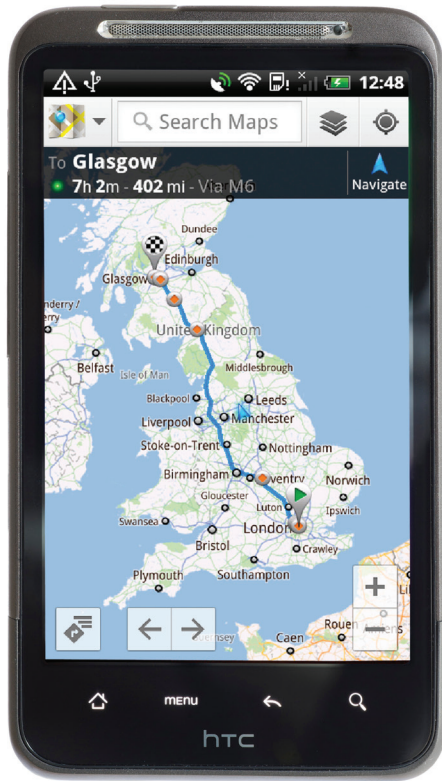


Chapter 1

Problem solving

Problem solving



Written instructions to the driver are given at the left of the map

Key terms

Unambiguous: this means that the instructions cannot be misunderstood. Simply saying 'turn' would be ambiguous because you could turn left or right.

All instructions given to a computer must be unambiguous or it won't know what to do.

Sequence: an ordered set of instructions.

Algorithm: a precise method for solving a problem. It consists of a sequence of step-by-step instructions.

1.1 Algorithms

Understanding algorithms

Learning outcomes

By the end of this section you should be able to:

- describe what an algorithm is.
- explain what algorithms are used for.
- express algorithms as flowcharts, pseudo-code and written descriptions.
- use and describe the purpose of arithmetic operators.

An example of an algorithm

An interactive map is a useful way to find a route between two locations. This image shows a route between two cities that was calculated by a mapping program.

- It is **unambiguous** in telling the driver exactly what to do, like 'turn left', 'turn right' or 'continue straight'.
- It is a **sequence** of steps.
- It can be used again and will always provide the same result.
- It provides a solution to a problem, in this case how to get from London to Glasgow.

A solution to a problem with these characteristics is called an **algorithm**. Most problems have more than one solution so different algorithms can be created for the same problem.

Did you know?

The computer program that created the algorithm to travel from London to Glasgow was following an algorithm of its own – an algorithm instructing it how to create another algorithm.

Successful algorithms

There are three criteria for deciding whether an algorithm is successful.

- **Accuracy** – it must lead to the expected outcome (e.g. create a route from London to Glasgow)
- **Consistency** – it must produce the same result each time it is run.
- **Efficiency** – it must solve the problem in the shortest possible time using as few computer resources as possible. In this example the mapping software is replacing a manual method, and if it were no faster than looking in an atlas then it would not be an improvement on the older method. Later in the topic there is a section on algorithms used to sort and search data. Some of these algorithms are more efficient than others and will sort the data far more quickly.

The relationship between algorithms and programs

Algorithms and programs are closely related, but they are not the same. An algorithm is a detailed design for a solution; a program is the implementation of that design.

This chapter is all about algorithms. We look at how algorithms are implemented in **high-level programming languages** in Topic 2. It's up to you whether you study these two topics sequentially or in parallel. So you could either study Topic 1 followed by Topic 2, or learn about algorithms and how to create them in this topic and at the same time consulting Topic 2 to find out how to translate algorithms into programs.

Displaying an algorithm

We carry out many everyday tasks using algorithms because we are following a set of instructions to achieve an expected result, for example making a cup of coffee. If we have performed the task many times before, we usually carry out the instructions without thinking, but if we are doing something unfamiliar, such as putting together a flat-pack chest of drawers, then we follow the instructions very carefully.

An algorithm can be expressed in different ways.

Written descriptions

A written description is the simplest way of expressing an algorithm. Here is an algorithm describing the everyday task of making a cup of instant coffee.

Algorithm for making a cup of instant coffee

Fill kettle with water.
Turn on kettle.
Place coffee in cup.
Wait for water to boil.
Pour water into cup.
Add milk and sugar.
Stir.

Flowcharts

Flowcharts can be used to represent an algorithm graphically. They provide a more visual display.

There are formal symbols that have to be used in a flowchart – you can't just make up your own because nobody else would be able to follow your algorithm.

Figure 1.1 shows the flowchart symbols that should be used.

Key terms

Flowchart: a graphical representation of an algorithm. Each step in the algorithm is represented by a symbol. Symbols are linked together with arrows showing the order in which steps are executed.

High-level programming language: a programming language that resembles natural human language.

Activity 1



Produce a written description of an algorithm for getting to school. It should start with leaving home and end with arriving at school.

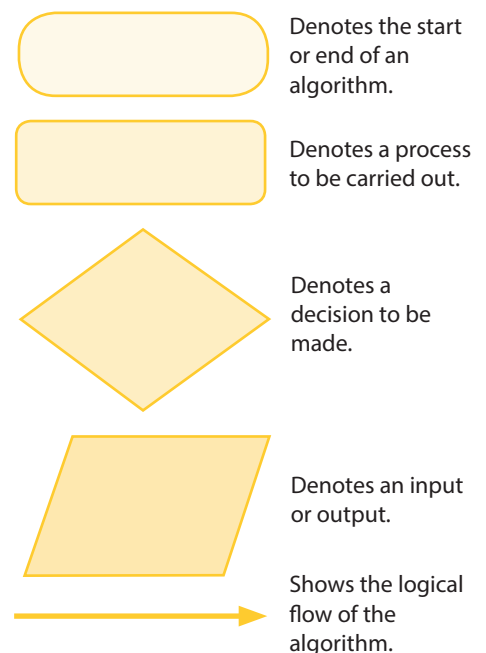


Figure 1.1 Flowchart symbols

Problem solving

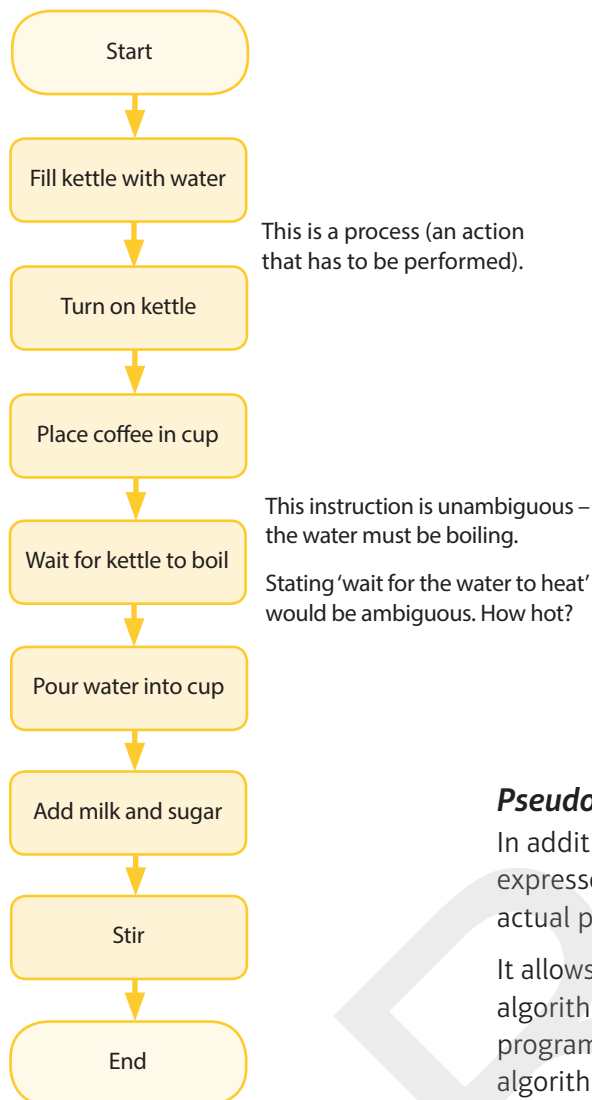


Figure 1.2 Flowchart of an algorithm to make a cup of coffee

Key term

Pseudo-code: a structured, code-like language that can be used to describe an algorithm.

The flowchart in Figure 1.2 is an alternative way of depicting the algorithm expressed above as a written description.

Activity 2



Display the 'journey to school' algorithm, which you created in the previous activity, as a flowchart.

The algorithms you have looked at so far are designed for humans to follow. Algorithms also form the basis of computer programs. A computer is a senseless machine that simply does exactly what it is told and follows a set of instructions, but computers can carry out these instructions far more quickly than humans. That is why they are so useful.

Did you know?



Algorithms for playing chess are widely used. After four moves by each opponent there are over 288 billion possible further moves – far too many for a human to consider, but within the range of computers. This is what makes it possible for a top-level computer program to defeat a chess grandmaster.

Pseudo-code

In addition to flowcharts and written descriptions, algorithms can also be expressed in **pseudo-code**. The pseudo-code is then translated into an actual programming language.

It allows the developer to concentrate on the logic and efficiency of the algorithm without having to bother about the rules of any particular programming language. It is relatively straightforward to translate an algorithm written in pseudo-code into any high-level programming language.

Activity 3



There are many different versions of pseudo-code and often they are unique to a particular organisation or examination board. Investigate the Edexcel pseudo-code that you will need for your GCSE course and which will be used in this book.

Example of a simple algorithm

To introduce the Edexcel pseudo-code, here is a simple algorithm that asks the user to input two numbers and then outputs the result of adding them together.

Written description

Algorithm for adding two numbers



Enter first number.
Enter second number.
Calculate total by adding first and second numbers.
Output total.

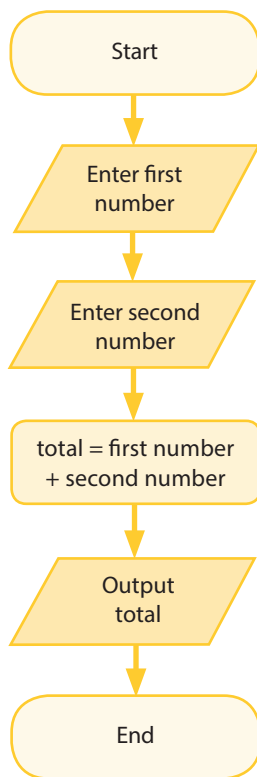


Figure 1.3 Flowchart showing the adding of two numbers

Pseudo-code

Algorithm for adding two numbers

```

SEND 'Please enter the first number.' TO DISPLAY
RECEIVE firstNumber FROM KEYBOARD
SEND 'Please enter the second number.' TO DISPLAY
RECEIVE secondNumber FROM KEYBOARD
SET total TO firstNumber + secondNumber
SEND total TO DISPLAY
  
```

The pseudo-code spells out the step-by-step instructions that the computer will be expected to carry out. It also introduces some important programming concepts.

- The numbers entered by the user are stored in two **variables** with the **identifiers** 'firstNumber' and 'secondNumber'.
- The result of adding the numbers together is stored in the variable 'total'.
- When some text is to be displayed, for example 'Please enter the first number.', it has to be enclosed in quotation marks, either single or double.
- When a variable is to be displayed, the quotation marks are not used. If they were, then, in the last instruction, the word 'total' would be displayed and not the number it represents.
- **Arithmetic operators** are used to perform calculations. This box shows the arithmetic operators.

