardware, 100  
 Harvard architecture, 130  
 hash table, 202  
 hashing algorithm, 202  
   folding method, 203  
 Haskell, 360, 361  
 heuristic methods, 256  
 hexadecimal, 70  
 hierarchy chart, 40  
 higher-order function, 367  
 high-level languages, 109  
 HTTP request methods, 314

**I**

I/O controller, 127, 129  
 IF ... THEN, 8  
 image resolution, 83  
 immutable, 363, 372

imperative language, 109  
 implementation, 344  
 infinite set, 266  
 infix expression, 284  
 information hiding, 54, 57, 188, 350  
 inheritance, 351  
 in-order traversal, 214, 225, 226  
 Instagram, 181  
 instantiation, 348  
 instruction set, 107, 110  
 interface, 23, 129, 357  
 Internet  
   registrars, 289  
   registries, 290  
   security, 172, 294  
   Service Providers, 289  
 Internet of things, 182  
 interpreter, 111, 112  
 interrupt, 136  
   handling, 105  
 Interrupt Service Routine, 136  
 intractable problems, 255  
 IP address, 291  
   private, 309  
   public, 309  
   structure, 307  
 irrational number, 68  
 ISBN, 76  
 ISP, 289  
 iteration, 13

**J**

Java Virtual Machine, 112  
 JSON, 315, 316

**K**

kibi, 72  
 kilobyte, 72

**L**

LAN, 164  
 laser  
   printer, 152  
   scanner, 150  
 latency, 161  
 legislation, 183  
 library programs, 101  
 limits of computation, 254  
 linear function, 230

linear search, 235  
 link layer, 300, 301  
 linking database tables, 324  
 list, 194, 371  
   appending to, 372  
   prepending to, 372  
 loader, 103  
 local area network, 164  
 logarithmic function, 231  
 logic gates, 114  
 logical bitwise operators, 144  
 logical operations, 127  
 low-level language, 108

**M**

MAC address, 167, 302  
 machine code, 106  
   instruction format, 138  
 mail server, 304  
 majority voting, 75  
 malicious software, 297  
 malware, 297  
 mantissa, 381  
 many-to-many relationship, 321, 326  
 map, 369  
 MAR, 133  
 maze, 247  
 MBR, 133  
 Mealy machines, 260, 261  
 mebi, 72  
 Media Access Control, 301  
 memory  
   address register, 133  
   buffer register, 133  
   data register, 133  
   management, 104  
 merge sort, 239  
   space complexity, 241  
   time complexity, 241  
 metadata, 84  
 meta-languages, 278  
 MIDI, 91  
 metadata, 91  
 mnemonics, 142  
 modelling data requirements, 343  
 modular programming, 25  
 module, 39  
 modulo 10 system, 76

**N**

NAND gate, 116  
 NAT, 310  
 natural number, 68, 265  
 nested loops, 15  
 network  
   client-server, 168  
   interface cards, 294  
   layer, 300, 301  
   peer-to-peer, 169  
   security, 172, 294  
   station, 171  
 Network Address  
   Translation, 310, 311  
 nibble, 72  
 NIC, 294  
 node, 207  
 non-computable problems, 256  
 NOR gate, 116  
 normal form  
   first 1NF, 324  
   second 2NF, 326  
   third 3NF, 326  
 normalisation, 327  
   of databases, 324  
   of floating point number, 382  
 NOT, 10, 11, 144  
   gate, 114  
 number  
   irrational, 68  
   natural, 68  
   ordinal, 68  
   rational, 68  
   real, 68  
 Nyquist's theorem, 90

**O**

object code, 110  
 object-oriented programming, 347  
 one-time pad, 97  
 opcode, 106, 138  
 operand, 106, 138  
 operating system, 100, 103  
 operation code, 106, 138  
 optical disk, 155  
 OR, 10, 144  
   gate, 115  
 ORDER BY, 332  
 ordinal number, 68

