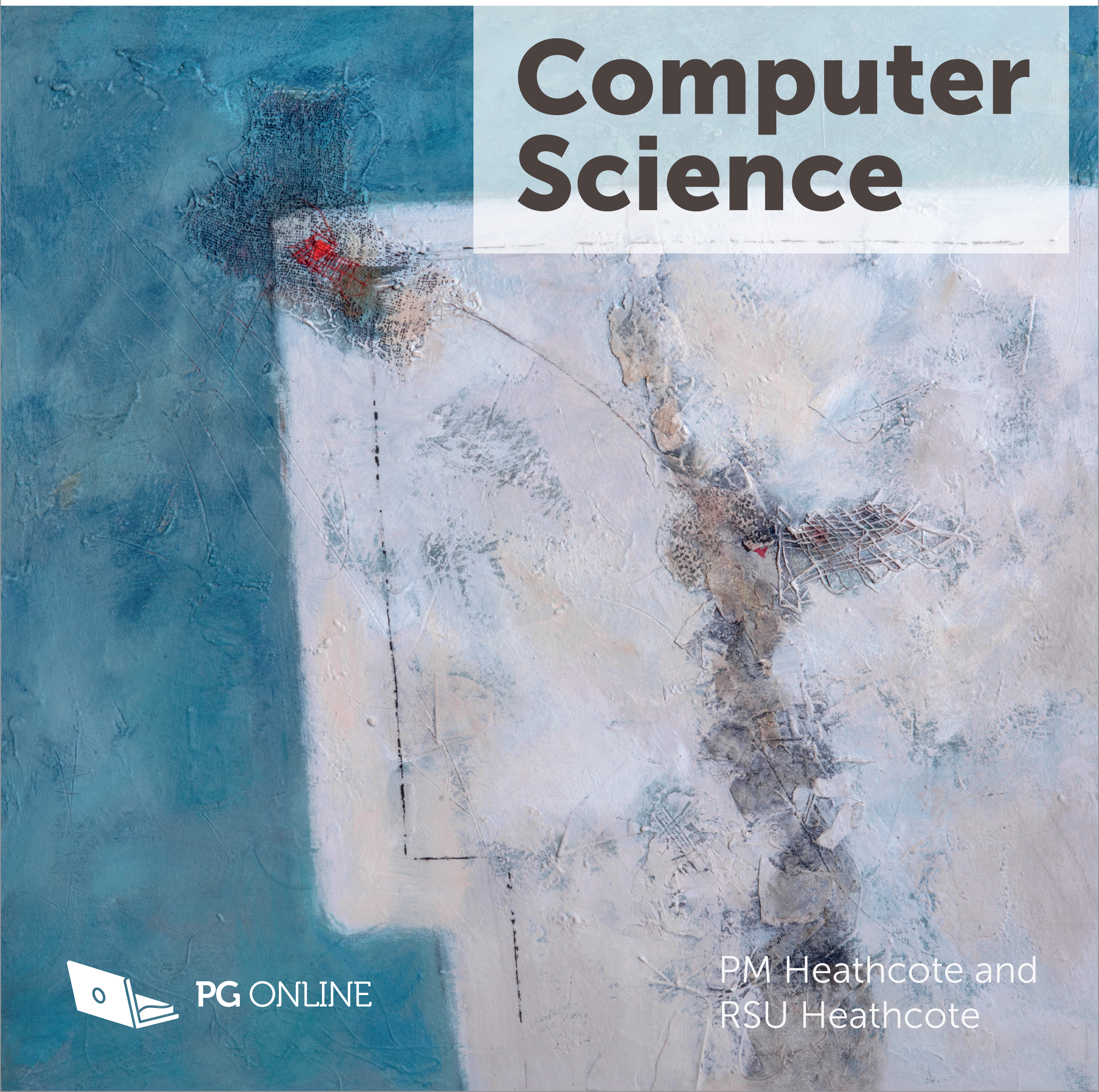
OCR
Oxford Cambridge and RSA
An OCR endorsed textbook

OCR AS and A Level

Computer Science



PM Heathcote and
RSU Heathcote

PG ONLINE

Contents

Section 1

Components of a computer		**1**
Chapter 1	Processor components	2
Chapter 2	Processor performance	7
Chapter 3	Types of processor	10
Chapter 4	Input devices	16
Chapter 5	Output devices	20
Chapter 6	Storage devices	25

Section 2

Systems software		**29**
Chapter 7	Functions of an operating system	30
Chapter 8	Types of operating system	36
Chapter 9	The nature of applications	39
Chapter 10	Programming language translators	44

Section 3

Software development		**51**
Chapter 11	Systems analysis methods	52
Chapter 12	Writing and following algorithms	57
Chapter 13	Programming paradigms	64
Chapter 14	Assembly language	69

Section 4

Exchanging data		**74**
Chapter 15	Compression, encryption and hashing	75
Chapter 16	Database concepts	82
Chapter 17	Relational databases and normalisation	88
Chapter 18	Introduction to SQL	95
Chapter 19	Defining and updating tables using SQL	101
Chapter 20	Transaction processing	106

Section 5
Networks and web technologies 110

Chapter 21	Structure of the Internet	111
Chapter 22	Internet communication	119
Chapter 23	Network security and threats	126
Chapter 24	HTML and CSS	130
Chapter 25	Web forms and JavaScript	136
Chapter 26	Search engine indexing	142
Chapter 27	Client-server and peer-to-peer	147

Section 6
Data types 154

Chapter 28	Primitive data types, binary and hexadecimal	155
Chapter 29	ASCII and Unicode	159
Chapter 30	Binary arithmetic	162
Chapter 31	Floating point arithmetic	167
Chapter 32	Bitwise manipulation and masks	174

Section 7
Data structures 178

Chapter 33	Arrays, tuples and records	179
Chapter 34	Queues	184
Chapter 35	Lists and linked lists	190
Chapter 36	Stacks	200
Chapter 37	Hash tables	204
Chapter 38	Graphs	209
Chapter 39	Trees	214

Section 8

Boolean algebra **222**

Chapter 40	Logic gates and truth tables	223
Chapter 41	Simplifying Boolean expressions	228
Chapter 42	Karnaugh maps	233
Chapter 43	Adders and D-type flip-flops	238

Section 9

Legal, moral, ethical and cultural issues **242**

Chapter 44	Computing related legislation	243
Chapter 45	Ethical, moral and cultural issues	249
Chapter 46	Privacy and censorship	255