Key terms

Variable: a 'container' used to store information. The information stored in a variable is referred to as a value. The value stored in a variable is not fixed. The same variable can store different values during the course of a program and each time a program is run.

Identifier: a unique name given to a variable or a constant. Using descriptive names for variables makes code much easier to read.

Arithmetic operator: an operator that performs a calculation on two numbers.

Arithmetic operators

Operator	Function	Example
+	Addition: add the values together.	8 + 5 = 13 myScore1 + myScore2
-	Subtraction: subtract the second value from the first.	17 - 4 = 13 myScore1 - myScore2
*	Multiplication: multiply the values together.	6 * 9 = 54 numberBought * price
/	Real division: divide the first value by the second value and return the result including decimal places.	13/4 = 3.25 totalMarks/numberTests
DIV	Quotient: like division, but it only returns the whole number or <i>integer</i> .	13 DIV 4 = 3 totalMarks DIV numberTests
MOD	Modulus: this will return the remainder of a division.	13/4 = 3 remainder 1 Therefore 13 MOD 4 = 1
^	Exponentiation: this is for 'to the power of'.	3^3 = 27 It is the same as writing 3 ³

Problem solving

Variables and constants

Variables play an important role in algorithms and programming. Although a variable can only store one value at a time, that value can change – it is variable. Variables are extremely useful in programming because they make it possible for the same program to process different sets of data.

A **constant** is the opposite of a variable. It is a 'container' that holds a value that always stays the same. Constants are useful for storing fixed information, such as the value of pi, the number of litres in a gallon or the number of months in a year.

Each variable and constant in an algorithm has to have a unique identifier. It is important to choose descriptive names for identifiers. This will make your code much easier to read. For example, a variable to hold a user's first name could be given the identifier 'firstName' so that it is indicative of the data it contains. If it were given the identifier 'X' then it would be anybody's guess what data it contained.

Key term

Constant: a 'container' that holds a value that never changes. Like variables, constants have unique identifiers.

Naming conventions for variables and constants

It is good practice to adopt a consistent way of writing identifiers throughout an algorithm.

A common convention is to use *camel case* for compound words (e.g. firstName, secondName) with no space between words and the second word starting with a capital letter. Alternatives are to capitalise the first letter of both words, e.g. FirstName, SecondName, or to separate the words with an underscore, e.g. first_name, second_name, known as snake case.

Activity 4

Here is a written description of an algorithm.

Enter the first number.

Enter the second number.

The third number is equal to the first number multiplied by the second number.

Display the third number.

Express this algorithm in pseudo-code.

Activity 5



This algorithm is displayed as a flowchart.

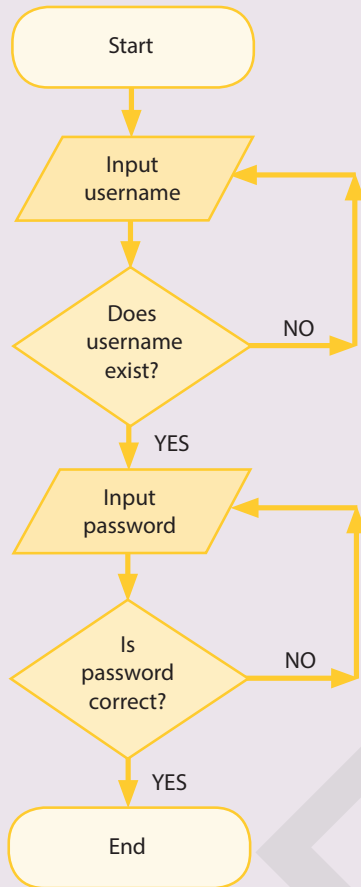


Figure 1.4 Flowchart of an algorithm

Produce a written description of this algorithm.

Extend your knowledge

When you enter a search term into Google®, a list of links to websites is returned. But why are they presented in that particular order? Research the PageRank algorithm that Google® uses to rate the importance of websites and write a short report about your findings that explains how the order is determined.

Summary

- An algorithm is a precise method for solving a problem.
- Algorithms can be displayed as written descriptions, flowcharts and in pseudo-code.
- Pseudo-code is a structured, code-like language.
- Pseudo-code is translated into program code.
- Arithmetic operators are used in calculations.
- Variables and constants are 'containers' for storing information. The value stored in a variable can change, whereas the value of a constant never changes.
- Selecting descriptive names for identifiers makes code easier to read.

Checkpoint

Strengthen

- S1** Produce a written description of an algorithm for borrowing a book from the library.
- S2** Describe what each of the seven arithmetic operators does.
- S3** Describe what a variable is and explain why variables are useful.
- S4** Explain the difference between a variable and a constant.

Challenge

- C1** Produce a flowchart describing an algorithm for making a cheese sandwich.
- C2** Write an algorithm expressed in pseudo-code that receives three numbers from the keyboard, calculates and displays the average.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following.

- For S1 reread page 3.
- For S2 study the table on page 5.
- For S3 and S4 look again at page 6.

Problem solving

Key terms

Construct: a component from which something is built. Letters and numbers (i.e. a to z and 0 to 9) are the constructs we use to build our language and convey meaning. Bricks and cement are the basic constructs of a building.

Selection: a construct that allows a choice to be made between different alternatives.

Iteration: a construct that means the repetition of a process. An action is repeated until there is a desired outcome or a condition is met. It is often referred to as a loop.

Creating algorithms

Learning outcomes

By the end of this section you should be able to:

- create an algorithm to solve a particular problem.
- use sequence, selection and iteration in algorithms.

Algorithms for computers

There was an ambiguous statement in the algorithm for making a cup of coffee on page 3. After filling the kettle with water and adding coffee to the cup, the next instruction was 'Wait for water to boil'.

A human can interpret this instruction as meaning that they have to keep checking the kettle over and over again until the water is boiling. But a computer is unable to interpret an instruction like that. It would just wait. And wait. Forever.

Even worse, the algorithm didn't explicitly say how to determine if the water was boiling. Through experience we humans assume the water is boiling when there is lots of steam, sound and bubbles; or, even better, when the kettle turns itself off. An algorithm for a computer would have to state that it was waiting until the water reached 100°C.

A version of this part of the algorithm, suitable for a computer, is shown in Figure 1.5.

This example introduces two new **constructs** from which algorithms are created.

We have already met the construct *sequence* – step-by-step instructions in the correct order. To add to this we now have **selection** and **iteration**.

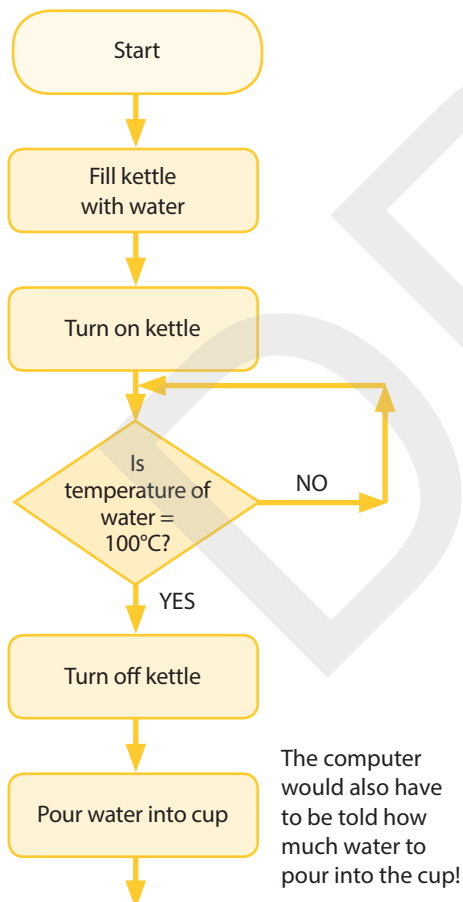


Figure 1.5 Part of an algorithm suitable for a computer for making coffee

Did you know?

We use iteration in our daily lives whenever we carry out an action over and over again. For example, at mealtimes we keep on eating until our plate is empty or we have had enough to eat. When we're travelling by car and the traffic lights are red we have to keep waiting until they change to green. An actor repeats their lines over and over again until they are word perfect.

Representing selection and iteration in a flowchart

Selection and iteration are represented in a flowchart as shown in Figure 1.6.

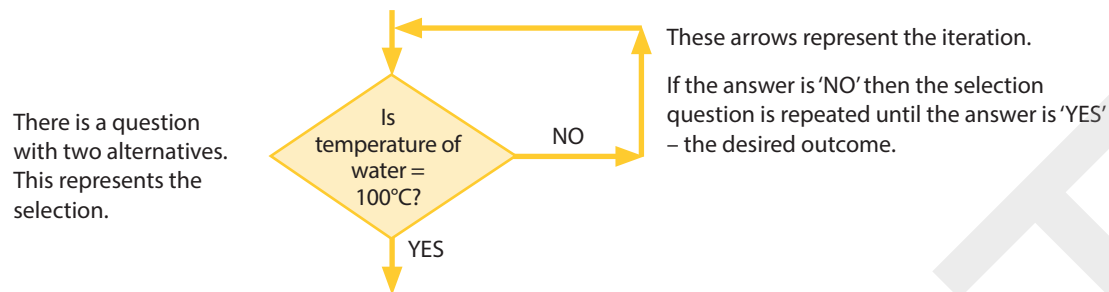


Figure 1.6 Selection and iteration in a flowchart

Representing selection and iteration in pseudo-code

Selection

Selection in pseudo-code is represented exactly as we would say it using an **IF...THEN...ELSE statement**.

IF...THEN...ELSE statement

```
IF Temperature = 100°C THEN
    Switch off kettle
ELSE
    Keep waiting
END IF
```

This is an oversimplification as the commands 'Switch off kettle' and 'Keep waiting' mean nothing to a computer.

An IF...THEN statement can be used without an ELSE if there is only one course of action to be taken, providing the condition in the IF statement is met.

IF...THEN statement

```
IF score >= 90 THEN
    SEND 'Excellent' TO DISPLAY
END IF
```

For selection, **relational operators** are used to compare the values.

Key terms

IF...THEN...ELSE statement:

the IF...THEN...ELSE statement allows a choice to be made between two alternatives based on whether or not a condition is met (e.g. IF it is cold THEN wear a jumper ELSE wear a T-shirt).

Relational operator: an operator that compares two values.

Problem solving

Relational operators

Relational operators are used to compare two values. The operators that you will be using are:

- = equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- <> not equal to

Using relational operators

```
SET passMark TO 75
RECEIVE mark FROM KEYBOARD
IF mark >= passMark THEN
    SEND 'Well done. You've passed.' TO DISPLAY
ELSE
    SEND 'Bad luck. You failed.' TO DISPLAY
END IF
```

Notice the use of indentation in an IF...THEN...ELSE statement. In this example there is only one statement for each alternative, but imagine if there were many. The indentation makes it easier to see which statement(s) belong to each alternative. Indentation is a useful technique for improving the readability of algorithms expressed in pseudo-code. You should get into the habit of using it. When you move on to implementing your algorithms in a high-level programming language you might find that the computer won't be able to execute your programs unless you have used indentation correctly.

