

PROBLEM SOLVING AND PROGRAM DESIGN

9.1 Introduction to problem solving

Software is the name given to computer programs that tell (instruct) the hardware how to work. Software therefore includes computer programs that create system and application software. Each program exists because a problem first existed, and a solution was needed to solve that problem.

Solving a problem

You should first develop an **algorithm** to solve the problem. An algorithm is a sequence of *precise* instructions which results in a solution. If your algorithm is vague and has conflicting logic, you will not get the correct answer.

Problem solving is usually broken into two phases.

Algorithm phase

In the first phase, there are five general steps:

- 1 Clearly define the problem that you want to solve.
- 2 Propose solutions and evaluate each one.
- 3 Select the most reasonable solution.
- 4 Design an algorithm that is precise and well thought out to solve the problem.
- 5 Test your algorithm. You must be sure your algorithm works correctly before you can write a program for it.

Implementation phase

In this phase, there are three main steps:

- 1 Translate your algorithm using a programming language such as BASIC, Pascal, C or VisualBasic.

- 2 Execute the program code.

- 3 Maintain the program.

For now, let's focus on the algorithm phase.

Defining the problem

When you first analyse a problem, you should specify your objectives, that is, what the program is meant to do. The following four steps are used to organise and summarise a program's objectives:

- 1 Specify the output: What kind of output are you expecting from this program? Will this output be one or more numbers, text, numbers *and* text, or even a graphic? Is this output in readable form or is it for input to another program? Will it be displayed on a monitor as soft copy, printed or stored in a file?
- 2 Specify the input: If you know what the output is, you can specify what values are needed for the input. Data can be captured by an input device such as a keyboard, transferred from a secondary storage device or even entered via voice recognition.
- 3 Specify the processing: What processing should be done on the input to get the necessary output? For example, what calculations are needed? Is the processing grouped as a batch or completed in real time as the data is captured?

Most programs also:

- 4 Specify the storage: Determine how the data and even the information or results will be stored temporarily or for future use.

Input–processing–output (IPO) charts

IPO charts are used to identify the:

- ♦ inputs: the information you need to solve the problem
- ♦ processing: the steps needed to convert the input data to the desired outputs
- ♦ outputs: the goal of the problem solution.

An IPO chart is not actually a chart, but a table with three columns. Each column represents three components: input, output and processing. The storage component is not usually documented in the chart.

Example 1

You are given a problem of finding the sum of three numbers. What is the goal or output? What input information do you have?

The output or goal is a number which is the sum of the three numbers. The input is made up of the values of the three numbers. What steps are needed to convert the input(s) into the goal(s) or output? The processing requires:

- ♦ the input of the three numbers
- ♦ adding the three numbers together and saving the result as a single number
- ♦ the output of the sum of the numbers.

Notice that the processing column is written in English. The completed IPO chart is shown below.

Input	Processing	Output
number1 number2 number3	Add the three numbers together	Result of the addition

You can follow an IPO chart by reading each column from left to right. For example, the input column indicates that you input three values. In the processing column you add the values that were input. The third column outputs the result of the addition.

Example 2

You are given the problem of finding the average temperature by using maximum and minimum temperature readings. You then calculate and output the average temperature:

$$(\text{max temp} + \text{min temp}) \div 2$$

The completed IPO chart is shown below.

Input	Processing	Output
max_temp min_temp	Calculate (max_temp + min_temp) / 2	average_temp

Example 3

Read a number, add 10% and output the result.

Input	Processing	Output
number	result = number + (number * 0.10) OR result = number * 1.10	result

Example 4

Input the regular price of an item; calculate the discount amount at 20% of the regular price and also the item's discounted price. Output the discount amount and the discounted price.

Input	Processing	Output
regular_price	discount amount = 0.20 * regular price discount price = regular price – discount amount	discount amount discount price

Questions

- 1 Explain the two major phases in problem solving.
- 2 What do the letters I, P and O refer to in an IPO chart?
- 3 Explain the purpose of an IPO chart.
- 4 Show the correct order when solving a problem. The steps are:
 - i select solution
 - ii design algorithm
 - iii define problem
 - iv test algorithm
 - v propose solutions

To help the simplification and development of the programming problem, algorithms can be created as pseudocode or flowcharts.

An algorithm is a set of step-by-step instructions required to obtain the solution to a problem. It must have a set of rules, be explicit and have a clear stopping point. It is really used to expand the 'processing' part of the IPO chart using diagrams for **flowcharts** or English-type statements for pseudocode. The algorithm should start with a title and end with the keyword END. Developing a program can be a major undertaking, so good algorithm design is very important. It is not always possible to satisfy all the requirements of a good algorithm completely at the same time. Some problems may be complex and hence require complex solutions. However, there are some general goals in designing algorithms. A good algorithm should be:

- ♦ *correct*: it should accept all inputs (even invalid inputs!) and output a correct answer or meaningful response or message
- ♦ *simple*: each step of the algorithm should perform one logical step in solving the problem
- ♦ *clear*: the algorithm should be easy to read and understand
- ♦ *precise*: the algorithm should present the solution steps precisely and concisely without referring to low-level (program code) details
- ♦ *easy to implement*: the algorithm should be relatively easy to translate into a programming language
- ♦ *efficient*: the algorithm should enable the program code to produce results quickly, depending on the problem size, and not waste any memory or time.

Now that you have been introduced to algorithms, you should become familiar with some terminology. These terms apply to both pseudocode and flowcharts.

Variables and constants

Let's explain how data is usually stored when it is input. A variable or constant is an area or space in memory that holds data that a program might use or manipulate. Both could contain text or numerical values.

A constant is a type of variable where the values do not change for that algorithm. Examples of constant values are 3.14 for pi (π), 0.175 for value-added tax (VAT) or 2002 for a year of birth.

The values in a variable can change throughout the algorithm as data is processed. Examples of variable names are *number* and *total*. Each **variable** is given a name so that you can assign values such as numbers or text to them and refer to them later to read the values. Variables typically store values of a given type. Some types of data are shown in Table 9.1.

Table 9.1 Basic data types

Data type	Description	Examples
Integer	Integer or 'whole' numbers, positive or negative	63, -12, 0
Real	Numbers including fractional numbers, positive or negative	12.63, 0.5, -8.0
Character	A single character such as a letter of the alphabet or punctuation, as shown on a keyboard	'G', 'b', '*'
String	A collection of characters such as a word, phrase or sentence	"A+" or "Good Job!"
Boolean	Can contain only one of two values	TRUE or FALSE; YES or NO; MALE or NOT MALE

You should use meaningful variable names in your programs so that if you or someone else needs to review the logic of the data being processed, then the variables will be easy to remember and understand.

Statements and keywords

A **statement** is a description of the processing that can include an action or condition. Instructions within the statement are called **keywords**. Examples of keywords are INPUT, READ, OUTPUT, DISPLAY, PRINT or WRITE.

Conditional statements

Conditional statements allow decisions to be made in a program. This includes deciding which statements are to be executed (carried out). For example, you might want a set of statements to execute only if certain conditions are met.

Loops

Loops are useful for repeating parts of a program. That is, they will repeatedly execute a section of program until the end condition is satisfied. In order to exit from a **loop**, you must have a method for checking to see if you have completed the task. Once a loop terminates, control is returned to the first statement after the block of statements in the loop.

Subroutines

For big programs, it is easier to work on small sections by separating the algorithm into individual sections. These sections are known as 'subroutines', 'functions', 'modules' or 'procedures'. A **subroutine** is a named section of a program that can be repeatedly called on to perform a given set of instructions by other parts of the program. Sometimes it returns information to another part of the program, such as accepting

a number between 1 and 5 and then returning the corresponding day of the week. Other subroutines might carry out instructions, such as printing the headings of an invoice.

Using subroutines saves writing the same code several times in the program. This method of focusing on the program's main goal, and then separating the program into manageable components is called 'divide and conquer' or top-down design. The individual subroutines can even be tested in isolation to check that they correctly perform their function.

Questions

- 1 State whether the following statements are true or false:
 - a An algorithm can be written as pseudocode.
 - b An example of pseudocode is a flowchart.
 - c Flowcharts use English-type sentences.
 - d Flowcharts use specific symbols.
 - e A flowchart can be developed into an algorithm.
- 2 Explain the purpose of using a variable.
- 3 Explain the difference between a variable and a constant.
- 4 State the most suitable data type for each of the following:
 - a the current year
 - b someone's height
 - c a mobile phone number
 - d is it day or night?
 - e your blood type.
- 5 For each of the descriptions in question 4, suggest a suitable variable name to store the data.

Pseudocode is not programming code, but uses English-type words and phrases that are clear enough to be easily converted into programming code. It uses the same statements, keywords, variables, conditional statements and loops described in the previous section to create the instructions for a solution to a programming problem. Pseudocode can also be developed from an IPO chart to specify what data is input and processed into information.

Let's first review the IPO chart that finds the sum of three numbers.

Input	Processing	Output
number1	Add the three numbers	The result of the
number2	together	addition
number3		

Note that the variables `number1`, `number2` and `number3` will store the first, second and third numbers that are input.

The processing column involves short statements explaining how the data is managed. Here, the values that were input into the variables `number1`, `number2` and `number3` are added. The third column indicates that the result of the addition is output.

The name of the segment of pseudocode is *Add_three_numbers*. The keywords **INPUT** and **OUTPUT** are used to indicate that data is input, usually using an input device such as a keyboard, while the output can be as soft copy to the screen of a monitor. The keyword **END** or **STOP** denotes the end of the statements.

The pseudocode in Example 5 is produced from the IPO chart.

Example 5

```
Pseudocode_5a: Add_three_numbers
INPUT number1
INPUT number2
INPUT number3
Add number1 + number2 + number3
OUTPUT result of number1 + number2 +
number3
END of pseudocode
```

If the three variables that contain the three numbers have the same data type (for example, an integer), then the statements to input the values can be placed on one line as shown in the updated pseudocode below.

```
Pseudocode_5b: Add_three_numbers
INPUT number1, number2, number3
Add number1 + number2 + number3
OUTPUT result of number1 + number2 +
number3
END of pseudocode
```

Now that we are familiar with variables, we can also place the result of the calculation in a variable. Let's name this variable *result*.

```
Pseudocode_5c: Add_three_numbers
INPUT number1, number2, number3
result ← number1 + number2 + number3
OUTPUT result
END of pseudocode
```

The new variable *result* on the left side of the statement will contain the answer from the calculation of the numbers from the right side of the statement.

Calculations can include the left arrow (\leftarrow) or the equals sign ($=$) to indicate that the result of a calculation is stored in a variable.

Here is yet another set of pseudocode for Example 5 using different keywords to READ in the data and to PRINT the result. The keyword END can also be used on its own to denote the end of the statements.

```
Pseudocode_5d: Add_three_numbers
READ number1, number2, number3
result = number1 + number2 + number3
PRINT result
END
```

Sequential statements

Sequential statements, such as those in Example 5, are executed one after the other from the first statement to the last.

Let's look again at the IPO chart from Example 2, which finds the average temperature from two values.

Input	Processing	Output
max_temp	Calculate	average_temp
min_temp	$(\text{max_temp} + \text{min_temp}) / 2$	

The following pseudocode is produced using sequential statements from the information in the IPO chart.

Example 6

Find the average temperature.

```
Pseudocode_6a: Find_average_temp
INPUT max_temp, min_temp
average_temp = (max_temp + min_temp) / 2
OUTPUT average_temp
END
```

First, the value for the maximum temperature is input into the variable (*max_temp*), then the value for the minimum temperature is input into the second variable (*min_temp*). The resulting number from the calculation is placed in the variable (*average_temp*). The result stored in *average_temp* is then output.

Conditional branching

Conditional branching is used when there is a choice between two options. Two types of conditional branching are IF-THEN and IF-THEN-ELSE statements.

IF-THEN statements

The IF-THEN statement suggests that one or more statements will only be considered based on a condition or the answer to a question.

```
IF (the condition is true)
THEN (carry out one or more statement)
ENDIF
```

Sometimes the keyword ENDIF is used to indicate the end of the IF-THEN statement. Any statement below the ENDIF in the pseudocode is therefore part of the general pseudocode and not specific to the IF-THEN statement.

Let's use an example. You wish to output the total of three test results, but only if the total mark is greater than 50. In this case, we use the IF-THEN statement.

The pseudocode below shows that the three marks are each input into a variable. The marks are added and the sum is stored in a variable named *total*. The IF-THEN statement is then used to check if the total mark is greater than 50. We do not know the value of the three marks that will be input. Therefore, IF the total mark is greater than 50 THEN its value is output. The sequence of statements stops there.

Example 7

Print the sum of three numbers if their total is greater than 50.

```
Pseudocode_7: Add_three_numbers
READ number1, number2, number3
total = number1 + number2 + number3
IF total > 50
THEN PRINT total
ENDIF
END
```


The condition (or question) asked is: is the total mark greater than 50?

If it is, then the condition is true, the mark is indeed greater than 50. The next statement in the sequence indicates to output the total mark. That is, if we used 20, 25 and 10 as the three marks to be input, then the total mark is $20 + 25 + 10 = 55$. Since 55 is greater than 50, it is expected that the value 55 will be output.

The IF–THEN statement does not consider if the total mark is less than or even equal to 50, as there are no statements in the pseudocode that instruct that result to be output. If the total mark is less than or equal to 50, the condition will be false, then the next statement (THEN PRINT total) in the sequence is skipped, and the pseudocode reaches the END without printing anything.

The keywords in this example are READ, IF, THEN, PRINT, ENDIF and END. The variables are *number1*, *number2*, *number3* and *total*.

The general format of an IF statement is:

```
IF (condition is true)
THEN T-statement(s)
ENDIF
```

If **condition is true**, execute **T-statement(s)**

IF is a keyword.

Condition is a Boolean expression, which means its value is either TRUE or FALSE.

T-statement(s) are one or more statements that are included only if the result of the condition is TRUE.

ENDIF tells the computer that the IF–THEN statements finish here.

If the condition is FALSE, skip **T-statement(s)** – that is, do not execute them.

Consider the following example to find out if a student is younger than 13 years old. If so, then we want to output the statement “Student is not a teenager”. For this, we need to use an IF–THEN statement.

Example 8

First, create an IPO chart.

Input	Processing	Output
age	Check if age < 13	If true then output “Student is not a teenager”

You can write the pseudocode much as it is written in the above sentence.

```
Pseudocode_8a: Is_student_a_teenager
INPUT age
IF age < 13
THEN output “Student is not a teenager”
ENDIF
END
```

Another variation of the pseudocode is to input the student’s year of birth and the current year to calculate the age of the student.

```
Pseudocode_8b: Is_student_a_teenager
thisyear = 2019
INPUT birthyear
age = thisyear - birthyear
IF age < 13
THEN OUTPUT “Student is not a teenager”
ENDIF
END
```

Note that *thisyear* = 2019 is an example of a constant where the value 2019 is not modified in the algorithm. The variable *birthyear*, however, changes when new data is input.

Example 9

Consider the following example that checks for exam marks of 80 or over:

```
Pseudocode_9a: Exam_results
INPUT mark
IF mark >= 80
THEN OUTPUT "Excellent!"
OUTPUT "Please see your teacher"
ENDIF
END
```

The pseudocode will output the following statements for values stored in the variable *mark*:

Value of mark	Example of mark	Statements output
80 or greater	86	Excellent! Please see your teacher
less than 80	71	Please see your teacher

Notice the use of indentation in the following IF-THEN statement:

```
Pseudocode_9b: Exam_results
INPUT mark
IF mark >= 80
THEN  OUTPUT "Excellent results!"
      OUTPUT "Please see your teacher"
ENDIF
END
```

This suggests that both statements will be output if the condition is true and the variable *mark* contains a value greater than or equal to 80. Nothing is printed if a value less than 80 is stored in the variable *mark*.

Value of mark	Example of mark	Statements output
80 or greater	86	Excellent results! Please see your teacher
less than 80	71	

IF-THEN-ELSE statement

The IF-THEN-ELSE statement directs the algorithm to one or more statements if the outcome of the

condition is valid or true. The algorithm is directed to another set of statements if the outcome of the condition is not valid or false.

```
IF (the condition is true)
THEN (carry out one or more statements)
ELSE (carry out one or more statements)
ENDIF
```

Example 10

As an example, suppose you wish to print 'the sum is less than 50' if the sum of three numbers is less than 50, but output their sum if it is greater than 50. This means you have two options: you can either print the comment or print the sum.

Input	Processing	Output
Read number1, number2, number3	Total = number1 + number2 + number3	If true, then Output Total
	Check if total > 50	If false, then Output 'The sum is less than 50'

In this case, use the IF-THEN-ELSE statement. The pseudocode shows the modification.

```
Pseudocode_10: Add_three_numbers
READ number1, number2, number3
total = number1 + number2 + number3
IF total > 50
THEN PRINT total
ELSE PRINT "The sum is less than 50"
```

The following examples also illustrate the IF-THEN-ELSE statement.

Example 11

```
Pseudocode_11: Is_Millee_a_teen
INPUT thisyear
INPUT birthyear
age = thisyear - birthyear
IF age < 13
THEN OUTPUT "Millee is a teenager"
ELSE OUTPUT "Millee is not a teenager"
ENDIF
END
```


You could also choose to output Millee's age to produce the following:

```
IF age < 13
THEN OUTPUT "Millee is a teen, her age
is", age
ELSE OUTPUT "Millee is not a teen"
ENDIF
```

The general form of an IF-THEN-ELSE statement is:

```
IF (condition is true)
THEN T-statement(s)
ELSE F-statement(s)
ENDIF
```

If **condition is true**, execute **T-statement(s)**. Do not execute **F-statement(s)**.

If **condition is false**, execute **F-statement(s)**. Do not execute **T-statement(s)**.

Nested conditions

Nested conditions involve the use of IF-THEN or IF-THEN-ELSE statements, either separately or combined. Example 12 shows a nested condition using an IF-THEN-ELSE and an IF-THEN statement:

Example 12

```
Pseudocode_12: Exam_results
INPUT mark
IF mark >= 80
THEN OUTPUT "Excellent results!"
    OUTPUT "Please see your teacher"
ELSE IF mark >= 70
    THEN OUTPUT "Satisfactory
    results!"
OUTPUT "Please await the end of term
report"
ENDIF
END
```

If the mark is 80 or greater, then the statements for 'Excellent results!' and 'Please see your teacher' will be output. Otherwise, if the mark is 70 or greater, then the statement for satisfactory results will be output. However, regardless of the mark, another statement to await the end of term report is also output.

Loops

The statements of a loop are arranged to depend on the result of a variable. Most loops cycle through the statements as follows:

- ◆ Input a starting value to a specific variable – this variable usually determines whether or not the loop executes or not.
- ◆ Test the variable against a condition.
- ◆ Execute the body of the loop.
- ◆ Update the value of the variable.