Section 10

Computational thinking **259**

Chapter 47	Thinking abstractly	260
Chapter 48	Thinking ahead	265
Chapter 49	Thinking procedurally	268
Chapter 50	Thinking logically, thinking concurrently	272
Chapter 51	Problem recognition	277
Chapter 52	Problem solving	282

Section 11
Programming techniques **287**

Chapter 53	Programming basics	288
Chapter 54	Selection	294
Chapter 55	Iteration	299
Chapter 56	Subroutines and recursion	303
Chapter 57	Use of an IDE	313
Chapter 58	Use of object-oriented techniques	319

Section 12
Algorithms **327**

Chapter 59	Analysis and design of algorithms	328
Chapter 60	Searching algorithms	334
Chapter 61	Bubble sort and insertion sort	340
Chapter 62	Merge sort and quick sort	345
Chapter 63	Graph traversal algorithms	351
Chapter 64	Optimisation algorithms	358
Index		364

Chapter 4 – Input devices

Objectives

- Describe different input devices
- Explain how different input devices can be applied as a solution to different problems

Barcodes

Barcodes first started appearing on grocery items in the 1970s, and today they are used for identification in thousands of applications from tracking parcels, shipping cartons, passenger luggage, blood, tissue and organ products around the world to the sale of items in shops and the recording of the details of people attending events. Keeping track of anything accurately is now almost unimaginable without barcodes.

A handheld barcode scanner used for scanning medical samples

There are two different types of barcode: Linear barcodes such as the one shown above and 2D barcodes such as the Quick Response (QR) code, which can hold more information than the 1D barcode.

A 2D barcode

2D barcodes are used for example in ticketless entry to concerts, or access through gates to board a Eurostar train or passenger airline. They are also used in mobile phone apps that enable the user to take a photo of the code which may then provide them with further information such as a map of their location, product details or a website URL.

Barcode readers

There are four different barcode readers available, each using a slightly different technology for reading and decoding a barcode. The four types are pen-type readers, laser scanners, CCD readers and camera-based readers.

A-Level only

The statements `input(radius)`

`area = pi * radius * radius`

could be 'tokenised' and stored as the lexical string 1 3

5 4 2 7 3 7 3

Q2: What further entries to the symbol table will the lexical analyser make on encountering the statement

```
circumference = 2 * pi * radius
```

Add the entries to the symbol table and then tokenise the statement.

Note that the lexical analyser puts the identifier and its run-time address in the symbol table, so that it can replace them in the source code by 'tokens'. It will not fill in the 'kind of item' and 'type of item'; this is done later by the **syntax analyser**.

Accessing the symbol table

Since the lexical analyser spends a great proportion of its time looking up the symbol table, this activity has a crucial effect on the overall speed of the compiler. The symbol table must therefore be organised in such a way that entries can be found as quickly as possible. The most common way of organising the symbol table is a **hash table**, where the keyword or identifier is 'hashed' to produce an array subscript. As with any hash table, synonyms (collisions) are inevitable, and a common way of handling them is to store the synonym in the next available free space in the table.

Syntax analysis and semantic analysis

Syntax analysis is the process of determining whether the sequence of input characters, symbols, items or tokens form a valid sentence in the language. In order to do this, the language has to be expressed as a set of rules, using for example **syntax diagrams** or **Backus-Naur** form.

Parsing is the task of systematically applying the set of rules to each statement to determine whether it is valid. **Stacks** will be used to check, for example, that brackets are correctly paired. The priorities of arithmetic operators will be determined, and expressions converted into a form (such as **reverse Polish notation**) from which machine code can more easily be generated.

The **semantics** of the program will also be checked in this phase. **Semantics** define the meaning rather than the grammar of the language; it is possible to write a series of syntactically correct statements which nevertheless do not obey the rules for writing a correct program. An example of a semantic error is the use of an undeclared variable in Pascal, or trying to assign a **real** value to an **integer** variable, or using a real number instead of an integer as the counter in a **for … next** loop.

Q3: Give other examples of a semantic error.

Code generation and optimisation

This is the final phase of compilation, when the machine code is generated. Most high-level language statements will be translated into a number of machine code statements.

Code optimisation techniques attempt to reduce the execution time of the object program by, for example, spotting redundant instructions and producing object code which achieves the same net

The compression of sound and video works in a similar way. **MP3** files use lossy compression to remove frequencies too high for most of us to hear and to remove quieter sounds that are played at the same time as louder sounds. The resulting file is about 10% of original size, meaning that 1 minute of MP3 audio equates to roughly 1MB in size.

Voice is transmitted over the Internet or mobile telephone networks using lossy compression and although we have no problem in understanding what the other person is saying, we can recognise the difference in quality of a voice over a phone rather than in person. The apparent difference is lost data.

Lossless compression

Lossless compression works by recording patterns in data rather than the actual data. Using these patterns and a set of instructions on how to use them, the computer can reverse the procedure and reassemble an image, sound or text file with exact accuracy and no data is lost. This is most important with the compression of program files, for example, where a single lost character would result in an error in the program code. A pixel with a slightly different colour would not be of huge consequence in most cases. Lossless compression usually results in a much larger file than a lossy file, but one that is still significantly smaller than the original.

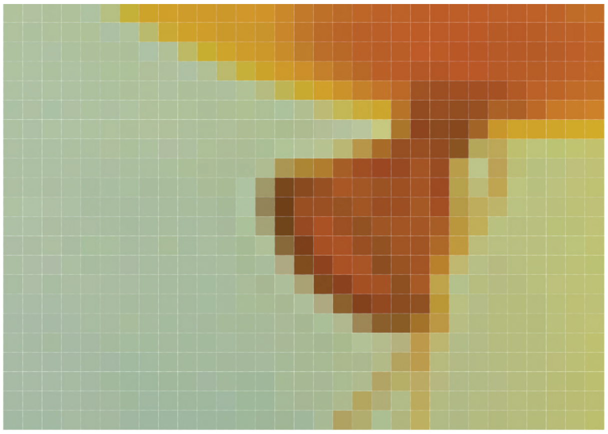
Q1: What type of compression is likely to be used for the following: a website image, a zipped file of long text documents and images, a PDF instruction manual?