Activity 6

```
IF score <= highScore THEN
    SEND 'You haven't beaten your high score.' TO DISPLAY
ELSE
    SEND 'You've exceeded your high score!' TO DISPLAY
END IF
```

State the output of the algorithm when

- score = 5 and highScore = 10
- score = 20 and highScore = 10
- score = 15 and highScore = 15

Activity 7



A driving school uses this rule to estimate how many lessons a learner will require.

- Every learner requires at least 20 lessons.
- Learners over the age of 18 require more lessons – two additional lessons for each year over 18.

Create an algorithm expressed in pseudo-code that inputs a learner's age and calculates the number of driving lessons they will need.

Nested selection

The IF...THEN...ELSE statement allows a choice to be made between two possible alternatives. However, sometimes there are more than two possibilities. This is where a **nested IF statement** comes in useful.

Key term

Nested IF statement: a nested IF statement consists of one or more IF statements placed inside each other. A nested IF is used where there are more than two possible courses of action.

Worked example

A learner handed in three homework assignments, which were each given a mark out of 10. All the marks were different. Write an algorithm that would print out the highest mark.

Figure 1.7 shows the algorithm expressed as a flowchart:

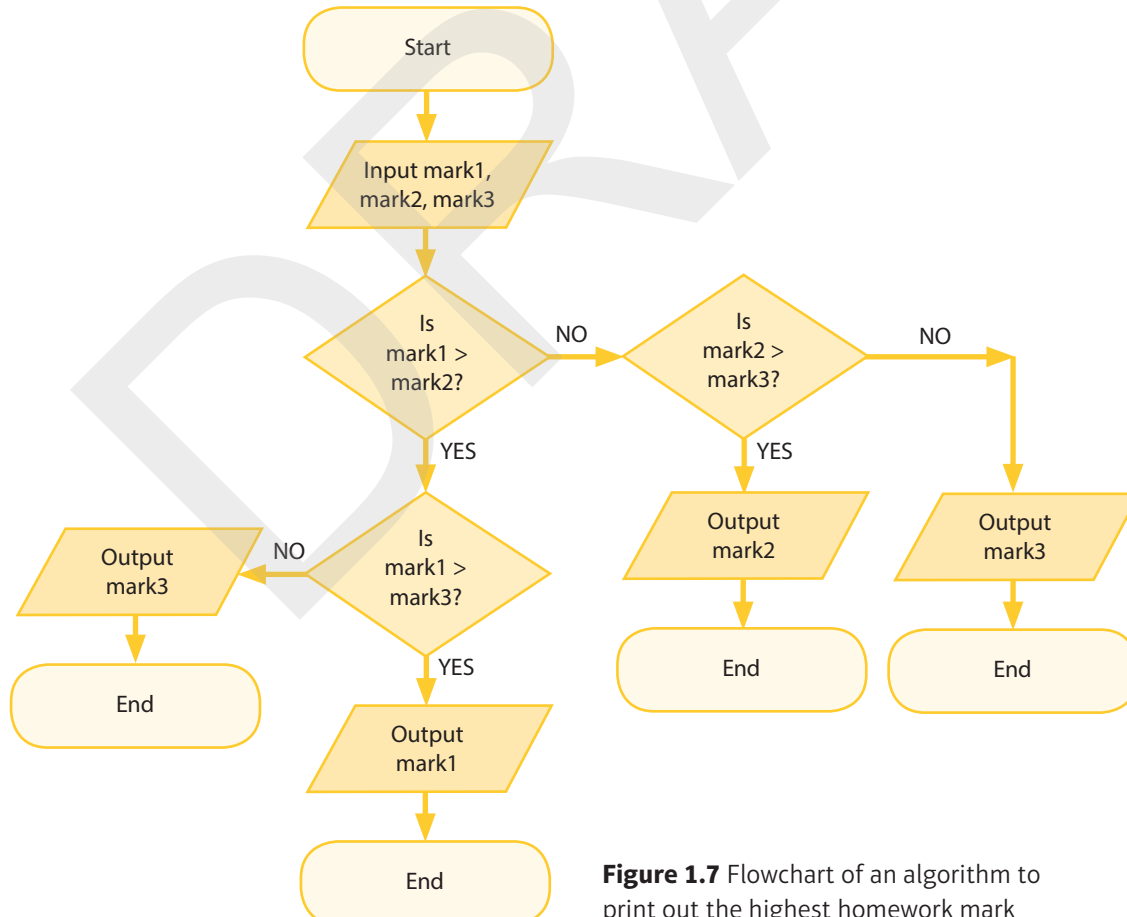


Figure 1.7 Flowchart of an algorithm to print out the highest homework mark

Problem solving

The variables mark1 and mark2 are compared using a relational operator. If mark1 is greater than (>) mark2 then it is compared with mark3. If it is not greater than mark2, then mark2 must be greater than mark1 and it is then compared with mark3.

This algorithm can also be expressed in pseudo-code:

```
RECEIVE mark1 FROM KEYBOARD
RECEIVE mark2 FROM KEYBOARD
RECEIVE mark3 FROM KEYBOARD
IF mark1 > mark2 THEN
    IF mark1 > mark3 THEN      #This is an IF statement
                              #within another IF statement.
                              #It is called a nested IF.
        SEND mark1 TO DISPLAY
    ELSE
        SEND mark3 TO DISPLAY
    END IF
ELSE
    IF mark2 > mark3 THEN      #This is another nested IF
                              #statement.
        SEND mark2 TO DISPLAY
    ELSE
        SEND mark3 TO DISPLAY
    END IF
END IF
```

Top tip

When you are creating nested IF statements you have to ensure that each one is completed with an END IF statement at the correct indentation level.

In this algorithm there are three IF...THEN...ELSE statements. Two of them are completely nested within the outer one.

Top tip

The # symbol indicates a comment. This is some text used to explain the code and the # symbol shows that it is not to be executed. It can be on a line of its own or at the end of the line to which it applies. You should get into the habit of adding comments to your algorithms to explain how they work and should do the same when writing program code.

Activity 8



A learner is creating a guessing game. A player has to enter a number no greater than 10. If it is too high, they are informed that they have made an error, but if it is within the range 1 to 10, they are told whether or not they have guessed the correct number. (Assume that the correct number is 3.) Create an algorithm to solve this problem and express it as a flowchart and in pseudo-code.

Activity 9



A school uses this algorithm to calculate the grade learners achieve in end-of-topic tests.

```
RECEIVE testScore FROM KEYBOARD
IF testScore >= 80 THEN
    SEND 'A' TO DISPLAY
ELSE
    IF testScore >= 70 THEN
        SEND 'B' TO DISPLAY
    ELSE
        IF testScore >= 60 THEN
            SEND 'C' TO DISPLAY
        ELSE
            SEND 'D' TO DISPLAY
        END IF
    END IF
END IF
```

What would be the output of this algorithm for these test scores: 91, 56 and 78?

Iteration

When writing programs it is often necessary to repeat the same set of statements several times. Rather than simply making multiple copies of the statements you can use iteration to repeat them. The algorithm for making a cup of coffee includes an instruction to keep waiting until the water in the kettle boils.

Waiting for the kettle to boil

```
IF temperature = 100°C THEN
    Switch off kettle
ELSE
    Keep waiting
END IF
```

Selection would be useless without iteration. The question would be asked once and then the program would move on or just stop. There has to be a method for repeating the question until there is a desired outcome. In a flowchart this is easy to implement – you just have to draw some arrows.

In both pseudo-code and program code you have to construct a loop, or iteration. There are two types of iteration: **indefinite iteration** and **definite iteration**.

Key terms

Indefinite iteration: this is used when the number of iterations is not known before the loop is started. The iterations stop when a specified condition is met. This sort of loop is said to be condition controlled.

Definite iteration: this is used when the number of iterations, or turns of the loop, is known in advance. It can be set to as many turns as you want. This sort of loop is said to be count controlled.

Problem solving

Indefinite iteration

Obviously in this example it is not known how many times the program will have to check until the water temperature reaches 100°C.

Therefore we have to use indefinite iteration. There are two ways of doing this in pseudo-code – you can use a REPEAT...UNTIL loop or a WHILE...DO loop.

A REPEAT...UNTIL loop checks the condition when it gets to the end of the loop. This means that the statements contained within the loop will be executed at least once. A WHILE...DO loop checks the condition at the start of the loop so in some circumstances the statements contained within the loop will not be executed.

Top tip

It is good practice to indent commands that occur within a statement block because it makes the algorithm clearer.

Using REPEAT...UNTIL

```
REPEAT                                     #This starts the REPEAT...UNTIL loop.
    RECEIVE temp FROM SENSOR              #The temperature of the water is
                                          #input from a temperature sensor.
UNTIL temp = 100                          #This sets the condition for the loop
                                          #to stop. When 'temp' is equal to 100
                                          #the loop will stop.

Switch off kettle                         #This is the next command to be
                                          #executed when the loop has finished.
```

Using a WHILE...DO loop the algorithm shown above would be

Using WHILE...DO

```
RECEIVE temp FROM SENSOR
WHILE temp < 100 DO
    RECEIVE temp FROM SENSOR
END WHILE
Switch off kettle
```

Top tip

Use the pseudo-code command SET to assign a value between 1 and 6 to diceRoll before the start of the loop.

When using a WHILE...DO loop, the statement RECEIVE guess FROM KEYBOARD, which allows the user to enter their guess, should be included in the algorithm twice – once before the start of the loop and once inside the loop.

Activity 10

- 1 Write an algorithm expressed in pseudo-code that asks the user to enter a number between 1 and 6. If the number entered matches the value stored in the variable diceRoll the message 'Well done you guessed correctly.' is displayed. Otherwise the user is invited to guess again. Use a WHILE...DO loop and include comments to explain what each line of code does.
- 2 Produce a second version of the algorithm using a REPEAT...UNTIL loop.

Definite iteration

This is used when the number of iterations, or turns of the loop, is known in advance.

One method is to use a REPEAT...TIMES loop.

Using REPEAT... TIMES

```
REPEAT 50 TIMES
  SEND '*' TO DISPLAY
END REPEAT
```

A FOR loop is another method of repeating a sequence of instructions a fixed number of times.

Worked example

A learner is designing a program to help younger children with their times tables. When a user enters a number the program will output the times table up to 12.

RECEIVE number FROM KEYBOARD	#The number entered is assigned to the variable 'number'.
FOR index FROM 1 TO 12 DO	#The loop is set up using the variable 'index' which will change from 1 to 12 at each turn of the loop.
SEND number * index TO DISPLAY	#The value of 'number' is multiplied by the value of 'index' at each turn of the loop.
END FOR	#This command is used to close the loop.

This loop will be repeated 12 times. At each turn of the loop the variable 'index' is incremented by one.

Top tip

Variables in a FOR loop can be used to indicate the start and end values of the loop, for example

```
start = 25
finish = 50
FOR index FROM start TO finish DO
  SEND 'Congratulations' TO DISPLAY
END FOR
```

Activity 11

Create an algorithm expressed in pseudo-code that asks a user to enter a start number and an end number and then outputs the total of all the numbers in the range. For example, if the start number was 1 and the end number was 10, the total would be 55 (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10).

Tip:

You should initialise the variable total, used to hold the sum of the numbers 1 to 10, to zero before the start of the loop.

Problem solving

Another example of a fixed number of iterations is when you have to enter a password and you only get three attempts. It could be implemented in the following way.