There are two types of loop statements:

- ◆ indefinite: when you do not know in advance how many times to repeat the loop (WHILE or REPEAT loops)
- ◆ definite: when you know in advance how many times to repeat the loop (FOR loop).

Indefinite loops

WHILE loop

Start value ____ (then) ____ check if condition is true
____ (then) ____ perform statements

REPEAT loop

Start value ____ (then) ____ perform statements ____
(then) ____ check if condition is true

WHILE loop

The WHILE loop repeatedly executes one or more statements as long as the condition is true. The condition in a WHILE loop is tested at the beginning of the loop, so it is possible for the statement not to be executed at all.

The general form of the WHILE loop is:

WHILE (condition is true)

Statement(s)

ENDWHILE

The ENDWHILE keyword indicates the end of the statements in the loop.

Note that statements are indented between WHILE and ENDWHILE statements.

Here are some examples that illustrate the WHILE loop. Where the number of times that the loop should be repeated may be known or unknown, the WHILE loop can still be used when the condition for stopping the loop is known.

Example 13

Pseudocode_13: Numbers_in_a_loop

number = 1

WHILE (number <= 3)

BEGIN

number = number + 1

OUTPUT "the number is", number

ENDWHILE

OUTPUT "out of loop"

END

These two statements are executed only in the loop

Note that the value 1 is assigned to the variable *number*. This is also called *initialising* a variable.

Here is how the loop is executed:

- a Number containing the value 1 enters the loop.
The condition (number <= 3) is checked. That is (1 <= 3) is true.
- b Enter the loop:
Number is increased to 2.
Output is 'the number is 2'.
The condition (number <= 3) is checked. That is (2 <= 3) is true.
- c The loop is repeated:
Number is increased to 3.
Output is 'the number is 3'.
The condition (number <= 3) is checked. That is (3 <= 3) is true.

- d The loop is repeated:

Number is increased to 4.

Output is 'the number is 4'.

The condition (number <= 3) is checked. That is (4 <= 3) is now false.

- e Leave the loop:

The next statement following the loop is executed (after the ENDWHILE statement).

The output is:

the number is 2

the number is 3

the number is 4

out of loop

Example 14

In this example, the indentation is used to indicate those statements that are within the WHILE loop, without using the BEGIN and ENDWHILE keywords.

Pseudocode_14: Younger_than_20

age = 15

WHILE (age < 20)

Output "You are a teenager"

age = age + 1

Output "You are not a teenager"

While the value of the variable age is less than 20, the statement 'You are a teenager' will be output. age is then increased by 1 and checked in the condition (age < 20) before again passing through the loop. As soon as age reaches the value 20 then the statement 'You are not a teenager' is output.

Example 15

Let's now try an example of a loop where the number of times to repeat the loop is unknown. First, create an IPO chart that will repeatedly read each students' exam mark into a variable. It is not known how many exam marks are to be entered, but a mark with the value of -1 will indicate that it is the last mark. Once all marks are entered, the average mark is calculated and output.

Input	Processing	Output
mark	While (mark is not equal to -1) Add mark to total marks Calculate the average	average mark

An example of the corresponding pseudocode using the WHILE loop is shown below:

```
Pseudocode_15: Average_of_exam_marks
total = 0
count = 0
average = 0
OUTPUT 'enter a mark'
INPUT mark
WHILE (mark is not equal to -1)
    total = total + mark
    count = count + 1
    OUTPUT 'enter a mark'
    INPUT mark
ENDWHILE
average = total/count
DISPLAY average
END
```

The logic of the pseudocode is as follows:

- ♦ Three variables are initialised to store the total marks, the number of marks entered and the average mark.
- ♦ The user is prompted to enter the first exam mark.
- L **a** Once the mark is not -1, the loop is entered.
- O **b** The first mark is added to the total and counted as the first mark.
- O **c** The user is prompted to again to enter another mark which is stored on input.
- P **d** The loop is repeated at **a**.
- ♦ Once a mark of -1 is entered, the loop is exited.
- ♦ The average of the marks is calculated.
- ♦ The average mark is output.

Sometimes algorithms require the use of sequential statements, conditional statements and even loops. Example 15 illustrates the use of a WHILE loop; however, it can be further expanded to include a conditional statement. As you practise more with writing pseudocode and even writing programming code, you would want to test the program by entering some marks. Testing the algorithm is explained in a later section, but for now, consider what the output could be if a user enters -1 as the first exam mark.

- ♦ The condition in the WHILE loop is tested.
- ♦ The condition is now true since mark = -1.
- ♦ The loop is bypassed to calculate the average mark which would be average = 0/0.
- ♦ This would cause an error since 0/0 cannot be determined.

To avoid this error, after the ENDWHILE statement the following pseudocode for IF-THEN-ELSE statement can be added:

```
IF count = 0
THEN DISPLAY 'no marks entered'
ELSE
    average = total/count
    DISPLAY average
```

REPEAT loop

The REPEAT construct repeatedly executes one or more statements as long as the specified condition is false. Note that this condition is tested at the end of the loop, so the statement will always be executed at least once.

The general form of the REPEAT statement is:

```
REPEAT
    Statement(s)
UNTIL (condition is true)
```

Example 16

This example uses the REPEAT loop to output some numbers between 1 and 3, unlike Example 13, which used a WHILE loop.

```
Pseudocode_16: Numbers_in_a_loop
number = 1
REPEAT
    number = number + 1
    OUTPUT "the number is", number
UNTIL (number = 3)
OUTPUT "out of loop"
END
```

The variable number is first initialised to 1. Here is how the loop is executed:

- a Number with the value 1 enters the loop.
- b Number is increased to 2.
Output is 'the number is 2'.
The condition (number > 3) is checked.
2 > 3 is not true so the statements in the loops are executed again.
- c Number is increased to 3.
- d Output is 'the number is 3'.
Since 3 = 3, 'out of loop' is now output.
The output is:
the number is 2
the number is 3
out of loop

Example 17

```
Pseudocode_17: Attend_classes
INPUT day
REPEAT
    OUTPUT 'attend classes'
UNTIL (day = weekend day)
END
```

In this example, the statement 'Attend classes' will be output before it is checked to determine whether the data is, say, Saturday or Sunday. The statement is

therefore output at least once before the condition is checked in the UNTIL statement. When the condition is checked and is true, then the next statement following the UNTIL statement is executed. This means that you will attend classes when there may be no school!

Example 18

Alternatively:

```
Pseudocode_18: Attend_classes
INPUT day
REPEAT
    OUTPUT "do not go to school"
UNTIL (it is a school day)
END
```

As an alternative example, the statement 'do not go to school' is output until the condition 'it is a school day' becomes true. Again, the statement is output at least once before the condition is checked in the UNTIL statement. This means that you will stay home one day when there may be school!

Example 19

```
Pseudocode_19: Under_20
age = 18
REPEAT
    OUTPUT "you are under 20 years old"
    age = age + 1
UNTIL (age = 20)
END
```

The statement will be output even though the value of age may be 20. This is because the condition (age = 20) is checked *after* the statement is output.

Example 20

Along with the WHILE loop, the REPEAT loop is also used when the number of times to repeat the loop is unknown. Let's look again at Example 15 which will repeatedly read students' exam marks into a variable.

Since it is not known how many exam marks are to be entered, a value of -1 will indicate that it is the last mark. Once all marks are entered, the average mark is calculated and output.

An example of the corresponding pseudocode using the REPEAT and IF-THEN-ELSE statements is shown below:

```
Pseudocode_20: Average_of_exam_marks_
using_REPEAT_loop
total = 0
count = 0
average = 0
REPEAT
    DISPLAY 'Enter mark'
    INPUT mark
    IF mark = -1
    THEN DISPLAY 'end of marks'
    ELSE
        total = total + mark
        count = count + 1
UNTIL mark = -1
IF count = 0
THEN DISPLAY 'no marks entered'
ELSE
    average = total/count
    OUTPUT average
END
```

The logic of the pseudocode is as follows:

- ♦ Three variables are initialised to store the total marks, the number of marks entered and the average mark.
- L **a** The user is prompted to enter the first exam mark.
- O **b** If the mark is -1, the loop is exited.
- O **c** Otherwise, the first mark is added to the total and counted as the first mark.
- P **d** The loop is repeated at **a**.

- ♦ A mark of -1 must be entered to be outside the loop.
- ♦ Check if count is 0. This means no marks were entered so nothing to count and no average to calculate.
- ♦ Otherwise, the average of the marks is calculated.
- ♦ The average mark is output.

FOR loop

The FOR loop is used only when the start value and the end value are known.

The general form of the FOR loop is:

```
FOR <variable> = <start value> TO/DOWNTO <final value> DO
    Statements(s)
ENDFOR
```

Note:

- ♦ The variable must be in order so that it can be counted.
- ♦ The variable begins with the <start value>.
- ♦ The variable changes by 1 every time the statements are executed. TO counts up DOWNTO counts down.
- ♦ The loop terminates when the variable reaches the final value.
- ♦ ENDFOR denotes the end of the FOR loop.

The following examples written in pseudocode will help you understand the FOR loop.

Example 21

```
Pseudocode_21: School
Saturday = 2
Sunday = 3
FOR day = Saturday to Sunday DO
    OUTPUT "It is weekend"
ENDFOR
END
```

The variable *Saturday* is initialised with the integer 2, and the variable *Sunday* is initialised with the integer 3.

In the FOR statement, the variable *day* is assigned the first value of *Saturday* which is the number 2.

'It is weekend' is output.

day is then given the next value which is *Sunday*, represented by the number 3.

'It is weekend' is again output.

Since the values for the start and end of the FOR statement have been reached, the statement 'It is weekend' is therefore output twice, once for Saturday and once for Sunday.

Example 22

```
Pseudocode_22: Countdown
FOR countdown = 10 DOWNT0 1
    OUTPUT countdown
ENDFOR
END
```

The numbers 10, 9, 8, 7, 6, 5, 4, 3, 2 and 1 are output.

Example 23

```
Pseudocode_23: Daylight
FOR time = daybreak to sunset DO
    OUTPUT "it is not night"
ENDFOR
END
```

If a day consists of daybreak, morning, noon, afternoon, evening, sunset and night, then 'it is not night' will be printed six times, for each time it passes through the loop from *daybreak* to *sunset*.

Example 24

```
Pseudocode_24: Printing_numbers
FOR number = 1 to 2 DO
    Output "The number is", number
OUTPUT "Out of loop"
ENDFOR
END
```

The output here is:

The number is 1

The number is 2

Out of loop

Consider another algorithm which adds all the even numbers between 1 and 20 inclusive and then displays the sum. It uses a REPEAT loop. The equivalent pseudocode is:

Example 25

```
Pseudocode_25: Sum_of_even_numbers
sum = 0
count = 1
REPEAT
    IF count is even
        THEN sum = sum + count
count = count + 1
UNTIL count > 20
DISPLAY sum
END
```


Questions

- 1 Correct the following statements where each contains one error:
 - a REPEAT (count = count - 2) THEN count = 25
 - b FOR (count <= 20) DO
count = count * 5
 - c IF total = count * 5
WHILE display count
- 2 State which of the following statements define WHILE and REPEAT loops:
 - a Loop A
 - i executed at least once
 - ii condition is checked after statements are completed
 - iii statements are performed until condition is TRUE.
 - b Loop B
 - i not always executed
 - ii condition is checked before statements are completed
 - iii statements are performed until condition is FALSE.
- 3 Consider the following algorithms:




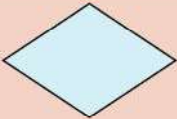


Pseudocode_9.3.3left BEGIN Blu = 4 Met = 5 INPUT Num Blu = 2 * Num + Blu Met = Blu + Num - Met DISPLAY Blu, Num, Met END	Pseudocode_9.3.3right WHILE word <> 'page' INPUT word IF word = 'page' THEN Display 'OK' ELSE Display 'Oh No' ENDWHILE
---	---
- a Write an example from each algorithm that represents a(n):
 - i variable
 - ii constant
 - iii keyword
 - iv arithmetic operator
 - v Boolean operator.
- b State whether the left algorithm contains any of the following:
 - i loop
 - ii conditional statement
 - iii sequential statement
 - iv output statement.
- 4 You wish to input each mark from four class tests, then calculate and print the total mark. Write the pseudocode using:

a sequential statements	c a REPEAT loop
b a WHILE loop	d a FOR loop.
- 5 Write pseudocode to input an unknown number of integers. The value -1 should stop reading numbers and calculate the average.

9.4 Flowcharts

Flowcharts are algorithms that use symbols to depict the input, processing and output of data and information. These symbols use the same terms to illustrate the flow of data as the statements in pseudocode. Table 9.2 shows the symbols with their descriptions.

Table 9.2 Flowchart symbols

Symbol	Name	Description
	Terminator	Used to identify the start and end of a flowchart
	Input / Output	Used to accept data or to output information
	Process	Statements in this symbol perform calculations
	Decision	Also called a conditional statement. Used to ask a question to determine the next step or option. One line in but more than one out
	Connector	A link to an external element or detail not included in the visible flowchart
	Flow lines	Lines with arrow heads determine the flow of data from one symbol to another

Each symbol represents an action or condition. Within most symbols, text is used to describe what is happening to the data, while the arrows indicate the flow of data from one symbol to the next.

Some rules for flowcharts include:

- ♦ Every flowchart must have a Start symbol and a Stop symbol, unless you are drawing a section of a flowchart. The Start and Stop symbols are called the terminals or terminators.

- ♦ Use arrow heads on flow lines where the direction of flow may not be obvious.
- ♦ The main symbols used in a flowchart are the Decision (also known as Selection) and the Process (or Sequence) symbols.
- ♦ The flow of sequence is generally from the top of the page to the bottom of the page. However, this can vary: sometimes there are loops which need to flow back to a process or decision.

There are also some important rules for the symbols. These rules also generally apply to algorithms and pseudocode:

- ♦ Processes have only one entry point and one exit point.
- ♦ Lines with arrow heads indicate the flow of sequence.
- ♦ Decisions have only one entry point, one TRUE (or Yes) exit point and one FALSE (or No) exit point. You should also know when to use the IF-THEN and IF-THEN-ELSE structures.
- ♦ The REPEAT loop has a process before the decision. That is, a REPEAT loop will always execute the process at least once. This is an important point to remember.
- ♦ The WHILE loop is generally the reverse of the REPEAT loop: the decision comes first, followed by the process. The WHILE loop is usually drawn so that it loops *while* the condition is true, the REPEAT loops *until* the condition becomes true.

It is important that you practise drawing flowcharts since they illustrate the orderly flow of data in an algorithm. If a flowchart is not created properly it can cause confusion and result in statements that are not logical or produce incorrect information. You should first practise drawing the symbols and then draw some of the sample flowcharts presented in the examples so that you can compare and improve as you continue to practice.

The following flowcharts represent some of examples that were written using pseudocode in the previous section.

Sequence statements

Similar to pseudocode, where a set of statements are executed one after the other from the first statement to the last, flowcharts also depict the sequence of statements by using symbols.

As shown in Figure 9.2, a flowchart has Start and Stop symbols. There are also small circles called connector symbols which indicate that the flowchart is part of a larger one (Fig 9.1). Sequential statements are joined by directional arrows. Each symbol has an arrow pointing to it and another arrow pointing away from it to another symbol. The symbol also contains text that explains the processing to be carried out.

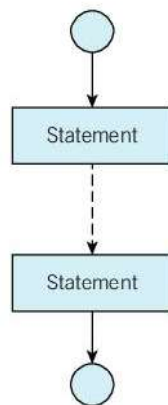


Fig 9.1 Flowcharts have sequential statements

Consider a flowchart created from Example 5 which finds the sum of three numbers (Fig 9.2). The IPO chart and pseudocode are also given here for comparison.

Example 26

IPO chart to find the sum of three numbers.

Input	Processing	Output
number1 number2 number3	Add the three numbers together	The result of the addition

```
Pseudocode_26: Add_three_numbers
INPUT number1, number2, number3
Add number1 + number2 + number3
OUTPUT result of number1 + number2 +
number3
END of pseudocode
```

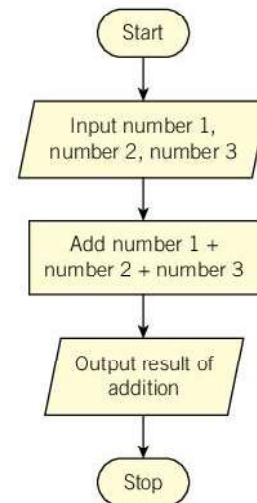


Fig 9.2 Flowchart for finding the sum of three numbers

The flowchart shows a sequence of symbols from the Start symbol to the Input symbol, Process symbols, Output symbol and Stop symbol. The text within each symbol is similar to that of the IPO chart and is an example of pseudocode.

Example 6 in the previous section finds the average of two temperatures. The flowchart is shown below. Notice that the text in the process symbol does not need to specify the precise calculation. A description of what processing is carried out or a series of symbols that detail each calculation are both acceptable.

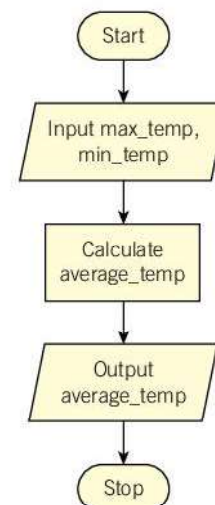


Fig 9.3 A flowchart for Example 6

Conditional statements

Conditional statements for flowcharts use IF–THEN or IF–THEN–ELSE structures.

IF–THEN structure

The flowchart for the IF–THEN structure is shown in Figure 9.4. Note that the condition symbol usually contains a question where the answer is one of only two options, such as yes or no, true or false. The conditional statement can selectively skip or include statements based on the outcome of the condition.

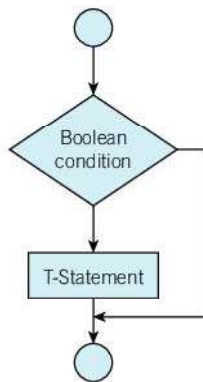


Fig 9.4 Flowchart for the IF–THEN structure

Figure 9.5 shows the flowchart based on Example 8b. The algorithm accepts a birth year, calculates the age based on the current year and then displays a comment if the student is a teenager. This flowchart contains Process, Input/Output and Decision symbols (also

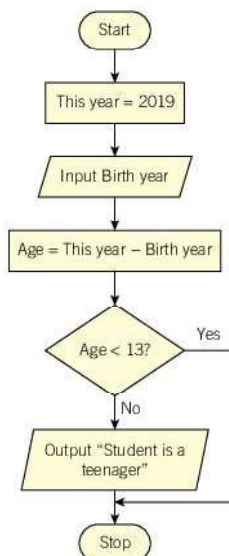


Fig 9.5 Flowchart that calculates an age to determine if the student is a teenager

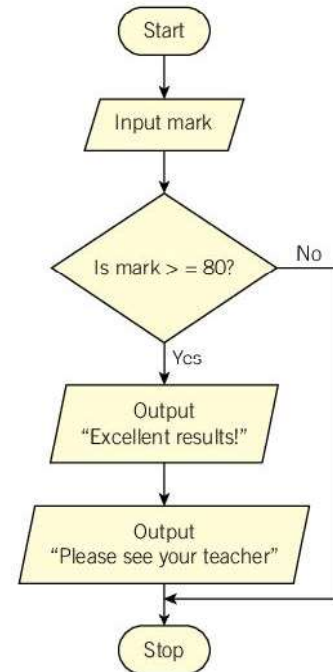


Fig 9.6 A flowchart for Example 9b

the Start/Stop symbols). A Process symbol is used to initialise the variable *This year*.

