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 flow charts and pseudocode
- 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 its 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 flow charts and pseudocode.
- Your challenge is to use this knowledge to help a taxi company calculate its fares.

Why algorithms?

Algorithms run our world! In every area algorithms are used to decide what action should be taken in a particular circumstance and 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 3 moves, there are over 9 million possible moves available; after 4 moves there are over 288 billion possible moves. 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. 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 something 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 also be further divided into sub-tasks as they may be made up of smaller steps.

For example, ‘showering’ could involve many different steps: turning on the shower, setting the correct temperature etc. 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 one successfully. Compare it with other members of your group and note any differences in sequence and sub-tasks.

**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.

**Key terms**

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

**Complete the Cambridge Computing Online activity**

www.cambridge.org/links/kose4001

**Download Worksheet 1.1 from Cambridge Elevate**
The getting to school example might seem pretty easy. Like a typical algorithm, it is just a list of steps. However, here is part of another algorithm, which is part 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 115g caster sugar, a dessert spoonful 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 cook has to beat the mixture and then stop and ask themselves if it resembles a fluffy cloud. If it doesn’t they have to beat again, check again, beat again, check again etc. 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. In other words, as well as doing things over and over again, they have to make a decision. Does it resemble a fluffy cloud?

Key terms

iteration: when a task is repeated until there is a required outcome
selection: a question is asked, and depending on the answer, the program takes one of two courses of action

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, iteration and selection.

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

Put plug in the bath.  Sequence
Turn on hot tap.  Sequence
Is the water at the correct temperature?  Selection
Turn cold tap until water is at the correct temperature.  Iteration
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 A**

- Get out of bed.
- Shower.
- Get dressed.
- Turn on kettle.
- Turn on toaster and put in bread.
- 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.

**List B**

- Get out of bed.
- Turn on kettle and ask someone to watch it for you.
- Put bread in toaster, turn it on and ask someone to watch it for you.
- Shower.
- Get dressed.
- Make tea.
- Butter toast and add marmalade.
- Drink tea and eat toast.
- Gather school books and put in bag.
- Put on shoes and coat.
- Leave the house.

**Watch out**

Although the algorithm is now more efficient, it would not have been safe to leave the kettle and toaster unattended. Computer scientists must consider health and safety issues when designing real-life solutions.

**List B** is more efficient as it could be implemented in less time. The kettle and the 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 heaters and open the ventilators.
3. If the temperature is less than 20°C then turn on 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 action taken.
- **Iteration**: instruction 4 instructs 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.

### Flow diagrams

Flow diagrams can be used to represent algorithms visually.

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

The symbols used are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="start.png" alt="Start/End" /></td>
<td>This represents the start or end point of the flow diagram. You always start your flow diagram with this symbol and use it to finish the flow diagram.</td>
</tr>
<tr>
<td><img src="input.png" alt="Input/Output" /></td>
<td>You use this to represent data input or data output. For example it could be a number or name entered by a user.</td>
</tr>
<tr>
<td><img src="decision.png" alt="Decision" /></td>
<td>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 isn’t then an arrow will point to a different action.</td>
</tr>
<tr>
<td><img src="process.png" alt="Process" /></td>
<td>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 which has been asked, e.g. ‘Turn on the heaters’ or ‘Turn off the heaters’.</td>
</tr>
</tbody>
</table>
Here is the flow diagram of an algorithm to calculate the area of a rectangle:

Start → Input length of rectangle → Input width of rectangle → Area = length of rectangle * width of rectangle → Output area of rectangle → End

This start/stop symbol is used at the beginning and end of an algorithm.

This is input. The user is being asked to enter the length of the rectangle.

This is input. The user is being asked to enter the width of the rectangle.

This is a process to be carried out by the computer. It calculates the area of the rectangle.

This output. The area of the rectangle is displayed (output) for the user.

This start/stop symbol is used at the beginning and end of an algorithm.

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.

Complete Interactive Activity 1b on Cambridge Elevate

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 flow diagram of an algorithm that would output the total amount collected during the day.
Here is a flow diagram of an algorithm to identify a vertebrate animal. It includes sequence and selection.

Start

Is it warm blooded?

YES

Does it have feathers?

YES

It is a bird

End

NO

It is a mammal

End

NO

Does it have scales?

YES

It is an amphibian

End

NO

Does it have gills?

YES

It is a fish

End

NO

It is a reptile

End

Does it have feathers?

YES

NO

Does it have scales?

YES

NO

NO

End

End

End

ACTIVITY 1.4

A teacher is marking his students’ test papers on a computer. If they achieve over 50% he would like the message ‘Very well done’ displayed. If they achieve over 90%, they should also receive a second message stating ‘This is an excellent result’. If they score 50% or lower, the message will be ‘You must try harder next time’.