Worked example

```
SET correct TO 'LetMeIn'      #The variable 'correct' is assigned
                               the value of the password stored in
                               the system.

FOR index FROM 1 TO 3 DO
    SEND 'Please enter your password.' TO DISPLAY
    RECEIVE password FROM KEYBOARD
    IF password = correct THEN
        SEND 'You entered the correct password.' TO DISPLAY
    END IF
END FOR
```

The loop will ask for the password to be input three times, but what if the user gets the password correct on the first attempt? They do not want to have to enter it twice more.

Another solution would be to use indefinite iteration and keep count of the number of iterations.

Worked example

```
SET count TO 0                #The variable 'count' is assigned
                               the value 0.

SET correct TO 'LetMeIn'      #The variable 'correct' is assigned
                               the value of the password stored in
                               the system.

REPEAT
    SET count to count + 1     #The variable 'count' is
                               incremented by 1 on each turn.
    SEND 'Please enter your password.' TO DISPLAY
    RECEIVE password FROM KEYBOARD
    IF password = correct THEN
        SEND 'Correct password.' TO DISPLAY
    ELSE
        SEND 'Incorrect password.' TO DISPLAY
    END IF
UNTIL password = correct OR count = 3
```

The loop will end if either of the conditions is met – if the password is correct or the number of attempts is equal to 3.

We have used a compound comparison by joining two conditions together using an 'OR', which is a **logical operator**.

Key term

Logical operator: a Boolean operator using AND, OR and NOT.

Logical operators

AND	If two conditions are joined by the 'AND' operator, then they must both be true for the whole statement to be true.
OR	If two conditions are joined by the 'OR' operator, then either one must be true for the whole statement to be true.
NOT	With the NOT operator both of them must be false for the whole statement to be true.

Activity 12

In Activity 8 you had to create an algorithm for a guessing game. This game is more difficult. It should generate a **random number** between 1 and 20.

- Ask the user to guess the number.
- Allow the user three attempts.
- Display a message if the attempt is correct.
- Display a message if the attempt is incorrect and inform the player if their attempt is too high or too low.
- Display a message to the player after three incorrect attempts informing them of the correct number.

Nested loops

A **nested loop** comprises a loop within a loop. When one loop is nested within another, each iteration of the outer loop causes the inner loop to be executed until completion.

In this example a nested loop is used to calculate and display the average mark achieved by a group of twenty students in a series of five tests.

The outer loop iterates through each student in turn. The inner loop receives each set of five marks and adds them together. The outer loop calculates and displays the average mark before moving on to the next student.

Worked example

```

FOR student = 1 TO 20
  SET sum = 0
  FOR mark = 1 TO 5
    RECEIVE nextMark FROM KEYBOARD
    SET sum TO sum + nextMark
  END FOR
  SET averageMark TO sum/5
  SEND averageMark TO DISPLAY
END FOR

```

Key terms

Random number: a number within a given range of numbers that is generated in such a way that each number in the range has an equal chance of occurring.

There are many devices for generating random numbers. A die is used in games to get a random number from 1 to 6. Computer programming languages have a function for generating random numbers across variable ranges.

In the Edexcel pseudo-code there is a useful built-in RANDOM command.

`RANDOM(upperLimit)`

For example, `number = RANDOM(6)` would generate a random number from the numbers 1 to 6.

Nested loop: a loop that runs inside another loop. The inner one executes all of its instructions for each turn of the outer loop.

Problem solving

Worked example

A learner has a Saturday job selling cups of tea and coffee. The tea is £1.20 per cup and the coffee is £1.90. He is supposed to keep a record of the number of cups of each he sells.

Unfortunately he has been so busy that he has lost count but he knows that did not sell more than 100 of each.

He has collected £285.

Write a program that will calculate how many cups of tea and coffee he sold.

```
SET teaCost TO 1.2
SET coffeeCost TO 1.9
FOR numCoffees FROM 1 TO 100 DO
  FOR numTeas FROM 1 TO 100 DO      #This loop for the teas is nested
                                    #inside the loop for coffees.
    SET total TO (numCoffees * coffeeCost) + (numTeas *
    teaCost)
    IF total = 285 THEN
      SET teas TO numTeas           #These 'teas' and 'coffees'
                                    #variables are needed as the loops
                                    #will continue and 'numCoffees'
                                    #and 'numTeas' will change.
    SET coffees TO numCoffees
  END IF
END FOR
END FOR
SEND 'The number of teas is' & teas & 'and the number of
coffees is' & coffees TO DISPLAY
```

Key term

Concatenation: the linking together of two or more items of information.

In this SEND command, four items of information are displayed. First there is some literal text enclosed in quotation marks – 'The number of teas is' – followed by the value of the variable teas, followed by some more literal text – 'and the number of coffees is' – followed by the value of the variable coffees. The four are joined by an '&' symbol – known as the append operator. Joining items of information in this way is called **concatenation**.

If this algorithm was converted into program code and executed this sentence would be displayed on the screen:

'The number of teas is 95 and the number of coffees is 90'

Activity 13

Create an algorithm that will print out the times tables (up to 12 times) for the numbers 2 to 12.

Summary

- The constructs sequence, selection and iteration are the basic building blocks of algorithms.
- Nested IF statements allow for more than two alternatives.
- There are two types of iteration – definite and indefinite.
- Another name for iteration is loop. Loops can be nested.
- Comments make algorithms easier to understand. The # symbol denotes a comment in pseudo-code.

Checkpoint

Strengthen

- S1** Explain, using examples, how sequence, selection and iteration are used in algorithms.
- S2** Explain what each of the six relational operators does.

Challenge

- C1** Design an algorithm in pseudo-code that asks the user to enter their height (in metres) and weight (in kilograms) and displays their body mass index (BMI). The formula for calculating BMI is $\text{weight} / \text{height}^2$.
- C2** Design an algorithm expressed as a flowchart to control the heating in a house. A thermostat monitors the temperature within the house. During the week the temperature should be 20°C between 6.00 and 8.30 in the morning and between 17.30 and 22.00 at night. At weekends it should be 22°C between 8.00 and 23.00. If the temperature in the house falls below 10°C at any time the boiler is switched on.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at the key terms on pages 2 and 8.
- For S2 have a look at the table on page 10.

Problem solving

Working with algorithms

Learning outcomes

By the end of this section you should be able to:

- describe the purpose of a given algorithm and explain how it works.
- determine the correct output of an algorithm for a given set of data.
- identify and correct errors in algorithms.

The purpose of an algorithm

When you look at an algorithm, expressed either in pseudo-code or as a flowchart, it is sometimes easy to see its purpose.

Have a look at the algorithm in Figure 1.8.

The purpose of this algorithm is to find the area of a rectangle. It works like this.

- The length of the rectangle is input and is stored in the variable 'length'.
- The width of the rectangle is input and is stored in the variable 'width'.
- The variable 'length' is multiplied by the variable 'width' to find the area of the rectangle, which is stored in the variable 'area'.
- The value of the variable 'area' is output.

Here is another algorithm, this time displayed in pseudo-code.

```
SET totalFor TO 0
SET totalAgainst TO 0 #These variables hold the running totals
                        and are initialised to 0 at the start of the
                        algorithm.

SET win TO 0
SET loss TO 0
SET draw TO 0
SET anotherEntry TO 'Y'
SET totalFor TO 0
SET totalAgainst TO 0

WHILE anotherEntry = 'Y' DO
    SEND 'Enter goals for.' TO DISPLAY
    RECEIVE goalsFor FROM KEYBOARD
    SEND 'Enter goals against.' TO DISPLAY
    RECEIVE goalsAgainst FROM KEYBOARD
    SET totalFor TO totalFor + goalsFor
    SET totalAgainst TO totalAgainst + goalsAgainst
    IF goalsFor > goalsAgainst THEN
        SET win TO win + 1
```

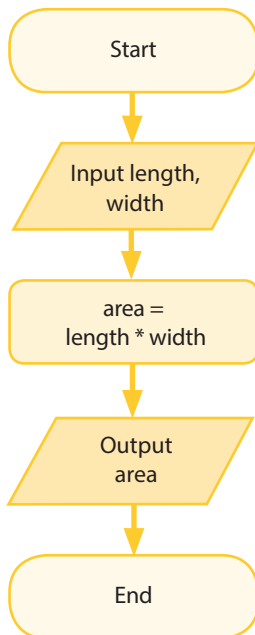


Figure 1.8 Flowchart of an algorithm showing area of a rectangle


```

ELSE
    IF goalsFOR < goalsAgainst THEN
        SET loss TO loss + 1
    ELSE
        SET draw TO draw + 1
    END IF
END IF
SEND "Press 'Y' to enter another result." TO DISPLAY
RECEIVE anotherEntry from KEYBOARD
END WHILE
SEND 'Total wins: ' & win TO DISPLAY
SEND 'Total losses: ' & loss TO DISPLAY
SEND 'Total draws: ' & draw TO DISPLAY

```

So what can we say about this algorithm?

- Its purpose is to allow a user to enter match scores for a particular team and sport. It could be something like football or hockey.
- It calculates the total goals that were scored for and against the team.
- It calculates the number of matches that were won, lost and drawn.
- It displays the number of wins, losses and draws to the user.
- The variables to hold these values are initialised – declared and assigned a start value at the beginning of the algorithm.
- The algorithm uses indefinite iteration. It loops until the user presses any key other than the 'Y' key.

Checking the output

A good way to follow the reasoning behind an algorithm, and also to find any **logic errors**, is to use some sample data and check if the output is what you expect.

Activity 14

Use the following scores as test data.

1–0

3–2

0–2

1–1

Calculate the expected end states of the variables from these results.

Track the entry of each score through the algorithm and see if they are the same as your expected results.

Trace tables

The formal way of checking the logic of an algorithm is to use a **trace table**.

Top tip

In the algorithm the SEND statement at the end of the loop uses double quotation marks. This is because the text to be displayed includes the letter 'Y' in single quotes.

You can use double quotation marks in a SEND statement where the text includes single quotes or an apostrophe. Pseudo-code is very forgiving, so it doesn't matter if you don't do this, but it does matter when the algorithm is converted into program code.

Key terms

Logic error: an error in an algorithm that results in incorrect or unexpected behaviour.

Trace table: a technique used to identify any logic errors in algorithms. Each column represents a variable or output and each row a value of that variable.

Problem solving

Worked example

Create a trace table for the following algorithm.

```
SET number TO 3
FOR index FROM 1 TO 5 DO
    SET number1 TO number * index
    SET number2 TO number1 * 2
    IF number2 > 20 THEN
        SEND number2 TO DISPLAY
    END IF
END FOR
```

This algorithm can be traced in the following table.

number	index	number1	number2	output
3				
3	1	3	6	
3	2	6	12	
3	3	9	18	
3	4	12	24	24
3	5	15	30	30

The value of the variable number remains at 3 throughout, but as the index increases from 1 to 5, then so do the values of number1 and number 2. When the value of number2 is greater than 20, its value is output.

Activity 14



Complete a trace table for this algorithm.

```
SET number1 TO 2
SET number2 TO 3
FOR index FROM 1 TO 5 DO
    SET number1 TO number1 * index
    SET number2 TO number2 + number1
END FOR
```

Exam tips

Spend some time studying the algorithm to ensure that you fully understand it.

