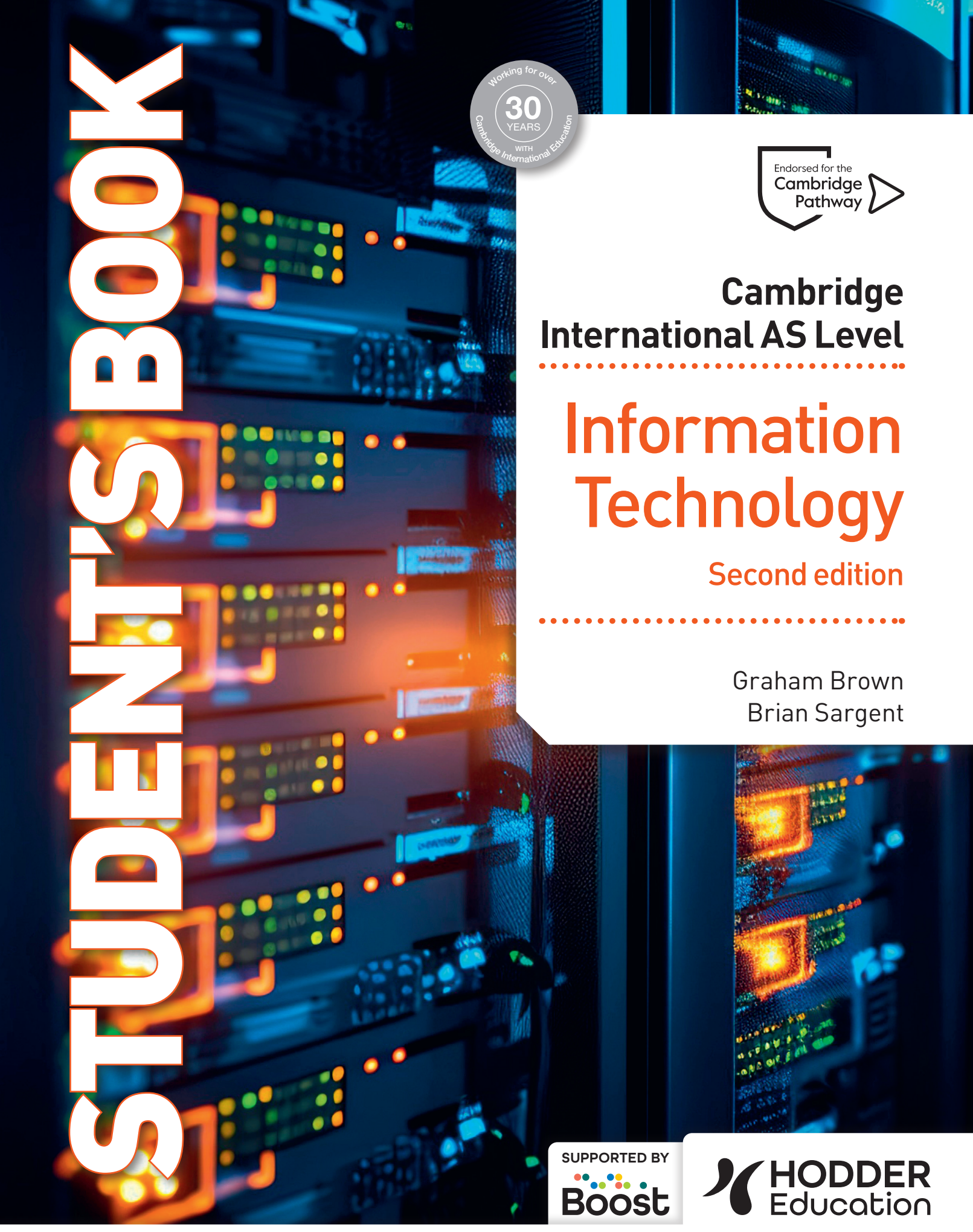


Working for over 30 YEARS with Cambridge International Education

Endorsed for the Cambridge Pathway

Cambridge International AS Level

Information Technology

Second edition

Graham Brown
Brian Sargent

SUPPORTED BY Boost

HODDER Education

Cambridge International AS Level

Information Technology

Second edition

Graham Brown
Brian Sargent

HODDER Education

Endorsement indicates that a resource has passed Cambridge International Education's rigorous quality-assurance process and is suitable to support the delivery of a Cambridge syllabus. However, endorsed resources are not the only suitable materials available to support teaching and learning, and are not essential to achieve the qualification. Resource lists found on the Cambridge website will include this resource and other endorsed resources.

Any example answers to questions taken from past question papers, practice questions, accompanying marks and mark schemes included in this resource have been written by the authors and are for guidance only. They do not replicate examination papers. In examinations the way marks are awarded may be different. Any references to assessment and/or assessment preparation are the publisher's interpretation of the syllabus requirements. Examiners will not use endorsed resources as a source of material for any assessment set by Cambridge International Education.

While the publishers have made every attempt to ensure that advice on the qualification and its assessment is accurate, the official syllabus, specimen assessment materials and any associated assessment guidance materials produced by the awarding body are the only authoritative source of information and should always be referred to for definitive guidance.

Our approach is to provide teachers with access to a wide range of high-quality resources that suit different styles and types of teaching and learning.

For more information about the endorsement process, please visit www.cambridgeinternational.org/endorsed-resources.

Third-party websites and resources referred to in this publication have not been endorsed by Cambridge International Education.

Answers to the practice questions and activities and all source files needed for the Student's Book can be downloaded from www.hoddereducation.com/cambridgeextras.

Photo credits: p50 [top] © Courtesy of International Business Machines Corporation, © International Business Machines Corporation; p50 [bottom] © Julian Herzog/CC BY (https://creativecommons.org/licenses/by/4.0); p93 [left] © Alvey & Towers Picture Library/Alamy Stock Photo; p93 [right] © David Jones/PA Images/Alamy Stock Photo; p93 [bottom] © Justin Kase zsixz/Alamy Stock Photo; p267 © MclittleStock/stock.adobe.com; p269 [left] © imageBROKER.com GmbH & Co. KG/Alamy Stock Photo; p269 [right] © Firas Nashed/stock.adobe.com; p271 © Designua/stock.adobe.com

Text credits: p46 Table 1.14 reproduced by permission of Lloyd's Register; p151 Figure 6.3 adapted from, 'Americans with lower incomes have lower levels of technology adoption' Pew Research Center, Washington, D.C. (21 June 2021) https://www.pewresearch.org/short-reads/2021/06/22/digital-divide-persists-even-as-americans-with-lower-incomes-make-gains-in-tech-adoption/ft_2021-06-22_digitaldivideincome_01/; Microsoft, (Access, Excel, Movie Maker, Photos, Windows and Word) are trademarks of the Microsoft group of companies; Apple Mac is a trademark of Apple Inc., registered in the U.S. and other countries and regions.

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

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

To order, please visit www.hoddereducation.com or contact Customer Service at education@hachette.co.uk/+44 (0)1235 827827.