4-15

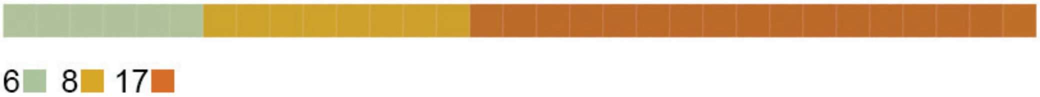
A-Level only

Run Length Encoding (RLE)

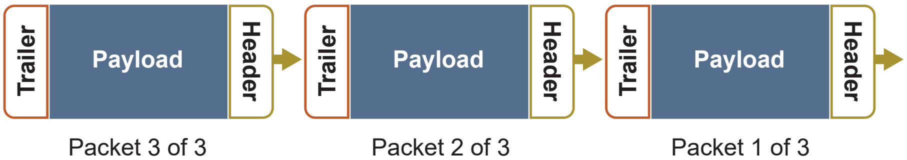
If you were ordering food from a takeaway restaurant for a group of five friends, it is likely that you might ask for "5 pizzas" rather than "one pizza, and another pizza, and another pizza etc." **Run Length Encoding** exploits the same principle. Rather than recording every pixel in a sequence, it records its value and the number of times it repeats.



For this section of the balloon image, the encoding for the first row might crudely translate to:
6 green, 8 yellow and 17 orange, using one binary value for the colour value and another for the number of contiguous matching pixels in the run. This would reduce the data necessary to store this row to 6 bytes (00000110 00000001 00001000 00000010 00010001 00000011) rather than 31 bytes assuming a bit depth of 8 and values for each colour of 00000001, 00000010 and 00000011.



The header (much like the box(es) of a consignment you might send or receive through the post) includes the sender's and the recipient's IP addresses, the protocol being used with this type of packet and the number of the packet in the sequence being sent, e.g. packet 1 of 8. They also include the **Time To Live** (TTL) or **hop limit**, after which point the data packet expires and is discarded.



Data packets queueing to be sent

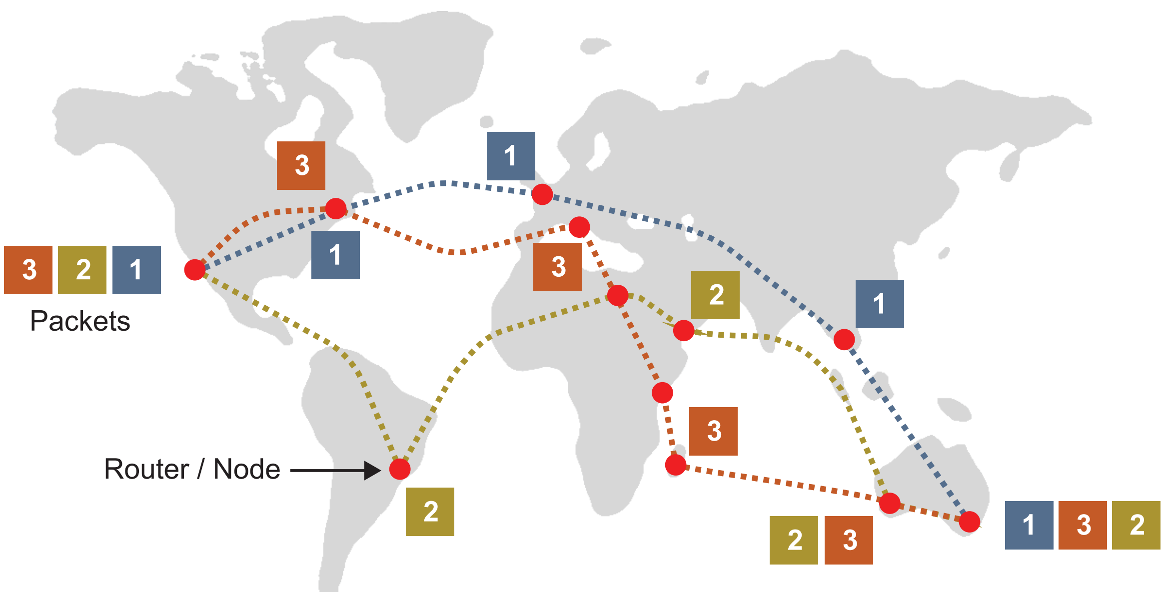
Q1: Why is the sender's IP address included in the packet header?

The payload of the packet contains the actual data being sent. Upon receipt, the packets are reassembled in the correct order and the data is extracted.

Routing packets across the Internet

The success of packet switching relies on the ability of packets to be sent from sender to recipient along entirely separate routes from each other. At the moment that a packet leaves the sender's computer, the fastest or least congested route is taken to the recipient's computer. They can be easily reassembled in the correct order at the receiving end and any packets that don't make it can be requested again.

5-22



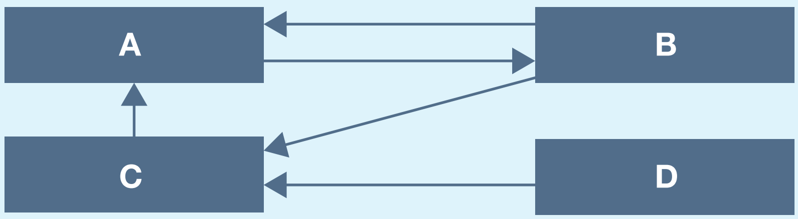
Q2: What information is included in the packet header to enable the receiving computer to reassemble packets in the correct order?

A-Level only

Example 2 shows the iterative process used to calculate and recalculate the PageRank (PR) of a group of webpages where the starting point is unknown.

Example 2

As the number of web pages grows, more complex link structures are created. After the addition of one extra web page, the PageRank is recalculated and adjusted to reflect the new pages and links.



First iteration: (Assumes a PR of 1 for each page where not known.)

d	$= 0.85$		
$PR(A)$	$= (1 - d) + d(PR(B)/2 + PR(C)/1)$	$PR(A)$	$= 0.15 + 0.85 * (0.5 + 1) =$ **1.425**
$PR(B)$	$= (1 - d) + d(PR(A)/1)$	$PR(B)$	$= 0.15 + 0.85 * 1.425 =$ **1.361**
$PR(C)$	$= (1 - d) + d(PR(B)/2 + PR(D)/1)$	$PR(C)$	$= 0.15 + 0.85 * (0.681 + 1) =$ **1.578**
$PR(D)$	$= (1 - d) + d(0)$	$PR(D)$	$=$ **0.15**

Second iteration: (Uses new PR figures from first iteration.)