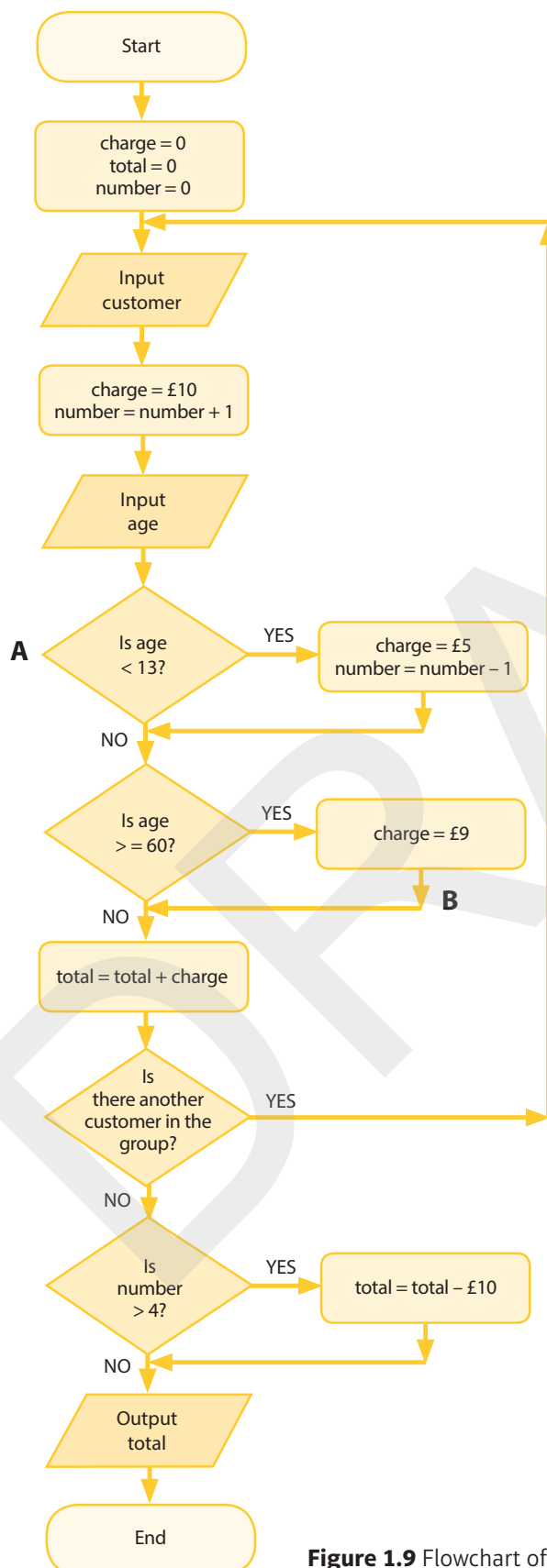
In **1** you are asked to 'explain' how the algorithm works. A longer answer is required that includes all of the stages of the algorithm. Use the correct terms to explain the constructs.

In **2** and **3** short answers are sufficient.

In **4** calculate the charge for each person using the rules of the algorithm. Then calculate the overall charge and check to see if the family qualifies for a group discount.

Exam-style question

This flowchart displays an algorithm used by Holiday Theme Parks Limited.



- 1 Explain how the algorithm calculates the total amount that should be paid.
- 2 Give **two** variables that are used in the algorithm.
- 3 In the flowchart, two of the constructs are labelled A and B. State the type of each construct.
- 4 The Smith family is visiting the park. The family consists of two children, one aged 8 and one aged 10, their two parents and their grandfather, who is aged 65. Use the algorithm to calculate how much the family should have to pay for entry.

Figure 1.9 Flowchart of an algorithm showing charges in a theme park

Problem solving

Key term

Simulation: a representation of a real-world process or system.

Activity 16



Here is part of a **simulation** for a payment system at a car park.

```
SET parkCharge to RANDOM(20) #As this is a simulation, a parking
                                charge is generated as a random
                                number between 1 and 20, i.e. £1
                                and £20.

SET payment TO 0

WHILE payment < parkCharge OR payment > 20 DO
    SEND 'The charge is ' & parkCharge TO DISPLAY
    SEND 'enter payment up to £20 maximum.' TO DISPLAY
    RECEIVE payment FROM KEYBOARD
    SET changeDue TO payment - parkCharge
    WHILE changeDue >= 10 DO
        SEND '£10 note' TO DISPLAY
        SET changeDue TO changeDue - 10
    END WHILE
    WHILE changeDue >= 5 DO
        SEND '£5 note' TO DISPLAY
        SET changeDue TO changeDue - 5
    END WHILE
    WHILE changeDue >= 2 DO
        SEND '£2 coin' TO DISPLAY
        SET changeDue TO changeDue - 2
    END WHILE
END WHILE
```

What is the maximum charge at the car park?

Explain how the algorithm calculates the change due and decides how many notes or coins should be given. Complete the algorithm so that it includes the payment of £1, 50p, 20p, 10p, 5p, 2p and 1p coins and add comments to show how the code works. If the charge is £6.90 and the person pays with a £20 note, use the completed algorithm to determine how many of each type of denomination (£20, £10 and £5 notes and £2, £1, 50p, 20p, 10p, 5p, 2p and 1p coins) will be issued.

Identifying errors

There are three types of error in computer programs.

- Syntax errors occur when algorithms are being converted into program code.
- Runtime errors when the program is executed.
- Logic errors are errors in the design of algorithms.

Syntax and runtime errors will be covered in Chapter 2. Pseudo-code purposefully doesn't really have any syntax so that the developer can concentrate on noting down the logic without having to be bothered with the rules of syntax. Obviously, you won't have syntax or runtime errors in pseudo-code.

Worked example

Here is an algorithm to find the average of two numbers.

```
RECEIVE number1 FROM KEYBOARD
```

```
RECEIVE number2 FROM KEYBOARD
```

```
SET average TO number1 + number2 / 2
```

```
SEND average TO DISPLAY
```

This seems logical. Two numbers are input, they are added together and then they are divided by 2.

However, if this algorithm was given 12 and 6 as the two numbers it would return 15 as the average instead of 9. There is a logic error.

Instead of adding the two numbers and then dividing by two, as the developer intended, it is dividing the second number by 2 and then adding the result to the first number.

The developer should have written the third line as

```
SET average TO (number1 + number2) / 2
```

Did you know?

So far you have been using the Edexcel pseudo-code that is provided in the specification. In the examination, questions will be asked using this version, but if you have to answer a question by writing pseudo-code, you do not have to use Edexcel's version. As long as your answer is logical and can be understood by a competent person, then it will be accepted.

Did you know?

In computer programming the order of precedence (the order in which you do each calculation) is the same as in mathematics and science – BIDMAS.

This is how $3^2 \times 9 + (5 - 2)$ would be evaluated.

Brackets $3^2 \times 9 + (3)$

Indices $9 \times 9 + (3)$

Division

Multiplication $81 + (3)$

Addition 84

Subtraction

The best way to find logic errors is to use the technique in the section above – use sample data and check if the actual output from the algorithm is as expected. You'll learn more about this in Chapter 2.

Problem solving

Activity 17

This is part of a larger algorithm designed to ensure that input data falls within a certain range. It is part of a school management system and checks that the 'year group' entry is acceptable. It has learners aged 11 to 18 years with year groups of 7 to 13.

The staff using the system congratulated themselves on never making an error when entering the year group, but when learner lists were printed out they immediately received complaints.

What is the logic error in the algorithm?

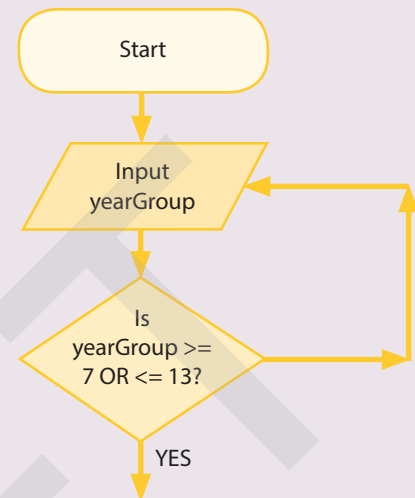


Figure 1.10 Flowchart showing an error in an algorithm

Often logic errors occur when designing loops.

Worked example

Study this algorithm.

```
WHILE index < 10 DO
    SET index TO 1
    SEND index TO DISPLAY
    SET index TO index + 1
END WHILE
```

The expected output is 1, 2, 3, 4, 5, 6, 7, 8, 9.

But there is a logic error. The variable 'index' is initialised in the wrong place. It should be done before the start of the WHILE loop.

This algorithm would loop forever because at each turn in the loop the variable index is set to 1. It will never reach 10. This is an example of an '**infinite loop**'. The algorithm should have been written as shown below.

```
SET index TO 1
WHILE index < 10
    SEND index TO DISPLAY
    SET index TO index + 1
END WHILE
```

Key term

Infinite loop: a loop that is never-ending since the condition required to terminate the loop is never reached.

Activity 18

Find and correct the errors in these algorithms.

- Example 1

```
SET index TO 1
WHILE index < 10
    SEND index TO DISPLAY
END WHILE
```

- Example 2

```
SET index TO 1
WHILE index < 10
    SEND index TO DISPLAY
    SET index TO index - 1
END WHILE
```

- Example 3

```
SET index TO 1
WHILE index < 1
    SEND INDEX TO DISPLAY
    SET index TO index + 1
END WHILE
```

Summary

- A logic error in an algorithm will produce an incorrect result.
- Tracing the value of variables in an algorithm helps to identify logic errors.
- Understanding the order of precedence of arithmetic operators and the significance of brackets will help you to avoid making logic errors.

Checkpoint

Strengthen

- S1** Describe the purpose of an algorithm and explain how it works.
- S2** Explain, using examples, what a logic error is.
- S3** Use a trace table to check the output of an algorithm and identify any logic errors. Explain what BIDMAS means and demonstrate how this expression would be evaluated.
 $4^3 \times 10 / 2 + (8 - 3)$

Problem solving

Challenge

C1 Describe the purpose of this algorithm and explain how it works.

```
SEND 'Enter the weight of your parcel in kilograms  
'TO DISPLAY  
RECEIVE parcelWeight FROM KEYBOARD  
IF parcelWeight <= 2 THEN  
    postage = 8  
ELSE  
    IF parcelWeight <= 10 THEN  
        postage = 8 + ((parcelWeight - 2) * 2.5)  
    ELSE  
        postage = 8 + (8 * 2.5) + ((parcelWeight - 10) *  
        3.5)  
    IF END  
IF END  
SEND 'The cost of posting your parcel is ' & postage  
TO DISPLAY
```

C2 People often want to know the human-equivalent age of their dog or cat. The rules for calculating this are:

A 1-year-old cat is equivalent in age to a 15-year-old human, a 2-year-old cat is equivalent in age to a 24-year-old human. Add four years for every year after that.

A 1-year-old dog is equivalent in age to a 12-year-old human, a 2-year-old dog is equivalent in age to a 24-year-old human. Add four years for every year after that.

Design an algorithm expressed in pseudo-code that allows the user to input the age of their dog or cat and outputs its equivalent human age. It should allow the user to have another go.

Use a trace table and test data to check the logic of your algorithm.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 reread pages 20–21 and have another go at all the activities.
- For S2 and S3 reread the section on trace tables on pages 21–22 and the section on identifying errors on pages 24–25.