Draw a flow diagram of an algorithm that would output these messages.
A flow diagram 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.

**Key terms**

**decision:** when a question is asked (as in selection) the answer will lead to one or more different alternative actions

**process:** action that is taken, sometimes as a result of a decision, in order to achieve a desired outcome

**ACTIVITY 1.5**

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

It should include sequence, selection and iteration.

(Hint: Look back at your answers for Activity 1.1.)

**Input and output**

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

A common request for user input is to enter a password.
Here is a flow diagram of an algorithm to authenticate a user’s password.

When a user enters a password it has to be confirmed that it is the same as the one stored; it has to be authenticated.

In this flow diagram 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 flow diagram 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 doesn’t keep the same value throughout the algorithm; it can change because it is a variable.

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

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 flow diagram shows the algorithm used to create usernames for a school network.

1. Flow diagrams are used to show the sequence, selection and iteration in an algorithm.
2. Symbols for start, end, input, output, decision and process are used in flow diagrams.
3. Values used in an algorithm can be stored as variables.
As we have seen, flow diagrams make algorithms easy for people to understand. However, computers can’t understand flow diagrams they can only understand programming languages. In addition to flow diagrams algorithms can also be expressed using pseudocode. Pseudocode is a kind of structured English for describing algorithms. It is a generic, code-like language that can be easily translated into any programming language.

Writing in pseudocode 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 than when you have carefully translated it into code!

There are many different varieties of pseudocode; some programming teams or organisations have their own versions of pseudocode. However, all pseudocode 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 pseudocode, 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 and 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’.

---

Follow the algorithm shown in the flow diagram and then answer the following questions.

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

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

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

---

**Key term**

**pseudocode**: a language that is similar to a real programming language, but is easier for humans to understand although it doesn’t actually run on a computer. It can easily be converted to a regular programming language.

**Key term**

**identifier**: the ‘name’ given to a variable.

---

© Cambridge University Press 2016
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:

\[ X = 10 \]
and
\[ \text{distanceToSchool} = 10 \]

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

Here are some golden rules for choosing variable identifiers.

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 e.g. ‘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 (also known as CamelCase or camelCase) for compound words.
  
  e.g.
  
  FirstName, LastName.

  Often the first word is given a lower case initial, e.g.
  
  firstName, lastName.

- An alternative is to use an underscore. This method is often called snake case.
  
  e.g. 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 pseudocode.

Tip

People have variables! We all have a variable that is given the identifier ‘Age’ and stores the length of time we have been alive. It changes every second but is celebrated when it changes at the end of a year. So you could say ‘Happy variable change!’

Our pulse rate, blood pressure and temperature are also our variables.

The school stores variable data about you in variables identified as ‘Year’ and ‘Tutor Group’. Their values change each school year.

Tip

Study the OCR Pseudocode Guide to see how assignment is made in the variety of pseudocode 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”

**Watch out**

Unfortunately the = operator is used to assign variables and this can cause confusion. In this case it does not mean ‘equal to’. It means ‘assign the variable this value’. It can be assigned other values as the program is running.

**Constants**

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

**Tip**

Constants are also given identifiers.

For example a constant identified as ‘conversionFactor’ could be assigned the value of 0.39. It could then be used to convert a distance in centimetres into inches:

**Code**

lengthInInches = lengthInCm * conversionFactor.

**Remember**

1. A variable is a value that can change while a program is running.
2. Variables are given names, called identifiers.
3. Variable identifiers should be descriptive of the data they are storing e.g. age or firstName and not identifiers such as X or Y.
4. Variables should be written in a consistent way, e.g. they should all be camel case or all snake case and not a mixture of the two.
5. Variables are assigned a value using the = symbol.
6. Constants are values that cannot be altered as a program is running.

OK. Now back to using pseudocode.
When writing pseudocode it is a good idea to add comments to explain to others, and often to remind yourself, what the code is intended to do.

Adding comments

To separate these comments from the actual code, two forward slashes /** are used.

The pseudocode example used above could have been commented in the following way:

**Code**

```
length = input ("Please enter the length")  // Ask the user for the length of the rectangle
width = input ("Please enter the width")   // Ask the user for the width of the rectangle
area = length * width                      // Find the area by multiplying the length by the width
print (area)                                // output the area
```

**WORKED EXAMPLE**

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

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

- **length** = input ("Please enter the length")
  - This is how the pseudocode 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.

- **width** = input ("Please enter the width")
  - The variable ‘width’ will be assigned the value entered by the user.

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

- **print area**
  - The value stored by the variable ‘area’ is printed on the screen.

**Tip**

Check the commands and keywords used in the OCR Pseudocode 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 have to have been included in speech marks to show that it is literal text and not a variable name. It would then have to have been joined to the variable ‘area’.

```
print("The area is " + area)
```

The computer will print the literal text “The area is ” and then the value for the variable ‘area’ as they have been joined together (concatenated) using the ‘+’ symbol.

