1 Algorithms

Learning outcomes
By the end of this chapter you should be able to:
• explain what an algorithm is and create algorithms to solve specific problems
• use sequence, selection and iteration in algorithms
• use input, processing and output in algorithms
• express algorithms using flowcharts and pseudo-code
• analyse, assess and compare different algorithms
• create, name and use suitable variables
• use arithmetic, relational and Boolean operators
• use conditional statements.

Challenge: create an algorithm to help a taxi company calculate fares

By the end of this chapter, you should have a thorough knowledge of how algorithms can be used to solve complex problems and how they can be displayed using flowcharts and pseudo-code.
• An algorithm is a step-by-step procedure for solving a problem. It can be followed by humans and computers.
• Your challenge is to create an algorithm to help a taxi company calculate fares.

Why algorithms?
Algorithms run our world! In every area algorithms are used to decide what action should be taken in a particular circumstance. As computers can consider all the possibilities far more quickly than a human brain, they are becoming more important to the running of the world. Here are just a few examples:

• In a game of chess, when each player has made three moves, there are over 9 million possible moves available; after four moves there are over 288 billion. Computers have the ability to consider all these possible moves far more quickly than humans. That is why no chess grandmaster has beaten a top computer chess algorithm since 2005.
• Algorithms are used by financial organisations to trade shares on the stock market. A computer following an algorithm can decide which deal to make far more quickly than a human and a split second difference can be worth millions of pounds.
• Closely guarded algorithms are used for internet searches to make them quicker and the results more relevant to the user. They will even auto-complete the search terms based on previous searches.

Algorithms are used to control automatic-pilot systems in airplanes. If you have flown in an aeroplane, you have probably been piloted by an algorithm!
What is an algorithm?

An algorithm is a step-by-step procedure for solving problems. It is a set of instructions that can be followed by humans and computers.

We use algorithms to carry out everyday tasks often without thinking about them. For example, an algorithm to solve the problem of getting ready for school might be:

- Get out of bed.
- Shower.
- Get dressed.
- Turn on kettle.
- Put bread in toaster and turn on.
- Wait for kettle to boil and make tea.
- Wait for bread to toast, butter it and add marmalade.
- Drink tea and eat toast.
- Gather school books and put in bag.
- Put on shoes and coat.
- Leave the house.

The algorithm shows the sequence of tasks. Different people will design different algorithms, as they will do things in a different order, meaning there can be many solutions to the same problem. Some of these tasks could be further divided into sub-tasks as they might be made up of smaller steps.

For example, ‘showering’ could involve many different steps including turning on the shower and setting the correct temperature. If all the possible sub-tasks were included, the complete algorithm would get very large and complicated.

**ACTIVITY 1.1**

Create an algorithm for someone who has never made a cup of tea before to follow, in order to make a cup of tea successfully.

Compare it with other members of your group. There will probably be differences, for example some may include the addition of sugar.

**Key terms**

- **sequence**: the order in which tasks are carried out
- **sub-tasks**: small steps making up a larger task

**Watch out**

In an algorithm, the order in which the tasks are carried out is very important to its success or failure. For example, this algorithm would not be very successful if ‘shower’ was placed after ‘get dressed’. The sequence is very important.
This example seems pretty easy. However, like a typical algorithm, it is simply a list of steps. Here is part of another algorithm which is the start of a recipe to make a perfect meringue.

1. Tip 4 large egg whites into a large clean mixing bowl (not plastic).
2. Beat them on medium speed with an electric hand whisk.
3. Keep beating until the mixture resembles a fluffy cloud and stands up in stiff peaks when the blades are lifted.
4. Now turn the speed up and start to add 115 g caster sugar, a dessertspoonful at a time until there is none left.
5. Continue beating for 3–4 seconds between each addition.

In addition to sequence, this algorithm has two new elements: iteration and selection.

Iteration means doing things over and over again. The cooks have to beat the mixture and then stop and ask themselves if it resembles a fluffy cloud. If it does not, they have to beat again, check again, beat again and check again until they are convinced they have made a fluffy cloud. There is also repetition when adding the sugar. It has to be added a spoonful at a time until there is none left.

Selection means making decisions. As well as doing things over and over again, the cooks have to make a decision. Does it resemble a fluffy cloud?