Sorting and searching algorithms

Learning outcomes

By the end of this section you should be able to:

- explain the following sorting algorithms and be able to apply them.
 - Bubble sort
 - Merge sort
- explain the following searching algorithms and be able to apply them.
 - Linear search
 - Binary search
- explain how the choice of algorithm is influenced by the data structures and data values that need to be manipulated.
- evaluate the fitness for purpose of algorithms in meeting specified requirements efficiently using logical reasoning and test data.

Two of the most common tasks in computer programs are sorting data into a particular order and searching for particular items of information.

There might be millions of items of stored data and searching for information would be very inefficient if the data was not sorted. Imagine the confusion and difficulty of having to find something in a dictionary that wasn't in alphabetical order or planning a trip with train timetables that weren't sorted into time order. Even small lists such as football league tables or the top-20 music charts are much more useful if they are sorted into order.

Arrays

All sorting and searching algorithms work on lists of data. As you already know, a variable stores just a single value (e.g. SET age TO 15 or SET firstName TO 'David').

But what if a programmer wants to store lots of related values, such as the names of a group of friends? A tedious way of doing it might look something like this.

```
SET friend1 TO 'Alice'
SET friend2 TO 'Barry'
SET friend3 TO 'Catherine'
```

It would be much easier if all the names could be stored in one list to which the names of new friends can be added. That's exactly what an **array** allows you to do.

An array with the identifier 'arrayFriends' would store the data items like this.

```
arrayFriends = ['Alice', 'Barry', 'Catherine']
```

This array has three elements.

Key term

Array: an organised collection of related values that share a single identifier.

Problem solving

Array indexes

Each element in an array has an index number. In the example above the index number of 'Alice' is 0 and the index number for 'Catherine' is 2. (Index numbering starts at 0.)

A new element can be inserted into an array like this.

```
SET arrayFriends[3] TO 'David'
SET arrayFriends[4] TO 'Eva'
```

This would insert the name 'David' at index position 3 and the name 'Eva' at index position 4 – the fourth and fifth positions in the array.

Top tip

Pseudo-code has a built-in LENGTH command, which you can use to find the number of elements in an array, for example `SET size TO LENGTH(arrayFriends)` would set the variable 'size' to 5.

Remember: the elements have indexes 0 to 4.

Top tips

- Use the LENGTH function to find out how many marks there are altogether.
- Use a variable maxMark – set initially to 0 – to find the highest mark.
- Each mark in the array is compared in turn with the value stored in maxMark.
- If it is higher then maxMark is updated.

Key terms

Ascending order: this is arranging items from smallest to largest (e.g. 1, 2, 3, 4, 5, 6 or a, b, c, d, e, f).

Descending order: this is arranging items from largest to smallest (e.g. 6, 5, 4, 3, 2, 1 or f, e, d, c, b, a).

Worked example

This code will traverse an array named 'arrayFriends' and print out each element of the array.

```
FOR index FROM 0 TO LENGTH(arrayFriends) - 1 DO
    #The loop has to run to the length of
    #the array minus 1 as indexing starts at
    #0. For example, if there are 10 items
    #then they will be indexed as 0 to 9 (10
    #-1).
    SEND arrayFriends[index] TO DISPLAY
END FOR
```

Activity 19

An array named 'arrayScores' contains a set of marks that a learner has obtained during her computer science course.

Create an algorithm expressed in pseudo-code to find and display her highest mark and her average mark.

Activity 19 would have been a whole lot easier if the elements of the array had been sorted into order. We will next be looking at two ways this can be done.

Sorting algorithms

As sorting is such a widely used procedure, many algorithms have been created to carry it out. As with all algorithms, some are more efficient than others.

Bubble sort

When data is sorted, different items must be compared with each other and moved so that they are in either **ascending order** or **descending order**.

The bubble sort algorithm starts at one end of the list and compares pairs of data items. If they are in the wrong order, they are swapped. The comparison

of pairs continues to the end of the list, each complete **traversal** of the list being called a pass. This process is repeated until there have been no swaps during a pass, indicating that the items must all be in the correct order.

The algorithm can be described as follows.

Bubble sort (ascending order)

- 1** Start at the beginning of the list.
- 2** Compare the values in position 1 and position 2 in the list – if they are not in ascending order then swap them.
- 3** Compare the values in position 2 and position 3 in the list and swap if necessary.
- 4** Continue to the end of the list.
- 5** If there have been any swaps, repeat steps 1 to 4.

Key term

Traversal: travel across or through something. An array can be traversed by moving from the first to the last element in order to examine the data stored at each index position.

Worked example

Here is an example of a bubble sort in action.

Pass 1

4	2	6	1	3	Items 1 and 2 must be swapped.
2	4	6	1	3	Items 1 and 2 are swapped.
2	4	6	1	3	Items 2 and 3 are already in ascending order.
2	4	6	1	3	Items 3 and 4 must be swapped.
2	4	1	6	3	Items 3 and 4 have been swapped.
2	4	1	6	3	Items 4 and 5 must now be swapped.
2	4	1	3	6	Items 4 and 5 have been swapped.

Figure 1.11 A bubble sort

It would take a human three passes to carry out this bubble sort, but a computer would need four passes because it must continue until there have been no swaps. A computer cannot just look at all of the numbers at once and see that they are all in order.

Pass 2

2	4	1	3	6	Items 1 and 2 are in correct order.
2	4	1	3	6	Items 2 and 3 must be swapped.
2	1	4	3	6	Items 2 and 3 have been swapped.
2	1	4	3	6	Items 3 and 4 must be swapped.
2	1	3	4	6	Items 3 and 4 have been swapped.
2	1	3	4	6	Items 4 and 5 do not need to be swapped.

Pass 3

2	1	3	4	6	Items 1 and 2 must be swapped.
1	2	3	4	6	Items 1 and 2 have been swapped.
1	2	3	4	6	All items are now in the correct order.

Did you know?

Do you know why it is called 'bubble sort'? If you look carefully, you can see that gradually the largest items move to the end, like bubbles rising in water. After the first pass, the largest number is in its correct position. Then after the second pass, the next largest is in its correct position. This happens on each pass and so if the algorithm is to be made more efficient the last comparisons can be omitted.

Problem solving

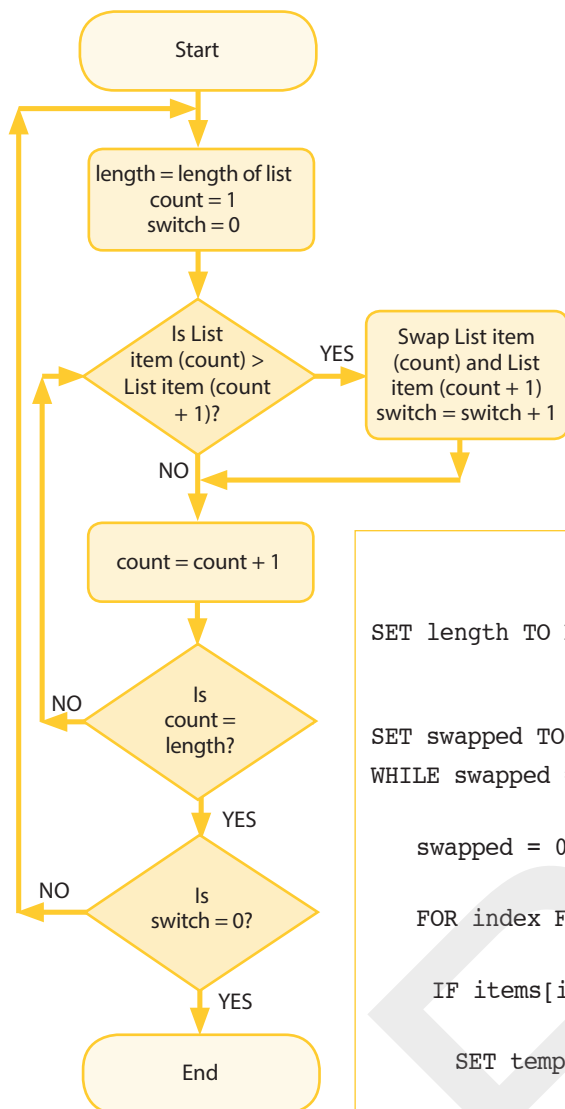


Figure 1.12 Bubble sort algorithm written as a flowchart

The bubble sort algorithm can be represented as a flowchart as shown in Figure 1.12.

Activity 20



Study the flowchart of the bubble sort algorithm.

Using the variables declared, explain the logic behind the algorithm – explain how it functions to sort a list.

The bubble sort algorithm can also be expressed in pseudo-code. It assumes that the items to be sorted are stored in an array.

```
SET length TO LENGTH(items) - 1
```

```
SET swapped TO 1
```

```
WHILE swapped = 1 DO
```

```
    swapped = 0
```

```
    FOR index FROM 1 TO length DO
```

```
        IF items[index-1] > items[index] THEN
```

```
            SET temp TO items[index-1]
```

```
            SET items[index-1] TO items[index]
```

```
            SET items[index] TO temp
```

```
            SET swapped TO 1
```

```
        END IF
```

```
    END FOR
```

```
END WHILE
```

#The items to be sorted are in an array with the identifier 'items'.

#The variable 'length' is set to the length of the array minus one to represent the last index position.

#The variable 'swapped' is assigned the value of 1. #The loop will run while the value of 'swapped' is equal to 1.

#The variable 'swapped' is changed to 0. It will be changed back to 1 if a swap occurs.

#This will set up a loop from 1 to the value of 'length' which is the length of the array minus one.

#On the first loop this will check the value of the item at index 1 with that at index 0.

#This block of code will swap the two items if the first is larger than the second.

#Swapped is set to 1 to indicate that a swap has occurred and so the 'WHILE' loop will turn again.

#This ends the 'FOR' loop and so the variable 'index' will be incremented by 1 for the next iteration until it is greater than the 'length' variable when it will stop.

Merge sort

Merge sort is a sorting algorithm that divides a list into two smaller lists and then divides these until the size of each list is one. Repeatedly applying a method to the results of a previous application of the method is called **recursion**.

In computing a problem is solved by repeatedly solving smaller parts of the problem. A part of a program can be run and rerun on the results of the previous run (e.g. repeatedly dividing a number by 2).

Key term

Recursion: a process that is repeated. For example, a document can be checked and edited, checked and edited and so on until it is perfect.

Worked example

Here is an example of a merge sort.