**Key term**

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

If you look through the Pseudocode Guide you will see words like ‘print’, ‘for’, ‘next’, ‘if’ and ‘endif’, which are used in commands. They are reserved words 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 an operand.

Key terms

- **operator**: the symbol that tells the computer what to do
- **operand**: the data the operator is working on

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 firstResult + secondResult</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction: subtract the second value from the first.</td>
<td>6 - 3 = 3 dailyProfit - dailyCosts</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication: multiply the values together.</td>
<td>3 * 6 = 18 length * 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 totalSweets/totalChildren</td>
</tr>
<tr>
<td>DIV</td>
<td>Quotient: like division, but it only returns the whole number or integer.</td>
<td>13 DIV 3 = 4 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. Therefore: 13 MOD 3 = 1</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation: this is for ‘powers of’.</td>
<td>3^3 = 27. It is the same as writing 3³.</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.

**Key term**

**parentheses**: brackets

Therefore:

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

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

**ACTIVITY 1.8**

Using pseudocode, 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.

**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 4 times.
- The **DIV** operator returns just the quotient and so in some pseudocode 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:

\[ 3^3 \times 6 + (16 - 7) \]

- **Brackets**: \( 3^3 \times 6 + (9) \)
- **Indices**: \( 9 \times 6 + (9) \)
- **Division**: \( 54 + (9) \)
- **Multiplication**: \( 13 \times 3 + 6 = 45 \) and not 117
- **Addition**: 63
- **Subtraction**: 33 × 6 + (16 – 7)

The area of a circle can be found by the formula \( \pi r^2 \) where \( r \) equals the radius.
In this flow diagrams 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>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to: checks if two values are equal. Note that the double equals sign are used (==) to differentiate it from assigning a value to a variable when one equal sign is used.</td>
<td>length == width</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to: checks to see if two values are not equal.</td>
<td>temperature != 20</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than: checks to see if one value is less than another.</td>
<td>temperature &lt; 20</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than: checks to see if one value is greater than another.</td>
<td>temperature &gt; 20</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to: checks to see if one value is less than or equal to another.</td>
<td>temperature &lt;= 20</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to: checks to see if one value is greater than or equal to another.</td>
<td>temperature &gt;= 20</td>
</tr>
</tbody>
</table>

As mentioned above we used two of these operators in the flow diagram 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 isn’t then something else is done.

We can rewrite this using the ‘if’, ‘then’ and ‘else’ statement in the pseudocode.

**Code**

```
if the temperature is greater than 20°C then turn off the heater and open the ventilators.

else do something else.
```

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

**Code**

```
if the temperature is greater than 20°C then turn off the heater and open the ventilators.

if the temperature is less than 20°C then turn on the heater and close the ventilators.

else go back and check the temperature.
```
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 doesn’t need to run the second one as the temperature can’t 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, ‘elseif’, is 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>elseif x = 2 do this</td>
</tr>
<tr>
<td>if x = 3 do this</td>
<td>elseif x = 3 do this</td>
</tr>
<tr>
<td>if x = 4 do this</td>
<td>elseif x = 4 do this</td>
</tr>
<tr>
<td>if x = 5 do this</td>
<td>elseif x = 5 do this</td>
</tr>
<tr>
<td>if x = 6 do this</td>
<td>elseif 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 won’t 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 it should output the message ‘Pass’ and if it is below 50 the message ‘Fail’.

```plaintext
test1 = input("Please enter first test result.")
test2 = input("Please enter second test result.")
test3 = input("Please enter third test result.")

average = (test1 + test2 + test3) / 3

if average >= 50 then
    print ("Pass")
else
    print("Fail")
endif
```

The user is asked to input the three test results which are stored in the variables test1, test2 and test3.

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 pseudocode 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 don’t. Indentation helps to show the logic in your algorithm.

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

```
if average >= 50 then
    print(“Pass”)  This is indented as it is dependent on the ‘if’ statement.
else
    print(“Fail”)  This is indented as it is dependent on the ‘else’ statement.
endif
```

**ACTIVITY 1.9**

Create an algorithm, expressed as pseudocode, that asks a user to enter a number between 1 and 10.

If the number is 5 or less the message: (the number input) “ is a low number.” is output to the screen and if it is over 5 the message: (the number input) “ is a high number.” is output.

```
 Complete Interactive Activity 1g on Cambridge Elevate
```

**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 10.

```
score = input(“Please enter the test result.”)
if score < 5 then
    print(“You must try harder next time.”)
elseif score > 5 then
    print(“You have gained half marks.”)
elseif score > 7
    print(“This is a good result.”)
elseif score > 8
    print(“This is an excellent result.”)
endif
```

The teacher input a score of 9 and expected the message ‘This is an excellent result.’ to be printed. However, the comment produced was not as he 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, which shows the number of rainy and sunny days in a fortnight.