Complete Interactive Activity 1a on Cambridge Elevate

ACTIVITY 1.2

Using sequence, selection and iteration write an algorithm that a person (who has never done this before) could follow in order to successfully prepare a bath with the water at the correct temperature. Annotate the algorithm to indicate sequence, selection and iteration.

You could set it out as shown in the table below where the first four tasks have been done for you.

<table>
<thead>
<tr>
<th>Task</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put plug in the bath.</td>
<td>Sequence</td>
</tr>
<tr>
<td>Turn on hot tap.</td>
<td>Sequence</td>
</tr>
<tr>
<td>Is the water at the correct temperature?</td>
<td>Selection</td>
</tr>
<tr>
<td>Turn cold tap until water is at the correct temperature.</td>
<td>Iteration</td>
</tr>
</tbody>
</table>
What makes a successful algorithm?
The two most important criteria are:

- **Correctness**: it successfully solves the problem
- **Efficiency**: it solves the problem in the least possible time.

List A is the ‘getting up’ algorithm we looked at earlier. List B is similar but with the sequence slightly altered.

List B is more efficient as it could be implemented in less time. The kettle and toaster are turned on before taking a shower and so the water will boil while the person is showering. There will be no waiting time.

**An algorithm for a computer**

Now let’s look at a simple algorithm that we could create for a computer to follow, instead of a human. Computers are ideal for obeying orders and carrying out actions over and over again. In fact, that is their main function.

It is important for the temperature in a shopping mall to be kept at a set value. It will keep the shoppers comfortable and it will help to prevent condensation on glass shop windows and slippery floor surfaces. Here is an algorithm intended to be used to control the temperature in a shopping mall and maintain a temperature of 20°C.

1. Check the temperature.
2. If the temperature is greater than 20°C, then turn off the heaters and open the ventilators.
3. If the temperature is less than 20°C, then turn on the heaters and close the ventilators.

4. Go back to instruction 1.

This is a simple algorithm but it includes the basic building blocks:

- **Sequence**: there is a list of instructions in the correct order.
- **Selection**: the ‘if’ statements in instructions 2 and 3 allow a decision to be made and an action to be taken.
- **Iteration**: instruction 4 tells the computer to go back to instruction 1 and so the sequence will run over and over again indefinitely.

---

**Remember**

1. An algorithm is a step-by-step procedure for solving a problem in a finite number of steps.
2. The basic building blocks of algorithms are sequence, selection and iteration.
3. The criteria for a successful algorithm are correctness and efficiency.
4. An algorithm must be translated into a programming language before it can be executed by a computer.

---

**Flowcharts**

Flowcharts can be used to represent algorithms visually.

They use symbols connected by arrows to show the flow of the algorithm.

The symbols used are:

- This represents the start or end point of the flowchart. You always start and finish your flowchart with this symbol.

- You use this to represent data input or data output. For example it could be a number or name entered by a user.

- You use this where a decision has to be made. It is also called selection. It will contain a question, for example: ‘Is the temperature greater than 20°C?’ If it is, then an arrow will point to a task to be carried out and if it is not, then an arrow will point to a different action.

- You use this to represent a process that must be carried out by the algorithm. In this example it is used as a result of one of the questions that has been asked, e.g. ‘Turn on the heaters’ or ‘Turn off the heaters’.
WORKED EXAMPLE

Here is the flowchart of an algorithm to calculate the area of a rectangle:

There are two inputs of the dimensions of the rectangle, a process to calculate the area and an output of the area.

In this example, there is just sequence: a list of tasks to be performed.

ACTIVITY 1.3

At the end of each day, an ice cream seller calculates how much money he has collected. Assuming that the ice creams all cost the same amount, draw a flowchart of an algorithm that would output the total amount collected during the day.
A teacher is marking his students’ test papers on a computer. If they achieve over 50 per cent, he would like the message ‘Well done!’ displayed. If they achieve over 90 per cent, they should also receive a second message stating ‘This is an excellent result.’ If they score 50 per cent or lower, the message will be “you must try harder next time.”

Draw a flowchart of an algorithm that would output these messages.
A flowchart to represent the algorithm to control the temperature of the shopping mall that we mentioned earlier would look like this. It contains sequence, selection and iteration.

There are three processes: one to check the temperature in the mall and two to either turn off the heaters and open the ventilators, or turn them on and close the ventilators.