The IF–THEN Decision symbol in Figure 9.6 shows that the two statements are output if the condition is true, otherwise they are skipped to join the directional arrow below the last output statement. The figure also shows two consecutive Output symbols, but depending on the algorithm, both output statements can be placed in one symbol. Notice that the ‘Yes’ and ‘No’ placed near to the Decision symbol are in different locations when compared to the flowchart in Figure 9.5. These labels must match the logic of the algorithm and can therefore be used at the appropriate point of the symbol.

IF–THEN–ELSE structure

The flowchart for the IF–THEN–ELSE structure is shown in Figure 9.7. Note that the condition symbol again contains a question. However, depending on the result of the question, the flow of data is directed to different statements.

Figure 9.8 shows the flowchart based on Example 10, which illustrates the IF–THEN–ELSE structure. Three numbers are input. They are added, and the

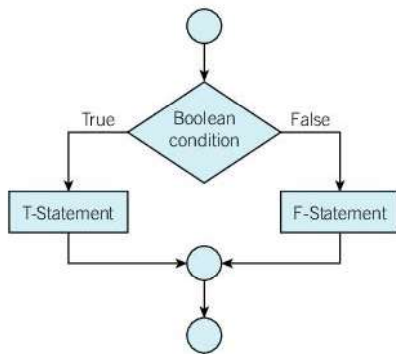


Fig 9.7 Flowchart for the IF-THEN-ELSE structure

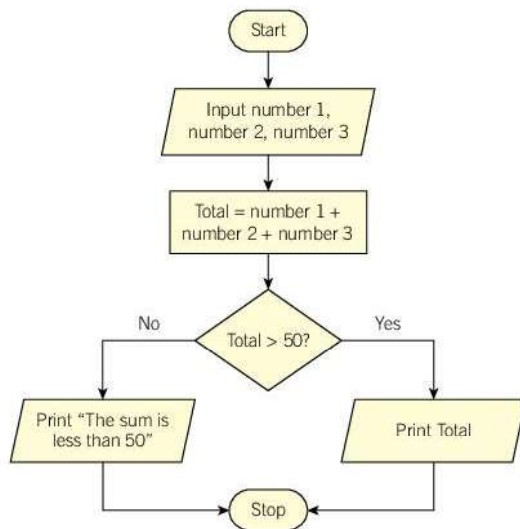


Fig 9.8 Flowchart for the IF-THEN-ELSE structure that determines whether a number is greater than 50

result is stored in a variable named total. If the total is greater than 50, then the flowchart displays the total, otherwise it displays a comment.

Figure 9.9 expands the flowchart of Figure 9.5 to show the IF-THEN-ELSE structure on determining whether a student is a teen or not.

Nested conditions

Figure 9.10 illustrates a combination of two conditional structures that output statements based on marks. The accompanying pseudocode is shown in Example 27. This flowchart shows how the flow of data does flow from the start to the stop symbol even though the statements are not from top to bottom. As you trace the flowchart, notice that regardless of the mark, the same final statement will be output.

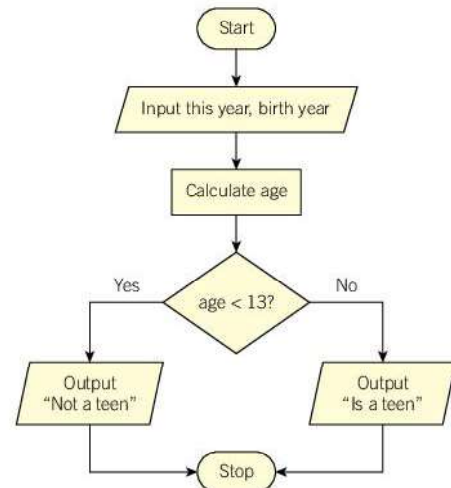


Fig 9.9 A flowchart that displays different comments based on an age

Example 27

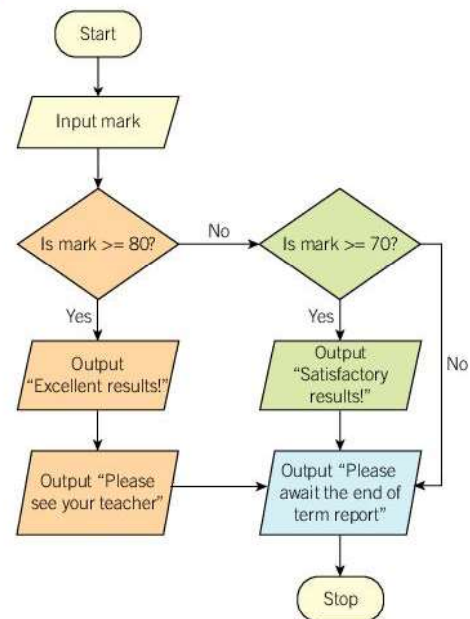


Fig 9.10 A flowchart can combine conditional structures that output statements based on the result of a mark

```

Pseudocode_27: Exam_results
INPUT mark
IF mark >= 80
    THEN OUTPUT "Excellent results!"
        OUTPUT "Please see your teacher"
ELSE IF mark >= 70
    THEN OUTPUT "Satisfactory results!"
OUTPUT "Please await the end of term
report"
ENDIF
END
  
```

Loops

The WHILE and REPEAT loops are used when you do not know in advance how many times to repeat the loop. The FOR loop is preferred when you *do* know the starting and ending values or the number of times to repeat the loop.

Figures 9.11 and 9.12 compare the flowcharts of the WHILE and REPEAT structures.

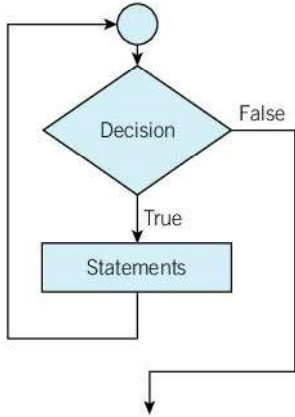


Fig 9.11 Flowchart for the WHILE loop

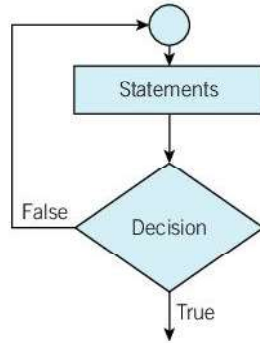


Fig 9.12 Flowchart for the REPEAT loop

Note where the Decision symbols are located in each loop. The condition in a WHILE loop is tested at the beginning of the loop, so it is possible for the statement not to be executed at all. The condition in the REPEAT loop is tested at the end of the loop, so the statement will always be executed at least once.

Examples 28 and 29

Compare the two flowcharts in Figures 9.13 and 9.14. Each flowchart displays some numbers between 1 and 4 based on a condition in each loop.

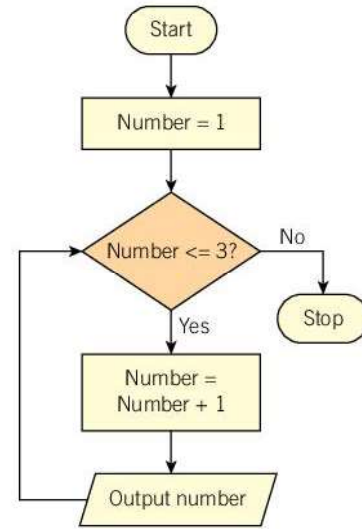


Fig 9.13 Using the WHILE loop to output numbers between 1 and 3 based on a condition

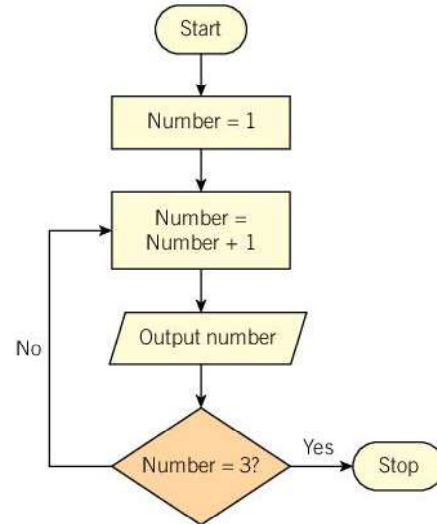


Fig 9.14 Using the REPEAT loop to output numbers between 1 and 3 based on a condition

The pseudocode for the WHILE and REPEAT loops are again shown for comparison. The FOR loop can be represented by either the WHILE or the REPEAT loop.

```

Pseudocode_28: while loop example
number = 1
WHILE (number <= 3) DO
BEGIN
    number = number + 1
    OUTPUT "the number is", number
ENDWHILE
OUTPUT "out of loop"
END
  
```

```

Pseudocode_29: repeat loop example
number = 1
REPEAT
    number = number + 1
    OUTPUT "the number is", number
UNTIL (number = 3)
OUTPUT "out of loop"
END
  
```


The flowchart in Figure 9.15 adds together all the even numbers between 1 and 6 and then displays the sum.

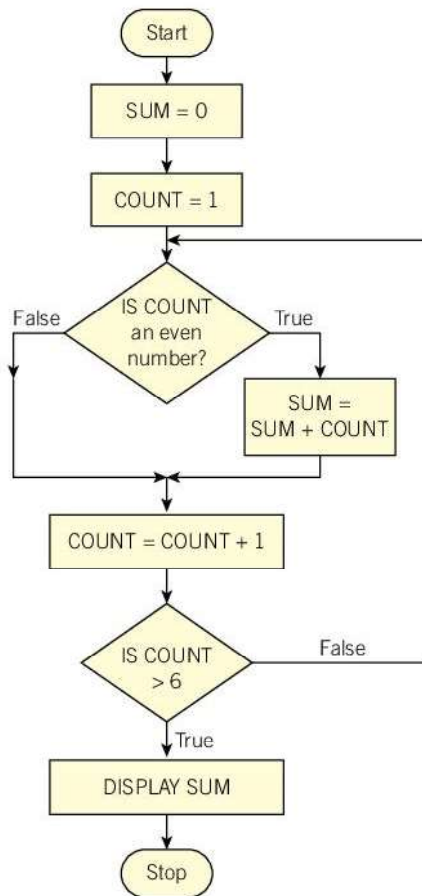


Fig 9.15 Flowchart adds all even numbers between 1 and 6

Pseudocode may appear quite simple by comparison to flowcharts, so why would you want to use flowcharts? The major reasons are that a flowchart:

- ♦ is easier to read
- ♦ more closely follows a standard, using symbols, unlike pseudocode
- ♦ lends itself more readily to conditional and control program structures.

As you continue to practice drawing flowcharts remember that:

- ♦ Every flowchart has a start and end point.
- ♦ The Process symbol can be anything from completing a calculation to describing an action. It is represented by a rectangle containing the text with the description.
- ♦ If there is a question or condition, it is most likely placed in a Decision symbol.
- ♦ The connection of 'what symbol follows next' is shown with arrows between symbols.

Questions

- 1 Name and draw the symbol that is suitable for each of the following statements:
 - a indicates the end of a flowchart
 - b performs a calculation
 - c directs the flow of data to another symbol
 - d asks a question
 - e prints a result.
- 2 Draw the flowchart symbol including suitable text for each of the following statements:
 - a display the message 'Call a taxi'
 - b read a value into a variable *age*
 - c add 15 to a variable *age* and store the result in a variable *old*
 - d double the value of a variable named *cost*
 - e ask if the taxi has arrived.
- 3 Use the pseudocode below to draw the corresponding flowchart segment:

Pseudocode_9.4.3:

```

WHILE word <> 'page'
    INPUT word
    IF word = 'page'
    THEN DISPLAY 'OK'
    ELSE DISPLAY 'Oh No'
ENDWHILE
  
```

In many of the examples in this chapter you may have seen fragments of pseudocode such as (age < 20) and (number < 3). These conditions are evaluated in the Arithmetic and logic unit (ALU) of the CPU. Let us look at them in more detail.

Arithmetic operators

These perform basic mathematical operations such as addition, subtraction, multiplication or division to produce a result which is a number.

Table 9.3 Arithmetic operators

Arithmetic operator	Meaning (example)
+	addition (3+3)
-	subtraction (3-1)
*	multiplication (3*3)
/	division (3/3)
MOD	produces the remainder in the result
DIV	integer division—produces only the whole number in result (5/2 = 2)

Relational operators

These compare two quantities with each other. You will see these operators in algorithms where there is a condition. The algorithm relies on whether the result of a condition is true or false to know how to proceed to the next set of statements. This may include exiting a loop or jumping to another set of statements. In Table 9.4 the variable *age* = 20 is used as the example to explain the different operators.

Table 9.4 Relational operators

Logic operator	Meaning	Example
=	equal to	age = 20
>	greater than	age > 19
>=	greater than or equal to	age >= 19
<	less than	16 < age
<=	less than or equal to	19 <= age
<>	not equal to	age <> 19

Logical operators

The conditional, WHILE, REPEAT and FOR statements compare values in order to determine a result. For example, WHILE number <= 3, IF mark = -1 and IF age < 13 must each produce a result that is either TRUE or FALSE. That is, when two values are compared using these operators, the result is either TRUE or FALSE.

Let's look at the example WHILE number <= 3 more closely.

If the variable *number* contains the value 2, then 2 is compared with 3 to determine if 2 is less than or equal to 3 (2<=3), which is TRUE. The outcome of *number* <= 3 is therefore TRUE. However, if *number* contains the value 4, then the result of the condition (4<=3) is FALSE.

The NOT operator

The result of a condition has one of two options. Examples are TRUE or FALSE, Yes or No, Accept or Not Accept, 1 or 0. That means a result is not TRUE, then it is FALSE. If is not 1, then it is 0.

The following tables use examples to illustrate how the NOT, AND and OR operators can be applied. These tables are also called **truth tables**.

Suppose a line of pseudocode contained the statement **IF day = Sunday**. Then based on the result (TRUE or FALSE) other statements will be performed.

Table 9.5 shows two examples for a typical condition IF day = Sunday and the use of the NOT operator. The condition can be written in these ways:

```
IF day is NOT Sunday
IF day is NOT equal to Sunday
IF day <> Sunday
NOT( IF day = Sunday)
```

If the result is TRUE, then a day other than Sunday would produce a result of FALSE.

Table 9.5 Truth table for NOT

If day = Sunday	NOT (If day = Sunday)	If day = Sunday	NOT (If day = Sunday)
TRUE	FALSE	1	0
FALSE	TRUE	0	1

The AND operator

Suppose a segment of pseudocode indicates that a student is promoted to the next class if an exam mark in the third term is 70 or greater.

```
IF (mark >= 70) AND (term = 3)
THEN Display 'Promoted'
```

The condition of both values must be true.

- ♦ If the value of one condition is TRUE and the value of the next condition is TRUE, then the result of both conditions is TRUE.
- ♦ If any one of the values is NOT TRUE (FALSE) then the result is also FALSE.

Table 9.6 Truth table for AND

mark >= 70	Term = 3	(mark >= 70) AND (term = 3)	(mark >= 70) AND (term = 3)
TRUE	TRUE	TRUE	Promoted
TRUE	FALSE	FALSE	Not promoted
FALSE	TRUE	FALSE	Not promoted
FALSE	FALSE	FALSE	Not promoted

The rightmost column shows the result for the example given.

In creating a truth table, you should list all the possible combinations for the conditions so that you can determine the outcome of each. Notice that Table 9.6 contains four rows of combinations of TRUE and FALSE. To calculate how many rows of options are needed, you can use the formula $2^{\text{number of conditions}}$. The number 2 represents the two options available, such as TRUE or FALSE, Yes or No, 1 or 0, Promoted or Not promoted.

Since there are two conditions (**mark >= 70**), (**term = 3**), the formula becomes $2^{\text{number of conditions}} = 2^2 = 4$ possible combinations of results.

Now that we know we need four rows:

- ♦ Use the first column to fill half of the rows (that is $4 / 2 = 2$) with TRUE and half with FALSE. So, rows 1 and 2 contain TRUE and rows 3 and 4 contain FALSE.
- ♦ Use the second column to alternate TRUE and FALSE.
- ♦ Use the third column and any additional columns to fill in the results.

Row	Column 1 Condition1	Column 2 Condition2	Column 3 Outcome/Result
1	TRUE	TRUE	
2	TRUE	FALSE	
3	FALSE	TRUE	
4	FALSE	FALSE	

The OR operator

OR works slightly differently. Suppose a segment of pseudocode indicates that a student is promoted to the next class if an exam mark is 70 or greater or the student ranked in the top 10 of the class.

```
IF (mark >= 70) OR (rank <= 10)
THEN Display 'Promoted'
```

- ♦ If the value of one condition is TRUE, then the result is TRUE.
- ♦ However, if the value of all the conditions are FALSE then the result is FALSE

Table 9.7 Truth table for OR

mark >= 70	rank <= 10	(mark >= 70) OR (rank <= 10)	(mark >= 70) OR (rank <= 10)
TRUE	TRUE	TRUE	Promoted
TRUE	FALSE	TRUE	Promoted
FALSE	TRUE	TRUE	Promoted
FALSE	FALSE	FALSE	Not promoted

Combinations of operators

As you practise the use of operators in algorithms as pseudocode or flowcharts, longer expressions can be expected.

For example, if an exam mark in the third term is 70 or greater and the student is also ranked in the top 10 of the class, then the student is promoted to the next class and given a plaque:

```
IF (mark >= 70) AND (term = 3) AND
(rank <= 10)
THEN Display 'Promoted and awarded a
plaque'
```

The values for all three conditions therefore must be TRUE in order to display 'Promoted and awarded a plaque'.

Expressions can also be replaced with a character, such as A or B, or a phrase for ease of completion. Then the basic expressions can be written as NOT A, A AND B, A OR B. This also makes it easier to write combinations of operators.

Example 30

```
IF (mark >= 70) OR (rank <= 10)
THEN Display 'Promoted'
```

Here we still have two conditions:

```
Let A represent mark >= 70
Let B represent rank <= 10)
```

Using the representations of A and B above, use TRUE and FALSE to find the result of A OR (NOT B).