d	$= 0.85$		
$PR(A)$	$= (1 - d) + d(PR(B)/2 + PR(C)/1)$	$PR(A)$	$= 0.15 + 0.85 * (0.681 + 1.578) =$ **2.07**
$PR(B)$	$= (1 - d) + d(PR(A)/1)$	$PR(B)$	$= 0.15 + 0.85 * 2.07 =$ **1.909**
$PR(C)$	$= (1 - d) + d(PR(B)/2 + PR(D)/1)$	$PR(C)$	$= 0.15 + 0.85 * (0.955 + 0.15) =$ **1.089**
$PR(D)$	$= (1 - d) + d(0)$	$PR(D)$	$=$ **0.15**

Third iteration:

d	$= 0.85$		
$PR(A)$	$= (1 - d) + d(PR(B)/2 + PR(C)/1)$	$PR(A)$	$= 0.15 + 0.85 * (0.955 + 1.089) =$ **1.887**
$PR(B)$	$= (1 - d) + d(PR(A)/1)$	$PR(B)$	$= 0.15 + 0.85 * 1.887 =$ **1.754**
$PR(C)$	$= (1 - d) + d(PR(B)/2 + PR(D)/1)$	$PR(C)$	$= 0.15 + 0.85 * (0.877 + 0.15) =$ **1.023**
$PR(D)$	$= (1 - d) + d(0)$	$PR(D)$	$=$ **0.15**

After three iterations, the PageRank of each page begins to settle. In reality many more iterations would be necessary before the figures stop moving, but three iterations get us close enough to understand the process and begin to see some results.

Page A now has a slightly higher ranking than B since it has another vote from page C. Page B has a higher rank than pages C and D because it has 100% of the votes from A, a high ranking page in itself. Page C has a comparatively moderate ranking since it has two inbound links from other pages that also have inbound links. C's vote from page D however is not given significant importance since page D has no inbound links and therefore has a low PageRank.

Q5: What factors may result in a web page A's rank rising or falling over time as it is revised?

A

5-26

A-Level only

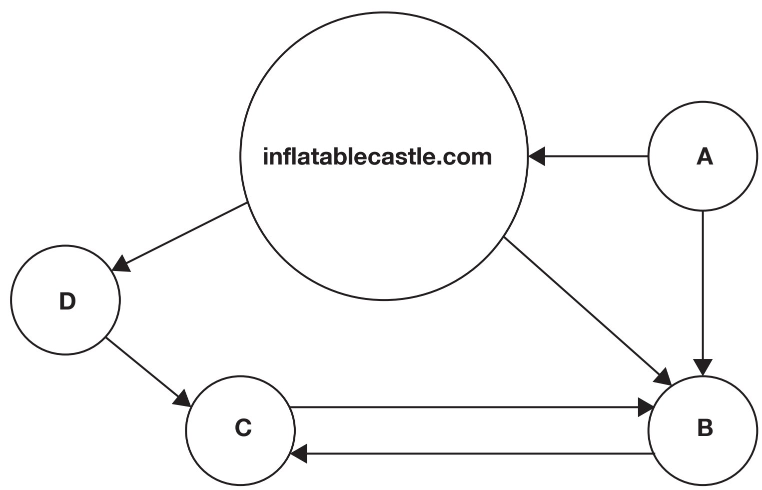
Exercises

1. The owner of website www.inflatablecastle.com is trying to improve the positioning of his homepage inflatablecastle.com/index.html in search engine listings.

   (a) Other than PageRank, give **three** design factors that may affect the company homepage's positioning in search results. [3]

   Google's PageRank algorithm *PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))* calculates a ranking for each web page that has a significant bearing on search results.

   (b) With reference to the diagram below, explain which page is likely to have the highest PageRank. You are not expected to perform any calculations. [2]



   (c) Looking at the algorithm, what factors directly influence the PageRank of the homepage index.html at inflatablecastles.com? [2]

   (d) PageRank uses a damping factor d in its algorithm. Explain the purpose of *d*. [2]

2. Search engines provide a listing of all web pages with content relevant to a set of search terms.

   (a) Explain how search engines produce this list. [2]

   (b) With reference to the screenshot below, state which line of code contains metatags. [1]

   (c) Briefly explain the purpose of the meta description. [2]

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
4  <TITLE>Fossils</TITLE>
5  <META NAME="Keywords" CONTENT="dinosaur, lyme, regis, limestone,
   ammonite, bone, jurassic, strata, rock, geology, paleontology">
6  <META NAME="Description" CONTENT="Fossils are the preserved remains
   of animals or plants, commonly found embedded in sedimentary
   layers of rock.">
7  <head>
8  <body>
9  </body>
10 </html>
```

5-26

A

Q5: Why not simply leave the array element names[2] blank after deleting Ken?

First, items are moved up to fill the empty space by copying them to the previous spot in the array:

Holly	**James**	**Nathan**	**Paul**	**Sophie**	**Sophie**	

Finally the last element, which is now duplicated, is replaced with a blank.

Holly	**James**	**Nathan**	**Paul**	**Sophie**		

A-Level only

Linked lists

Definition

A linked list is a dynamic data structure used to hold an ordered sequence, as described below:

- The items which form the sequence are not necessarily held in contiguous data locations, or in the order in which they occur in the sequence
- Each item in the list is called a **node** and contains a **data field** and a next address field called a **link** or **pointer field** (the data field may consist of several subfields.)
- The data field holds the actual data associated with the list item, and the pointer field contains the address of the next item in the sequence
- The link field in the last item indicates that there are no further items by the use of a **null** pointer
- Associated with the list is a pointer variable which points to (i.e. contains the address of) the first node in the list

7-35

Operations on linked lists

In the examples which follow we will assume that the linked list is held in memory in an array of records, and that each node consists of a person's name (the data field) and a pointer to the next item in the list.

We will explore how to set up or initialise an empty list, insert new data in the correct place in the list, delete an unwanted item and print out all items in the list. We will also look at the problem of managing the free space in the list.

A node record may be defined like this:

```
type nodeType
   string name
   integer pointer
endType

dim Names[0..5] of nodeType
```

Initialising a linked list

We need to keep two linked lists; one for the actual data, and one for the free space. When a new item is added, it is put in the node pointed to by `nextfree`. When a node is deleted, it is linked into the free space list.

Chapter 41 – Simplifying Boolean expressions

Objectives

A
- Use the following rules to derive or simplify statements in Boolean algebra:
  - o de Morgan's Laws
  - o commutation
  - o association
  - o distribution
  - o absorption
  - o double negation
- Write a Boolean expression for a given logic gate circuit, and vice versa

A-Level only

de Morgan's laws

Augustus de Morgan (1806-1871) was a Cambridge Mathematics professor who formulated two theorems or laws relating to logic. These laws can be used to manipulate and simplify Boolean expressions. Although his theoretical work had little practical application in his lifetime, it became of major significance in the next century in the field of digital electronics, in which TRUE and FALSE can be replaced by ON and OFF or the binary numbers 0 and 1.

8-41

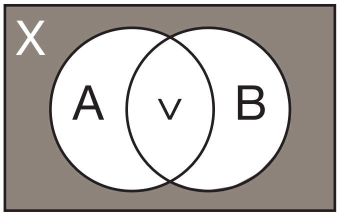
Using de Morgan's laws, any Boolean function can be converted to one which uses only NAND functions or only NOR functions, and these can be further converted to an expression using all NAND functions or all NOR functions.

Thus, any integrated circuit can be built from just one type of logic gate. This is an advantage in manufacturing where costs can be kept down by using only one type of gate.

de Morgan's first law

$$\neg(A \vee B) = \neg A \wedge \neg B$$

The truth of this is clear from the Venn diagram on the right. Suppose we have a variable X defined by



$$X = \neg(A \vee B)$$

Looking at the Venn diagram, A ∨ B is represented by the white area. Since X is not in A ∨ B, it consists of all the grey area. This can be defined as everything not in A and not in B, i.e.

$$X = \neg A \wedge \neg B$$

Q1: Complete the following truth table to show that $\neg(A \vee B) = \neg A \wedge \neg B$

A	B	¬A	¬B	A ∨ B	¬(A ∨ B)	¬A ∧ ¬B
0	0					
0	1					
1	0					
1	1					

A \ BC	00	01	11	10
0			1	1
1	1		1	1

Here, the group outlined in green "wraps around" but is still a single group. The expression simplifies to

B ∨ (A ∧ ¬C)

Q3: Simplify the same expression as above, (¬ A ∧ B) ∨ (B ∧ ¬ C) ∨ (B ∧ C) ∨ (A ∧ ¬ B ∧ ¬ C), but this time use a Karnaugh map with the following headings:

C \ AB	00	01	11	10
0				
1				

The four-variable problem

With four variables, each row or column represents a combination of two variables.

Example 4

Represent the expression A ∨ (A ∧ ¬ B ∧ C ∧ D) in a Karnaugh map, and hence simplify the expression.

AB \ CD	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

AB \ CD	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

This simplifies to A.

Summary of the Karnaugh map method

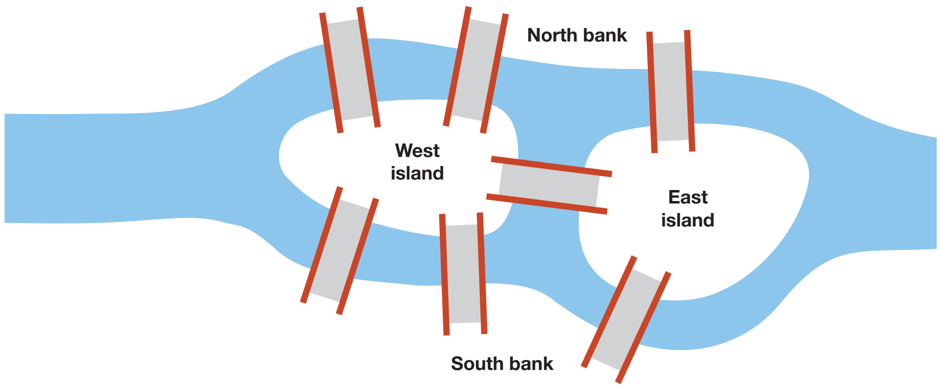
1. Construct the Karnaugh map step by step, placing 1s in the squares for each sub-expression separated by an OR symbol (∨)
2. Group any octet (8 squares)
3. Group any quad (4 squares that have not already been grouped, making sure to use the minimum number of groups
4. Group any pair which contains a 1 adjacent to only one other 1 which is not already in a group
5. Group any isolated 1s which are not adjacent to any other 1s.
6. Form the OR sum of all the terms generated by each group.

Abstraction by generalisation

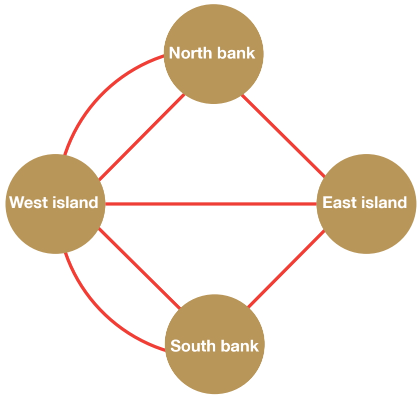
There is a famous problem dating back more than 200 years to the old Prussian city of Königsberg. This beautiful city had seven bridges, and the inhabitants liked to stroll around the city on a Sunday afternoon, making sure to cross every bridge at least once. Nobody could figure out how to cross each bridge once and once only, or alternatively prove that this was impossible, and eventually the Mayor turned to the local mathematical genius Leonhard Euler.

The map of 18th century Königsberg

Euler's first step was to remove all irrelevant details from the map, and come up with an abstraction:

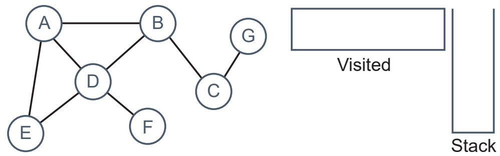


To really simplify it, Euler represented each piece of land as a circle and each bridge as a line between them.

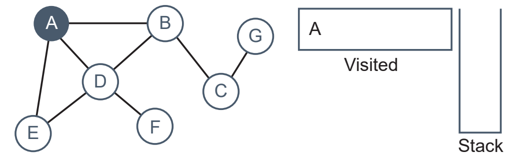


A-Level only

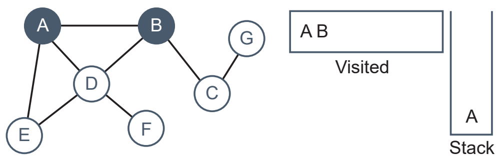
It is easiest to understand how this works by looking at the graphs below. This shows the state of the **stack** (here it just shows the current node when a recursive call is made), and the contents of the **visited** list. Each visited node is coloured dark blue.



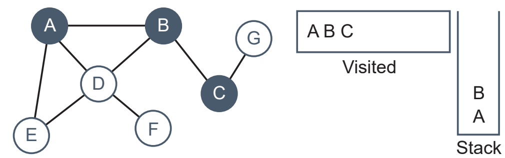
1. Start the routine with an empty stack and an empty list of visited nodes.



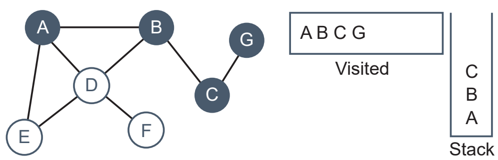
2. Visit A, add it to the visited list. Colour it to show it has been visited.



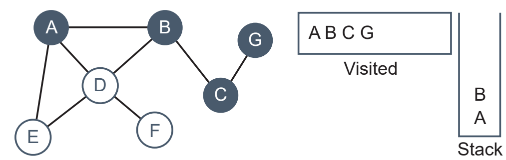