There are two decisions: is the temperature greater or is it less than 20°C? Both are needed as the temperature could in fact be equal to 20°C.

There are only two possible answers for each decision question: yes or no, and the arrows show the relevant action to be taken depending on the answer.

Iteration is shown in the flow diagram as the arrows always lead the flow back to the first process. So the algorithm will repeat over and over again indefinitely.

As the algorithm repeats forever, no end symbol is required.

**WORKED EXAMPLE**

A flowchart to represent the algorithm to control the temperature of the shopping mall that we mentioned earlier would look like this. It contains sequence, selection and iteration.

There are three processes: one to check the temperature in the mall and two to either turn off the heaters and open the ventilators, or turn them on and close the ventilators.

There are two decisions: is the temperature greater or is it less than 20°C? Both are needed as the temperature could in fact be equal to 20°C.

There are only two possible answers for each decision question: yes or no, and the arrows show the relevant action to be taken depending on the answer.

Iteration is shown in the flow diagram as the arrows always lead the flow back to the first process. So the algorithm will repeat over and over again indefinitely.

As the algorithm repeats forever, no end symbol is required.

**Tip**

Look back at your answers for Activity 1.1.

**ACTIVITY 1.5**

Draw a flowchart to illustrate an algorithm for making a cup of tea.

It should include sequence, selection and iteration.

**Input and output**

In some of the previous examples, user input was required and information was output to the user.

**Key terms**

- **decision:** when a question is asked (as in selection) the answer will lead to one or more varied alternative actions
- **process:** an activity that a computer program is performing

A common request for user input is to enter a password.
Here is a flowchart of an algorithm to authenticate a password.

When a user enters a password, it has to be confirmed that it is true; it has to be authenticated.

In this flowchart there is iteration. If the password is incorrect, then in this particular algorithm, the user is asked to enter it again, and again, and again, forever or until it is correct.

Now assume that the user is given three attempts and then the account is locked.
Here is a flowchart to authenticate a password and lock the account after three incorrect attempts.

The algorithm will have to keep a count of the number of attempts that have been made.

In this algorithm, we have used a container called ‘attempts’ to keep a count of the number of attempts that have been made.

When incorrect attempts are made, the value of ‘attempts’ changes. It does not keep the same value throughout the algorithm, but it can change because it is a variable.

On the first attempt, it is changed to 1, on the second attempt it is changed to 2, and on the third attempt it is changed to 3.

If three attempts are made, then the container ‘attempts’ equals 3 and if there is still no correct password, then the account is locked.

Containers like ‘attempts’ are used in algorithms to store values that can change as the algorithm is running. As the values they contain can change, they are called variables.
ACTIVITY 1.6

The following flowchart shows the algorithm used to create usernames for a school network.

1. Flowcharts are used to show the sequence, selection and iteration in an algorithm.
3. Values used in an algorithm can be stored as variables.
Follow the algorithm shown in the flowchart and then answer the following questions.

1. Identify, and list the variables that have been used in this algorithm.

2. State the usernames that the flowchart will give to the following students (assume that there are no other students with the same username):
   i. Catherine Jones who joined the school in 2005.
   ii. Fred Green who joined the school in 2006.

3. A student has been given the username of 03SSmith13. State four facts that you can work out from this username.

**Pseudo-code**

As we have seen, flowcharts make algorithms easy for people to understand. However, computers cannot understand flowcharts, they can only understand programming languages.

In addition to flowcharts, algorithms can also be expressed using pseudo-code.

Pseudo-code is a form of structured English for describing algorithms. It is a generic, code-like language that can be easily translated into any programming language.

Writing in pseudo-code helps you to concentrate on the logic (the process) and efficiency of your algorithm before you have to start thinking about the code that you will be using. It is important to check that you have included all the stages in your process as it is easier to spot anything missing at this stage before you carefully translate it into code!

There are many different varieties of pseudo-code; some programming teams or organisations have their own versions of pseudo-code. However, all pseudo-code must be able to express the basic programming constructs that we will be looking at.

**Variables**

Before we look at examples of algorithms expressed in pseudo-code, we should look at variables in more detail.

As we mentioned above, a variable can be changed and manipulated as an algorithm is running.

So that programmers can keep track of variables, they are given names or identifiers.

An algorithm might contain many variables, so it is important to give them meaningful identifiers.