First, create the truth table with A and B, then add another column and use the pattern from Table 9.5 to find NOT B:

A:	B:	NOT B
mark >= 70	rank <= 10	
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE

Next, using the values of A and NOT B, and the pattern for the truth table for the OR operator from Table 9.7, gives:

A: mark >= 70	B: rank <= 10	NOT B	A OR (NOT B)	Display
TRUE	TRUE	FALSE	TRUE	Promoted
TRUE	FALSE	TRUE	TRUE	Promoted
FALSE	TRUE	FALSE	FALSE	Not promoted
FALSE	FALSE	TRUE	TRUE	Promoted

The only outcome that is FALSE for A OR (NOT B) indicates that if a student's exam mark is *not* 70 or greater or the student is ranked in the top 10 of the class, then the student is *not* promoted.

Questions

- Identify the arithmetic or relational operator used in each of the following statements:
 - An area code must be three digits.
 - The cashier deducted \$2 for the sale item.
 - There were two more passengers in the car.
 - There must be 15 or more passengers on board.
- Write the following statements using relational operators. The first one is done for you.
 - There are three sisters. Answer: sisters = 3
 - Citizens must be at least 18 in order to obtain a picture ID.
 - There are 65 students with passes at Grade II.
 - Employees have at most four weeks of vacation.
 - Pension is from age 65 and older.
 - A cheque is not equal to cash.
- The variable LET contains the number 8 and CHR contains the number 24. Write the answer to the following statements, as either TRUE or FALSE:
 - (LET > 16)
 - (CHR < LET)
 - (LET > 16) OR (CHR < LET)
 - (LET > 16) AND (CHR < LET)
 - (CHR < 16) OR (LET < CHR)

- 4 Consider the following algorithm:

```
password = 'page'
count = 1
word = ' '
WHILE (word <> password) and
(count <=3)
INPUT word
count = count + 1
IF word = password
THEN Display 'Access'
ELSE Display 'Forgot password'
```

Identify:

- a an arithmetic operator
 - b a relational operator
 - c a logical operator.
- 5 You are to create a truth table for the following statement:
- ```
IF (word = password) AND (count <=3)
THEN Display 'Access'
ELSE Display 'Forgot password'
```
- a Identify:
    - i the conditions
    - ii the two outcomes (results) from the statement
    - iii the number of possible options/rows needed for the truth table.
  - b Draw the truth table.
- 6 How many possible combinations of options (rows) will be needed for a truth table that has three conditions?
- 7 Let A represent Saturday and B represent It's raining. The outcomes are if TRUE, do some gardening, and if FALSE, go to town. Create truth tables for the following Boolean expressions:
- a (NOT A) OR B
  - b NOT (A OR B)
  - c NOT (A AND B)
  - d A AND (NOT B).

Desk checking is the process of reviewing an algorithm for the correct processing and output. You do this by manually executing the steps in the algorithm one by one, while keeping track of the results. We started desk checking with Example 12 by going through each loop and explaining the output.

The logic of your algorithm is an important part of testing. Using a trace table will help you to test and verify your algorithms.

An algorithm is a sequence of steps which seek to solve a problem. If you 'freeze' the algorithm at any point then you have a 'snapshot' of the state of all the variables at that point.

The trace table is a very useful tool, which allows you to see the state of your algorithm with as much detail as you wish. Each row of the table shows the state of one step in the algorithm and each column shows the value of a variable at that step. The trace table allows you to check the algorithm for errors.

### Example 31

Complete the trace table for the following algorithm, given that the number 4 is the input value for QNTY.

```

READ QNTY
FOR SOLD = 1 to QNTY DO
 INV = QNTY - SOLD
 VAL = 5 * INV - SOLD
END
PRINT VAL

```

| QNTY | SOLD | INV | VAL |
|------|------|-----|-----|
| 4    |      |     |     |
| 4    |      |     |     |
| 4    |      |     |     |
| 4    |      |     |     |

Start by replacing each QNTY in the algorithm with the number 4:

```

READ 4
FOR SOLD = 1 to 4 DO
 INV = 4 - SOLD
 VAL = 5 * INV - SOLD
END
Print VAL

```

Now, the FOR loop tells you that SOLD must go from 1 to 4. You can therefore fill in the column labelled 'SOLD' with the numbers 1, 2, 3 and 4.

The next step is to perform the two calculations within the loop.

When SOLD is 1,  $INV = 4 - 1 = 3$ .

Write the number 3 in the table under INV, in the row where SOLD = 1.

The number 3 can now be used in the second calculation, giving  $VAL = 5 * 3 - 1$ .

Remember the rules in mathematics: BODMAS (Brackets – Order – Division – Multiplication – Addition – Subtraction). They tell you in which order to do the arithmetic operations. Computers follow the same rules.

Therefore  $VAL = (5 * 3) - 1 = 15 - 1 = 14$

Write 14 in the row where INV = 3.

| QNTY | SOLD | INV | VAL |
|------|------|-----|-----|
| 4    | 1    | 3   | 14  |
| 4    | 2    |     |     |
| 4    | 3    |     |     |
| 4    | 4    |     |     |

When SOLD is 2,  $INV = 4 - 2 = 2$ . Write the number 2 in the table in the row where SOLD = 2.

Now  $INV = 2$ , so  $VAL = 5 * 2 - 2 = (5 * 2) - 2 = 8$



Write 8 in the row where  $INV = 2$ .

| QNTY | SOLD | INV | VAL |
|------|------|-----|-----|
| 4    | 1    | 3   | 14  |
| 4    | 2    | 2   | 8   |
| 4    | 3    |     |     |
| 4    | 4    |     |     |

When SOLD is 3,  $INV = 4 - 3 = 1$ . Write the number 1 in the table in the row where SOLD = 3.

Now  $INV = 1$ , so  $VAL = 5 * 1 - 3 = (5 * 1) - 3 = 2$

Write 2 in the row where  $INV = 1$ .

| QNTY | SOLD | INV | VAL |
|------|------|-----|-----|
| 4    | 1    | 3   | 14  |
| 4    | 2    | 2   | 8   |
| 4    | 3    | 1   | 2   |
| 4    | 4    |     |     |

When SOLD is 4,  $INV = 4 - 4 = 0$ . Write the number 0 in the table in the row where SOLD = 4.

Now  $INV = 0$ , so  $VAL = 5 * 0 - 4 = (5 * 0) - 4 = -4$

Write -4 in the row where  $INV = 0$ .

| QNTY | SOLD | INV | VAL |
|------|------|-----|-----|
| 4    | 1    | 3   | 14  |
| 4    | 2    | 2   | 8   |
| 4    | 3    | 1   | 2   |
| 4    | 4    | 0   | -4  |

The maximum value in the loop has been reached, so you move on to the next line in the algorithm, which is to print VAL. The very last value of VAL in the algorithm is -4.

So when you freeze the algorithm, the last row of the table shows the values at that time. For this example, when QNTY = 4, you have SOLD = 4, INV = 0 and VAL = -4.

## Questions

- The marks 25, 30, 12, 10, -1 are to be entered using the following algorithm. Trace the values of total, count and average as the marks are entered.

**Pseudocode\_9.6.1: Calculate\_average**

```
START
total = 0
count = 0
average = 0
OUTPUT "enter a mark"
INPUT mark
WHILE (mark is not equal to -1)
 total = total + mark
 count = count + 1
 OUTPUT "enter a mark"
 INPUT mark
ENDWHILE
average = total/count
DISPLAY average
END
```

- Trace the variable mark and the output when the marks 65, 71 and 82 are entered:

**Pseudocode\_9.6.2: Exam\_results**

```
START
INPUT mark
IF mark >= 80
 THEN OUTPUT "Excellent results!"
 OUTPUT "see your teacher"
ELSE IF mark >= 70
 THEN OUTPUT "Satisfactory
results!"
ENDIF
OUTPUT "await end of term report"
ENDIF
END
```

### Multiple choice questions

- 1 The two main phases of problem solving comprise \_\_\_\_\_ and \_\_\_\_\_.
- algorithm, execution
  - algorithm, implementation
  - definition, execution
  - definition, implementation.

Questions 2 to 4 refer to the diagram below.

| Column 1      | Column 2               | Column 3    |
|---------------|------------------------|-------------|
| Accept letter | IF letter is A, B or C | Print grade |
|               | THEN grade = 'P'       |             |
|               | ELSE grade = 'F'       |             |

- 2 Columns 1, 2 and 3 represent:
- read, store and write
  - output, store and input
  - write, process and read
  - input, process and output.
- 3 Column 2 is an example of a \_\_\_\_\_ statement.
- logical
  - looping
  - sequential
  - conditional.
- 4 The terms 'letter' and 'grade' in column 1 are:
- constants
  - keywords
  - variables
  - subroutines.
- 5 The \_\_\_\_\_ loop executes a process statement before a decision.
- DO
  - REPEAT
  - WHILE
  - UNTIL.
- 6 An example of a relational operator is:
- /
  - >
  - NOT
  - AND.
- 7 All flowcharts have a:
- truth table
  - start symbol
  - WHILE loop
  - conditional statement.
- 8 A(n) \_\_\_\_\_ loop must have start and end values.
- FOR
  - IF
  - REPEAT
  - WHILE.
- 9 The process of reviewing an algorithm for the correct processing and output is known as:
- compiling
  - debugging
  - pseudocode
  - desk checking.
- 10 'Freezing' an algorithm at any point to view a 'snapshot' of the state of all the variables at that point is achieved by using a:
- program
  - trace table
  - flowchart
  - truth table.



## Short answer questions

- 11** You were asked to write a short program that will accept information about companies in six countries and the number of employees requiring training. There is one trainer for every 40 employees. However, another trainer is needed if there are at least 15 or more employees over the requirement. A sample of the data to be entered is shown below:

| CO_ID | Country  | Company     | NumtoTrain |
|-------|----------|-------------|------------|
| 6985  | Guyana   | Rumaba      | 56         |
| 6987  | Trinidad | Mariob      | 45         |
| 7295  | Jamaica  | Courtstreet | 87         |
| 7324  | Barbados | EveryInc    | 42         |
| 7361  | Antigua  | St. Micks   | 50         |
| 7455  | Belize   | Maggow      | 68         |

- a** On entering a country's code, name, company and the number of employees to be trained, the program will calculate and print the number of trainers assigned.
- i** Identify the variables required.
  - ii** State the data type for each of the variables listed in part i.
  - iii** Determine the output for each of the following statements if `numberofempl` is 56:  
`numberofempl DIV 30`  
`numberofempl MOD 30`
  - iv** Explain the purpose of the two statements in part iii.
  - v** Create an IPO chart for entering one country's data.
- b** The IPO chart should be drawn and sent to ten programming personnel for comments by the end of the day.
- i** Suggest an appropriate application that should be used to draw the IPO chart.
  - ii** Describe the best method to send the chart to the personnel.
  - iii** Explain one disadvantage of the method you describe in part ii.
- 12** The following pseudocode was written in preparation for writing the program:
- Initialise variables  
 Prompt to enter data in each variable  
 Calculate the number of trainers  
 Calculate the number of extra trainers required  
 Output total number of trainers needed
- a** Draw a flowchart to represent the five lines of pseudocode.
  - b** Write additional pseudocode to add further detail to line 1.
  - c** Write a conditional statement that would calculate the number of extra trainers required for line 4.
  - d** The program should be able to enter data for a maximum of six countries. Explain which loop would be most suitable.
  - e** Identify three examples of data that could be used to test the algorithm for correctness.

## 10.1 Programming languages

Chapter 9 introduced the concept of problem-solving by writing algorithms and drawing flowcharts. It is during the implementation phase that an algorithm must be converted into actual programming language statements called **source code**. In this chapter, we share some examples of programming languages to introduce the process of writing short programs.

### Implementation phase

In this phase, there are four main steps:

- 1 Translate your algorithm into a programming language such as BASIC, Pascal, C or C++, or Visual Basic for Applications (VBA). If you have a correct and precise algorithm, the translation should be almost line-by-line.
- 2 Locate and correct any errors in the code such as:
  - ♦ syntax errors, which are the errors resulting from incorrect use of the programming language syntax or violation of syntax rules
  - ♦ logic errors made by the programmer, such as those made by using wrong signs or arithmetic operators.
- 3 Execute the program code. This includes using test data to ensure that the program is working as expected, and can produce error messages as needed.
- 4 Maintain the program. This includes documenting the program throughout the coding process by writing comments on how to use the program, as well as comments within the program on how it works.

### Choosing the programming language

Programming is the art of writing the solution to a problem using a language that a computer can interpret. A programmer actually instructs the computer how to solve a problem since it cannot come up with a solution to a problem itself.

A computer program is a set of instructions that tells a computer what to do and how to do it. These instructions are usually converted into a sequence of numeric codes called machine code which is stored in memory. The central processing unit (CPU) interprets this code in order to carry out the instructions of the program.

First, you select the type of programming language that is suitable for your task. There are many programming languages that can be used to write programs or create other kinds of software. These languages are grouped into two categories:

- ♦ low-level languages
- ♦ high-level languages.

#### Low-level languages

These languages are machine-dependent. That is, the code written can only be understood by the particular computer or processor that was used to write the code.

Machine language uses the digits 0 and 1 to make up the binary code. An instruction might be written as 10110000 01100001.

- ♦ Advantage: code runs very fast and efficiently because it is directly executed by the CPU.



- ◆ Disadvantage: the programmer may become confused with the massive amount of 0s and 1s in the program. It is also machine-dependent.

Assembly language has the same structure and commands as machine language but allows programmers to use abbreviated words, called mnemonics, instead of binary. So instead of writing code as 10110000 01100001, the equivalent assembly language code may be 'add A, B', generally meaning 'add the contents of A and the contents of B'.

- ◆ Advantage: can be easily converted to machine code by a program called an assembler.
- ◆ Disadvantage: still difficult to understand compared to the high-level languages. Still machine-dependent.

### High-level languages

High-level languages are different from low-level languages in that they are not machine-dependent. Therefore, programs written on one computer can generally be used on another similar computer. They also use keywords similar to English and are easier to write.

These languages are designed to be easier for you to understand. They are converted to machine code, rather like translating from one language to another, so that

the computer can carry out the instructions in the CPU.

- ◆ Advantage: can use English-type words to write program code, making it easier to create.
- ◆ Disadvantage: programs have to be converted to machine language.

**Table 10.1** Examples of high-level programming languages

#### Pascal

Named after the 17th century mathematician Blaise Pascal. A language that uses structured programming, mostly used for teaching purposes. Pascal is an easy to learn language that is an alternative to BASIC.

#### VBA

Visual Basic for Applications, is a programming language developed by Microsoft and derived from BASIC. Programming in VBA uses a graphical user interface using drag-and-drop techniques on a form (window). Controls, such as text boxes and buttons, are used to design the layout on the form to work with Microsoft applications such as Word, Excel and Access.

The other sections in this chapter focus on writing the program, running or executing the code, debugging techniques, testing the program with data and documentation.

### Question

- 1 Explain the difference between low-level languages and high-level languages.

Although different programming languages have slightly different code and data types, for the most part writing the code becomes easy with practice. The 2020 syllabus has not specified a recommended programming language, but it is best to choose a language that is not too complicated to learn. Examples of programs are shown in four languages for illustrative purposes.

Once the programming language has been selected, you can start writing the program. Once you have designed your algorithm and tested it on paper, you need to use a text editor to type the program. Most programs provide an editor that can be used to type your programming code. Most programming languages have different syntax and semantics which you must follow.

**Syntax** is the very precise way in which the statements in a program must be written in order to be understood. It is a set of rules for combining the various elements that make up a programming language. The syntax allows the programmer to create a correctly structured statement that is 'legal' – although it does not guarantee that the statement will be useful!

**Semantics** is the meaning associated with the words, symbols and punctuation that make up the language. For example, in Pascal, a semicolon (;) represents the end of a statement, while in other languages it may have another meaning. Statements are constructed from the various program elements; the overall meaning of the statement is determined by the language semantics.

### Program structure

Nearly all structured programs share a similar overall structure:

- ♦ statements to establish the start of the program: could include program headers or comments to state the name of the author and a date that the program was written
- ♦ declarations of variables and types
- ♦ Program statements (blocks of code), which include:
  - ♦ constructing arithmetic, relational and Boolean expressions (AND, OR, NOT) using appropriate operators
  - ♦ implementing the remaining pseudocode constructs such as conditional (IF), iteration constructs (looping) and terminating conditions
  - ♦ formatting output in a user-friendly manner.

### Sample program

The following is a simple example of a program written in three different programming languages: BASIC, Pascal and VBA in Microsoft Excel. These programs each show the program header, if required, and code to output the line 'Hello to everyone' on the computer screen. There is also a comment indicating the purpose of the program. You should include comments within the program when you write your code. There are no variable declarations in this first example, and you do not need to understand every line of code; just become familiar with the pattern of how the code is written. Some code may also vary since each compiler has slight variations in their syntax requirements.

#### Example 1

**Programming language: BASIC**

|                         |                                                  |
|-------------------------|--------------------------------------------------|
| REM Output a sentence   | ← Optional comment on the purpose of the program |
| PRINT Hello to everyone | ← Output statement                               |



**Programming language: Pascal**

|                               |                                                                   |
|-------------------------------|-------------------------------------------------------------------|
| Program sentence;             | ← Keyword Program followed by name of program then a semicolon(;) |
| {To output a sentence}        | ← Comment on the purpose of the program                           |
| Begin                         | ← Keyword to indicate statement(s) will follow below              |
| Writeln('Hello to everyone'); | ← Output statement                                                |
| End.                          | ← Keyword for end of program code                                 |

**Visual Basic for Applications (VBA)**

|                                       |                                               |
|---------------------------------------|-----------------------------------------------|
| ' To output a sentence                | ← Comment on purpose of the program           |
| Private Sub<br>CommandButton1_Click() | ← Keywords for start of code                  |
| MsgBox "Hello to everyone"            | ← Keyword to send the statement to the screen |
| End Sub                               | ← Keyword for end of program code             |

In order to use a variable within a program, the compiler needs to know in advance the type of data that will be stored in it. For this reason, you must declare or state the variables you are using at the very start of the program. Variable declaration means giving a new name and a data type for the variable, for example, age: integer.

You should use meaningful variable names in your programs, so that if you or someone else needs to review the programming code later, the variables will be easy to remember and understand. In the following example programs, three variables named age, cost and class are declared. Each variable is then used in the programs as part of a statement to be displayed on the screen. Make sure you note the variable names to understand their purpose.