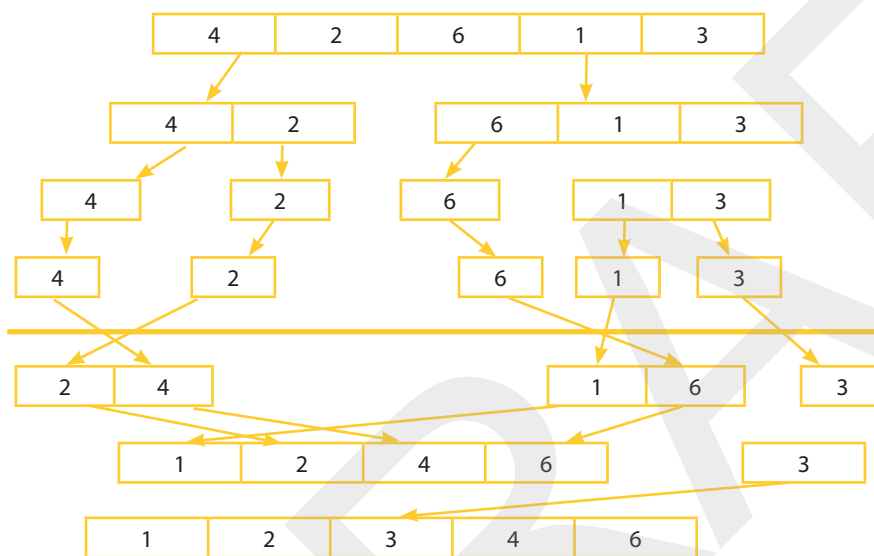


Figure 1.13 Example of a merge sort

The list is recursively divided until the size becomes one. The lists are then recursively merged, with the items in the correct order, until the complete list has been merged.

Instead of sorting the complete list, the problem is broken down into smaller problems, which are then solved independently. It can be explained by the following description.

- 1** Divide the list into two parts.
- 2** Recursively divide these lists until the size of each contains one item.
- 3** Recursively merge the lists with the items in the correct order.

Activity 21

A tutor has stored a set of class examination results in an array named 'exam1'. Write an algorithm expressed in pseudo-code to sort the results into descending order and then output the highest and lowest result.

Activity 22

Using a table like the one in Figure 1.13, show how the following list would be sorted into descending order using merge sort.

48, 20, 9, 17, 13, 21, 28, 60

Problem solving

Key terms

Brute force: an algorithm design that does not include any techniques to improve performance, but instead relies on sheer computing power to try all possibilities until the solution to a problem is found.

Divide and conquer: an algorithm design that works by dividing a problem into smaller and smaller sub-problems, until they are easy to solve. The solutions to these are then combined to give a solution to the complete problem.

Extend your knowledge

Only two sorting algorithms are required for the specification: bubble sort (the slowest) and merge sort (one of the most efficient). There are far more and many of them are relatively easy to code. Research the insertion and selection sorts.

Linear search

- 1 Start at the first item in the list.
- 2 Compare the item with the search item.
- 3 If they are the same then stop.
- 4 If they are not then move to the next item.
- 5 Repeat 2 to 4 until the end of the list is reached.

Efficiency of sorting algorithms

This graph compares the performance of the bubble and merge sort algorithms.

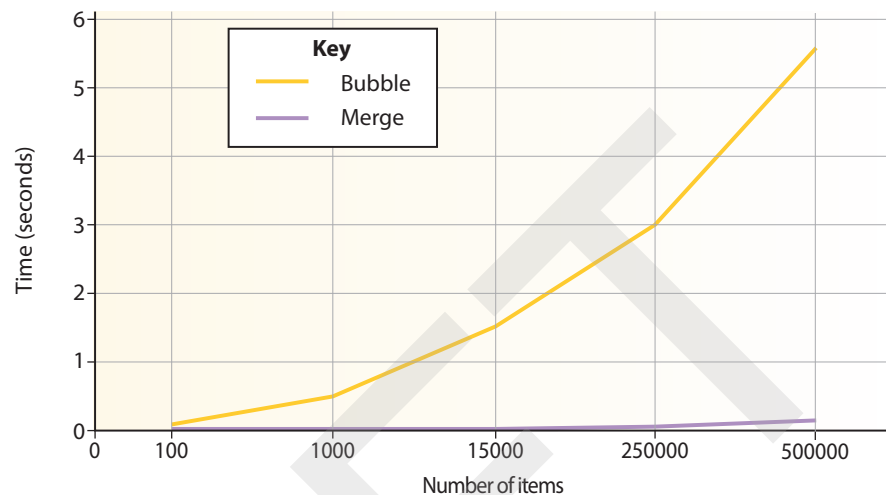


Figure 1.14 A graph comparing the performance of bubble and merge sort algorithms

The bubble and merge sort algorithms demonstrate two alternative approaches to algorithm design.

The bubble sort algorithm is said to be using **brute force** because it starts at the beginning and completes the same task over and over again until it has found a solution.

The merge sort uses the '**divide and conquer**' method because it repeatedly breaks down the problem into smaller sub-problems, solves those and then combines the solutions.

The graph shows that a bubble sort is far slower at sorting lists of more than 1000 items, but for smaller lists the time difference is negligible.

As the bubble sort algorithm is easier to code it could be advantageous to use it for smaller lists of less than 1000 items.

Searching algorithms

To find a specific item in a list involves carrying out a search. Like sorting, some methods of searching are more efficient than others.

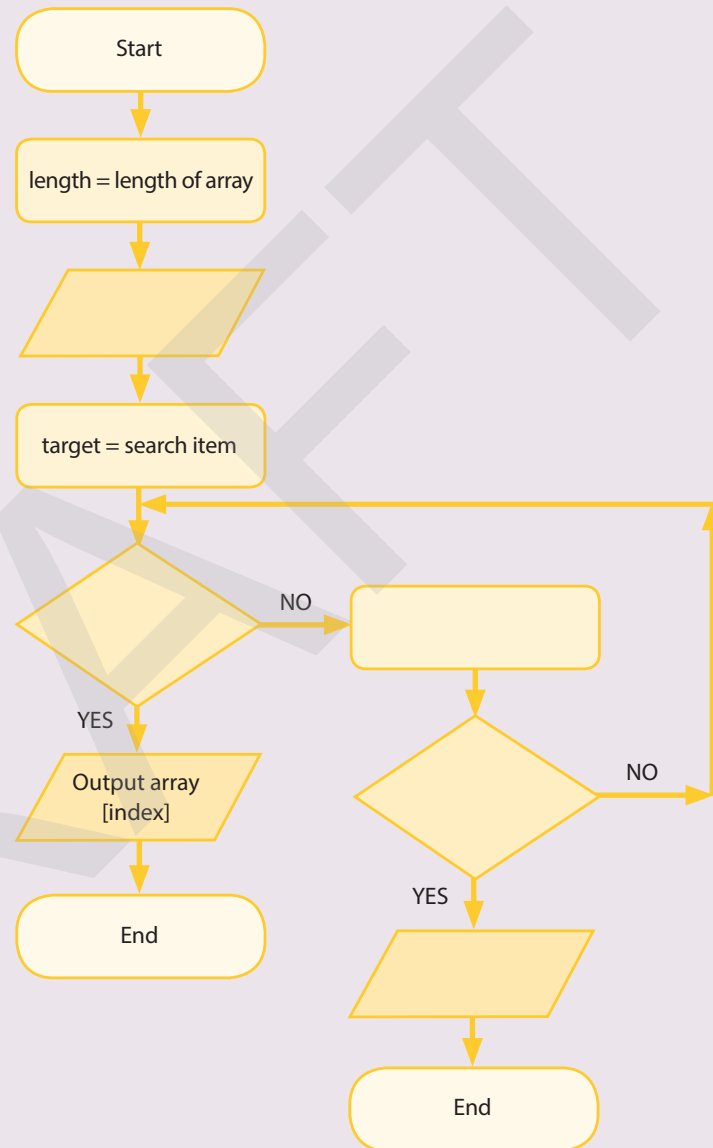
Linear search

A linear search is a simple algorithm and it is not subtle. It simply starts at the beginning of the list and goes through it, item by item, until it finds the item it is looking for or reaches the end of the list without finding it.

A linear search is sequential as it moves through the list item by item.

Activity 23

This flowchart displays a linear search algorithm, but some of the symbols have not been labelled. Draw and complete the flowchart using the labels provided.



Labels

Output search item
not found

Is index > length?

Does array [index] =
target?

index = index + 1

Input search item

Activity 24



An array contains the names of the one hundred most downloaded performers on an online music streaming site. Produce an algorithm expressed in pseudo-code that enables a user to see if their favourite performer is in the top 100.

Figure 1.15 A flowchart displaying a linear search algorithm with labels missing

Problem solving

Key term

Median: the middle number when the numbers are put in ascending or descending order, e.g. if there are 13 numbers, then the 7th number is the median. If there are an even number of items in an ascending list, choose the item to the right of the middle (e.g. if there are 10 numbers, then choose the 6th as the median).

Binary search

Like a merge sort, a binary search uses a 'divide and conquer' method.

Did you know?

You have probably used a binary search method when trying to guess a number between two limits. If you are asked to guess the number between 1 and 20 you will probably start at 10, the middle number. If you are told this is too high, you will then guess 5, the middle number between 1 and 10 and then repeat this method until you find the correct one.

In a binary search the middle or **median** item in a list is repeatedly selected to reduce the size of the list to be searched – another example of recursion. If the selected item is too high or too low then the items below or above that selected item can then be searched.

To use this method the list must be sorted into ascending or descending order. It will not work on an unsorted list.

Binary search (items in ascending order)

- 1 Select the median item of the list.
- 2 If the median item is equal to the search item then stop.
- 3 If the median is too high then repeat 1 and 2 with the sub-list to the left.
- 4 If the median is too low then repeat 1 and 2 with the sub-list to the right.
- 5 Repeat steps 3 and 4 until the item has been found or all of the items have been checked.

Worked example

In this list, the search item is the number 13.

As this is too high, the sub-list to the left of the median must be searched.

3	13	24	27	31	39	45	60	69	Select the median number.
---	----	----	----	----	----	----	----	----	---------------------------

As this is too high, the sub-list to the left of the median must be searched.

3	13	24	27	The median number of this sub-list is now selected.
---	----	----	----	---

This is again too high and so the sub-list to the left must be searched.

3	13	The median number is now the search item.
---	----	---

Figure 1.16 Binary search including sub-lists

In this example, it took three attempts to find the search item. A linear search would have accomplished this with only two attempts.

Activity 25



Display the stages of a binary search, as in the worked example above, to find the number 13 in this list.

3 9 13 15 21 24 27 30 36 39 42 54 69

Here is part of the algorithm displayed as pseudo-code. Some of the lines are incomplete as indicated by 'x' symbols.

SET start TO 0	#The items to be sorted are in an array with the identifier 'aList'.
SET end TO LENGTH(aList) - 1	#The variable 'start' is set to 0 – the index of the first item in the list.
SET found TO False	#The variable 'end' is set to the index of the last item in the list.
SEND 'Please enter the search item.' TO DISPLAY	#The Boolean variable 'found' is assigned the value 'false'.
RECEIVE target FROM KEYBOARD	#The user is asked to enter the search item.
WHILE start <= end AND found = False DO	#A while loop is set up with these conditions as the loop should stop either when the target is found or all of the list has been searched, i.e. when the 'start' and 'end' values are the same.
SET middle TO (start + end) / 2	#The median is found and is assigned to the variable 'middle.'
IF aList[middle] = target THEN	#If the target is found, the user is informed and 'found' is set to 'true' to stop the WHILE loop.
found = True	
SEND target + ' is in the list' TO DISPLAY	
END IF	
IF target < aList[middle] THEN	
end = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
END IF	
IF target xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
END IF	
END WHILE	
IF found = False THEN	
SEND ('The search item is not in the list.') TO DISPLAY	
END IF	

Activity 26



Complete the missing sections of the pseudo-code.

Problem solving

Efficiency of searching algorithms

In the example on page 36, the linear search was more efficient because it only had to carry out two comparisons instead of the three for binary search. But is this always the case?