If you were creating an algorithm that stores people’s ages, it would be sensible to name or identify that variable as ‘age’ and not something like ‘X’ or ‘Y’.
Choosing variable identifiers

Variable identifiers should be as descriptive as possible so that anyone reading the code will be able to see what they represent. For example, look at these identifiers:

Code

\[ X \leftarrow 10 \]
and

\[ \text{distanceToSchool} \leftarrow 10 \]

Anyone reading the code would know immediately what the value ‘10’ represented in the second example.

1. Shorter identifiers are easier to type and spell. A longer identifier could easily be misspelt.

2. Longer identifiers may be used if they are more descriptive of the data they represent.

3. Some identifiers may be reserved words used by the programming language and cannot be used, for example: ‘print’.

4. In many programming languages, identifiers cannot begin with a number.

Naming conventions

Again, it is important for all variables to be written in a similar way throughout an algorithm. This makes the program consistent and easy to understand. This is even more important when a team of programmers is working on the same project.

- A commonly used convention is to use camel case (or CamelCase) for compound words.

  For example:
  
  FirstName, LastName,

  Often the first word of a compound word is given a lower case initial, for example ‘f’ for ‘first’ and ‘l’ for ‘last’:

  firstName, lastName.

  An alternative is to use an underscore. This method is often called snake case.

  For example:

  Code

  first_name, last_name

Assigning values to variables

All variables have to be given or assigned a value. This is done in assignment statements. It is done differently in different varieties of pseudo-code.

Tip

Study the AQA Pseudo-code Guide to see how assignment is made in the variety of pseudo-code that you will be using.
Here are two examples, one that assigns a number and the other that assigns text.

Code

myAge ← 21
myName ← “David”

Several variables can be assigned in the same statement:

Code

myAge ← 21, myName ← “David”

**Constants**

A constant is a value that cannot be altered by the program during normal execution: the value stays the same.

For example, a constant could be used to hold the number of hours in a day or rate of value added tax (VAT).

In pseudo-code these could be declared as:

Code

numberOfHoursInDay ← 24
vatRate ← 20

These can then be used in code such as:

Code

OUTPUT “Please enter the net cost of the item.”
netCost ← USERINPUT
fullCost ← netCost + (netCost/100*vatRate)

The use of the identifier makes the code far more understandable than multiplying by 20.

If there is a later change to the VAT rate then instead of having to search through the code for every occurrence of the VAT rate all that needs to be done is to change the value of the constant identifier ‘vatRate’ at the one place in the program.

Different programming languages use different key words for declaring constants.

In C++, ‘vatRate’ would be declared using ‘const’ and the data type:

Code

const int vatRate = 20

In Java the numberOfHoursInDay could be declared using ‘final’:

Code

final int numberOfHoursInDay = 24

Error messages will be generated if any attempt is made to change the values of these constants while the program is running.
Adding comments

When writing pseudo-code it is a good idea to add comments to explain to others, and often to remind yourself, what the code is intended to do.

To separate these comments from the actual code, the hash sign # is used.

The pseudo-code example used above could have been commented in the following way.

**Code**

```
OUTPUT “Please enter the length.”
length ← USERINPUT

OUTPUT “Please enter the width.”
width ← USERINPUT

area ← length * width

OUTPUT area
```

**WORKED EXAMPLE**

The first flowchart we looked at illustrated an algorithm to find the area of a rectangle. Here it is expressed in pseudo-code. It asks a user for the width and the length of a rectangle. It then calculates the area and prints it on screen.

Where variables are first assigned a value, they are shown in red.

```
OUTPUT “Please enter the length.”
length ← USERINPUT

OUTPUT “Please enter the width.”
width ← USERINPUT

area ← length * width

OUTPUT area
```

This is how the pseudo-code allows users to input data. Note how the prompt text is enclosed in speech marks. The variable ‘length’ will be assigned the value entered by the user.

The variable ‘width’ will be assigned the value entered by the user.

The variable ‘area’ will be assigned the value of the variable ‘length’ multiplied by the variable ‘width’.

The value stored in the variable ‘area’ is printed on the screen.

---

**Key term**

*comment:* a piece of information for the programmer. It does not form part of the program and is not executed by the computer. It is for information only.

**Tip**

Check the commands and keywords used in the AQA Pseudo-code Guide.