In each example the statements are to display the following output:

```
Minimum age to obtain licence:
18
Cost of drivers permit: 250.95
Class of vehicle licence: B
```

**Example 2****Programming language: BASIC**

```
REM example2
Dim age As Integer
Dim cost As Double
Dim grade As String
age = 18
cost = 250.95
class = 'B'
PRINT "Minimum age to obtain licence: "
PRINT age
PRINT "Cost of drivers permit: ", cost
PRINT "Class of vehicle licence: ", grade
```

**Programming language: Pascal**

```
Program example2;
var
 age: integer;
 cost: real;
 class: char;

Begin
 age := 18;
 cost := 250.95;
 grade := 'B';
 Write('Minimum age to obtain
 licence: ');
 Writeln('age');
 Writeln('Cost of drivers permit: ',
 cost);
 Writeln('Class of vehicle licence:',
 grade);

End.
```

**Programming language: VBA**

The cell locations A2, B2 and C2 in the spreadsheet in Figure 10.1 are referred to in the VBA programming code.

|   | A   | B      | C     |
|---|-----|--------|-------|
| 1 | Age | Cost   | Class |
| 2 | 18  | 250.95 | B     |
| 3 |     |        |       |

Fig 10.1 VBA Example 1

```
Private Sub CommandButton1_Click()
Dim age As Integer
Dim cost As Double
Dim class As String
 age = Range("A2").Value
 cost = Range("B2").Value
 grade = Range("C2").Value
MsgBox " Minimum age to obtain licence: "
 & vbCrLf &
 MsgBox Range("A2").Value
 MsgBox "Cost of drivers permit: " &
 Range("B2").Value
 MsgBox "Class of vehicle licence: "
 & Range("C2").Value
End Sub
```

These examples illustrate that different programming languages have slightly different syntax and data types. However, for the most part, variable declaration is straightforward. As the program requirements become more complex, so do the variable declarations.

## Program statements

Program statements are the instructions which will carry out the requirements of the program. The following examples illustrate the use of conditional statements such as IF-THEN-ELSE. A variable *form* is defined and a value is assigned to it. A **conditional statement** then determines which message should be displayed.

## Conditional statements

### Example 3

**Programming language: BASIC**

```
form = 3
IF form = 3 THEN PRINT "Promoted to Form 3"
ELSE PRINT "Not promoted to Form 3"
```

**Programming language: Pascal**

```
Program Conditional;
Var form: integer;
Begin
 form := 3;
 If (form = 3)
 then writeln('Promoted to Form 3')
 else writeln('Not promoted to Form
3');
End.
```

**Programming language: VBA**

```
Private Sub CommandButton1_Click()
Dim form As Integer, result As String
form = Range("C3").Value
 If form = 3 Then
 result = "Promoted to Form 3"
 Else
 result = "Not promoted to Form 3"
 End If
Range("C4").Value = result
MsgBox Range("C4").Value
End Sub
```

## Looping

The following examples illustrate the use of looping constructs such as WHILE and FOR.

In each of the following examples, the code will output 'TT for CXC' 10 times.



*WHILE loop***Example 4****Programming language: BASIC**

```

line = 0
WHILE line < 10 DO
line = line + 1
PRINT "IT for CXC"
ENDWHILE

```

**Programming language: Pascal**

```

Program whileloop;
Var line: integer;
Begin
 line := 0;
 While (line < 10) Do
 Begin
 line := line + 1;
 Writeln('IT for CXC');
 end;
 End.

```

**Programming language: VBA**

```

Private Sub CommandButton1_Click()
Dim row As Integer, line As String
row = 1
line = "IT for CXC"
Do While row <= 10
 MsgBox "IT for CXC"
 Cells(row, 2).Value = line
 row = row + 1
Loop
End Sub

```

*FOR loop***Example 5****Programming language: BASIC**

```

FOR I = 1 to 10 DO
PRINT "IT for CXC"
ENDFOR

```

**Programming language: Pascal**

```

Program forloop;
Var line: integer;
Begin
 for line := 1 to 10 do
 writeln('IT for CXC');
 End.

```

**Programming language: VBA**

```

Private Sub CommandButton1_Click
Dim i As Integer
For i = 1 To 10
 MsgBox "IT for CXC"
Next i
End Sub

```

**Questions**

- 1** Explain the difference between syntax and semantics in a program.
- 2** What does it mean to declare a variable?
- 3** Why should you use meaningful variable names in your programs?

Usually, you will go through several steps before you can even type your program instructions, let alone view output from your program. At this stage, you are ready to see your program run. After you have typed your program using an editor it is called source code. Next, it needs to be tested for correctness. To do so, the source code must be translated into machine language called object code so that the CPU can carry out the program instructions.

### From source code to object code

Converting a program from source code to object code is performed by a 'translator' program. Sometimes a program listing is shown, which is a printout or soft copy of the source program instructions as a reference while working with and coding the program.

These are programs that translate a specific program from one language to another – from a high-level language to a low-level language. Interpreters, compilers and assemblers are all translators.

An interpreter translates the source program line-by-line, and if an error is detected then translation is aborted (stopped) (Fig 10.2). If no errors are detected the interpreter instructs the control unit to execute the translated instruction. This cycle will be repeated for every instruction in the program. This is an easy but inefficient way of executing (running) programs not written in machine code. BASIC is a language which is interpreted.

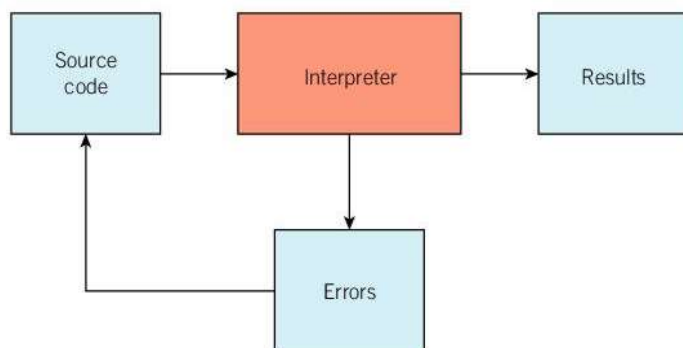


Fig 10.2 Illustration of the process of an interpreter

A compiler translates all program instructions at one time and produces a 'stand-alone' object program that can be executed (run) on its own (Fig 10.3). Error checks for syntax and logic errors are performed, and an error summary is issued (A). Once the program is compiled and no errors are detected, the compiler will then instruct the control unit to start executing the program. If the program produces incorrect output, or stops unexpectedly, then another type of error may be the cause (B). Otherwise, the object program will be executed each time it is run, but the source code only needs to be compiled once. If there are any modifications to the source code, then it must be recompiled. COBOL is a language which is compiled.

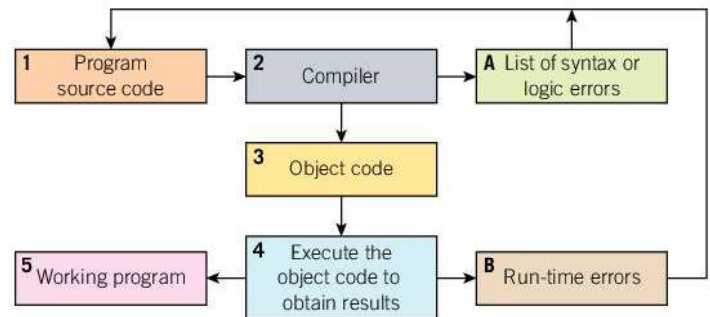


Fig 10.3 Illustration of the process of a compiler

### Programming errors

Often, when the computer is compiling or interpreting your source code, an error, whether minor or major, will cause it to either output wrong results or not reach the stage of output at all. There are different types of errors which will cause your program to crash.

#### Syntax errors

Syntax errors occur when a mistake is made in the language rules or sentence structure of the programming language. Examples of syntax errors include misspelling of a variable or keyword. Syntax errors stop the program instructions or source code being converted to machine code. Tools used with the program usually detect syntax errors quite easily.



### Examples of syntax errors

|                                |                   |
|--------------------------------|-------------------|
| <code>if (age 42) then</code>  | Incorrect syntax  |
| <code>print "OK"</code>        | since an operator |
|                                | such as < or > is |
|                                | missing           |
| <code>Bgin</code>              | Spelling error in |
| <code>For X = 1 to 5 Do</code> | keyword BEGIN     |
| <code>End</code>               |                   |

### Logic errors

Logic errors occur when a programmer makes mistakes in the sequence of the program sentences, such as using the wrong mathematical formula, wrong operator in an expression or incorrect use of looping structures. The program will usually compile; that is, the compiler will be able to convert it from source code to object code. Compilation with **logic errors** does not generate any syntax errors or warning messages, but when the program is run it produces the wrong results.

### Example of a logic error

This segment of code will output 'A page' when  $x < 10$ :

```
IF (x < 10) THEN PRINT ("A page")
```

However, the programmer intended the statement to be:

```
IF (x > 10) THEN PRINT ("A page")
```

However, the error of putting the wrong symbol in the code means the output will be printed when  $x$  is less than 10 and not when it becomes greater than 10.

## Debugging

Debugging is the process of finding the errors in the source code (detection), understanding why they occurred (diagnosis) and correcting them. Errors are often found through error messages generated by the program or the operating system, or because the program does not behave as expected.

### Executing the program

If you have translated your program to object code without errors, you can now execute it and see the results. The final two terms commonly used with program execution are:

- ♦ **program loading:** copying a program from hard disk to the main memory in order to put the program in a ready-to-run state
- ♦ **linking:** combining various pieces of code and data together to form a single executable object code that can be loaded in memory. Linking can be done at compile time, at load time and also at run time.

Once your program compiles, you may either see your results on the screen, or you may see another type of error.

### Runtime errors

Runtime errors occur as the program compiles or 'runs'. These errors are usually due to unexpected events such as division by zero or lack of memory for the computer to manipulate the data. Runtime errors can be very difficult to trace as the program may produce results most of the time.

### Example of a runtime error

```
FOR average = 1 to 5 DO
 Results = results/(average - 1)
```

The program will produce an error message or cause the computer to 'freeze' when the value of average becomes 1.

The statement will be:  $\text{results} = \text{results}/(1 - 1)$

... which is division by zero!

## Questions

- 1 What is the name given to the printout or soft copy of the source program?
- 2 Explain the purpose of a translator and give three examples.
- 3 Explain why debugging is necessary.

Testing and debugging are necessary stages in the development cycle, and they are best incorporated early. Testing begins when you start to plan the program and continues until the program is completed and put into daily use. It is used to ensure that a computer application (program) is complete and does what it was meant to do.

Testing attempts to find problems in your code; debugging isolates and fixes the problems. Typically, when you test and debug your program, you need to ensure that it:

- ♦ runs without crashing or generating error messages that the end user may be unable to resolve
- ♦ carries out a reasonable action or produces error messages for a range of scenarios
- ♦ helps the user to continue or restart the program if unexpected user or system errors occur.

Errors can occur at any stage of a software application. They may happen frequently if you make errors while using the application, for example, but they also occur as a result of problems such as a power failure.

Testing cannot prove that your program is fully correct. It can certainly find defects (bugs) in the program, but there is no way to be sure that there are absolutely no errors in the logic of the program code. Therefore, the earlier a defect is found, the easier it is to fix it. Testing the program involves:

- 1 creating a set of test cases
- 2 running the program with each test case
- 3 checking that the performance of the software is as expected.

### Test cases

A test case documents the values that are input and compares the predicted results with the actual results when a program is executed. When choosing test cases

you must consider a range of values. It is important to ensure that every conditional branch, loop and statement of the program is actually tested by using a variety of test cases.

Black box testing (Fig 10.5) is used to check that the output of a module is as expected given certain inputs. The term 'black box' is used because the actual code being executed is not examined. The black box test provides no information about whether all statements or loops of the test module are actually necessary or not. Black box testing attempts to:

- 1 input values to the program
- 2 process the data
- 3 display the values returned from the program.

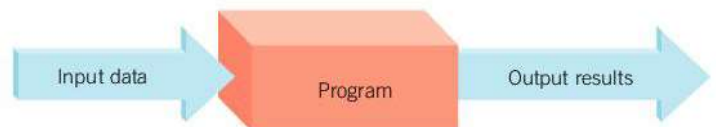


Fig 10.5 Black box testing checks the output from the values input

White box test cases (Fig 10.6) are designed to examine the inner structure of the program and are one of the most important test methods. The test checks the accuracy of the module from input, through every possible path through the test object, to the output.

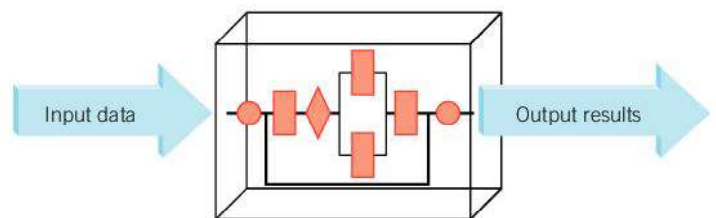


Fig 10.6 White box testing examines the inner structure of the module

Since testing every value is impractical, you should choose a range of values to check specific error conditions (Table 10.2). You should also be aware of which test data, if any, would be expected to produce errors.



**Table 10.2** Testing strings and numbers – examples of values to test

| Strings                                               |
|-------------------------------------------------------|
| Empty string                                          |
| String consisting solely of white space               |
| String with leading or trailing white space           |
| Special characters such as -, #, ", ' & and <         |
| 'Foreign' characters typed on international keyboards |
| Numbers                                               |
| No number (that is, leave input blank), if possible   |
| Zero 0                                                |
| Small and large positive numbers                      |
| Small and large negative numbers                      |
| Positive numbers out of required range                |
| Negative numbers out of required range                |
| Numbers with leading zeros such as 0034               |
| Combinations of letters and numbers                   |

Consider the program that gives a user access based on the correct username and password entered in the log-in screen (Fig 10.7).

Fig 10.7 Testing a username and password access screen

Examples of usernames and passwords as test cases can include combinations of values as shown below:

| Test# | Username:   | Password:  |
|-------|-------------|------------|
| 1     | Admin       | newday2019 |
| 2     | 01boss      | Admin      |
| 3     | 00112233    | newday2019 |
| 4     | New day2019 | Empty      |

Testing has to be planned. Some sections of the program often have to be tested before their completion, and this often includes testing of other subroutines with which they interact. The results of these tests determine the quality of the application. Although quality varies from system to system, some common attributes include:

- ◆ **Reliability:** This is the probability that the inputs to and use of the software will not cause it to crash for a specified time under certain conditions.
- ◆ **Stability:** A software system is stable or robust if an error in its operation does not result in irreversible damage to the application.
- ◆ **Maintainability:** This means how easy it is to debug, modify and expand the program at the appropriate locations without undesirable side effects.
- ◆ **Usability:** The software system should allow flexible input by the user. The results output should also be in a clear and well-structured form and be easy to interpret, and error messages must be provided in a form that the user can understand.
- ◆ **Portability:** This is the ease with which a software system designed for one type of computer can be adapted to run on different computers and operating systems.

### Questions

- 1 Explain the difference between:
  - a testing and debugging
  - b black box and white box testing.
- 2 Give two examples of what should be done when testing a program.
- 3 List three attributes of a good program that has completed rigorous testing.
- 4 You have written a program to input two integers which will add these numbers and output the answer. Write examples of values that could be used to test the program.

Documentation is an important part of the software design process. Several types are produced during the process.

## Documenting programming code

Documenting your code means typing short phrases or comments describing what your code is doing. Comments placed alongside your programming code are useful if the program needs to be debugged or updated. If you write your code for your SBA and then review it a month later, you may spend some time trying to remember what some sections were supposed to achieve. Comments therefore help you and others to understand the purpose of your code, especially since your teacher or a moderator will be reviewing it.

Many students write their comments after the program is finished and working. If you leave the comments for later, you may not remember why you used a certain method, used different variables or used another set of code that doesn't resemble the requirements. Then you may resort to typing in comments that are similar to the code. This is not a good method.

Comments in programming code should include the following:

- An overview of the process and tasks, rather than rephrasing each step of your code.
- The name(s) of the authors of the program and even contact information for some programs.
- The date that the program was created or modified or reviewed. This provides a good guideline of how recently the program was updated.
- If you have finally found a solution to a part of the code that did not work, you should document what you did and include any comments that would help to explain that section of the code.

Adding comments to your programming code may seem like a tedious process. However, if you add them while writing your code, you may find that explaining

the sections will help you understand more about programming.

Example 6 shows the listed programming languages with the keywords or symbols used for adding comments.

### Example 6

#### Programming language: BASIC

```
REM Written by Ali James
REM Apr 28, 2020
REM *****
REM This program will output a
single sentence
REM *****
```

#### Programming language: Pascal

```
{Written by Ali James }
{Apr 28, 2020 }
{***** }
{This program will output a
single sentence }
{***** }
```

#### Visual Basic for Applications (VBA)

```
' Written by Ali James
' Apr 28, 2020
' *****
' This program will output a single
sentence
' *****
```

## User documentation

User documentation is concerned with what the program does and how the end user makes the program do what it is supposed to do. It is usually the first



contact they have with the system. It may give details of how to input data, how to format output (use the printer, save files and so on), how to access features of the program and how to interpret any system messages.

User documentation should be structured in such a way that it is not necessary to read it all before starting to use the application. It is usually integrated as part of online help, with lots of blogs and online chatrooms to share information on how to troubleshoot and use the system more efficiently. Five common components of user documentation are:

- ♦ a system overview which explains what the system can and cannot do
- ♦ an installation document, which explains how to install the system and tailor it for particular hardware configurations. The document suggests how to recover from errors and basic problems when things go wrong. It should include illustrations and examples
- ♦ an introductory manual which explains, in simple terms, how to get started with the system
- ♦ a reference manual, which describes in detail all the system facilities available to the user and how these

facilities can be used. This manual assumes that the reader is familiar with the system and understands its concepts and terminology

- ♦ an optional system administrator's guide which explains how to react to situations which arise while the system is in use. It also carries out system housekeeping tasks such as making a system backup.

### Questions

- 1** Give two reasons why documentation is important when designing an application.
- 2** Explain why should you include comments as you write your programming code.
- 3** State the keyword or symbol used to identify a comment in each of the following programs:
  - a** Pascal
  - b** VBA.
- 4** Describe the difference between program documentation and user documentation.
- 5** Explain two components of the following:
  - a** program documentation
  - b** user documentation.

### Multiple choice questions

- 1 The part of the computer that interprets programming code in order to carry out the instructions of the program is the:
  - a CU
  - b CPU
  - c ALU
  - d ROM.
- 2 Errors resulting from incorrect use of programming language rules are called:
  - a data
  - b logic
  - c syntax
  - d runtime.
- 3 Using \_\_\_\_\_ data ensures that the program is working as expected.
  - a correct
  - b output
  - c sample
  - d source.
- 4 An example of a programming language that uses a graphical user interface is:
  - a VBA
  - b BASIC
  - c COBOL
  - d PROLOG.
- 5 A variable that stores a collection of characters such as a word, phrase or sentence is a(n):
  - a string
  - b integer
  - c number
  - d character.
- 6 Which of the following are true for FOR statements?
  - a start value must be known
  - b end value must be known
  - c both start and end values are optional
  - d both start and end values must be known.
- 7 Logic errors in a program can produce:
  - a test data
  - b syntax errors
  - c wrong results
  - d error messages.
- 8 Debugging is the process of finding errors in the:
  - a test data
  - b messages
  - c source code
  - d object code.
- 9 A test case documents the values that are \_\_\_\_\_ to compare them with the actual results when a program is \_\_\_\_\_.
  - a input, compiled
  - b input, executed
  - c output, compiled
  - d output, executed.
- 10 Adding comments to a program is best achieved:
  - a when a user has tested it
  - b while writing the program
  - c after the program is written
  - d when it is maintained in the future.
- 11 The keyword REM in BASIC refers to a(n):
  - a error message
  - b comment
  - c data type
  - d loop.
- 12 Testing a program involves each of the following, *except*:
  - a creating a set of test cases
  - b fixing errors in the program
  - c running the program with each test case
  - d checking the performance of the software.