3. Push A onto the stack to keep track of where we have come from and visit A's first neighbour, B. Add it to the visited list. Colour it to show it has been visited.



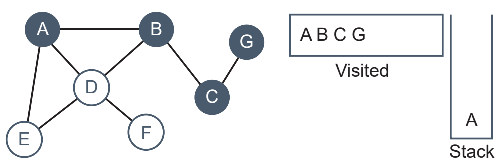
4. Push B onto the stack and from B, visit the next unvisited node, C. Add it to the visited list. Colour it to show it has been visited.



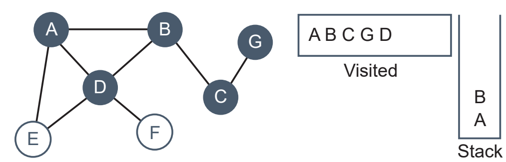
5. Push C onto the stack and from C, visit the next unvisited node, G. Add it to the visited list. Colour it to show it has been visited.



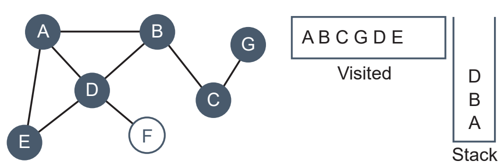
6. At G, there are no unvisited nodes so we backtrack. Pop the previous node C off the stack and return to C



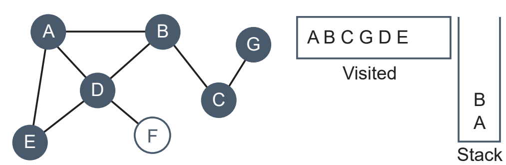
7. At C, all adjacent nodes have been visited, so backtrack again. Pop B off the stack and return to B.



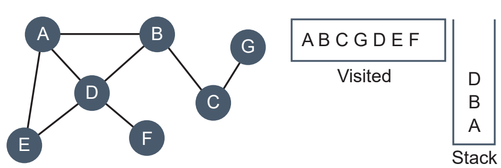
8. Push B back onto the stack to keep track of where we have come from and visit D. Add it to the visited list. Colour it to show it has been visited.



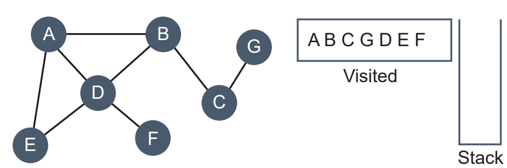
9. Push D onto the stack and visit E. Add it to the visited list. Colour it to show it has been visited.



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.

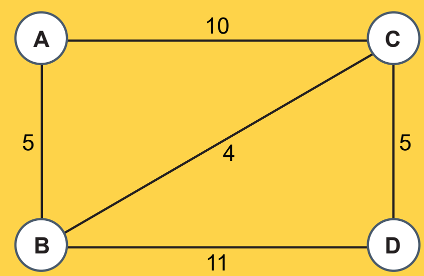
12-63

A-Level only

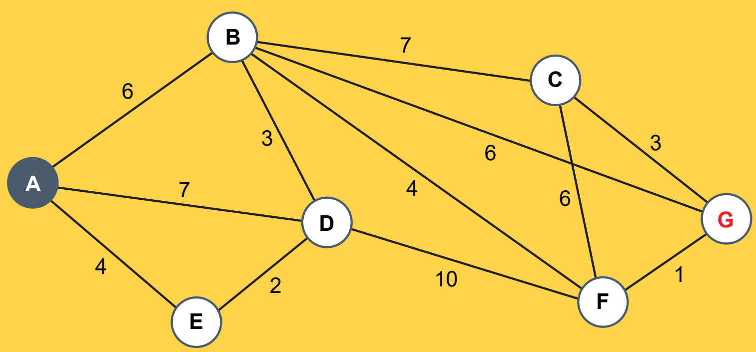
The queue is empty, all the nodes have now been visited so the algorithm ends.

We have found the shortest distance from A to every other node, and the shortest distance from A is marked in blue at each node.

Q1: Copy the graph below and use the method above to trace the shortest path from A to all other nodes. Write the shortest distance at each node.



Q2: Use a similar method to trace the shortest path from A to all other nodes. Write the shortest distance at each node. What is the shortest distance from A to G?



12-64

The A* algorithm

Dijkstra's algorithm is a special case of a more general **path-finding algorithm** called the **A* algorithm**. Dijkstra's algorithm has one **cost function**, which is the real cost value (e.g. distance) from the source node to every other node.

The A* algorithm has two cost functions:

1. g(x) – as with Dijkstra's algorithm, this is the real cost from the source to a given node.
2. h(x) – this is the approximate cost from node x to the goal node. It is a **heuristic function**, meaning that it is a good or adequate solution, but not necessarily the optimum one. This algorithm stipulates that the heuristic function should never overestimate the cost, therefore the real cost should be greater than or equal h(x).

The total cost of each node is calculated as f(x) = g(x) + h(x).

The A* algorithm focusses only on reaching the goal node, unlike Dijkstra's algorithm which finds the lowest cost or shortest path to every node. It is used, for example, in video games to enable characters to navigate the world.

A

Index

A
1NF, 89
2NF, 91
3NF, 91
A* algorithm, 361
abstract data type, 184
abstraction, 260
  by generalisation, 262
  data, 263
  problem, 279
  procedural, 268
accumulator, 4
ACID, 107
active tag, 19
actuator, 23
adder
  full, 238
  half, 238
address bus, 2, 3, 8
addressing modes, 72
adjacency
  list, 210
  matrix, 210
ADT, 184
agile modelling, 55
algorithm, 57, 288
  recursive, 337
algorithms
  and ethics, 252
  comparing, 328
  sorting, 340
alpha testing, 53
ALU, 4
Amazon, 249
analysing personal
  information, 245
analysis, 52
AND, 176, 296
AND gate, 224
API, 152
application layer, 122
application software, 41
arithmetic
  operations, 289
  shift instructions, 174
Arithmetic-Logic Unit, 2, 4
array, 186
  1-dimensional, 179
  2-dimensional, 181
  n-dimensional, 181
artificial intelligence, 253
ASCII code, 160
assembler, 44, 69
assembly language, 9, 69
attributes, 319
automated decision making, 252
automatic
  backup, 40
  updating, 40
automation, 280

B
backtracking, 283
barcode
  2D, 16
  linear, 16
  reader, 16
base case, 307
behaviours, 319
beta testing, 53
Big-O notation, 330
binary
  addition, 162
  fixed point, 165, 167
  floating point, 167
  number system, 155
  search tree, 215, 218, 338
  subtraction, 164
binary search, 335
  recursive algorithm, 337
BIOS, 38
bit, 159
bitwise manipulation, 174
black box testing, 53
block-structured languages, 272
Blu-Ray disk, 26
Boolean algebra rules, 229
Boolean operators, 296
branch instruction, 70
breadth-first
  search, 212, 356
  traversal, 353, 354
bubble sort, 340
bus, 2
  address, 2, 3, 8
  control, 3
  data, 2, 3
  system, 2
byte, 159
bytecode, 46

C
cache memory, 7
caching, 267
Caesar cipher, 77
call stack, 202, 309
camera-based reader, 18
Captcha, 138
capturing data, 106
Cascade Style Sheets, 130
case statement, 296
CD-ROM, 26
censorship, 255
Central Processing Unit, 2
character set, 159, 258
CIR, 4
circuit switching, 119
circular
  queue, 186
  shift instructions, 175
CISC, 12
class, 320
client-server networking, 147
client-side processing, 150
clock speed, 7
closed source, 42
cloud computing, 148
code
  generation, 48
  optimisation, 48