**Tip**

The printed message could have been made more user-friendly by including some text rather than just the area value. To do this, the text would need to be in speech marks to show that it is literal text and not a variable name. It would need to be joined to the variable ‘area’.

```
OUTPUT “The area is” + area
```

The literal text ‘The area is’ and then the value for the variable ‘area’ have been joined together (concatenated) using the ‘+’ symbol.
Keywords

If you look through the pseudo-code guide you will see words like ‘OUTPUT’, ‘FOR’, ‘ENDFOR’, ‘IF’ and ‘ENDIF’ that are used in commands. They are reserved or keywords as they have specific meanings for the language and therefore cannot be used as variable identifiers.

Operators

The algorithm you wrote in Activity 1.7 just printed out the data a user had input. Often you want the computer to do something with that data, usually a calculation.

In the worked example, the values of two variables were multiplied together to find the area.

$$\text{area} = \text{length} \times \text{width}$$

An operator is a symbol that tells the computer to perform a specific action on the data and manipulate it in a particular way. The data on which it performs the action is called the operand.

### Arithmetic operators

These are operators we have been using all our mathematical lives.

The following list shows the most common arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition: add the values together.</td>
<td>$3 + 6 = 9$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>firstResult + secondResult</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction: subtract the second value from the first.</td>
<td>$6 - 3 = 3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dailyProfit - dailyCosts</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication: multiply the values together.</td>
<td>$3 \times 6 = 18$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length \times width</td>
</tr>
<tr>
<td>/</td>
<td>Real division: divide the first value by the second number and return the result including decimal places.</td>
<td>$13/3 = 4.333$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>totalSweets/totalChildren</td>
</tr>
<tr>
<td>DIV</td>
<td>Quotient: like division, but it only returns the whole number or integer.</td>
<td>$13 \text{ DIV } 3 = 4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>totalSweets/totalChildren</td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus: this will return the remainder of a division.</td>
<td>$13/3 = 4$ remainder 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Therefore: $13 \text{ MOD } 3 = 1$</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation: this is for ‘powers of’.</td>
<td>$3^{^3} = 27$.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It is the same as writing $3^3$.</td>
</tr>
</tbody>
</table>
Order of operations
In computer programming, the order of precedence (the order in which you do each calculation) is the same as in mathematics and science.

To change the order of operations parentheses are used.

Therefore:

\[ 13 \times (3 + 6) = 13 \times 9 = 117 \]

Multiplication and division are carried out before addition and subtraction. Brackets (parentheses) are calculated before multiplication and division.

**ACTIVITY 1.8**

Using pseudo-code, create an algorithm that will ask a user to input the diameter of a wheel.

It should then calculate the area (assume Pi is 3.142) and output the result.

**Relational operators**
Let’s look again at the temperature control in the shopping mall.

![Flowchart of temperature control](chart.png)

**Tip**
In a division such as \(13/3\) the ‘13’ is called the **dividend** and the ‘3’ is called the **divisor**.

The **quotient** is the number of times the divisor divides into the dividend: in this case four times.

The DIV operator returns just the quotient and so in some pseudo-code dialects it is called **integer division** and the symbol used is a backslash \.

**Maths skills**
Remember BIDMAS!

This is how the following would be evaluated: \(3^3 \times 6 + (16 - 7)\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brackets</td>
<td>(3^3 \times 6 + (9))</td>
<td>(9 \times 6 + (9))</td>
</tr>
<tr>
<td>Indices</td>
<td>(9 \times 6 + (9))</td>
<td>(54 + (9))</td>
</tr>
<tr>
<td>Division</td>
<td>(54 + (9))</td>
<td>(63)</td>
</tr>
<tr>
<td>Multiplication</td>
<td>(63)</td>
<td>(13 \times 3 + 6 = 45)</td>
</tr>
<tr>
<td>Addition</td>
<td>(13 \times 3 + 6 = 45)</td>
<td>(63)</td>
</tr>
</tbody>
</table>
| Subtraction | \(63\) | \(13 \times 3 + 6 = 45\)

The area of a circle can be found by the formula \(\pi r^2\) where \(r\) equals the radius.
In this flowchart, two questions are being asked:

‘Is the temperature greater than 20°C?’

and

‘Is the temperature less than 20°C?’