## Short answer questions

13 Consider the following samples of pseudocode:

```
Pseudocode_A: average of exam marks
using While loop
total = 0
count = 0
average = 0
OUTPUT 'Enter a mark'
INPUT mark
WHILE (mark is not equal to -1)
 total = total + mark
 count = count + 1
 OUTPUT 'Enter a mark'
 INPUT mark
ENDWHILE
IF count = 0
THEN DISPLAY 'No marks entered'
ELSE
 average = total/count
 DISPLAY average
END
```

```
Pseudocode_B: average of exam marks
using Repeat loop
total = 0
count = 0
average = 0
REPEAT
 DISPLAY 'Enter mark'
 INPUT mark
 IF mark = -1
 THEN DISPLAY 'End of marks'
 ELSE
 total = total + mark
 count = count + 1
 UNTIL mark = -1
 IF count = 0
 THEN DISPLAY 'No marks entered'
 ELSE
 average = total/count
 OUTPUT average
 END
```

- a For one or both samples, write suitable code with documentation using a programming language assigned by your teacher.
- b Enter the following sample data to test the program. For each sample, explain whether you were able to enter all of the data and state the final results or errors, if any.
  - i 23, 15, 18, A+, -10, -1, 14
  - ii 23, 15, 18, -10, 14, -1, A+
  - iii -1, 23, 15, 18, -10, 14, A+
  - iv 23, 15, 18, 14, -1, A+

12 Consider the Training algorithm opposite that will accept information about the number of employees requiring training.

A country's code, name, company and number of employees to be trained are entered when prompted.

There is one trainer for every 40 employees. However, another trainer is needed if there are at least 15 or more employees over the requirement.

The program will calculate and print the number of trainers assigned.

### Algorithm: Training

Declare variables code, numberofempl, trainers, extras as number

Declare country, company as literals/string

Prompt to enter a code

Accept code

Prompt to enter country

Accept country

Prompt to enter company

Accept company

Prompt to enter numberofempl

Accept numberofempl

Calculate number of trainers = numberofempl / 30

Calculate number of additional trainers (extras) from (numberofempl / 30)

If extras >= 15

Then trainers = trainers + 1

Output the number of trainers needed

End of algorithm

- a Convert the following algorithm into code for a programming language approved by your teacher.
- b Insert lines to appropriately document the program.
- c Update the program to accept the information for six countries.

### 11.1 Introduction to Pascal

The programming language Pascal is used to teach introductory computer programming in many schools. This is because Pascal promotes a systematic, well-organised and logical approach to learning programming.

This chapter shows the practical aspects of implementing the programs introduced in Chapter 10. If this is your first programming language, you will need a few basics to get started. Think of a programming language as a recipe for baking a cake or instructions on building a bench. Pascal is one programming language, just as English is one spoken and written language.

Your mobile phone has a keypad that you use to type the number or text message. Then you click send and your call is made, or message sent, without the knowledge of how it is done. However, if you type a wrong digit when you are dialling a number, you will not be able to contact the correct person.

Similarly, a computer uses an application called a compiler to produce a result by translating the instructions in a program you write into a form the computer can interpret. If you type the wrong instructions, the compiler cannot produce any results. Instead it produces syntax errors or runtime errors.

### Finding a Pascal compiler

There are a number of Pascal compilers available. In this chapter, Ezy Pascal is used. You should search online for 'Ezy Pascal Free Download' from Dolphin Bay Software.

### Writing your first program

When you start Ezy Pascal, the Pascal editor will appear (Fig 11.1). The upper green area is to show the result of your programs, while the lower blue area is used for typing the program code.

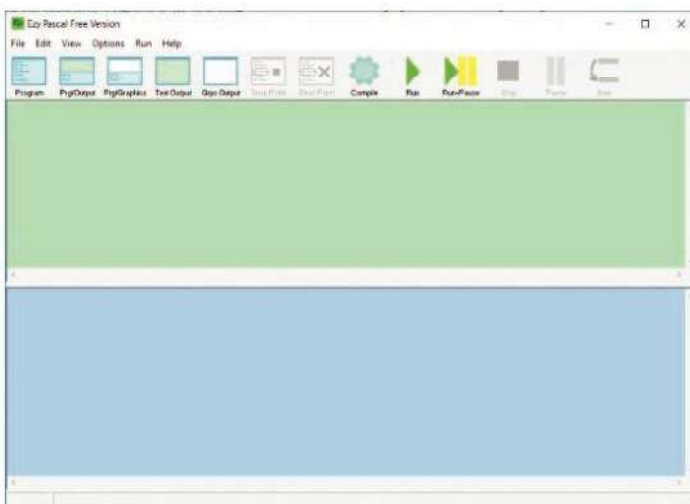


Fig 11.1 Ezy Pascal opening screen

In your Pascal editor, type the short Pascal program shown below. In the first line, be sure to type your name in place of 'type your name here' and today's date in place of type 'today's date here'.

```
{Name: type your name here}
{Title: My first simple program written
in Pascal}
{Purpose: To output one line}
{Date: type today's date here}
Program Sentence;
Begin
 Writeln('Hello to everyone');
End.
```



The word `Program` tells the Pascal compiler that this is the first line of code that contains the name of the program. In this example, the name of the program is `Sentence`.

The main section of a program starts at the word `Begin` and finishes at `End`. Within this section, the specific instructions that need to be executed by the compiler are included to solve our programming problem.

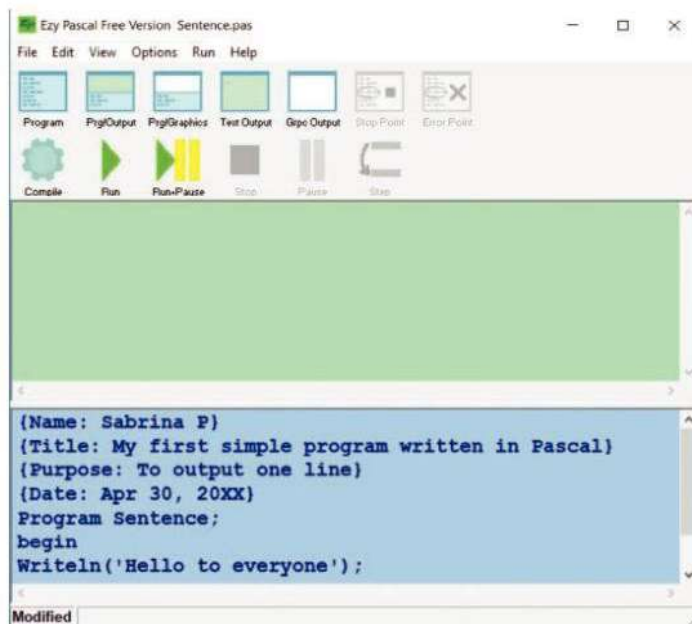


Fig 11.2 Typing Pascal code into the editor

Notice the first four lines of code. These are comments and are ignored by the Pascal compiler. You should always start your programs with the following comments:

- ♦ your name
- ♦ the title of the program
- ♦ a brief description of the purpose of the program
- ♦ the date the program was created.

## Saving the program

You should save every program you type. To save the program, click File, Save. Browse to where you want to save your program (such as the Documents folder or

a memory stick). Then type the name `'Sentence'` as the name of the program. If the name of the program will be two or more words, then remove the spaces between the words. So, for example, `'One Sentence'` will become `'OneSentence'`. Pascal will add the default extension `.pas` to the program. You will see the name of the program above the EzyPascal menu (Fig 11.3).

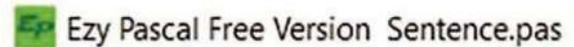


Fig 11.3 Your program is given a name and shown at the top of the Ezy Pascal screen

## Compiling the program

The Pascal compiler needs to convert the Pascal program from the statements you typed in the editor into machine language so that the central processing unit can execute the program correctly. The compiler first checks your program to ensure that no typing or syntax errors were made. Select Run, Compile or click the Compile icon on the menu. If you typed everything correctly, the compiler will output a message `'Compile successful'` at the bottom of the screen (Fig 11.4). Otherwise an error message will be displayed (also at the bottom of the screen) and a possible location of the error will be highlighted on a line of code.



Fig 11.4 Compiling the program shows a message at the bottom of the screen that there were no syntax errors

In Figure 11.5, the compiler has highlighted the line immediately below the error. The orange arrow shows where the error is located: the semicolon was omitted at the end of the line.

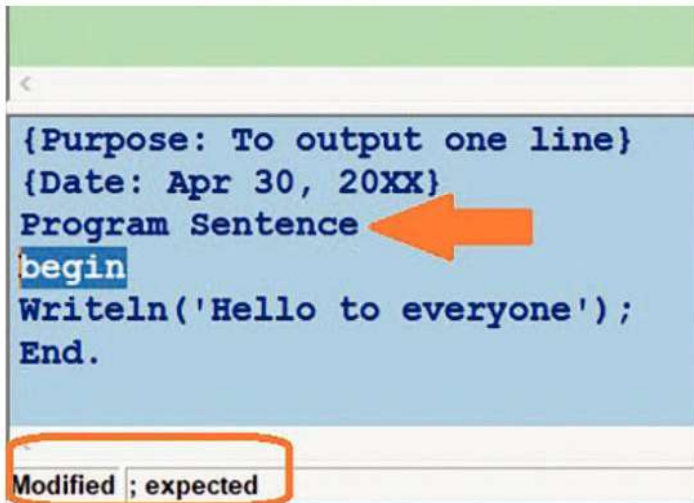


Fig 11.5 The compiler has highlighted the line immediately below the error

When compiling and removing errors, if you make multiple corrections, and more errors appear, then you may not know which correction caused the additional errors. Therefore, it is best to make corrections in this order:

- 1 Identify and correct each error individually.
- 2 Save the program.
- 3 Compile the program again.

You should continue to compile the program until there are no errors and the 'Compilation successful' message is shown.

## Running the program

Once you have no compilation errors, you can execute (run) the program to see if it produces the correct output. From the Run menu, select the Run option or click the Run icon. The program will execute in a program as shown in Figure 11.6.

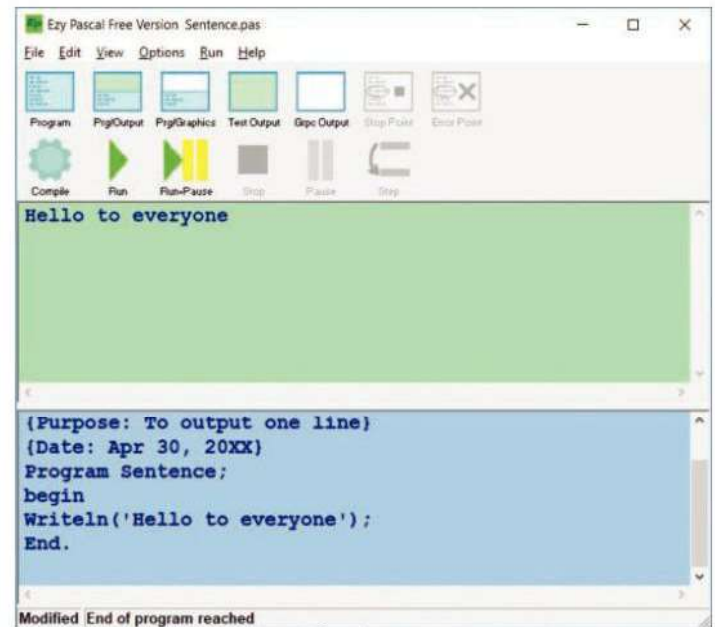


Fig 11.6 The output of the program named Sentence

## Questions

- 1 State which of the following are valid names for a Pascal program:
  - a Add numbers
  - b 10Lines
  - c For Loop
  - d ThreeChoices
  - e maxValues
- 2 List three steps to follow when compiling a program.



Congratulations! Now that you have completed your first Pascal program, you should compile and execute it

to see your results. Table 11.1 shows the basic structure of a Pascal program.

**Table 11.1** Structure of Pascal program

|                                 | Example(s)                                                                                                               | Explanation                                                                                                                                                                                                                                                      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Program header</b>           |                                                                                                                          |                                                                                                                                                                                                                                                                  |
| Program name;                   | Program Sample;<br><i>For Ezy Pascal</i><br>Or<br>Program Sample (input, output);<br><i>For some other compilers</i>     | All Pascal programs must have a <b>Program</b> heading. The first word, Program, is compulsory.                                                                                                                                                                  |
| <b>Declarations</b>             |                                                                                                                          |                                                                                                                                                                                                                                                                  |
| Const                           | Const pi = 3.14;                                                                                                         | Const is short for constant. You <b>assign</b> a <b>value</b> to a constant when you create it.                                                                                                                                                                  |
| Var                             | Var Mark, Count: integer;<br>Test: real;<br>Grade: char;<br>Title = 'Hello';                                             | Var is short for variable. Before you can use a <b>variable</b> , you must first <b>declare</b> it. These examples declare Mark and Count as integer, Test as real and Grade as character. Variable types include integer, real, character and literal (string). |
| <b>Procedures and functions</b> |                                                                                                                          |                                                                                                                                                                                                                                                                  |
| Begin                           | Begin                                                                                                                    | Denotes the start of one or more programming statements                                                                                                                                                                                                          |
| <b>Statements</b>               |                                                                                                                          |                                                                                                                                                                                                                                                                  |
|                                 | Read(Mark);<br>Writeln('Thank You');<br>If (Mark > 30) Then Grade := 'P'<br>Else Grade := 'F';                           | These include:<br>input,<br>output,<br>conditional (IF-THEN-ELSE)<br>assignment.                                                                                                                                                                                 |
|                                 | For count := 1 to 5 Do<br>Begin<br>Test := Mark * count;<br>Writeln('The test output is ; test');<br>End;                | For loop<br><b>Begin</b> (of a compound statement)<br>compound statements have two or more consecutive statements<br><b>End;</b> (of a compound statement)                                                                                                       |
|                                 | While Mark > 30 Do<br>Begin<br>Writeln('This is a sample while loop');<br>Writeln('More statements can go here')<br>End; | While loop<br><br>Can include compound statements enclosed in Begin and End                                                                                                                                                                                      |
|                                 | Repeat<br>Writeln('this is a sample Repeat loop');<br>Writeln('More statements can go here')<br>Until Mark > 30;         | Repeat loop<br>Can include compound statements that may not need Begin and End                                                                                                                                                                                   |
| End.                            | End.                                                                                                                     | <b>End.</b> (with a full stop) indicates the very last line or end of the program.                                                                                                                                                                               |


## Punctuation

The semicolon is used to separate declarations and most statements. Many Pascal programs are written to follow a structure, such as those in previous examples. However, since the semicolon separates the individual declarations and statements, the following Pascal program will also compile and run:

```
Program Sentence; Begin Writeln('Hello
to everyone'); End.
```

The keywords 'Begin' and 'End' act as brackets for multiple statements, so there is no semicolon after begin but usually after End to show the end of the statements. There are some occasions when no semicolon is used after End, as in the following example:

```
If (line >80)
Then Begin {perform the following two
 statements}
 Writeln('This is over the line');
 Newline := Newline - line;
```



```
End
Else Writeln('This is within the
line', Newline);
```

If a semicolon is placed with the End, this would cause a syntax error for the IF-THEN-ELSE statement.

The last line of every Pascal program's code must be 'End.' The full stop denotes the end of the program. Comments in { } can however be written after this last statement.

## Statements in Pascal

Statements give instructions to the computer. A simple statement is a single instruction while a compound statement comprises two or more statements that are within 'Begin' and 'End' keywords. A compound statement is useful if you need to perform some instructions in sequence or repetitively before moving on to the next set of statements. Various types of statements are explained in Table 11.2.

**Table 11.2** *Types of statements*

| Statement                                                                                                                            | Example                                     | Explanation                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input statement</b><br>Read and ReadLn reads data from keyboard, and waits for the <i>Enter</i> key on the keyboard to be pressed | Read;                                       | The cursor stays next to the text entered                                                                                                                                                                                             |
|                                                                                                                                      | ReadLn;                                     | The cursor moves to the next line after the text is entered                                                                                                                                                                           |
| <b>Assignment statement</b><br>Places data in a variable                                                                             | Mark := 30;                                 | := is called the assignment operator<br>The variable is on the left of the assignment operator<br>The value is on the right of the assignment operator<br>The semicolon (;) ends the statement<br>Here, Mark is assigned the value 30 |
|                                                                                                                                      |                                             |                                                                                                                                                                                                                                       |
| <b>Output statement</b><br>Shows the output on the monitor screen                                                                    | Write;                                      | The cursor remains at the same place on the screen                                                                                                                                                                                    |
|                                                                                                                                      | Writeln;                                    | The cursor goes to the start of the next line                                                                                                                                                                                         |
|                                                                                                                                      | Write(Mark);                                | Outputs 30                                                                                                                                                                                                                            |
|                                                                                                                                      | Writeln(Mark);                              | Outputs 30<br>         (  is the cursor)                                                                                                                                                                                              |
|                                                                                                                                      | Write('Score of',Mark);                     | Outputs Score of 30                                                                                                                                                                                                                   |
|                                                                                                                                      | Writeln('Score of',Mark);                   | Outputs Score of 30<br>         (  is the cursor)                                                                                                                                                                                     |
| <b>Compound statement</b>                                                                                                            | Begin                                       | Begin (of a compound statement)                                                                                                                                                                                                       |
|                                                                                                                                      | Mark := Mark + 1;<br>Writeln(Mark);<br>End; | Two or more consecutive statements<br>End; (of a compound statement)                                                                                                                                                                  |



## Questions

- 1 Explain the difference between:
  - a a compound statement and a single statement
  - b write and writeln
  - c := and =
- 2 Consider these two fragments of Pascal code:
 

**Fragment 1**

```
If (line > 80)
Then Writeln('Extra line');
Newline := Newline - line;
Writeln('Same line ',Newline);
```

**Fragment 2**

```
If (line > 80)
Then Begin
 Writeln('Extra line');
 Newline := Newline - line;
End
Else Writeln('Same line ',Newline);
```

  - a Determine the output if the values for the variable line were initialised to:
    - i line := 85
    - ii line := 55
  - b Convert Fragment 1 into a Pascal program. Initialise line to any value and Newline to 85. Compile and run the program.
    - i Initialise the value for the variable line to 85 and run the program again. Note the results.
    - ii Change the value for the variable line to 55 and run the program again. Note the results.
  - c Repeat part b for Fragment 2. Compare the results of both programs.