oscillator, 388  
 overflow, 78, 386  
 override, 354  
 Oyster card, 152

**P**

packet filters, 294  
 packet switching, 292  
 PageRank algorithm, 209  
 parallel data transmission, 160  
 parity, 162  
   bit, 74  
 partial dependency, 326  
 partial function application, 368  
 passive tags, 152  
 PC, 133  
 pebi, 72  
 peer-to-peer network, 169  
 pen-type reader, 149  
 peripheral management, 105  
 permutations, 231  
 phishing, 299  
 piracy, 170  
 pixel, 83  
 plaintext, 96, 295  
 platform independence, 112  
 polymorphism, 354  
 polynomial function, 230  
 polynomial-time solution, 255  
 POP3, 304  
 port forwarding, 311  
 Post Office Protocol (v3), 304  
 postfix  
   expression, 284  
   notation, 283  
 post-order traversal, 214, 227  
 precedence rules, 283  
 pre-order traversal, 213, 227  
 prepending, 372  
 primary key, 319  
 priority queue, 192  
 private, 348  
   key encryption, 296  
   modifier, 356  
 problem solving strategies, 36  
 procedural programming, 347  
 procedure, 21  
 procedure interface, 56  
 processor, 127

instruction set, 138  
 performance, 134  
 scheduling, 104  
 program  
   constructs, 8  
   counter, 133  
 programming paradigm, 360  
 proper subset, 266  
 protected access modifier, 356  
 protocol, 163  
 prototype, 343  
 proxy server, 294, 295  
 pseudocode, 2  
 public, 348  
   modifier, 356

**Q**

quad-core processor, 134  
 queue, 188  
   operations, 189  
 Quick Response (QR) code, 148

**R**

Radio Frequency Identification, 151  
 range, 79  
 raster, 83  
 rational number, 68, 265  
 real number, 265  
 record, 29  
 record locking, 340  
 recursion, 224  
 recursive algorithm, 237  
 reference variable, 349  
 referential transparency, 363  
 register, 127  
 regular expressions, 269  
 regular language, 270  
 rehashing, 204  
 relation, 323  
 relational database, 320, 323  
 relational operators, 8  
 relationships, 320  
 relative error, 385  
 REPEAT ... UNTIL, 14  
 Representational State Transfer, 314  
 resolution, 83  
 resource management, 100  
 REST, 314

- Reverse Polish notation, 283
- RFID, 151
- RLE, 94
- root node, 211
- rooted tree, 211
- rounding errors, 384
- router, 171, 293
- RTS/CTS, 173
- Run Length Encoding, 94
  
- S**
- sample resolution, 88
- scaling vectors, 220
- Second normal form, 326
- secondary storage, 154
- Secure Shell, 304
- SELECT .. FROM .. WHERE, 330
- selection statement, 8
- serial data transmission, 159
- serialisation, 340
- server
  - database, 168
  - file, 168
  - mail, 168
  - print, 168
  - web, 168
- Service Set Identification, 172
- set, 265
  - compact representation, 266
  - comprehension, 266
  - countable, 266
  - countably infinite, 266
  - difference, 267
  - intersection, 267
  - union, 267
- setter messages, 349
- side effects, 363
- simulation, 188
- Snowden, Edward, 176
- social engineering, 299
- software, 34, 100, 102
  - application, 102
  - bespoke, 102
  - development, 342
  - off-the-shelf, 102
  - system, 100
  - utility, 101
- solid-state disk, 156
- sorting algorithms, 44, 238
- sound sample size, 89
- source code, 110
- space complexity, 241
- spam filtering, 299
- specifier
  - private, 356
  - protected access, 356
  - public, 356
- SQL, 330, 338
- SSD, 156
- SSH, 304
- SSID, 172
- stack, 198
  - call, 200
  - frame, 201
  - overflow, 200
- underflow, 200
- state, 347
  - transition diagrams, 260
  - transition table, 261
- stateful inspection, 295
- stateless, 363
- static data structure, 190
- static filtering, 294
- Static IP addressing, 310
- stored program concept, 129
- string conversion, 5
- structured programming, 39
- Structured Query Language, 330
- subclass, 351
- subnet mask, 308, 310
- subnetting, 309
- subroutines, 21
  - advantages of using, 25
  - user-written, 22
  - with interfaces, 23
- subset, 266
- substitution cipher, 96
- superclass, 351
- symmetric encryption, 296
- synchronous transmission, 162
- synonym, 202
- syntax diagrams, 280
- syntax error, 111
- system
  - bus, 127
  - clock, 132
  - vulnerabilities, 298
  