We used two different relational operators: greater than and less than. Relational operators test the relationship between two values. As they compare the values they are sometimes called comparison operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to checks if two values are equal, for example: length = width</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal to checks to see if two values are not equal, for example: temperature ≠ 20</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than checks to see if one value is less than another, for example: temperature &lt; 20</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than checks to see if one value is greater than another, for example: temperature &gt; 20</td>
</tr>
<tr>
<td>≤</td>
<td>Less than or equal to checks to see if one value is less than or equal to another, for example: temperature ≤ 20</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than or equal to checks to see if one value is greater than or equal to another, for example: temperature ≥ 20</td>
</tr>
</tbody>
</table>

As mentioned above, we used two of these operators in the flowchart for the algorithm to control the temperature of a shopping mall.

If the temperature is greater than 20°C, then one course of action is carried out and if it is not, then something else is done.

We can rewrite this using the ‘IF’, ‘THEN’ and ‘ELSE’ statements in the pseudo-code.

Code

```
IF the temperature is greater than 20°C
THEN turn off the heaters and open the ventilators.
ELSE do something else.
ENDIF
```

If we put both of the selections together we could write:

Code

```
IF the temperature is greater than 20°C
THEN turn off the heaters and open the ventilators.
check the temperature again
ENDIF

IF the temperature is less than 20°C
THEN turn on the heater and close the ventilators.
check the temperature again.
ENDIF
```
When the computer is running a program coded from these ‘IF’ statements, it will run both of them. But if the first one is true, it does not need to run the second one as the temperature cannot be greater than and less than 20°C at the same time!

The computer will be wasting time. The algorithm is inefficient!

Therefore another term, ‘ELSE IF’ can be used.

In this example, $x$ is actually equal to 2.

<table>
<thead>
<tr>
<th>Inefficient algorithm</th>
<th>Efficient algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF $x = 1$ do this</td>
<td>IF $x = 1$ do this</td>
</tr>
<tr>
<td>IF $x = 2$ do this</td>
<td>ELSE IF $x = 2$ do this</td>
</tr>
<tr>
<td>IF $x = 3$ do this</td>
<td>ELSE IF $x = 3$ do this</td>
</tr>
<tr>
<td>IF $x = 4$ do this</td>
<td>ELSE IF $x = 4$ do this</td>
</tr>
<tr>
<td>IF $x = 5$ do this</td>
<td>ELSE IF $x = 5$ do this</td>
</tr>
<tr>
<td>IF $x = 6$ do this</td>
<td>ELSE IF $x = 6$ do this</td>
</tr>
<tr>
<td>ELSE do this</td>
<td>ELSE do this</td>
</tr>
<tr>
<td>ENDIF</td>
<td>ENDIF</td>
</tr>
</tbody>
</table>

In this example, it will not make much difference as there were only six ‘if’ statements but in large programs there might be hundreds or thousands of ‘IF’ statements and going through them all would waste a significant amount of time.

**WORKED EXAMPLE**

A teacher would like a program that allows her to enter three test results and calculate the average.

If the average is 50 or above, the program should output the message ‘Pass’ and if it is below 50, it should output the message ‘Fail’.

```
OUTPUT “Please enter first test result.”
test1 ← USERINPUT
OUTPUT “Please enter the second test result.”
test2 ← USERINPUT
OUTPUT “Please enter the third test result.”
test3 ← USERINPUT
average ← (test1 + test2 + test3)/3

IF average ≥ 50 THEN
    OUTPUT “Pass”
ELSE
    OUTPUT “Fail”
ENDIF
```

The user is asked to input the three test results which are stored in the variables test 1, test 2 and test 3.

The average is calculated and stored in the variable ‘average’. Notice how the additions are in brackets to ensure that they are done first.

This ‘IF’ statement checks to see if the average is equal to or greater than 50. If it is, the message ‘Pass’ is output.

There is no need for another ‘IF’ statement as if the average is not 50 or above it must be less than 50. Therefore an ‘ELSE’ statement is used.

In the pseudo-code dialect that we are using, an ‘ENDIF’ statement must be placed at the end of the selection block of code.
Indentation

It is considered good practice to indent statements that occur within an ‘IF’ statement. Some programming languages demand it, but most will accept it if you do not indent these statements. Indentation helps to show the logic in your algorithm.