## 11.3 Formatting the output of Pascal programs

Sometimes the output of a Pascal program is just what you expected. There are times however, when you need to look at your code again to see what is missing, has errors, or is not set out in logical steps. The following sample programs illustrate various results. Type the program in Example 1 into your Pascal editor. Try to maintain the indentation of the code, then compile and run it. If you typed it correctly, your output should look like the result given below.

### Example 1

```
Program Bigger;
Var
 num1, num2: integer;
Begin
 Write('Enter two integers ');
End.
```

The result is:

```
Enter two integers |
```

The vertical bar '|' in the output represents the cursor. Notice where it is located.

For now, you cannot enter any data, since the program did not have any input statements. Modify the program code to change the output statement as follows:

```
Write('Enter two integers ');
```

to

```
Writeln('Enter two integers ');
```

Compile and run the program again. Where is the cursor now?

The result is:

```
Enter two integers
|
```

Suppose you were to output statements that combine numbers or text. Update your program once again (as in Example 2) to include input and output statements.

### Example 2

```
Program Bigger;
var
 num1, num2: integer;
Begin
 Write('Enter two integers ');
 Readln(num1,num2);
 Writeln('You entered ',num1,' and
 ', num2');
End.
```

When you run this program, type the number 3, then press the *Spacebar*, then type the second number 56, then press the *Enter* key. The program ends there.

The result is:

```
Enter two integers: 3 56
You entered 3 and 56
```

## Dividing two numbers

Adding, subtracting and multiplying integers and real numbers in a Pascal program involves assignment statements. However, division is slightly different if the numbers are integers or real numbers. Try compiling and running the following two programs:



### Example 3

```

Program Int_division;
Var int1, int2, result : integer;
Begin
 int1 := 10;
 int2 := 2;
 result := int1 div int2;
 Writeln(int1, ' divided by ',int2,
 'equals ',result);
End.

```

Your result should be similar to this:

```
10 divided by 2 equals 5
```

### Example 4

```

Program Real division;
Var realA, realB, result : real;
Begin
 realA:=10;
 realB:=2;
 result := realA div realB;
 Writeln(realA, ' divided by
 ',realB,' equals ',result);
End.

```

This time, your answer is not quite what you expected. It may look like:

```

1.0000000000000000E+0001 divided
by 2.0000000000000000E+0000 equals
5.0000000000000000E+0000

```

You need to apply the formatting feature to produce more user-friendly output. Example 5 has now been modified from the version in Example 4 by formatting the output of the numbers.

### Example 5

```

Program Real_division_2;
Var realA, realB, result : real;
Begin
 realA := 10;
 realB := 2;
 result := realA div realB;
 Write(realA:2:2, ' divided
 by ',realB:2:2, ' equals
 ',Result:2:2);
End.

```

The result now looks like:

```
10.00 divided by 2.00 equals 5.00
```

## Questions

Convert each of the following sets of pseudocode to a Pascal program. Be sure to:

- ♦ include comments in the program
- ♦ declare all variables
- ♦ include appropriate indentation.

1

```

Algorithm Calculate_age
INPUT thisyear
INPUT birthyear
age = thisyear - birthyear
IF age < 13
THEN OUTPUT "Millee is a teen"
ELSE OUTPUT "Millee is not a teen"
ENDIF

```

2

```

number = 1
WHILE (number <= 3) DO
BEGIN
 number = number + 1
 Output "the number is", number
ENDWHILE
OUTPUT "out of loop"

```

3

```

age = integer
FOR age = 13 to 19 DO
 Output "You are a teenager"
ENDFOR

```



4

```
Pseudocode: average_of_exam_marks
total = 0
count = 0
average = 0
REPEAT
 DISPLAY 'Enter mark'
 INPUT mark
 IF mark = -1
 THEN DISPLAY 'End of marks'
 ELSE
 total = total + mark
 count = count + 1
UNTIL mark = -1
IF count = 0
THEN DISPLAY 'No marks entered'
ELSE
 average = total/count
 OUTPUT average
END
```



## PROGRAMMING WITH VISUAL BASIC FOR APPLICATIONS

### 12.1 Introduction to Visual Basic for Applications

Visual Basic for Applications (VBA) is part of Microsoft's programming language. You can write customised programming code in Access, Excel and Word to complement their built-in functions. Each application also has the built-in Visual Basic Editor. If you use any of these applications for your School-Based Assessment (SBA), then you will not need to submit additional files for the programming component since it is included in the file. Microsoft Excel will be used to illustrate the programming examples in this chapter.

There are two ways to write Excel VBA code. Either place a Command Button on the spreadsheet to access the code or write VBA code within the editor.

The following steps summarise one way to access the editor in Microsoft Excel, and how to write programs using VBA code:

- 1 Add the Developer tab if it is not visible.
- 2 Place a Command Button on the spreadsheet.
- 3 Change the name of the Command Button to an appropriate name.
- 4 Type the VBA code into the editor and set up the spreadsheet.

Test your program by clicking on the Command Button. You can also bring up the VBA editor by pressing *Alt + F11*.

### Adding the Developer tab

Once you have added the Developer tab in an application, the editor becomes available to write your

code and output your results. To add the Developer tab in earlier versions of Excel (such as 2007):

- 1 Click on the Office button (top-left corner of the screen)
- 2 Select Excel Options, in lower-right corner of the Option box.
- 3 Select the Popular tab on the left and check the Show Developer tab in the Ribbon option (Fig 12.1).

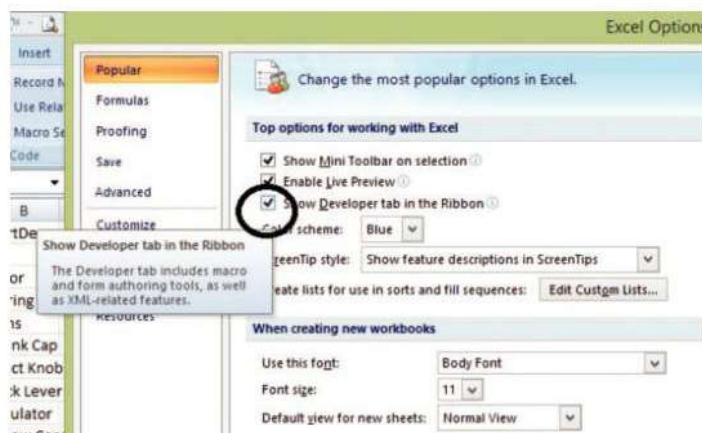


Fig 12.1 Adding the Developer tab in Microsoft Excel 2007

To add the Developer tab in later versions of Excel, such as (2013 and 2016):

- 1 Right-click anywhere on the ribbon, and then click Customize the Ribbon (Fig 12.2).
- 2 Under Customize the Ribbon, on the right side of the dialogue box, select Main tabs (if necessary).
- 3 Tick the Developer check box (Fig 12.3). Click OK.

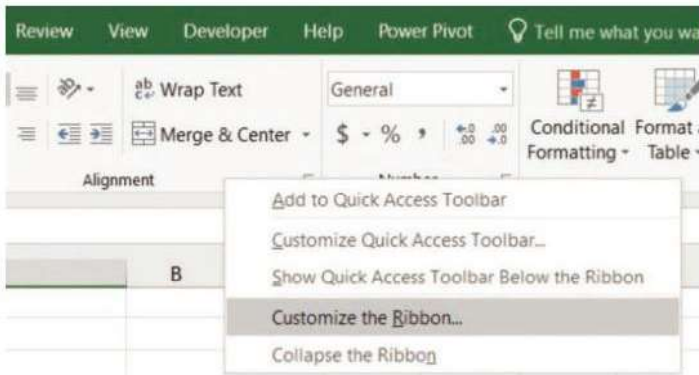
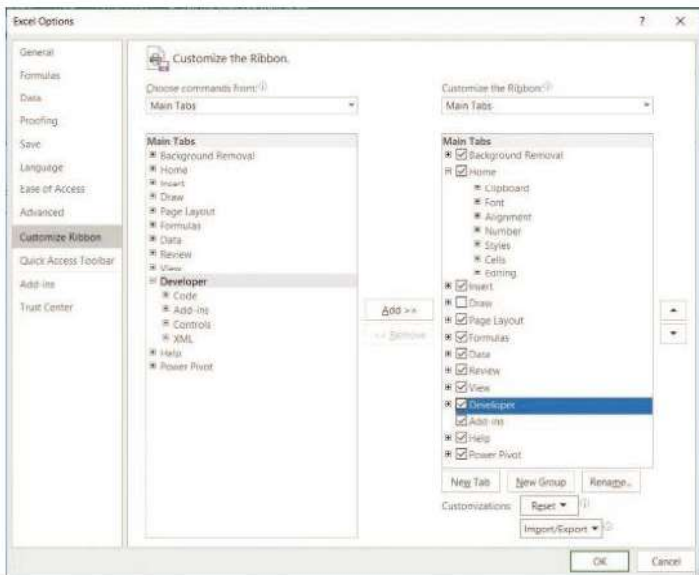
Fig 12.2 Right-click on the ribbon and select *Customize the Ribbon*

Fig 12.3 Check the option to Add the Developer tab

## The Command Button

The **Command Button** is similar to the Submit, OK, Cancel and Close buttons that you would use to close a form or dialogue box. It is used to execute code in the Visual Basic editor.

To place a Command Button on your worksheet, use the following steps.

- 1 Click the Developer tab, then click the Insert button.
- 2 In the ActiveX Controls group, click the Command Button icon (Fig 12.4).
- 3 Click on the worksheet or drag to create the rectangle that will be the Command Button on your worksheet (Fig 12.5).

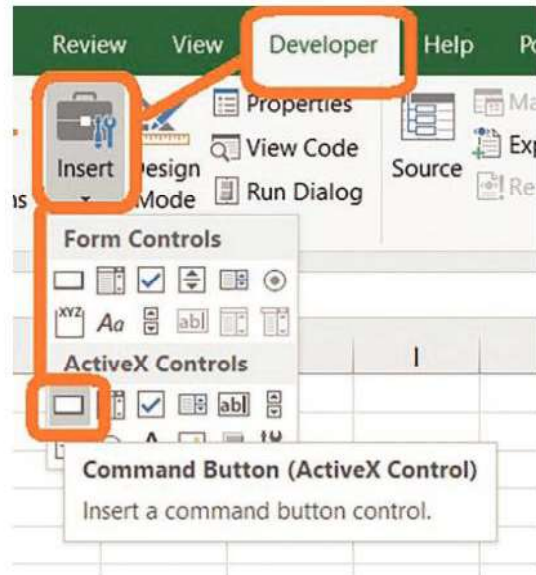


Fig 12.4 Locating the Command Button on the Developer tab

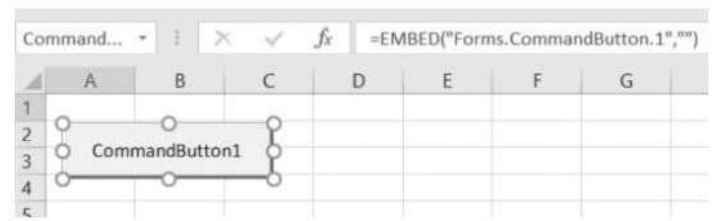


Fig 12.5 Inserting a Command Button on the worksheet

If you wish to move the Command Button to another location:

- 1 Click on Design Mode to select it, then click on the Command Button.
- 2 Once the button shows circles at the edges, you can drag it or cut and paste it in another location.

As you create more Command Buttons on the sheet, the number for the Command Button increases from **CommandButton1** to **CommandButton2** and so on.

## Using the VBA editor

Once your Command Button has been placed on the worksheet, you can now add your VBA code:

- 1 Right-click on the CommandButton1 icon. Note that the Design Mode icon on the ribbon should be selected.



- 2 Click View Code. The Visual Basic editor appears (Fig 12.6).

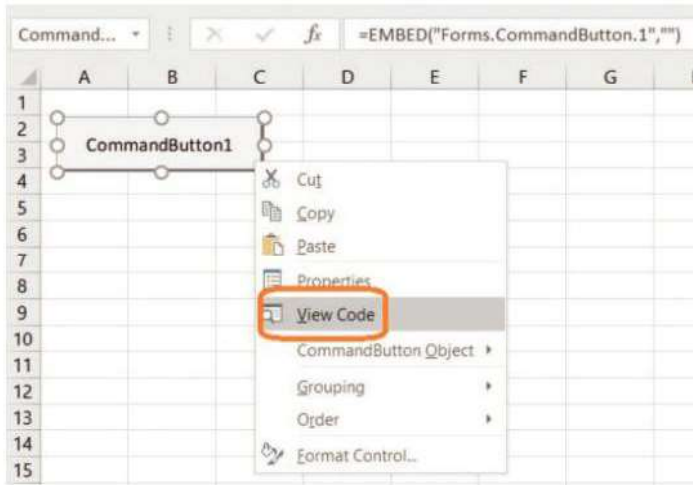


Fig 12.6 Getting ready to add code to your Command Button

- 3 Place your cursor between the Private Sub CommandButton1\_Click() and End Sub lines. Your code should always be typed between these two lines.
- 4 As your first example, type the code shown below:

MsgBox "This is my first message in VBA!"

### Edit the Command Button

In the VBA editor window (Fig 12.7), to rename the Command Button make sure the Properties window is visible. If it is not, then click View in the menu and select Properties window.

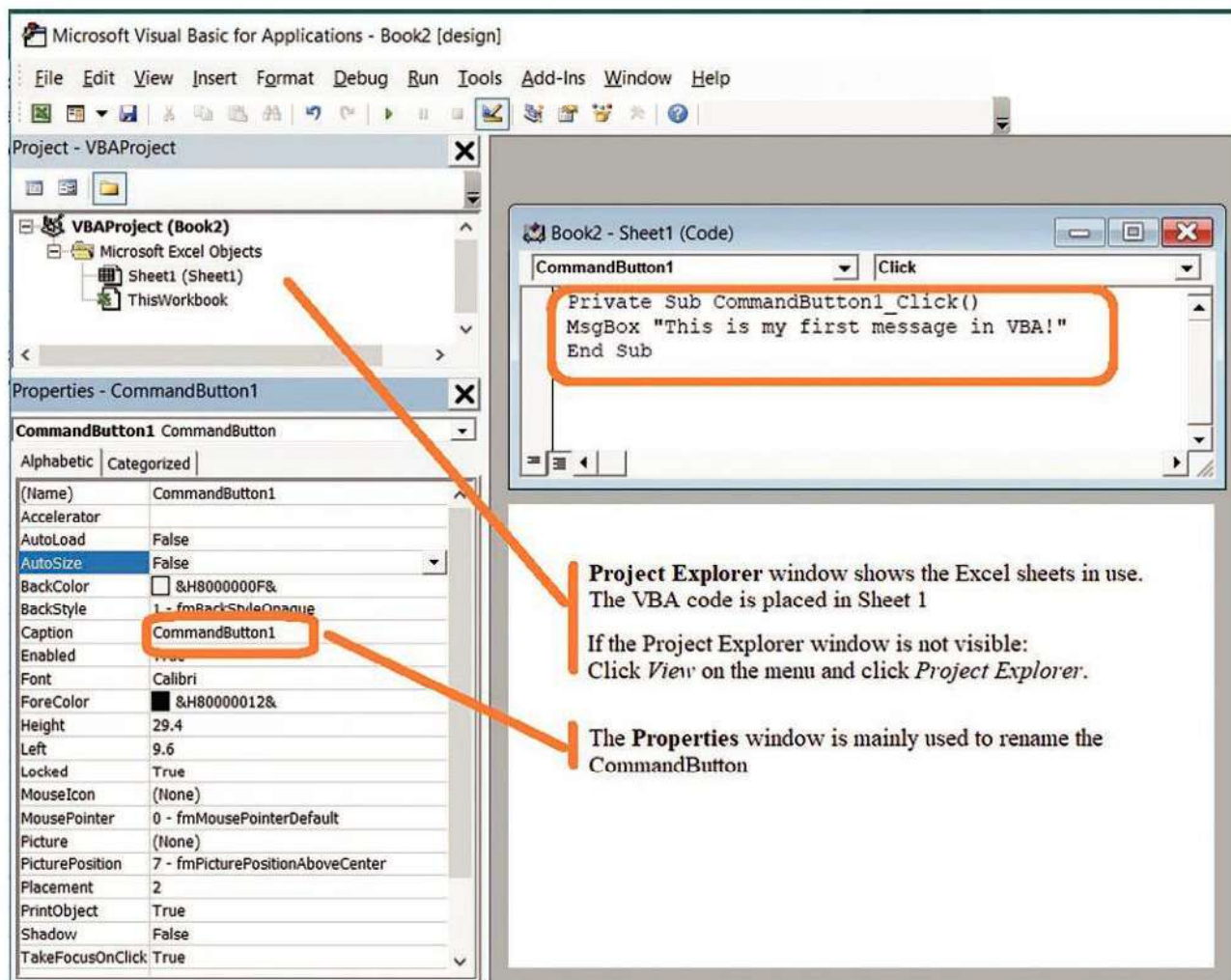


Fig 12.7 The Visual Basic editor where programs are typed and edited

You can use the Properties window to make the following changes to CommandButton1:

- 1 Click the caption option to change the name from CommandButton1 to Click Here.
- 2 Click the Font option to the type and size of the font to Calibri 14 point and bold. You can access this by clicking the box with three dots to the right of the font name.

As you create another set of code in the same sheet, add another Command Button. A sheet can therefore have many Command Buttons each for a different task. Each Command Button will also be numbered so that you can differentiate among them. Alternatively, you can rename them so that they relate to the purpose of their tasks.

### Testing your program

First you need to close the Visual Basic editor:

- ♦ Select File and click Close and Return to Microsoft Excel.
- ♦ Alternatively, you can switch to the Excel sheet by clicking the Excel icon under the File menu option in the editor.

Once you are in the spreadsheet:

- 1 Make sure Design Mode is deselected by clicking on it.
- 2 Then click the Command Button on the sheet.

You should see results on the screen. Figure 12.8 shows the message that should be displayed.



Fig 12.8 Clicking the Command Button shows the result

### Saving the program

Your code is part of your Excel worksheet so there is no need to save it as another file. However, you will need to save your workbook as a Macro-Enabled Workbook. In the Save As dialogue box, use the drop-down menu to do this.

### Correcting errors

Sometimes the program will display error messages and not produce output as expected. The VBA editor highlights the text to indicate that an error occurs after that point. It also shows the line and column number of the location of the error. Figure 12.9 shows that the error is found on line 2, column 7. The error in this code is that a single quote was used instead of a double quote.