ISBN: 978 1 0360 0560 3

First published in 2021
This edition published in 2024 by
Hodder Education,
An Hachette UK Company
Carmelite House
50 Victoria Embankment
London EC4Y 0DZ
www.hoddereducation.com

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

Year 2028 2027 2026 2025 2024

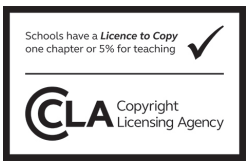
Cover photo © cookiecutter – stock.adobe.com

Illustrations by Barking Dog Art

Typeset in India by Aptara Inc

Printed in Slovenia

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



Contents list

Introduction

This textbook has been written to provide the knowledge, understanding and practical skills required by those studying the AS Level content (Topics 1–11) of the Cambridge International AS & A Level Information Technology syllabus (9626) for examination from 2025. Students studying the full A Level will also need to become familiar with the A Level content (Topics 12–21), covered in Hodder Education's *Cambridge International A Level Information Technology*.

How to use this book

This textbook, endorsed by Cambridge International Education, has been designed to make your study of Information Technology as successful and rewarding as possible.

Organisation

The book comprises 11 chapters, the titles of which correspond exactly with the topics in the syllabus. Each chapter is broken down into sections, which largely reflect the subtopics in the syllabus.

Features

Each chapter contains a number of features designed to help you effectively navigate the syllabus content.

At the start of each chapter, there is a blue box that provides a summary of the content to be covered in that topic.

In this chapter you will learn:

- ★ about the sensors and calibration used in monitoring technologies
- ★ about the uses of monitoring technologies
- ★ about the sensors and actuators used in control technologies
- ★ how to write an algorithm and draw a flowchart.

There is also a box that lists the knowledge you should have before beginning to study the chapter.

Before starting this chapter you should:

- ★ be familiar with the terms 'observation', 'interviews', 'questionnaires', 'central processing unit (CPU)', 'chip and PIN', 'direct access', 'encryption', 'file', 'key field', 'RFID', 'sort', 'validation' and 'verification'.

Chapters that require you to do practical work also feature a list of source files that you will need to use. These can be found here: www.hoddereducation.com/cambridgeextras.

For this chapter you will need these source files:

- TuckShop.csv
- Widget.csv

The practical chapters contain Tasks. The text demonstrates the techniques used to carry out the tasks. It provides easy-to-follow step-by-step instructions, so that practical skills are developed alongside the knowledge and understanding. Tasks often include the use of source files that you can download from www.hoddereducation.com/cambridgeextras.

Task 8e

Open and examine the file **Stock.csv**. Split this so that both types of stock can be viewed together. Save the spreadsheet as **Task_8e**.

Each chapter also includes Activities to allow you to check your understanding of the concepts covered and practise the skills demonstrated in the Tasks. In the practical chapters, these often require the use of source files from the website.

Activity 1a

Explain the difference between data and information.

Advice and shortcuts for improving your ICT skills are highlighted in Advice boxes.

Advice

A common error made by people writing algorithms with nested loops is not matching up the number of WHILE statements with the same number of ENDWHILEs. The same error can happen with REPEATs and UNTILs. You must always check this and make sure they are correctly indented.

Finally, each chapter ends with practice questions. These practice questions and their sample answers, as well as the activities throughout the book have been written by the authors.

Answers to the practice questions and activities and the source files needed for the Student's Book can be downloaded from www.hoddereducation.com/cambridgeextras.

Practice questions

1 A collection of data could be this: johan, Σ, $, <, AND

Explain why they are regarded as just items of data. In your explanation give a possible context for each item of data and describe how the items would then become information. [5]

2 A company uses computers to process its payroll, which involves updating a master file.

a State what processes must happen before the updating can begin. [2]

b Describe how a master file is updated using a transaction file in a payroll system. You may assume that the only transaction being carried out is the calculation of the weekly pay before tax and other deductions. [6]

3 a Name and describe **three** validation checks other than a presence check. [3]

b Explain why a presence check is not necessary for all fields. [3]

4 A space agency controls rockets to be sent to the moon.

Describe how real-time processing would be used by the agency. [3]

5 Describe **three** different methods used to carry out verification. [3]

Text colours

Some words or phrases within the text are printed in **red**. Definitions of these terms can be found in the glossary at the back of the book. In the practical section, words that appear in **blue** indicate an action or location found within the software package, for example 'Select the **Home** tab.' In the database sections of the book, words in **orange** show fieldnames. Words in **green** show the functions or formulas entered into the cell of a spreadsheet, for example a cell may contain the function **=SUM(B2:B12)**.

Assessment

The information in this section is taken from the Cambridge International Education syllabus. You should always refer to the appropriate syllabus document for the year of examination to confirm the details and for more information. The syllabus document is available on the Cambridge International Education website at www.cambridgeinternational.org.

If you are following the AS Level part of the course, you will take two examination papers: Paper 1 Theory (1 hour 45 minutes); Paper 2 Practical (2 hours 30 minutes).

Command words

The table below, taken from the syllabus, includes command words used in the assessment for this syllabus. The use of the command word will relate to the subject context. Make sure you are familiar with these.

Command word	What it means
Analyse	Examine in detail to show meaning, identify elements and the relationship between them
Assess	Make an informed judgement
Compare	Identify/comment on similarities and/or differences
Contrast	Identify/comment on differences
Define	Give precise meaning
Describe	State the points of a topic/give characteristics and main features
Discuss	Write about issue(s) or topic(s) in depth in a structured way
Evaluate	Judge or calculate the quality, importance, amount or value of something
Explain	Set out purposes or reasons/make the relationships between things clear/say why and/or how and support with relevant evidence
Identify	Name/select/recognise
Justify	Support a case with evidence/argument
State	Express in clear terms
Suggest	Apply knowledge and understanding to situations where there is a range of valid responses in order to make proposals/put forward considerations

Notes for teachers

Key concepts

These are the essential ideas that help learners to develop a deep understanding of the subject and to make links between the different topics. Although teachers are likely to have these in mind at all times when they are teaching the syllabus, the following icons are included in the textbook at points where the key concepts relate to the text (note that not all of these key concepts are relevant to the AS Level course and some will only feature in the A Level book).

Hardware and software

Hardware and software interact with each other in an IT system. It is important to understand how these work and how they work together with each other and with us in our environment.

Networks

Computer systems can be connected together to form networks allowing them to share data and resources. The central role networks play in the internet, mobile and wireless applications and cloud computing has rapidly increased the demand for network capacity and performance.

The internet

The internet is a global communications network. It uses standardised communications protocols to allow computers worldwide to connect and share information in many different forms. The impact of the internet on our lives is profound. While the services the internet supports can provide huge benefits to society they have also introduced issues, for example security of data.

System life cycle

Information systems are developed within a planned cycle of stages. They cover the initial development of the system and continue through to its scheduled updating or redevelopment.

New technologies

As the information industry changes so rapidly, it is important to keep track of new and emerging technologies and consider how they might affect everyday life.

Additional support

The *Cambridge International AS Level Information Technology Skills Workbook* is a write-in resource designed to be used throughout the course. It provides students with extra opportunities to test their understanding of the knowledge and skills required by the syllabus.

4 Algorithms and flowcharts

In this chapter you will learn how to:

- ★ write a basic algorithm that demonstrates a decision-making process
- ★ use conditional branching within an algorithm
- ★ use loops within an algorithm
- ★ use nested loops within an algorithm
- ★ include procedures/subroutines within an algorithm
- ★ edit a given algorithm
- ★ write an algorithm to solve a given problem
- ★ edit a given flowchart
- ★ draw a basic program flowchart that demonstrates a decision-making process
- ★ draw a program flowchart to solve a given problem
- ★ identify errors in an algorithm or program flowchart for a given scenario.