The statements above should be set out in the following way:

```
Code
IF average ≥ 50 THEN
    OUTPUT “Pass”  This is indented as it is dependent on the ‘IF’ statement.
ELSE
    OUTPUT “Fail”  This is indented as it is dependent on the ‘ELSE’ statement.
ENDIF
```

ACTIVITY 1.9

Create an algorithm, expressed as pseudo-code, which asks a user to enter a number between 1 and 10.

If the number is five or less, the message: (the number input) ’is a low number.’ is output to the screen. If the number is over five, the message: (the number input) ’is a high number.’ is output.

ACTIVITY 1.10

A teacher created an algorithm to automatically generate one comment for a student’s test result based on the score out of ten.

```
OUTPUT “Please enter the test result.”
score ← USERINPUT
IF score < 5 THEN
    OUTPUT “You must try harder next time.”
ELSE IF score ≥ 5 THEN
    OUTPUT “You have gained half marks.”
ELSE IF score > 7
    OUTPUT “This is a good result.”
ELSE IF score > 8
    OUTPUT “This is an excellent result.”
ENDIF
```

The teacher input a score of nine and expected the message ‘This is an excellent result.’ to be printed. However, the comment produced was not as expected. What would have been generated? Explain why this was the case.
Boolean operators

These operators are named after George Boole, a 19th century English mathematician who formulated an algebraic system of logic.

They are sometimes referred to as logical operators.

Look at this Venn diagram that shows the number of rainy and sunny days in a fortnight.

<table>
<thead>
<tr>
<th>RAIN</th>
<th>SUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

There were three days where there was RAIN AND SUN.

There were six days where there was RAIN AND NOT SUN.

There were two days where there was SUN AND NOT RAIN.

There were 11 days where there was SUN OR RAIN: six days with RAIN only, two days with SUN only and three days where there were both.

The words in bold are all Boolean operators.

### Operator Function Example

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
</table>
| AND      | Logical AND operator If all of the operands are true, then the condition becomes true. | IF length > 6 AND width > 3 THEN  
area ← length * width  
ELSE  
OUTPUT “Rectangle is not large enough.”  
ENDIF |
| OR       | Logical OR operator If any of the operands are true, then the condition becomes true. | IF score < 0 OR score > 100 THEN  
OUTPUT “The score is invalid.”  
ENDIF |
| NOT      | Logical NOT operator Used to reverse the logical state of the operand. | IF NOT (length > 6 AND width > 3) THEN  
OUTPUT “Rectangle is not large enough.”  
ELSE  
area ← length * width  
ENDIF |

The words in bold are all Boolean operators.
Remember to indent all of the statements which follow an ‘IF’ statement as they are only executed because of it.

**WORKED EXAMPLE**

A student would like to select a suitable T-shirt from local shops. The colour could be red, blue or white, the size needs to be medium and the shop must be no more than 10 miles away.

Create an algorithm to help the student find suitable T-shirts.

OUTPUT “Please enter the colour of T-shirt.”

colour ← USERINPUT

The user is asked to input the T-shirt colour and size and the distance to the shop.

The values entered are stored in variables.

OUTPUT “Enter size as S, M or L.”

size ← USERINPUT

OUTPUT “Enter distance to shop in miles.”

distance ← USERINPUT

IF (colour = “red” OR colour = “blue” OR colour = “white”) AND size = “M” AND distance ≤ 10 THEN

IF all of the variables meet the criteria, then this message is printed:

OUTPUT “This T-shirt is suitable.”

ELSE

IF all the variables do not meet the criteria, then this message is printed:

OUTPUT “No. This T shirt is not suitable.”

ENDIF

The ‘ENDIF’ statement must be placed after the ‘IF, ELSE’ code.

**ACTIVITY 1.11**

An 11–18 senior school would like you to design software to help with student administration. The software should allow the user to input the year and tutor group of each student. There are seven year groups designated from 7 to 13.

In each year, there are four tutor groups: red, green, blue and yellow.

Enter these details and check that the data entered is acceptable, that is, check that the year group or tutor group entered actually exists.
Nested ‘IF’ statements

A ‘nested IF’ refers to two ‘IF’ statements, one running inside the other.

**WORKED EXAMPLE**

A shop gives a discount of 5 per cent on purchases over £100, up to a maximum of £50. If the amount to be discounted would be greater than this, it has to be reduced back to £50.

```
OUTPUT “Please enter the money spent in £.”
purchase ← USERINPUT
IF purchase > 100 THEN
    discount ← purchase / 100 * 5
    IF discount > 50 THEN
        discount ← 50
    ENDIF
ENDIF
```

The user is asked to enter the purchase price.