There were 3 days where there was RAIN AND SUN.
There were 6 days where there was RAIN AND NOT SUN.
There were 2 days where there was SUN AND NOT RAIN.
There were 11 days where there was SUN OR RAIN (6 with rain only, 2 with sun only and 3 where there were both.).

The words underlined in bold are all Boolean operators.

<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  
print(“Rectangle is not large enough.”)  
endif  

Note: In this example the length must be greater than 6 AND the width must be greater than 3 to work out the area. |
| **OR**   | Logical OR operator  
If any of the operands are true then the condition becomes true. | if score < 0 OR score >100 then  
print(“The score is invalid.”)  
endif  

Note: In this example if the score entered is less than 0 OR if it is greater than 100 it will not be accepted. |
| **NOT**  | Logical NOT operator  
Used to reverse the logical state of the operand. | if NOT (length >= 6 AND width >= 3) then  
print(“Rectangle is not large enough.”)  
endif  

Note: This produces the same result as the first example. |
A student would like to select a suitable T-shirt from local shops. The colour could be red, blue or white, the size should be medium and the shop must be no more than 10 miles away.

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

colour = input(“Enter colour of T-shirt.”)
size = input(“Enter size as S, M or L.”)
distance = input(“Enter distance to shop in miles.”)

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.

if(colour == “red” OR colour == “blue” OR colour == “white”) 
    AND size == “M” AND distance <= 10 then

The ‘if’ statement is used to select the desirable characteristics.

The items for colour selection are enclosed in brackets so that they are evaluated together.

    print(“This T shirt is suitable.”)

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

else

    print(“No. This T shirt is not suitable.”)

If all the variables do not meet the criteria, then this message is printed.

endif

The ‘endif’ statement must be placed after the ‘if’, ‘else’ code.
Create an algorithm that would allow a user to enter these details and check that the data entered is acceptable, i.e. check that the year group or tutor group entered actually exists.

‘Nested if’ statements

The ‘if … elseif … else’ statements are sometimes referred to as nested if statements as the ‘elseif’ and ‘else’ statements run inside the overall ‘if’ statement.

But strictly speaking, ‘nested if’ should refer to two complete ‘if’ statements, one running inside the other.

WORKED EXAMPLE

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

```
purchase = input("Please enter the money spent in £.")
```

The user is asked to enter the purchase price.

```
if purchase > 100 then
```

If the purchase price is greater than £100 the discount is calculated.

```
discount = purchase / 100 * 5
```

```
if discount > 50 then
```

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

```
discount = 50
```

If the discount is more than £50 it is reduced back to £50.

```
endif
endif for the inner ‘if’.
endif for the outer ‘if’.
```
Switch/case

As well as the ‘if … elseif … else’ statements, switch/case is another method that 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 their answer the users have to enter one of those letters. They will then be informed if they are correct or incorrect. There should also be a method to inform them if they enter a letter other than the four allowed. (In this example the correct answer is option C.)

Using ‘if … elseif … else’ statements, this could be coded as:

```python
answer = input("Please select an option.")
if answer == "A":
    print("Sorry. That is incorrect.")
elseif answer == "B":
    print("Sorry. That is incorrect.")
elseif answer == "C":
    print("Well done. That is the correct answer.")
elseif answer == "D":
    print("Sorry. That is incorrect.")
else:
    print("That option is not recognised.")
endif
```

Using switch/case it would be coded as:

```python
answer = input("Please select an option.")
switch answer:
    case "A":
        print("Sorry. That is incorrect.")
    case "B":
        print("Sorry. That is incorrect.")
    case "C":
        print("Well done. That is the correct answer.")
```

Tip

Study the OCR Pseudocode Guide to see the keywords for this method and how it is used.
case “D”:
    print(“Sorry. That is incorrect.”)
default:
    print(“That option is not recognised.”)
endswitch

**Practice questions**

1. Using pseudocode examples, explain the following terms:
   a. variables
   b. relational operators
   c. Boolean operators
   d. ‘nested if’ statements

**Complete Interactive Activity 1j on Cambridge Elevate**

**Your final challenge**

- ‘We drive anywhere’ is a taxi firm who have the following criteria when working out the customer charge.

  Between 8am and 8pm the following rules apply:
  1. £3 for the first mile and £2 for every additional mile.
  2. If there are more than 4 passengers there is a charge of £2 for each extra passenger.

  Between 8pm and 8am, the overall charge is doubled.
  a. Design an algorithm that will allow the taxi drivers to input the required information and will then output the total charge.
  b. Display your algorithm as a flow diagram and as pseudocode.

**Remember**

1. Pseudocode is a generic, code-like language which 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 takes 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.