Searching algorithms can be compared by looking at the 'worst case' and the 'best case' for each one.

Worked example

If you wanted to find a particular item in a list of 1000 items these are the best and worst case scenarios for the linear search and binary search algorithms.

Linear search

A linear search starts at the first item and then works through sequentially.

The best case would be if the item is first in the list.

The worst case would be if it is the last in the list.

Therefore in this example the average would be 500 comparisons.

Binary search

The best case would be if the item is in the median position in the list. The search would require only one comparison.

For the worst case it would have to choose the following medians until it finally hit the target.

(This assumes that the target is always smaller than the median.)

Attempt	Median
1	500
2	250
3	125
4	63
5	32
6	16
7	8
8	4
9	2
10	1

Therefore the worst case for the binary search is ten comparisons.

The binary search is therefore far more efficient than the linear search.

So should a binary search be used every time? That depends on the circumstances. The binary search has one great disadvantage – the list must be already sorted into ascending or descending order and therefore a sorting algorithm must be applied before the search.

If the list is to be searched just once then a linear search would be better, but if there is a large list that will be searched many times then sorting the list and using binary search would be better. Once the list has been sorted new items can be inserted into the correct places.

Exam-style question

A tutor has stored learner surnames in an array as shown below.

Marek	Jackson	Bachchan	Wilson	Abraham	French	Smith
-------	---------	----------	--------	---------	--------	-------

1 Show the stages of a bubble sort when applied to this data.

The tutor has a sorted list of names from another class as shown below.

Azikiwe	Bloom	Byrne	Davidson	Gateri	Hinton	Jackson	Linton	Smith	Wall
---------	-------	-------	----------	--------	--------	---------	--------	-------	------

2 Show the stages of a binary search to find the name 'Jackson' when applied to this list.

Summary

- An array stores multiple items of data.
- There are many algorithms for sorting and searching data.
- The choice of algorithm depends on the data that is to be processed.

Checkpoint

Strengthen

- S1** Describe the differences between the 'bubble sort' and 'merge sort' algorithms.
- S2** Describe how a binary search algorithm finds the search item.

Challenge

- C1** Explain when a linear search might be preferable to a binary search even though the binary search algorithm is more efficient.
- C2** Discuss the advantages of using arrays in algorithms.

Exam tips

These questions are testing knowledge of the sort and search algorithms.

- 1** The answer should be set out to show how the data is progressively sorted using the bubble sort and the result of each pass should be shown.
- 2** This question is to check that you know that this is a recursive method where the median is repeatedly selected.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at pages 30–33.
- For S2 have a look at pages 29–30.

1.2 Decomposition and abstraction

Learning outcomes

By the end of this section you should be able to:

- analyse a problem by investigating requirements (inputs, outputs, processing, initialisation) and design a solution.
- decompose a problem into smaller sub-problems.
- explain how abstraction can be used effectively to model aspects of the real world.
- program abstractions of real-world examples.

Key terms

Computational thinking: the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by a computer.

Decomposition: breaking a problem down into smaller, more manageable parts, which are then easier to solve.

Abstraction: the process of removing or hiding unnecessary detail so that only the important points remain.

Problem solving

The tasks of a computer scientist include defining and analysing problems; creating structured solutions – algorithms; coding the solutions into a form that can be implemented by a computer.

These tasks are part of what is known as **computational thinking**.

One of the skills required for computational thinking is algorithm design, which we've covered in detail in this chapter. If there is a fault in the algorithm design then the program will not work, however good a coder you are. Two other skills are **decomposition** and **abstraction**.

Decomposition

Decomposition is usually the first step in the problem-solving process. Once a problem has been broken down and the sub-problems have been identified, algorithms to solve each of them can be developed.

Decomposition means that sub-problems can be worked on by different teams at the same time. As smaller algorithms are developed for each sub-problem, it is easier to spot and correct errors and when the algorithm is developed into a program, code can be reused.

Worked example

A learner has been set the task of creating a computer version of the 'noughts and crosses' game where a user plays against the computer.

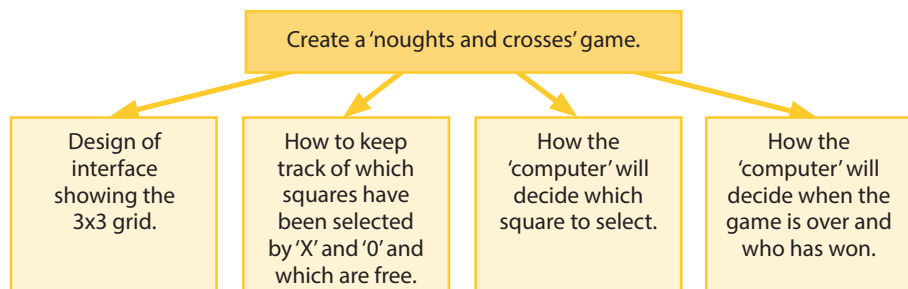


Figure 1.17 Sub-problems to be solved to create a noughts and crosses computer program

The diagram shows some of the sub-problems that must be solved in order to solve the complete problem and create a version of the game.

Abstraction

We use abstraction all the time in our daily lives. We abstract the essential features of something so that we can understand what people are trying to communicate.

Somebody might say, 'I was walking down the street when I saw a cat.' You immediately understand what they mean by 'street' – probably a road with a pavement and houses or shops along the side of it. Similarly you can picture the cat – a smallish animal with fur, four legs and a tail. An animal that is basically 'cattish'. You have extracted the basic properties of animals called cats so that you can recognise one when you see one or imagine one when somebody talks about a cat.

What you picture is very unlikely to be exactly like the actual street and cat that the person experienced. But because of our ability to abstract, the person did not have to go into unnecessary painstaking detail about exactly where they were and what they saw. They wouldn't get very far with the story if they did.

When we create algorithms we abstract the basic details of the problem and represent them in a way that a computer is able to process.

Worked example

A learner is designing a computer version of a game in which users have to throw a die to determine their number of moves.

In the computer game the users can't have an actual die, so the designer will have to have a 'pretend' or virtual die that behaves in exactly the same way as a real-life die.

The designer will have to use their powers of abstraction to work out the essential features of a die and then represent them in computer code.

To represent the die the designer will have to create a routine that will select a random number from 1 to 6 because that's what a die does.

The designer has used abstraction to model a real-life event.

Levels of abstraction

There are different levels or types of abstraction. The higher the level of abstraction, the less is the detail that is required. We use abstraction all the time in accomplishing everyday tasks.

When programmers write the 'print' command they do not have to bother about all of the details of how this will be accomplished. They are removed from them. They are at a certain level of abstraction.

A driver turning the ignition key to start a car does not have to understand how the engine works or how the spark to ignite the petrol is generated. It just happens and they can simply drive the car. That is abstraction.



Is this the street and cat you imagined?

An example – noughts and crosses

The diagram on page 40 showed some of the sub-problems that the problem of creating a noughts and crosses game could be divided into. The following could be written at a high level of abstraction.

- The computer goes first. Then the user. This continues until either one wins or all of the squares have been used.

Immediately a pattern can be recognised – a loop will be needed.

Inputs and outputs

The following inputs from the user will be needed.

- Start the game.
- Entries for the user.
- Select a new game or finish.

The following outputs will be needed.

- A message to inform the user when it is their turn.
- A message to inform the user if they try to select a square that has already been used.
- A message to inform the user if the game is a draw.
- A message to inform the user if they or the computer has won.
- A message to ask the user if they want to play another game or want to finish.

Processing and initialisation

The following processing will be needed.

- Set up the grid with the nine squares.
- Initialise all variables to a start value.
- Decide which square the computer will select.
- Allow the user to select a square.
- Check if the user has selected an already used square.
- Check if the computer or the user has won.
- Check if all squares have been used and the game is a draw.
- Allow the user to select a new game or finish.

The solution is still at a high level of abstraction and more details will need to be added.

For example, the programmer will need to decide how the game will record which player has selected each square; how the computer will decide which square to select; how the game will decide if the computer or the user has won.

The programmer will have to go into more and more detail or move to lower levels of abstraction.

Eventually the programmer will be able to design an algorithm for the game and code it using a high-level programming language such as Python or Java. Even before they start to implement the game, they will need to plan how they will test the finished program to make sure that it works correctly; what test data they will use; what outcomes it should produce.

Coding an algorithm

High-level programming languages make it easier for a programmer to write code. Unfortunately, the processor, which has to execute the program, cannot understand the language it is written in so needs a translator to translate it into the only language it does understand – a stream of 1s and 0s.

These high-level languages are therefore at a high level of abstraction – very far removed from the actual language of a computer.

The processing can be split into parts. For example, in the example of the noughts and crosses game there could be separate algorithms for

- deciding where the computer should make its next selection – it could be called ‘computer entry’;
- checking if the computer or the player has won – it could be called ‘check if won’;
- checking if there are any empty squares left – it could be called ‘check draw’.

These separate algorithms could be used when they are needed. It is efficient because it means that the same code doesn’t have to be rewritten whenever it is needed.

These items of code are called **subprograms**.

In Chapter 2 we’ll look in detail at how subprograms are used to reduce the complexity of programs and to make them easier to understand.

In the die example above, the designer could write a subprogram called ‘die’ that generates a random number from 1 to 6. In the main program the designer could just call the ‘die’ subprogram without having to think about how to implement it each time.

Key term

Subprogram: a self-contained module of code that performs a specific task. It can be ‘called’ by the main program when it is needed.

Activity 27

In a game each player spins a wheel that is divided into four colours: red, blue, green and yellow. Each player has to answer a question on a particular topic depending on the colour next to a pointer when the wheel stops. Red is for science, blue for history, green for general knowledge and yellow for geography. A player scores two points if they answer correctly on the first attempt and one point for being correct on the second attempt. The first player to reach 30 points is the winner.

Your task is to design a computer version of the game for up to four players. You must analyse the problem and list all of the requirements; decompose the problem, list all the sub-problems and write a brief description of each; list all of the input, output and processing requirements.

One of the requirements that will have to be modelled is the spinning of the wheel. Using a written description and pseudo-code show how this could be done.

Did you know?

Computer models or simulations of real life are widely used. It is far cheaper and safer to train pilots on flight simulators than on real aircraft. They are also used in weather forecasting, designing and testing new cars and bridges and even teaching people to drive. A computer model is used by the Chancellor of the Exchequer to predict what will happen if changes are made in the budget (e.g. if taxes are raised or lowered).

Problem solving

Extend your knowledge

Complete the noughts and crosses example. See if you can end up with a working game.

Summary

- Computational thinking is an approach to solving problems that includes techniques such as decomposition and abstraction.
- Problems are easier to solve if they are decomposed into smaller sub-problems.
- Abstraction is used to remove unnecessary detail to make a problem easier to understand and solve.
- When designing a solution to a problem the inputs, outputs and processing requirements should be identified at the outset.

Checkpoint

Strengthen

- S1** Explain what is meant by 'decomposition' and the benefits it provides for programmers.
- S2** Explain what is meant by 'abstraction'.

Challenge

- C1** Describe examples of the use of 'decomposition' and 'abstraction' when solving a problem.
- C2** In your own words explain what is meant by 'computational thinking'.

How confident do you feel about your answers to these questions?
If you're not sure you answered them well, reread pages 40–42.