IF the purchase price is greater than 100 the discount is calculated.

The nested ‘IF’ is then used to check ‘IF’ the discount is more than 50.

Notice how the second ‘IF’ statement is indented.

IF the discount is more than 50, it is reduced back to 50.

‘ENDIF’ for the inner ‘IF’.

‘ENDIF’ for the outer ‘IF’.

**‘CASE’ statements**

As well as the ‘IF’ … ‘ELSE IF’ … ‘ELSE’ statements, this is another method which can be used for selection.

It is very useful where users have to select an item from a list.

**WORKED EXAMPLE**

In a multiple-choice question there are four possible answers, labelled A, B, C and D. To select the answer, the users have to enter one of these letters. They will then be informed if they are correct or incorrect. There should also be a method to inform if they enter a letter other than the four allowed. (In this example, the correct answer is option C.)

Using ‘IF’ … ‘ELSE IF’ … ‘ELSE’ statements, it could be coded as:

```
OUTPUT “Please select an option.”
answer ← USERINPUT
IF answer = “A” THEN
    OUTPUT “Sorry. That is incorrect.”
```

Download Worksheet 1.5 from Cambridge Elevate
A student is writing code to ask a user to enter the month number, for example: 1 = January and 12 = December. The user should then receive a message giving the name of the month.

Write an algorithm in pseudo-code, using ‘CASE’ statements, to solve this problem.

ELSE IF answer = “B” THEN
    OUTPUT “Sorry. That is incorrect.”
ELSE IF answer = “C” THEN
    OUTPUT “Well done. That is the correct answer.”
ELSE IF answer = “D” THEN
    OUTPUT “Sorry. That is incorrect.”
ELSE
    OUTPUT “That option is not recognised.”
ENDIF

This could also be written as:

IF answer = “A” OR answer = “B” OR answer = “D” THEN
    OUTPUT “Sorry. That is incorrect.”
ELSE IF answer = “C” THEN
    OUTPUT “Well done. That is the correct answer.”
ELSE
    OUTPUT “That option is not recognised.”
ENDIF

Using CASE it would be coded as:

OUTPUT “Please select an option.”
answer ← USERINPUT
CASE answer OF
    “A”: OUTPUT “Sorry. That is incorrect.”
    “B”: OUTPUT “Sorry. That is incorrect.”
    “C”: OUTPUT “Well done. That is the correct answer.”
    “D”: OUTPUT “Sorry. That is incorrect.”
ELSE
    OUTPUT “That option is not recognised.”
ENDCASE

ACTIVITY 1.12

A student is writing code to ask a user to enter the month number, for example: 1 = January and 12 = December. The user should then receive a message giving the name of the month.

Write an algorithm in pseudo-code, using ‘CASE’ statements, to solve this problem.
## Remember

1. Pseudo-code is a generic, code-like language that can be easily translated into any programming language.
2. An operator is a symbol that tells the computer to perform a specific action on the data and manipulate it in a particular way.
3. An arithmetic operator is a mathematical function that can take one or two operands and performs a calculation on them.
4. Relational operators test the relationship between two values.
5. Boolean or logical operators test all the operands in a complex statement and return a value of true or false.
6. A ‘nested IF’ is a complete ‘IF’ statement running inside another ‘IF’ statement.
7. ‘Switch/case’ can be used to select from multiple options.

## Practice question

1. Using pseudo-code examples, explain the following terms:
   a. variables
   b. relational operators
   c. Boolean operators
   d. nested IF statements.

## Your final challenge

‘We drive anywhere’ is a taxi firm who have the following criteria when working out the customer charge. The following rules apply:
- Between 8 am and 8 pm, passengers are charged £3 for the first mile and £2 for every further mile.
- If there are more than four passengers, there is a charge of £2 for each extra passenger.
- Between 8 pm and 8 am, the overall charge is doubled.

Design an algorithm that will allow the taxi drivers to input the required information that will then output the total charge. Display your algorithm as a flowchart and as pseudo-code.

---

**Download Self-assessment 1 worksheet from Cambridge Elevate (this content has not been approved by AQA)**