Before starting this chapter you should understand the:

- ★ comparison operators >, <, =
- ★ arithmetic operators +, -, *, /
- ★ order of arithmetical operations in an equation.

4.1 Algorithms

Before we look at how to write or edit an algorithm, let us consider what is meant by an algorithm. In computer science, information technology and mathematics, an algorithm is a set of instructions sequenced to solve a problem or to represent a calculation. Some would say that an algorithm is basically a data or **program flowchart** without the boxes. It is actually a list of precise steps. The order in which these steps are carried out is always crucial to the way an algorithm works. We start at the first line of the algorithm and work downwards to the final instruction. When an algorithm is created to produce information, data is input, processed, then the result is output. An algorithm must be carefully designed. It must be written in a way that caters for all possible scenarios. Any steps that rely on decisions being made must be dealt with in sequence and the conditions must be clear and unambiguous. Depending on the way the algorithm is written, not all instructions are carried out for a particular scenario, but they still have to be written in a way that allows for all possible scenarios.

Algorithms can be written in many different ways, including everyday natural language, **pseudocode**, **flowcharts** or programming languages. Natural language can sometimes lead to instructions which are open to interpretation and so tends not to be used in very complex algorithms. Pseudocode and flowcharts are structured ways of writing algorithms and we will concentrate on these approaches. Programming languages are primarily intended for converting algorithms to a form that can be understood and executed by a computer.

When solving problems in this chapter, the algorithms will be written in pseudocode. Pseudocode is independent of any programming language. Once the pseudocode is written and checked to make sure that it solves the problem,

it would be fairly straightforward to translate it into any programming language we were familiar with.

Any algorithm consists of statements which have **linear progression**, that is the result of one statement is used in the statements that follow. It may also contain **conditional branching**, such as **IF…THEN…ELSE** or **CASE…ENDCASE**, which results in a decision being made between two or more courses of actions. It will probably also involve the use of **loops** such as **WHILE…ENDWHILE** or **REPEAT…UNTIL**. A loop is a sequence of statements that are repeated a number of times.

Here are some key actions that are performed and the common terms used in pseudocode:

- inputting data: **INPUT** or **READ**
- outputting data: **WRITE** or **PRINT**
- calculation: +, -, *, /
- comparison: >, <, =, <>
- setting values: ←

Do not worry if you have not met some of these before as we will be going into much greater detail when we begin to write our own algorithms.

4.1.1 Writing an algorithm

By the end of this chapter, you will be able to write a basic algorithm that demonstrates a decision-making process. In order for you to be able to do this you will need to become familiar with a number of pseudocode terms. One of the most basic units of an algorithm is a variable. It is best to think of a variable as an area of the computer's memory that stores one item of data, such as a number. The algorithm designer is able to choose the names of the variables, making writing an algorithm easier, and also it is possible to write an algorithm where we can change the values each time we work the algorithm. Assigning a value to a variable name can be done using an arrow symbol, ←. The variable name is to the left of the sign and the value is to the right, for example:

```
X ← 42
```

means we want to store the number 42 in X.

We can also assign calculations and other types of processing this way, for example:

```
Z ← W + X + Y
```

This assigns the result of adding the contents of W, X and Y together and stores the result of this calculation in Z.

Input and output

Consider your use of calculators. When you want to do a calculation such as 42 × 36, three things have to happen. First of all, you have to type in the numbers, then the calculator's internal computer calculates the answer and finally it outputs the answer on the screen. The last two occur so quickly that you probably do not realise that the calculation occurs before the answer appears on the screen.

If we were writing an algorithm to describe this calculation it would simply be:

```
INPUT X, Y          // input two numbers, one is X, the other is Y
Z ← X*Y             // multiply the two numbers and call the answer Z
PRINT Z             // output the answer Z
```

It is written like this so that X and Y represent any two numbers.

We can think of Z as somewhere we store the result of X multiplied by Y.

The last line causes the answer (Z) to be output.

Notice we have used // to make comments. This just helps us to see what is going on but is not part of an instruction to be carried out.

If we use 42 and 36 as our numbers, Z would be 42*36, which is 1512.

We could use any letters or words instead of X, Y and Z.

We could have written:

```
INPUT firstnumber, secondnumber
answer ← firstnumber*secondnumber
PRINT answer
```

This is an example of how writing an algorithm can be made simpler just by using variable names that make sense and result in the algorithm being easier to follow. Always try to use variable names which give a clue to what they are storing.

The last statement could also be written as:

```
PRINT "The answer is", answer
```

This would result, using our calculation, in the following output:

```
The answer is 1512
```


Conditional branching

This is sometimes referred to as selection. Conditional branching means that certain statements are carried out depending on the value stored within a particular variable name. There are generally considered to be two types, which are IF…THEN…ELSE and CASE…ENDCASE.

IF...THEN...ELSE

From now on, whenever you see words within angle brackets, this is just an explanation of what you will be typing in and not the exact words themselves, for example <statement> would mean that a statement must go here, such as:

```
A ← B+C
```

Sometimes, IF statements do not have an ELSE part, such as in this sequence of instructions:

```
IF <condition>
   THEN
   <statement or list of statements to be carried out>
ENDIF
```

An example of this type is:

```
INPUT number1, number2
X ← number1/number2
   IF number2 > number1
      THEN
         X ← number2/number1
   ENDIF
PRINT X
```

Here, two numbers are input and the first number is divided by the second number, with the result being stored in X. The IF condition checks to see if the second number is bigger than the first. If it is, then the second number is divided by the first number and that result is stored in X, overwriting the previous result. If it is not, then X remains unchanged. The value in X is then output.

It is important to note that IF statements are slightly indented. It does not matter how much the indentation is, as long as you are consistent and use the same amount of indentation for each IF.

THEN statements are further indented and again, you just need to be consistent. The statement after THEN is further indented. ENDIF must always be in line with the IF statement. Any statements that come after ENDIF are not indented.

IF statements with an ELSE clause are written as follows:

```
IF <condition>
   THEN
      <statement or list of statements to be carried out>
   ELSE
      <alternative statement or list of alternative statements to be
      carried out>
ENDIF
```

Using our example above, a better way of writing the algorithm would be:

```
INPUT number1, number2
   IF number2 > number1
      THEN
         X ← number2/number1
      ELSE
         X ← number1/number2
   ENDIF
PRINT X
```

Here, as before, the value of X is the bigger of the two numbers divided by the smaller number.

The algorithm inputs two numbers. It checks to see if the second number is bigger than the first and if it is then it divides the second number by the first number, storing the answer in X. If it is not bigger (ELSE) it divides the first number by the second number, storing the answer in X.

Again, it is important to note the indents and the fact that ELSE is immediately in line with THEN. The statement below THEN is in line with the statement below ELSE.

Sometimes it is necessary to have what are called nested IF statements. This is an IF condition within another IF. You will have met these when using spreadsheet software during your practical lessons.

Let us consider an exam system where if a student gains 75 or more marks they are awarded a distinction; if they get between 60 and 74 marks they are awarded a merit; between 40 and 59 marks the student is awarded a pass; otherwise they fail the course.

This is the type of problem which requires nested IF statements to solve it. One solution is as follows:

```
1  INPUT mark
2     IF mark <75
3        THEN
4           IF mark <60
5              THEN
6                 IF mark <40
7                    THEN
8                       PRINT "Sorry you have failed"
9                    ELSE
10                      PRINT "You have passed"
11                ENDIF
12             ELSE
13                PRINT "Well done. You have been awarded a merit"
14          ENDIF
15       ELSE
16          PRINT "Congratulations you have been awarded a distinction"
17    ENDIF
```

Each line of pseudocode is numbered. This does not normally tend to be the case but it has been done here to make the explanation of this algorithm easier to follow.