collision, 204
  resolution, 206
colour paradigms, 257
comments, 289
commitment ordering, 109
compiler, 44, 46
composite key, 84
compression, 75
  dictionary-based, 77
  lossless, 76
  lossy, 75
computable problems, 277
computational thinking, 260
Computer Misuse Act 1990, 244
computers in the workforce, 250
concurrent processing, 275
constants, 292

Index

constructor, 67, 321
control bus, 3
Control Unit, 2
co-processor, 13
Copyright Designs and Patents Act 1988, 244
core, 7, 13
CPU, 2
  multi-core, 13
cryptographic hash functions, 80
CSS, 130, 131
  script, 134
current instruction register, 4
cyber-bullying, 255

D

D-type flip-flop, 239
data
  abstraction, 184
  bus, 2, 3, 9
  capture, 106
  exchange, 107
  mining, 284
  redundancy, 92
  select and manage, 107
  types, 289
Data Protection Act 1998, 244
database
  locking, 108
  normalisation, 89
  relational, 88
de Morgan's laws, 228
decomposition, 269
DeepMind AlphaGo, 14
degree of a relationship, 83
denary, 155
depth-first
  search, 355
  traversal, 351
design, 52
device driver, 38
dictionary, 207
digital
  camera, 18
  certificate, 81
  signature, 80
digraph, 209
Dijkstra's algorithm, 358
direct addressing, 72
directed graph, 209
disk defragmenter, 40
divide and conquer, 58, 279
DNS, 113
domain name, 113
  system (DNS), 113
driverless cars, 252
dry run, 317
dual-core, 7
DVD-RW, 26
dynamic data structure, 182, 186

E

eBay, 249
embedded systems, 11
encapsulation, 184, 306, 322
encryption, 77
  asymmetric, 79
  private key, 79
  public key, 79
  symmetric, 79
entity, 82
Entity Relationship Diagram, 83
enumeration, 277
environmental issues, 253
evaluation, 53
exhaustive search, 277
extreme programming, 55

F

Fetch-Decode-Execute cycle, 5
FIFO, 184
File Transfer Protocol, 124
firewall, 126
First In First Out, 184
first normal form, 89
fixed point binary, 165
flat file database, 82
floating point
  addition and subtraction, 171
  binary, 167
folding method, 205
for … next, 301
foreign key, 84, 89
freeware, 42
FTP, 124
function
  exponential, 329
  linear, 329
  logarithmic, 330
  polynomial, 329
functional testing, 53
functions, 291, 303
  string-handling, 291

G

gateway, 121
getter messages, 321
Google, 249
GPU, 13
graph, 209
  traversals, 351
graphics processing unit, 13

H

hard disk, 25
Harvard architecture, 11
hash table, 204
hashing, 80
hashing algorithm, 204
  folding method, 205
heuristic methods, 285
hexadecimal number system, 156, 157
hierarchy chart, 269
HTML, 130
  form script, 138
  script, 133
  tags, 130
HyperText Markup Language, 130

I

IDE, 313
if … then … else, 294
immediate addressing, 72
indexed addressing, 72
indexing, 88
indirect addressing, 72
information hiding, 320, 323
inheritance, 65, 323
in-order traversal, 217, 219
installation53
instantiation, 321
Integrated Development Environment, 313
Internet, 111
  registrars, 112
  registries, 112

Index

interpreter, 45, 46
interrupt, 32
    service routine, 32
intractable problems, 284
IP address, 114
iteration, 299

J
JavaScript, 137
    arrays, 141
    code, 139

K
Karnaugh map, 233
kibibyte, 159

L
LAN, 114
laser scanner, 17
layout, 257
LCD monitor, 20
legislation, computing related, 243
lexical analysis, 47
library programs, 49
linear search, 334
link layer, 123
linked list, 192
    deleting an item, 197
    free space, 192
    inserting an item, 193
linker, 49
linking database tables, 89
list, 190
Little Man Computer, 69
loader, 30
local area network, 114
locking, 108
logic gates, 223
logical shift instructions, 174
low-level language, 44

M
MAC address, 121
machine code, 69
    instruction, 9, 72
mail server, 125
Many-to-Many relationship, 91
MAR, 4
masks, 176
MDR, 4
mebibyte, 159
Media Access Control address, 121
memory
    address register, 4
    buffer register, 4
    data register, 4
    management, 31
merge sort, 345
    space complexity, 347
    time complexity, 347
mesh network, 117
meta tags, 142
modular programming, 307
monitor, 20
monitoring behaviour, 257
multi-core CPU, 13
multimedia projector, 22
multi-tasking, 32

N
nested loops, 301
network layer, 123
normalisation
    of a binary number, 169
    of databases, 89
NOT, 176, 296, 297
NOT gate, 223
number bases, 155

O
object, 319
object code, 44
object-oriented programming, 319
offensive communications, 255
OLED, 20
opcode, 9
open source
software, 42
operand, 9
operating system, 30, 39
    distributed, 35
    embedded, 37
    mobile phone, 36
    multi-tasking, 35
    multi-user, 36
    real-time, 36
optical disk, 26
optimisation problems, 358
OR, 176, 296
OR gate, 224
output devices, 20
overflow, 162, 173
overriding, 324

P
packet
    filtering, 126
    switching, 119
PageRank, 144, 249
    algorithm, 143
paging, 31
parallel
    processing, 275
    systems, 13
partial dependency, 91
passing parameters
    by reference, 305
    by value, 305
passive tag, 19
path-finding algorithm, 361
PC, 4
peer-to-peer network, 148
pen-type reader, 17
performance modelling, 286
permutations, 330
pipelining, 8, 286
piracy, 149
polymorphism, 67, 324