- T**
- table structure, 336
- TCP/IP protocol stack, 300
- tebi, 72
- Telnet, 304
- test plan, 48
- testing, 48, 344
- text file, 29
- thick-client computing, 316
- thin-client computing, 316
- Third normal form, 326
- Tim Berners-Lee, 288
- time complexity, 229, 233, 235, 236
  - of merge sort, 241
- timestamp ordering, 340
- topology
  - logical, 166
  - physical, 166
  - physical bus, 164
  - physical star, 165
- trace table, 14, 49, 107
- tractable problems, 255
- transition functions, 276
- translators, 101
- transmission rate, 161
- transport layer, 300, 301
- travelling salesman problem, 254, 256
- traversing a binary tree, 213
- tree, 211
  - child, 211
  - edge, 211
  - leaf node, 211
  - node, 211
  - parent, 211
  - root, 211
  - subtree, 211
  - traversal algorithms, 225
- trojans, 298
- trolls, 181
- truth tables, 114
- TSP, 256
- Turing machine, 273
- two's complement, 80
- typeclasses, 365
  
- U**
- underflow, 386
- undirected graph, 207
- Unicode, 74

Uniform Resource Locators, 289  
 union, 267  
 universal Turing machine, 276  
 URLs, 289  
 user generated content, 181  
 user interface, 100  
 user-defined data type, 29  
 utility software, 101

**V**

variables, 6  
   global, 24  
   local, 24  
 vector, 217  
   adding and  
   subtracting, 218  
   calculating an angle, 221  
   convex  
   combination, 220  
   dot product, 220  
   scaling, 220  
 vector graphics, 85  
 Vernam cipher, 96  
 vertex, 207  
 virtual memory, 104  
 virus checker, 101  
 viruses, 297  
 von Neumann, 100  
   machine, 129

**W**

WAP, 171  
 web server, 305  
 WebSocket protocol, 314  
 weighted graph, 207  
 WHILE ... ENDWHILE, 13  
 whitelist, 172  
 Wi-Fi, 171  
   Protected Access, 172  
 Wilkes, Maurice, 100  
 WinZip, 101  
 wireless network  
   access point, 171  
   interface controller, 171  
 word, 128  
 word length, 135  
 World Wide Web, 288  
 worms, 297  
 WPA, 172  
 WWW, 288

**X**

XML, 315, 316  
 XOR, 11, 144  
   gate, 116

**Y**

yobi, 72

**Z**

zebi, 72



# AQA AS and A Level **Computer Science**



The aim of this textbook is to provide a detailed understanding of each topic of the new AQA A Level Computer Science specification. It is presented in an accessible and interesting way, with many in-text questions to test students' understanding of the material and their ability to apply it.

The book is divided into 12 sections, each containing roughly six chapters. Each chapter covers material that can comfortably be taught in one or two lessons. It will also be a useful reference and revision guide for students throughout the A Level course. Two short appendices contain A Level content that could be taught in the first year of the course as an extension to related AS topics.

Each chapter contains exercises, some new and some from past examination papers, which can be set as homework. Answers to all these are available to teachers only, in a Teachers Supplement which can be ordered from our website

[www.pgonline.co.uk](http://www.pgonline.co.uk)

## **About the authors**

**Pat Heathcote** is a well-known and successful author of Computer Science textbooks. She has spent many years as a teacher of A Level Computing courses with significant examining experience. She has also worked as a programmer and systems analyst, and was Managing Director of Payne-Gallway Publishers until 2005.

**Rob Heathcote** has many years of experience teaching Computer Science and is the author of several popular textbooks on Computing. He is now Managing Director of PG Online, and writes and edits a substantial number of the online teaching materials published by the company.

Cover picture:

**'South Coast Sailing'**

Oil on canvas, 60x60cm

© Heather Duncan

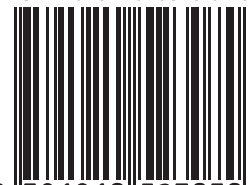
[www.heatherduncan.com](http://www.heatherduncan.com)

**This book has been  
approved by AQA.**



**PG ONLINE**

ISBN: 978-1-910523-07-0



9 781910 523070