Consider marks of 34, 45, 62 and 78. We will work through the algorithm for each mark.

With an IF statement we follow the THEN part when the IF statement is true but we only follow the ELSE part if the IF statement is false.

For the first mark of 34:

Statement 1 will store 34 in the variable name 'mark'.

Statement 2 is true so we go on to statements 3 and 4.

Statement 4 is true so we go on to statements 5 and 6.

Statement 6 is also true so we move on to statements 7 and 8, which cause the message 'Sorry you have failed' to be printed out.

We can go on to statement 11 as we can ignore statements 9 and 10 which are only followed if statement 6 was false.

Statement 11 is just there to tell us that statement 6 is now closed.

We can ignore statements 12 and 13 which would only be followed if statement 4 were false.

Statement 14 tells us we have reached the end of the IF in statement 4.

Again, we can ignore statements 15 and 16 as statement 2 is not false, which brings us to statement 17, which shows us that we have reached the end of the whole sequence of the nested IF.

Briefly then, for the mark of 45, statements 2 and 4 are true, so we reach statement 6 which is false. As a result, we jump to the ELSE statement 9 and reach statement 10 which causes the message 'You have passed' to be printed out. We then ignore all the remaining ELSE statements.

For the mark of 62, statement 2 is true but statement 4 is false, so we jump to the corresponding ELSE statement to the line 5 THEN statement, which is line 12. That causes us to go on to line 13, which causes the message 'Well done. You have been awarded a merit' to be printed out.

For the mark of 78, line 2 is false so we jump straight to the corresponding ELSE statement in line 15 which leads to line 16 being printed out 'Congratulations you have been awarded a distinction' and then we reach line 17.

Always make sure that the number of ENDIFs matches the number of IFs and that they are indented to match. Always make sure the ELSEs are in line with the THENs.

CASE...ENDCASE

With this type of condition, depending on the value input, the algorithm carries out one of a number of statements. The condition can be phrased differently but we shall use CASE <identifier> OF, though some would use CASE OF <identifier>. Whichever you choose to use, just be consistent.

The value input is usually a number which then determines which statement will be carried out.

```
CASE <identifier> OF
   <value 1> : <statement or list of statements>
   <value 2> : <statement or list of statements>
   OTHERWISE : <statement or list of statements>
ENDCASE
```

Depending on the value entered, either one statement or several statements can be carried out. For example:

```
INPUT grade
CASE grade OF
   'distinction':     X ← 75
                      Y ← 100
   'merit':           X ← 60
                      Y ← 74
   'pass':            X ← 40
                      Y ← 59
   'fail':            X ← 0
                      Y ← 39
ENDCASE
PRINT "Your mark must have been between ",X, " and ",Y
```


This algorithm allows you to type in your grade as a word. The values of X and Y are set accordingly depending on the word you type in. The range of marks that your mark must have been within is then printed out after the phrase 'Your mark must have been between'. Note the comma after the quotation marks. Commas are used to separate variable names from the words that are printed out.

For example, if 'distinction' is typed in, then X is set to 75 and Y is set to 100 and the algorithm jumps immediately to ENDCASE and then prints out the message: 'Your mark must have been between 75 and 100'.

Notice neither the quotes nor the commas are printed out.

If the word 'merit' had been typed in then the first case ('distinction') is ignored and the second case is carried out, then it jumps to ENDCASE and prints out the appropriate message. With 'pass', it goes straight to the third case then ENDCASE before printing the message, and with 'fail', it goes to the fourth case then ENDCASE and prints out the message.

It is quite common for CASE…ENDCASE to have numbers input and the appropriately numbered CASE statement to be carried out. Here is an example which includes an OTHERWISE statement.

```
INPUT number
CASE number OF
      1 : PRINT "This is the number 1"
      2 : PRINT "This is the number 2"
      OTHERWISE: PRINT "You have entered an unacceptable number"
ENDCASE
```


The OTHERWISE case must always be the last case statement.

There are a number of variations of the CASE…ENDCASE statement but you will need to be consistent with your use of any variation to the method given above.

Activity 4a

1 Write an algorithm using IF...THEN which will input a number, then print out the number multiplied by 8 if it is less than 3, multiplied by 4 if it is between 3 and 5, and multiplied by 2 if it is any other number.
2 Write an algorithm using CASE...ENDCASE which will input a number, then print out the number divided by 2 if it is less than 6, divided by 5 if it is between 6 and 10, and divided by 10 if it is any other number.

Loops

When referring to the writing of computer programs or algorithms, a loop is a sequence of repeated statements that are carried out a number of times. This number may be just one or it can be a very large number, depending on the problem which needs solving. There are three ways a loop operates.

- One way is to use a count-controlled loop. The first statement in the loop specifies the size of the loop (how many times the subsequent statements are to be carried out). The final statement in the loop is just a marker so that the first statement is then returned to. The statements in the loop are repeatedly carried out until the number of times the statements within the loop are to be carried out (as shown in the first statement) has been met.
- Another way is sometimes referred to as a pre-condition loop, meaning the conditional statement is checked before (pre) any of the sequence of statements is followed. This is when a condition is checked to see if it is met and if it is, the sequence of statements is carried out. When the last statement is reached, the conditional statement is returned to. The condition is checked again and the whole process is repeated. Once the condition has not been met, it jumps to the last statement and exits the loop.
- The third type of loop is sometimes referred to as a post-condition loop. As its name suggests, the conditional statement is only checked after the sequence has been carried out for the first time. If the condition is not met, then the first statement is returned to. This is repeated until the condition is met, in which case the next statement after the close of the loop is carried out. The loops which will be described here are the WHILE…ENDWHILE loop (a pre-condition loop) and the REPEAT…UNTIL loop (a post-condition loop).

Before describing the different types of loop, the concept of counting within an algorithm has to be considered. If a loop is to be repeated a certain number

of times, it is usual to set up a counter. This usually involves the initialising of a variable (setting it to 0). The variable name used is often just 'count'. It is quite common for the first statement in any algorithm to be:

```
count ← 0
```

This count is then incremented (increased by 1) every time a sequence of statements is carried out.

```
count ← count + 1
```

The loop can then be set up to be executed an exact number of times. This is done by basing the conditional statement on whether the count has reached the number of times we want the loop to be carried out.

Although, as we shall see, it is possible to have WHILE…ENDWHILE and REPEAT…UNTIL loops which use counts, these loops can be controlled using other means. The condition that has to be met doesn't necessarily have to involve a count. The FOR…NEXT loop on the other hand is purely a count-controlled loop.

FOR...NEXT

The FOR…NEXT loop looks like this

```
FOR <identifier> ← <lowvalue> TO <highvalue>
   <a statement or a list of statements to be carried out>
NEXT <identifier>
```

The identifier, lowvalue and highvalue must all be integers.

If lowvalue = highvalue then the statements are carried out once only. If lowvalue is greater than highvalue then the statements are not carried out. The NEXT instruction adds one to the count and goes back to the FOR statement.

When highvalue is greater than lowvalue (which is normally the case), the value of the identifier is set to that of the lowvalue and the statement or statements are carried out. The identifier's value then increases by one (it increments) and then the statements are carried out again. This process is repeated until the value of the identifier is the same as highvalue and the statement(s) are carried out for the final time.