POP3, 125
Post Office Protocol (v3), 125
post-order traversal, 217, 220
preconditions, 266
pre-order traversal, 216, 220
primary key, 83
primitive data type, 155
printer
    3-D, 22
    dot matrix, 22
    impact, 22
    inkjet, 22
    laser, 21
priority queue, 188
problem solving, 277
procedural
    abstraction, 268
    programming, 319
procedures, 303

Index

program
  constructs, 294
  counter, 4
programming language
  declarative, 64
  functional, 64
  object-oriented, 64, 65
  procedural, 64
programming paradigm, 64
proprietary software, 42
protocol, 122, 126, 152, 249
  stack, 81, 122
proxy server, 127
pseudocode, 288

Q
quad-core, 7
queue, 184
  operations, 185

R
RAD, 56
Radio Frequency Identification, 19
RAM, 25, 28
random access memory, 25, 28
rapid application development, 56
read-only memory, 28
record data structure, 182
recursion, 307
reference variable, 321
referential integrity, 85
register, 4
Regulation of Investigatory Powers Act 2000, 245
rehashing, 206
relation, 88
relational operators, 294
repeat … until, 300
resource management, 39
reusable program components, 266
RFID, 19
RISC, 12
RLE, 76
ROM, 28
root node, 214
rooted tree, 214
router, 121
Run Length Encoding, 76

S
scheduler, 32
scheduling algorithms, 33
  first come first served, 33
  multi-level feedback queue, 33
  round robin, 33
  shortest job first, 33
  shortest remaining time, 33
search engine indexing, 142
searching algorithms, 334
second normal form, 91
secondary
  key, 83
  storage, 25
segmentation, 31
selection statement, 294
serialisation, 108
server-side processing, 151
setter messages, 321
shift instructions, 174
sign and magnitude, 163
simulation, 184, 278
software, 41
  application, 41
  bespoke, 41
  development, 52
  freeware, 42
  off-the-shelf, 41
  open source, 42
  proprietary, 42
  system, 39
  utility, 40
solid state disk, 27
sorting algorithms, 340
  bubble sort, 340
  insertion sort, 343
  merge sort, 345
  quick sort, 347
source code, 44
space complexity, 347
speaker, 23
spiral model, 54
SQL, 95
  ALTER TABLE, 102
  DELETE, 104
  INSERT INTO, 103
  JOIN, 98
  ORDER BY, 97
  SELECT .. FROM .. WHERE, 95
  UPDATE, 104
SSD, 27
stack, 200
  call, 202
  frame, 203
  overflow, 202
  underflow, 202
state, 319
static data structure, 186
static filtering, 126
stored program concept, 10
string conversion, 291
structural testing, 53
structured approach, 272
Structured Query Language, 95
subclass, 323
subroutines, 303
  advantages of using, 307
  user-written, 304
superclass, 323
switch/case statement, 296
symbol table, 47
synonym, 204
syntax analysis, 48
system
  bus, 2
  clock, 7
  vulnerabilities, 129

T
TCP/IP protocol stack, 122
test
  plan, 315
  strategies, 315
testing
  alpha, 53
  beta, 53
  black box, 53
  functional, 53
  structural, 53
  white box, 53
thick-client, 152
thin-client, 152
third normal form, 91
time complexity, 285, 328, 331, 334, 336
  of merge sort, 347
timestamp ordering, 109
topology

Index

  physical bus, 115
  physical star, 115
trace table, 60, 272, 299
transaction processing, 107
transport layer, 123
traversing a binary tree, 216
tree, 214
  child, 214
  edge, 214
  leaf node, 214
  node, 214
  parent, 214
  root, 214
  subtree, 214
  traversal algorithms, 219
Trojan, 127, 128
trolls, 255
truth tables, 223
tuple, 182
two's complement, 163

U
underflow, 173
undirected graph, 209
Unicode, 161
Uniform Resource Locator, 112
unit nomenclature, 159
URL, 112
user interface, 39
utility software, 40

V
variables, 292
  global, 306
  local, 306
Vernam cipher, 78
virtual machine, 38
virtual memory, 28, 31
virus, 127
virus checker, 40
visualisation, 282
von Neumann architecture, 10

W
WAN, 114
WAP, 117
waterfall model, 54
wearable technology, 21
web forms, 136
web server, 150
while … endwhile, 299
white box testing, 53
wide area network, 114
Wi-Fi, 116
WinZip, 41
Wireless Access Point, 117
word, 3
  size, 8
World Wide Web, 111
worm, 127

X
XOR, 176, 297
  gate, 224

Y
Yobi, 159
Yotta, 159

Z
Zebi, 159
Zetta, 159

OCR AS and A Level

Computer Science

The aim of this book is to provide detailed coverage of the topics in the new OCR AS and A Level Computer Science specifications H046 and H446. 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 twelve sections and within each section, each chapter covers material that can comfortably be taught in one or two lessons. Material that is applicable only to the second year of the full A Level is clearly marked. Sometimes this may include an entire chapter and at other times, just a small part of a chapter.

Each chapter contains exercises and questions, some new and some from past examination questions. Answers to all these are available to teachers only in a free Teacher's Supplement which can be ordered from our website **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:

'Away Day'
Mixed media on canvas, 61x61cm
© Hilary Turnbull
www.hilaryturnbull.co.uk

This book has been endorsed by OCR.

PG ONLINE

ISBN: 978-1-910523-05-6

9 781910 523056