An example of a FOR…NEXT algorithm would be.

```
INPUT number
FOR count ← 1 TO number
   PRINT count, "x 8 = ", count  * 8
NEXT count
```

This algorithm takes as input the number of times the loop is to be carried out – 'number'. As long as the value of 'number' is a whole number greater than 0 it carries out the PRINT statement. To begin with, providing the number which has been input is a whole number greater than one, the identifier 'count' is set to 1 and the NEXT statement increases it by 1. When the count is equal to the value of 'number' the statement is carried out one more time before leaving the

loop except, of course, if the number input is 1. This algorithm is printing out the multiplication table for 8, often called the 8 times table, up to the number that is input. If the number input was 5 it would calculate 1*8, 2*8, 3*8, 4*8 and finally 5*8 before exiting the loop.

FOR...NEXT...STEP

It is possible to write a counting algorithm where the count does not increase by one every time but another number.

This looks like:

```
FOR <identifier> ← <lowvalue> TO <highvalue> STEP <incremental value>
   <a statement or a list of statements to be carried out>
NEXT <identifier>
```

The incremental value must be an integer such as 2, 3, 4, etc. The identifier will be given the values from lowvalue in successive increments of incremental value until it reaches highvalue. If identifier becomes greater than the highvalue, the loop stops without carrying out any more statements. For example, FOR count ← 1 TO 12 STEP 3 would result in count taking the values 1, 4, 7, 10, 13 on successive runs of the algorithm but when it got to count being 13 it would stop without carrying out the statements.

The incremental value can be negative. For example,

```
INPUT number
FOR count ← number TO 1 STEP -1
PRINT count, "x 8 = ", count  * 8
NEXT count
```

If we input the number as 5, this would produce the 8 times table but written backwards. It would calculate 5*8, 4*8, 3*8, 2*8 and finally 1*8 before exiting the loop.

WHILE...ENDWHILE

The WHILE…ENDWHILE loop looks like this:

```
WHILE <condition>
   <a statement or a list of statements to be carried out>
ENDWHILE
```

Some variations of this loop have the word 'DO' after the condition:

```
WHILE <condition> DO
```

We will not be using this in this book.

While the condition is met, the statements between the WHILE and ENDWHILE statements will be carried out. As soon as the conditional statement stops being true, the loop ends.

An example of a WHILE…ENDWHILE algorithm using a counter would be:

```
count ← 0
INPUT number
WHILE count < number
   count ← count + 1
   PRINT count, "x 10 = ", count*10
ENDWHILE
```

This algorithm takes as input the number of times the loop is to be executed – 'number'. It checks whether the count is less than this number before proceeding with the sequence of statements to be carried out. The first statement increments the counter so that it goes up by one before the calculation is performed. This algorithm is printing out the multiplication table for 10, often called the 10 times table, up to the number that is input. If the number input was 6 it would calculate 1*10, 2*10, 3*10, 4*10, 5*10 and finally 6*10 before exiting.

REPEAT...UNTIL

Unlike the WHILE…ENDWHILE loop, which tests the condition at the start of the loop, the REPEAT…UNTIL loop checks the condition at the end of the loop. It is similar to a WHILE…ENDWHILE loop, except that a REPEAT…UNTIL loop is executed at least once.

The REPEAT…UNTIL loop looks like this:

```
REPEAT
   <a statement or a list of statements to be carried out>
UNTIL <condition>
```

The statements between the REPEAT and UNTIL are carried out and from then on, for as long as the UNTIL condition is not met, they will be repeated. After the first time, they will only be carried out while the conditional statement is false. As soon as the condition becomes true, the loop ends.

We can convert the example above which uses the WHILE…ENDWHILE loop into the following:

```
count ← 0
INPUT number
REPEAT
   count ← count + 1
   PRINT count, "x 10 = ", count*10
UNTIL count = number
```

The only difference is that if you did not want anything printed out, you could type in 0 for the number. With the WHILE…ENDWHILE loop it would print nothing out. With the REPEAT…UNTIL loop it would print out:

```
1 x 10 = 10
```

despite the fact we might not want it to. This is a big disadvantage of the REPEAT…UNTIL loop, but as long as you want the loop to be executed at least once, it works well.

You may use another technique for finishing a loop. You can come out of a loop by inputting a rogue value (i.e. a value that is out of the ordinary). For example, supposing you needed to write an algorithm which added up a set of numbers. If all the numbers to be added consisted of numbers less than 100 you could end the loop by inputting a rogue value such as 999 as follows.

```
Total ← 0
number ← 0
WHILE number <> 999
total ← total + number
INPUT number
ENDWHILE
PRINT total
```

When the WHILE condition is met for the first time the value of number is 0, so the statements within the loop are executed. It will only terminate when number is 999. It will carry on increasing the total by adding the number that is input each time. As soon as 999 is input the loop stops and the total is output. The equivalent REPEAT…UNTIL algorithm would be:

```
Total ← 0
number ← 0
REPEAT
total ← total + number
INPUT number
UNTIL number = 999
PRINT total
```

Notice that in order to avoid adding 999 to the total, the input occurs *after* the total has been increased. The total has to be increased by the value of the number even when the number is equal to 0, which it is on the first pass.

Nested loops

A nested loop, as its name suggests, is a loop within a loop. Let us start with an example. We have already seen an algorithm which can output a multiplication table. Now we will look at an algorithm which will output several multiplication tables.

```
count1 ← 0
INPUT number1                  // this is the number of multiplication tables
                                  printed out
INPUT number2                  // this is the number we want to go up to in
                                  each table
WHILE count1 < number2
   count1 ← count1 + 1         // count1 is incremented so we do not
                                  get 0 × 0
   count2 ← 0
   WHILE count2 < number1
      count2 ← count2 + 1      // count2 is incremented so we do not get 1 × 0
      PRINT count1, "x ", count2, "= ", count1*count2
   ENDWHILE
ENDWHILE
```

The variable number1 is the number of multiplication tables we want and number2 is how far we want the table to go up to. For example, if 6 is input for number1, then the 1, 2, 3, 4, 5 and 6 multiplication tables will be printed out, and if 12 is input for number2 each table will go up to 12×.

Notice that both count1 and count2 are incremented immediately after the loop starts to prevent the algorithm from printing out 0×1 or 1×0. The WHILE… statement only checks whether the condition is met before the loop is allowed to start. When count2 gets to 6, it will have already printed out 1×6 and the count1 loop will increment then count2 will be set back to 0. This is repeated until count1 is 12 and count2 is 6.

The final output would be:

1 × 1 = 1	1 × 2 = 2	1 × 3 = 3	1 × 4 = 4	1 × 5 = 5	1 × 6 = 6
2 × 1 = 2	2 × 2 = 4	2 × 3 = 6	2 × 4 = 8	2 × 5 = 10	2 × 6 = 12
3 × 1 = 3	3 × 2 = 6	3 × 3 = 9	3 × 4 = 12	3 × 5 = 15	3 × 6 = 18
4 × 1 = 4	4 × 2 = 8	4 × 3 = 12	4 × 4 = 16	4 × 5 = 20	4 × 6 = 24
5 × 1 = 5	5 × 2 = 10	5 × 3 = 15	5 × 4 = 20	5 × 5 = 25	5 × 6 = 30
6 × 1 = 6	6 × 2 = 12	6 × 3 = 18	6 × 4 = 24	6 × 5 = 30	6 × 6 = 36
7 × 1 = 7	7 × 2 = 14	7 × 3 = 21	7 × 4 = 28	7 × 5 = 35	7 × 6 = 42
8 × 1 = 8	8 × 2 = 16	8 × 3 = 24	8 × 4 = 32	8 × 5 = 40	8 × 6 = 48
9 × 1 = 9	9 × 2 = 18	9 × 3 = 27	9 × 4 = 36	9 × 5 = 45	9 × 6 = 54
10 × 1 = 10	10 × 2 = 20	10 × 3 = 30	10 × 4 = 40	10 × 5 = 50	10 × 6 = 60
11 × 1 = 11	11 × 2 = 22	11 × 3 = 33	11 × 4 = 44	11 × 5 = 55	11 × 6 = 66
12 × 1 = 12	12 × 2 = 24	12 × 3 = 36	12 × 4 = 48	12 × 5 = 60	12 × 6 = 72

The same output would be produced using the following nested REPEAT… UNTIL loop:

```
count1 ← 0
INPUT number1
INPUT number2
REPEAT
   count1 ← count1 + 1
   count2 ← 0
   REPEAT
      count2 ← count2 + 1
      PRINT count1, "x ", count2, "= ", count1*count2
   UNTIL count2 = number1
UNTIL count1 = number2
```

The equivalent nested FOR…NEXT loop could be:

```
INPUT number1, number2
FOR count1 ← 1 TO number1
   FOR count2 ← 1 TO number2
      PRINT count1, "x", count2, " = ", count1*count2
   NEXT count2
NEXT count1
```

Advice

Common errors

An error sometimes made by people writing algorithms with nested loops is not matching up the number of WHILE statements with the same number of ENDWHILEs. The same error can happen with REPEATs and UNTILs. You must always check this and make sure they are correctly indented.

An IF statement checks if something is true and if it is, it carries out a matching statement. As we have seen above, if it is not true, the algorithm will either carry on or have an alternative statement to carry out which requires an ELSE statement. Sometimes, people forget to include a matching ELSE statement.

Procedures/subroutines

When writing algorithms, it is often good practice to use subroutines, which are sometimes called procedures, although technically a procedure is a subset of subroutines. A subroutine is a sequence of algorithm statements that perform a specific task. This subroutine can then be used in other algorithms as and when needed.

The subroutines or procedures follow the following pattern:

```
PROCEDURE <value>
   <a statement or list of statements>
ENDPROCEDURE
```

Let us consider a house builder who needs to know the area of the top part of a brick wall underneath the roof. He or she wants to calculate the number of bricks to be used. Such a house is shown in the figure below, with the area which needs to be calculated indicated as a red triangle. The other areas which would need to be calculated are rectangle shapes. An algorithm can be written to work out these areas and this could then be used to calculate the number of bricks needed. This could all be done by using simple subroutines.

The subroutines or procedures could be simply written as:

```
PROCEDURE triangle(width, height)
   area ← width*height*0.5
   PRINT area
ENDPROCEDURE
```

```
PROCEDURE rectangle(length, width)
   area ← length*width
   PRINT area
ENDPROCEDURE
```

The main algorithm could be:

```
INPUT width, height                    // type in the height and width of
                                          the triangle
CALL triangle (width, height)          // this would print out the area
INPUT number                           // type in the number of walls
count ← 0
WHILE count < number
   INPUT length, height
   count ← count + 1
   CALL rectangle (length, height)     // this would print out the area of
                                          each wall
ENDWHILE
```

Notice that although we used length and width as the variable names when we set up the rectangle procedure, we can use different variable names when we 'CALL' it. As long as we 'pass' two numbers to the procedure, it will perform the calculation.

Once the procedure or subroutine is set up, it can be called as many times as we want.

File handling

There are a number of file-handling functions used in pseudocode, but they are really outside the scope of this book. However, you will need to demonstrate an understanding of how file handling works, without necessarily knowing the technical terms. Let us consider the very basic algorithm that was constructed in Chapter 1.

```
First record in the transaction file is read
First record in the old master file is read
REPEAT
IDs are compared
IF IDs do not match, old master file record is written to new master
file
    IF IDs match transaction is carried out
        IF transaction is D or C, old master file record is not written
        to new master file
        IF transaction is C, data in transaction file is written to new
        master file
    IF IDs match, next record from transaction file is read
Next record from master file is read
UNTIL end of old master file
    Data in transaction file record is written to new master file
    Any remaining records of the transaction file are written to the
    master file
```

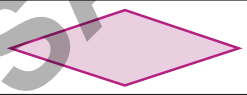

You should now be able to convert it into a properly structured algorithm such as this (notice we are using command words for the file input – READ and output – WRITE):

```
READ first record from the transaction file
READ first record from the old master file
REPEAT
   IF ID of old master file record <> ID of transaction file record
      THEN
         WRITE old master file record to new master file
      ELSE
         IF transaction = C
            THEN
                 WRITE data in transaction file record to new master file
         ENDIF
         READ next record from transaction file
   ENDIF
   READ next record from master file
UNTIL end of old master file
WRITE data in transaction file record to new master file
WRITE any remaining transaction file records to the master file
```

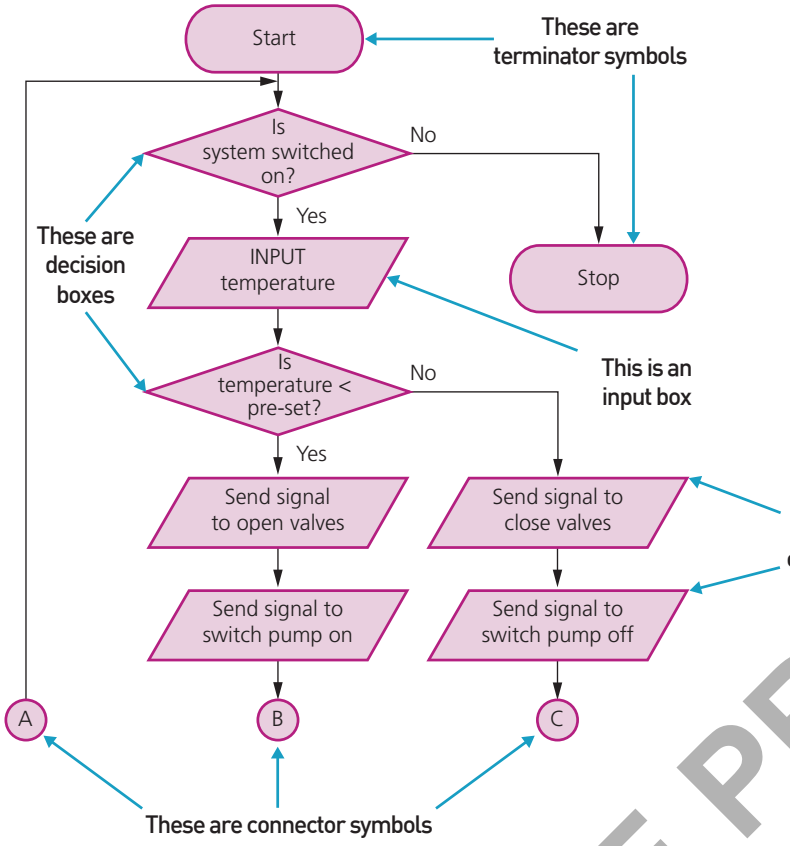
4.2 Flowcharts

There is another way of writing an algorithm and that is by using a program flowchart. A program flowchart is a diagrammatic way of representing an algorithm used to produce the solution to a problem. A flowchart consists of symbols, connected to each other by arrows which indicate the path to be followed when working through the flowchart. Each different shape symbol represents a different stage in the process. From now on when we mention the word flowchart we are referring to program flowchart. This is not the same as a system flowchart, which you will meet in the A2 text book. Below is a list of the more common symbols.

▼ **Table 4.1** Common flowchart symbols

Input/output		This shape symbol represents input or output. It is equivalent to either the INPUT or PRINT statement.
Decision		This is a decision box and is equivalent to the IF statement but is also used in loops.
Terminator (Start/Stop)		This is the symbol used to show where the flowchart begins and also where it ends. It is also used at the start and end of a subroutine.
Process box		This symbol represents any calculation or assigning of variables, usually contains the ← symbol.
Subroutine		This is the symbol used to call a subroutine from the main flowchart. It is equivalent to CALL.
Flow line		This is the symbol that shows which direction you should follow when working through the flowchart.
Connector		Sometimes a flowchart can extend over many pages. This symbol indicates the continuation of the flowchart.

Sometimes flowcharts are so complex that they occupy more than one page. To illustrate this, let us imagine that the central heating system flowchart shown in Chapter 3 has been extended. Here it is, labelled.



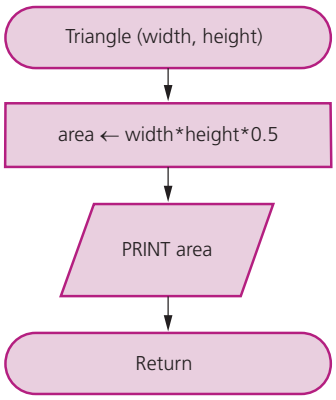
▲ **Figure 4.1** Labelled flowchart from Chapter 3

It is possible that this flowchart might have continued on to another page. The connector symbols have been added to the ends of the flow lines to show how they would be positioned, if this were the case. Connector symbols A, B and C would be drawn at the top of the next page with the appropriate flow lines flowing down to the next symbol(s) if the flowchart continued.

To illustrate how to draw a flowchart, let us consider the earlier algorithms:

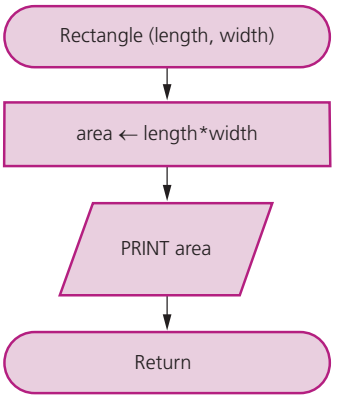
```
PROCEDURE triangle(width, height)
   area ← width*height*0.5
   PRINT area
ENDPROCEDURE
```

This becomes:



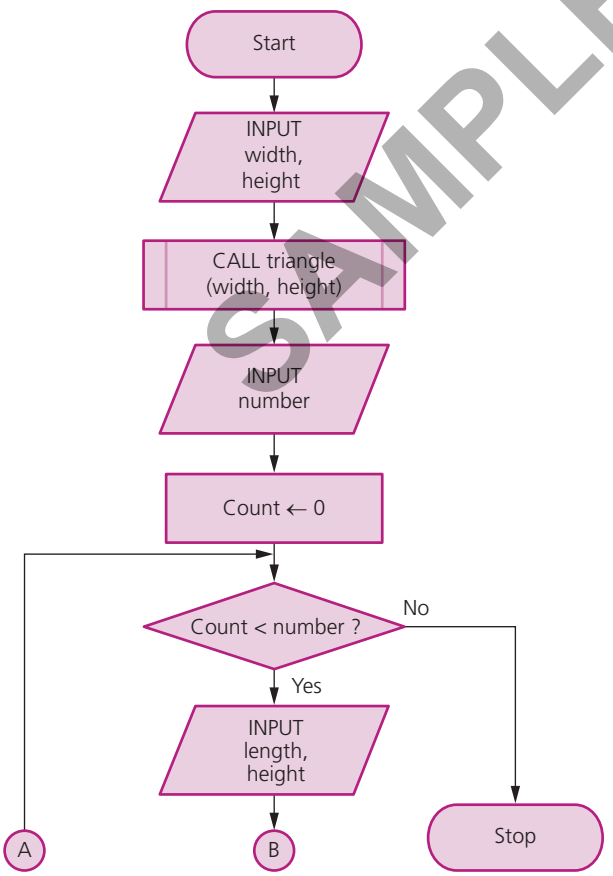
```
PROCEDURE rectangle(length, width)
   area ← length*width
   PRINT area
ENDPROCEDURE
```

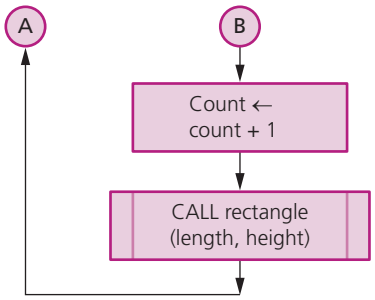
This becomes:




```
INPUT width, height                  // type in the height and width of the
                                        triangle
CALL triangle (width, height)        // this would print out the area
INPUT number                         // type in the number of walls
count ← 0
WHILE count <number
   INPUT length, height
   count ← count + 1
   CALL rectangle (length, height)
ENDWHILE
```

This becomes:



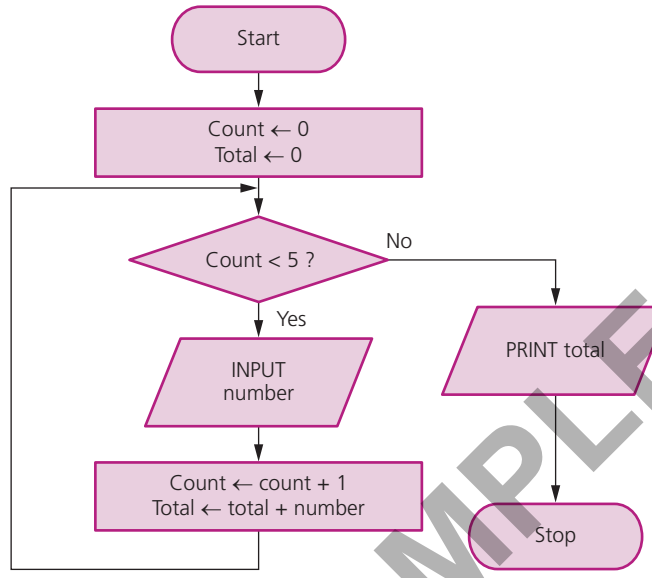


Notice the use of connector symbols because the flowchart goes over two pages.

As well as being able to increase the value of a counter, it is possible to add up a set of numbers in the same way, using:

```
total ← total + number
```

but you need to initialise the total as you would initialise a count. The flowchart to add up five numbers would look like this.



Common errors people make when drawing flowcharts and writing algorithms are:

- failing to initialise a count (leaving out count ← 0)
- failing to initialise a running total (leaving out total ← 0)
- getting the greater than (>) confused with the less than (<) symbol
- In flowcharts, having the yes and no the wrong way around.

Activity 4b

1 Look back at the section on greenhouses in Chapter 3. Refine the algorithms provided for temperature, moisture and light control to make them more efficient. Consider what should happen if the device is already switched off or switched on. Include extra statements to replace where the original algorithm has phrases like 'or leave on' and 'or leave off'.

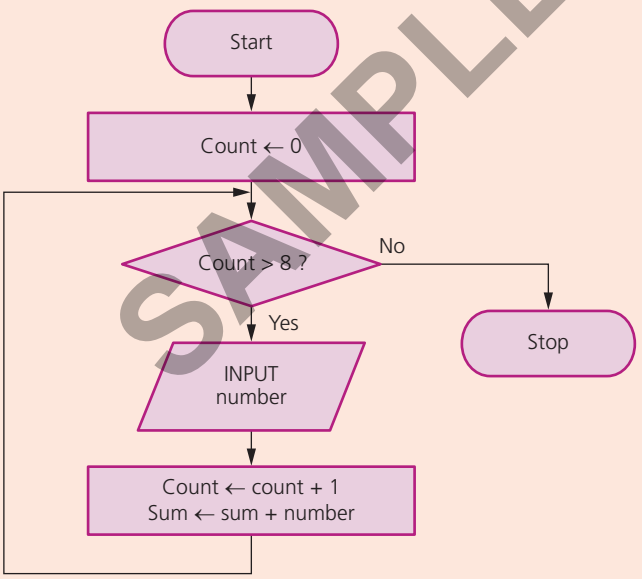
The sequence of statements needed for this will be similar to the following:

```
WHILE system switched on
   INPUT value
      IF value < pre-set
         THEN
            IF device off
                THEN
                      send signal to device to switch on
            ENDIF
         ELSE
            IF device on
                THEN
                      send signal to device to switch off
            ENDIF
      ENDIF
ENDWHILE
```

You will need to change the word 'device' to the name of the device being used. You will need to change 'value' to the physical variable being measured. Draw program flowcharts for each algorithm so that each is a subroutine with an appropriate name. You can assume the system is switched on.

2 Draw a simple program flowchart which would check if the system was on and would then call the three subroutines.

3 Here is a program flowchart to output the sum of eight numbers. Identify the errors in it and suggest improvements (the yes and no flow lines are positioned correctly).



Practice questions

1 Write an algorithm using WHILE...ENDWHILE conditions, which would input 10 exam marks and if the mark was greater than 40 the output would be 'You have passed'. If it was not, then the output would be 'You have not reached the pass mark'. [6]

2 Write an algorithm using a REPEAT...UNTIL loop which would input five numbers and would output the largest number. [4]

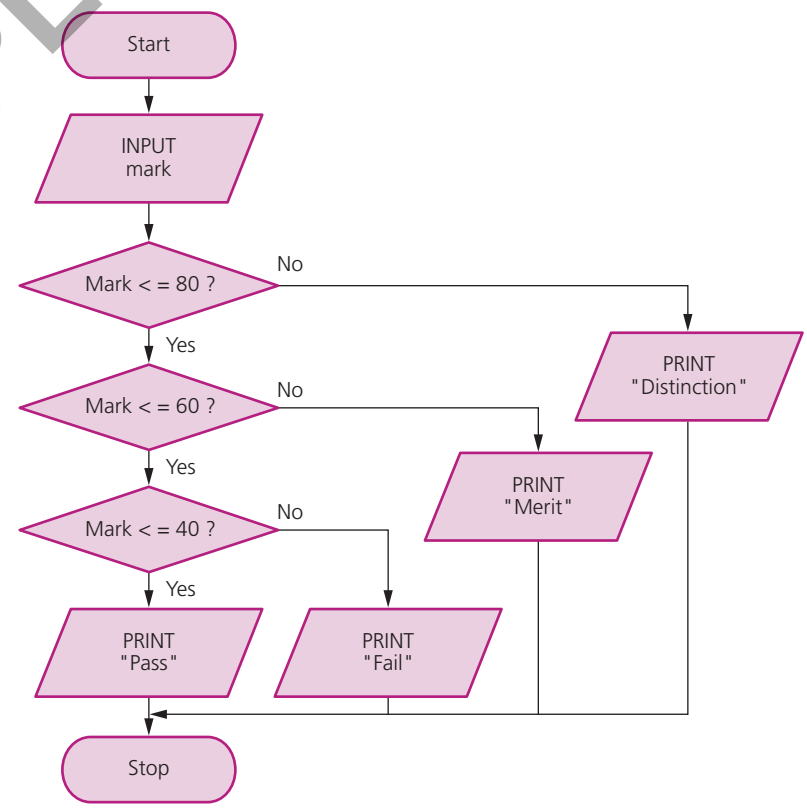
3 Draw a flowchart which will allow the average of six numbers to be calculated. [6]

4 A student wants to output the 10 multiplication (times) table up to a certain number, which will be input at the start of the algorithm. The student wishes to make sure that an invalid number is not input.

Identify the errors in this algorithm and suggest improvements. [5]

```
count ← 0
WHILE count = 0
   INPUT number
   IF number < 1
      THEN
         PRINT "number is invalid"
ENDWHILE
WHILE count < number
   PRINT count, "x 10 = ", count*10
ENDWHILE
```

5 This flowchart inputs an examination mark and then outputs an appropriate message. If the mark is 80 or more, the message is distinction; if it is 60 or more, it is a merit; if it is 40 or more, it is a pass; otherwise, it is a fail. Identify the errors in this flowchart and suggest how it might be edited to produce the required output. [2]



Develop theoretical and practical IT skills with this comprehensive Student's Book written by experienced authors and examiners. It is part of a two-volume set and fully covers Topics 1–11 which form the AS Level content of the updated Cambridge International AS & A Level Information Technology syllabus (9626) for examination from 2025.

- Improve understanding of concepts and terminology with clear explanations, photographs and diagrams, plus a glossary of key terms.
- Develop theoretical and practical skills with a range of exercises, practice questions, step-by-step instructions and example answers to ensure that skills are developed alongside knowledge.
- Follow a structured route through the course with in-depth coverage of the AS Level content.
- Answers to the practice questions and activities and all source files needed can be downloaded from www.hoddereducation.com/cambridgeextras.

This resource is endorsed for the Cambridge Pathway

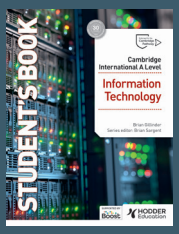
- ✓ Supports the AS Level content of the Cambridge International AS & A Level Information Technology syllabus 9626 for examination from 2025
- ✓ Has passed Cambridge International Education's detailed and independent quality-assurance process
- ✓ Developed by subject experts
- ✓ Accessible and appropriate for Cambridge schools worldwide

For over 30 years we have been trusted by Cambridge schools around the world to provide quality support for teaching and learning.
For this reason we are an Endorsement Partner of Cambridge International Education and publish endorsed materials for their syllabuses.

The Cambridge Pathway offers five stages of education from age 3 to 19, with curriculum, resources and assessment.

Registered Cambridge International Schools benefit from high-quality programmes, qualifications, assessments and a wide range of support so that teachers can effectively deliver in the classroom. Visit **www.cambridgeinternational.org** to find out more.

Also available:



Cambridge International A Level Information Technology

Boost

This title is also available as an eBook with learning support.

Visit hoddereducation.com/boost to find out more.

HODDER Education
education@hachette.co.uk
hoddereducation.com

ISBN 978-1-0360-0560-3