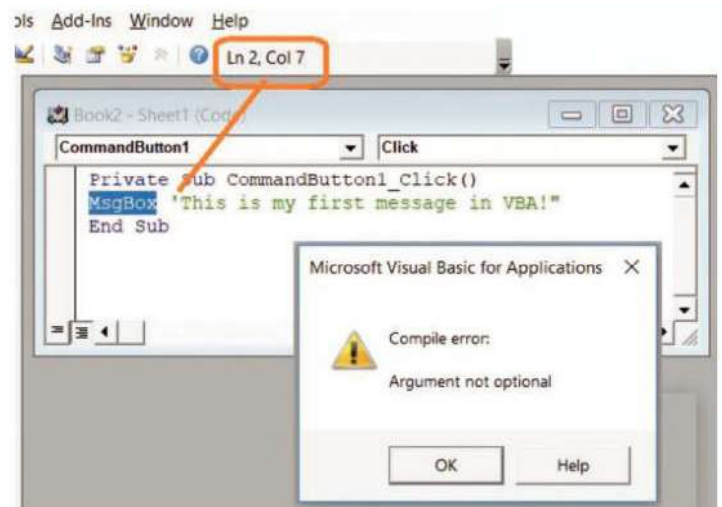


Fig 12.9 The VBA editor highlights the text to indicate that an error occurs after that point

### Questions

- 1 What must be done to access the VBA editor?
- 2 Give two examples of names that can be used for Command Buttons.
- 3 What is the name of the controls group where the Command Button icon is found?
- 4 What must be done to the Command Button before running the VBA code?
- 5 How are Excel sheets with VBA code saved?



Congratulations! Now that you have written your first VBA code, let's show how it can be used with your worksheet and SBA. Since you may be using data from your Excel sheet to create VBA code, this section will provide some examples of statements, variables, conditional and loop control structures. Remember that all code is written between the Private Sub CommandButton1\_Click() and the End Sub lines that are found in the code window editor.

## Declaring variables and data types

A VBA variable must be less than 255 characters, contain no spaces or special characters and not begin with a number. Each variable must have a data type. The main data types that you may want to use for your SBA are shown in Table 12.1.

**Table 12.1** VBA data types

| Numeric data types        | Non-numeric data types  |
|---------------------------|-------------------------|
| Integer                   | String (characters)     |
| Double (for real numbers) | Date                    |
| Currency                  | Boolean (true or false) |

The syntax to declare a variable is:  
Dim variableName as DataType.

Note that the following three lines:

```
Dim mark As Integer
Dim cost As Double
Dim exam As Date
```

can also be written on one line as:

```
Dim mark As Integer, cost As Double,
exam As Date
```

## Statements

Remember to include comments in your code. These are denoted by a single quote at the start of a statement. The following statements are quite useful when working with data on your spreadsheet. As you write your code,

you may notice that the editor sometimes adjusts the names of the variables to start with capital letters, indents some lines of code, and even changes the colour of some of the various parts of the syntax such as green for comments and dark blue for keywords in your code. Table 12.2 shows some examples of VBA statements.

**Table 12.2** Useful VBA statements

```
Private Sub CommandButton1_Click()
Must be the first line of code when using a Command Button

'This is a comment
Remember to use comments in your code

Range("cell address").Value
This statement accesses data in a cell

Range("C4").Value = info
Places data from the variable named info into cell C4

info = Range("B3").Value
Places data from cell B3 into a variable named info

Worksheets(1).Rows(1).Select
Selects row 1 in the current worksheet

Worksheets(1).Columns(3).Select
Selects column 3 or Column C

Worksheets(1).Cells(1,1).Select
Selects cell A1

Selection.Copy
Then copies the data in cell A1

Worksheets(1).Cells(2,1).Select
Selects cell B1

ActiveSheet.Paste
Pastes data in cell B1

End Sub
Must be the last line of code
```

## Displaying information

The MsgBox and InputBox are two keywords that display messages or information via a dialogue box.

```
MsgBox "text message" displays 'text
message' in a dialogue box.
MsgBox Range("B3").Value displays the
data stored in cell B3 in a dialogue box.
```

Unlike MsgBox, the InputBox keyword requires a variable to which it can return the results. Consider the two statements below that use the InputBox keyword. The variable `year` must be declared as an Integer data type.

`year = InputBox("Enter a Year", "Information Required")` shows a prompt in a dialogue box that waits for the user to input text to an empty area or click a button, and then outputs the contents of the text box to the screen (Fig 12.10).



Fig 12.10 The InputBox displays a prompt in a dialogue box and waits for the user to input text or click a button

Note the slight variation in the following Inputbox statement:

`year = InputBox("Enter a Year", "Information Required", 2019)` shows the default number 2019 in the input area.



Fig 12.11 A default value can be part of the dialogue box

Remember to save your workbook as a Macro-Enabled Workbook.

### Conditional statements

The following examples illustrate the use of the IF-THEN and IF-THEN-ELSE statements.

#### IF-THEN

In Example 1, the code places a message in a cell based on the value found in cell C3. Notice that no

message is displayed if a value of 50 or less is found in the cell (Fig 12.12).

#### Example 1

```
Private Sub CommandButton1_Click()
First line of code

Dim score As Integer, result As String
Two variables are declared

score = Range("C3").Value
value in cell C3 is placed in variable score

If score > 50 Then result = "Good!"
if score is greater than 50 message is placed in variable
named result

Range("C4").Value = result
message is pasted in cell C4

End Sub
Last line of code
```



Fig 12.12 Example of the IF-THEN statement

#### IF-THEN-ELSE

In Example 2, a different message is placed in cell J15, based on the value found in cell J8 (Fig 12.13).

#### Example 2

```
Private Sub CommandButton1_Click()
First line of code

Dim eligible As Integer, result As String
Declare two variables

eligible = Range("J8").Value
Value is cell J8 is placed in eligible

If eligible >= 5 Then
If the value in eligible is >= 5,
result = "Yes!"
Then place message 'Yes' in variable result

Else:
Otherwise
result = "Sorry!"
Place another message in variable result
```



```
End If
```

End of the IF-THEN-ELSE statements

```
Range("J15").Value = result
```

Place message in cell J15

```
End Sub
```

Last line of code



Fig 12.13 Example of the IF-THEN-ELSE statement

### Nested IF-THEN-ELSE (CASE)

The code in Example 3 determines whether the discount on the amount of sales entered in cell G10 is 0%, 5%, 10% or 15%. The discount amount is placed in cell G12 and the amount due is calculated and placed in cell G15 (Fig 12.14). The Case statement is used since it can become confusing with too many IF-THEN-ELSE statements. The syntax for the Case statement is:

```
Select Case Condition
Case value_1
 Code when Condition = value_1
Case value_2
 Code when Condition = value_2
Case value_3
 Code when Condition = value_3
Case Else
 Code to execute when all the other
 cases are False
End Select
```

### Example 3

```
Private Sub CommandButton1_Click()
Dim saleamt, disc As Double
Declare two variables
saleamt =
Round (Range("G10").Value
```

Place the integer value from G10 and place in saleamt

```
Select Case saleamt
```

Start of Case statement

```
Case Is >= 3000
```

If value is >= 3000

```
disc = 0.15
```

Assign 15% to the discount

```
Case Is >= 2000
```

If value is >= 2000

```
disc = 0.1
```

Assign 10% to the discount

```
Case Is >= 1000
```

If value is >= 1000

```
disc = 0.05
```

Assign 5% to the discount

```
Case Else
```

Otherwise

```
disc = 0
```

No discount

```
End Select
```

End of case statement

```
Range("G12").Value = disc
```

Place discount in cell G12

```
Range("G15").Value = (1 - disc) * saleamt
```

Calculate the amount due and place in cell G15

```
End Sub
```

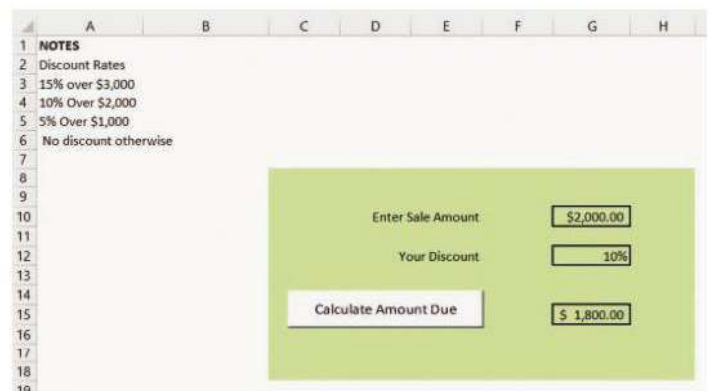


Fig 12.14 Example of using a Case statement

### Loops

You can use a single loop to cycle through a one-dimensional range of cells. This means one row or one column.

## 12 Programming with Visual Basic for Applications

Example 4 is used to calculate the updated cost of a list of items after a 10% discount is applied. The code works as follows:

- ♦ The While loop starts from row 2, column 2 (cell B2).
- ♦ Checks if it is a blank cell, which is denoted by "" (contains 'Shirt' for the first cycle).
- ♦ It takes the value in row 2 column 2 (cell B2), which is the cost of the item and calculates 90% of the cost. This is the same as calculating 10% of the item and then subtracting it from the cost. The updated cost is placed in the same row but in column 3.
- ♦ Cycles to the next row in column 2 (or column B) by adding one to the variable row.
- ♦ Returns to the top of the loop to check if the cell for the items is empty. Takes the cost for row 3, column 2 which is the blouse, and goes through the loop again.

### Example 4

```
Private Sub CommandButton1_Click()
Dim cost As Double
Declare variable cost
cost = 2
Start at the first item in the list (Shirt)
Do While Cells(cost, 2) <> ""
The While cell is not empty
```

```
Cells(cost, 3).Value = Cells(cost,
2).Value * 0.9
```

Calculate 90% of the value and place it in the next column

```
cost = cost + 1
```

Go to the next row

Loop

Return to Do While

End Sub

|   | A      | B       | C                 | D | E | F |
|---|--------|---------|-------------------|---|---|---|
| 1 | Item   | Cost    | With 10% Discount |   |   |   |
| 2 | Shirt  | \$55.00 |                   |   |   |   |
| 3 | Blouse | \$66.00 |                   |   |   |   |
| 4 | Skirt  | \$77.00 |                   |   |   |   |
| 5 | Pants  | \$88.00 |                   |   |   |   |
| 6 | Dress  | \$99.00 |                   |   |   |   |
| 7 |        |         |                   |   |   |   |
| 8 |        |         |                   |   |   |   |
| 9 |        |         |                   |   |   |   |

First, enter cost of each item

Calculate Discount

Fig 12.15 Example of loop before the Command Button is clicked

|   | A      | B       | C                 | D | E | F |
|---|--------|---------|-------------------|---|---|---|
| 1 | Item   | Cost    | With 10% Discount |   |   |   |
| 2 | Shirt  | \$55.00 | \$49.50           |   |   |   |
| 3 | Blouse | \$66.00 | \$59.40           |   |   |   |
| 4 | Skirt  | \$77.00 | \$69.30           |   |   |   |
| 5 | Pants  | \$88.00 | \$79.20           |   |   |   |
| 6 | Dress  | \$99.00 | \$89.10           |   |   |   |
| 7 |        |         |                   |   |   |   |
| 8 |        |         |                   |   |   |   |
| 9 |        |         |                   |   |   |   |

First, enter cost of each item

Calculate Discount

Fig 12.16 Example of loop after the Command Button is clicked

### Questions

- 1 Create a Command Button that will display your name in a dialogue box when clicked.
- 2 Create a Command Button that, when clicked, will display a message if the data stored in cell D5 is equal to 10.
- 3 Modify the code in Example 2 as follows:
  - ♦ if there are more than 4 passes at Grade II, then output 'Well done'
  - ♦ if there are fewer than 3 passes at Grade III, then output "Good work"
  - ♦ add the number of passes for all grades. If the number is greater than 6, then output 'Prize and Plaque'.
- 4 Use the Case statement to output whether a student is at primary, secondary or tertiary level based on the age entered.
- 5 Calculate the updated cost of a list of five drinks after a 15% tax is applied.
- 6 Convert each of the following sets of pseudocode to VBA code. Be sure to:
  - ♦ include comments in the program
  - ♦ declare all variables
  - ♦ place the data in separate cells
  - ♦ include appropriate indentation.





- a** Pseudocode: calculate\_age  
 Input thisyear  
 Input birthyear  
 age = thisyear - birthyear  
 IF age < 13  
 THEN output "Millee is a teen"  
 ELSE output "Millee is not a teen"  
 ENDIF
- b** Pseudocode: output numbers  
 number = 1  
 WHILE (number <= 3) DO  
 BEGIN  
 number = number + 1  
 Output "the number is", number  
 ENDWHILE  
 Output "out of loop"
- c** Pseudocode: determine whether a teenager  
 age = integer  
 FOR age = 13 to 19 DO  
 Output "You are a teenager"  
 ENDFOR
- d** Pseudocode: average of exam marks  
 total = 0  
 count = 0  
 average = 0  
 OUTPUT 'Enter a mark'  
 INPUT mark  
 WHILE (mark is not equal to -1)  
 total = total + mark  
 count = count + 1  
 OUTPUT 'Enter a mark'  
 INPUT mark  
 ENDWHILE  
 IF count = 0  
 THEN DISPLAY 'No marks entered'  
 ELSE  
 Average = total/count  
 DISPLAY average  
 END

- 7** Consider the following VBA code:

```
Private Sub CommandButton1_
Click()
Dim amount As Double
Dim interest As Double
Dim years As Integer
Dim payment As Double
amount = (Range("C2").Value)
interest = Range("C3").Value
years = Range("C4").Value
payment = PMT(interest / 12,
years * 12, -amount)
Range("C5").Value = payment
End Sub
```

- a** List the names of the variables in the code.  
**b** List the cells that are used to store data in a spreadsheet.  
**c** State the name of the Excel function that is used in this code.  
**d** You now wish to borrow \$50,000.00 and repay at 8.5% per annum for 7 years. Copy the grid of cells and use the code in C to write the data in the appropriate cells.

|   | A | B | C | D |
|---|---|---|---|---|
| 1 |   |   |   |   |
| 2 |   |   |   |   |
| 3 |   |   |   |   |
| 4 |   |   |   |   |
| 5 |   |   |   |   |

## Appendix 1

### Information Technology School-Based Assessment guidelines

The Information Technology (IT) School-Based Assessment (SBA) comprises an integrated project requiring the use of database, spreadsheet and word-processing software. It also involves the design of a web page using free online web page software or word-processing software. It is expected that data will be exported among most or all of these applications. The SBA should also involve some problem-solving skills and the ability to use some aspect of the project to write a short program using a programming language determined by the teacher. The SBA is expected to be designed within the school and administered by the IT teacher.

Term 2 of the fifth year is spent mainly working on the SBA, but the SBA can be started earlier if schedules allow. Interim deadlines should be set for each of the four components. Data from one or more components can be used to complete others. The order in which the SBA is completed is determined by the project itself and may vary each year. The SBA is marked by the teacher using guidelines provided in the syllabus.

Your teacher should provide you with the description of the project. It will be based on one of the application areas suggested under Types of Projects in the syllabus. The project should comprise five tasks to be performed in the recommended application. Read through the description several times so you can become familiar with *what* you are expected to do, *how* you are expected to complete the various tasks using the applications, and *when* you are expected to provide results to your teacher. Your teacher may also provide the marking criteria so that you can see exactly how the marks will be awarded for each task. It is important that you read through your mark scheme as well. It would also be a good idea to read the instructions and keep an electronic copy of the SBA stored on your memory stick.

## Group or individual SBA

If you are working on your own, then make a plan for how you intend to complete the tasks. You will be responsible for working through the entire SBA on your own.

If you have been assigned to a group, then arrange to meet and determine who will be group leader, as well as to make sure that everyone has the project description. It would also be useful to set up an online forum so that members of the group can collaborate to share ideas and progress on the SBA. It is also a good idea to work through the SBA individually so that all group members are familiar with the project. That way each member can make suggestions and learn from each other about suitable methods to approach or complete a task and to troubleshoot problem areas.

### Preparing your data

Use the description of the project to decide where to start. Gather the data needed, such as names, addresses, dates of birth, items and so on. Start to work out the most suitable layout for the data. Then type in data in the application indicated in the project or using the most suitable application based on the description.

Create a folder for your SBA files. You may want to create multiple folders for each component so that your work is organised.

Make sure to name *every* document with a short description that includes your name or your group members' last names. Remember that your teacher will have many SBAs to review, and so having each document named appropriately will help you to get feedback quickly. You may want to start the name or label of each document with D1 for Draft 1 of your submission, then D2 for the second draft so that both you and your teacher know which submission is the most recent one.



Keep backup copies of your work at all times or email it to yourself. Keeping backups will mean that you do not have to start your project again from the beginning if anything happens to your files!

## Working with your teacher

It is best to meet the interim deadlines set by your teacher in order for the SBA to be completed on time. Failure to complete and submit the SBA means that you forfeit 25% of your final mark and possibly fail the subject. If a section (database, spreadsheet, word processing, web page design, problem solving or programming) is incomplete, you should still submit your SBA for marking as the incomplete sections may still be worth marks.

Your teacher cannot help you if you do not have a file, a sketch of a layout or something to indicate your progress. Once you have entered data in a suitable format – continue to keep your teacher informed of your progress. Let your teacher provide feedback about the layout of your data and the accuracy of any calculations, tables, queries and so on. Make the effort to show where you or the group may need assistance or what stage you have reached.

Try to meet draft submission deadlines – have your submission ready for sending or delivering to your teacher. The intention is to provide you with a preliminary mark for your submission. This may be provided along with the details of the marks awarded. This feedback is an important part of the process and should be used to try to adjust your preliminary work before resubmitting for review. Ask your teacher about any tasks that you may need assistance with or any marks that have not been awarded based on the mark scheme.

## Word processing

Your word-processing task and web page design should be developed with the overall theme of the project in mind. Try to incorporate the requirements based on the marking criteria.

Have a friend or a group member review all documents to make suggestions. If it is part of your SBA, you may want to perform the mail merge a few times or regenerate the table of contents to correct errors and tweak any formatting. Ask other people to give their critique of the documents – your English teachers, for example, and if possible, people in the workplace. The submission of this document should include your primary, secondary and final documents created by merging.

## Web page design

Your web page can be created using a word processor or a free online web page software. Remember that it is limited to *one* web page. If a word processor is being used, then it is recommended that the design of the web page is created at the end of the word-processed document or in a separate document.

If an online web page builder is used, then paste the hyperlink to the online web page on the last page of your word-processed document. However, include screenshots in case the web page has not been published so that the moderator can still review the design.

It is important that you place your candidate number near the top of the web page as a form of identification.

## Spreadsheet

Usually, the data used in your spreadsheet will be needed to complete tasks in the other applications. If this is the case for your SBA assignment, then an early start to planning your spreadsheet model is necessary. Consider the following:

- ◆ What is the best way to organise the data using one or more spreadsheets?
- ◆ What suitable row and column titles are needed?
- ◆ How many data items are required in each sheet?
- ◆ What functions and formulae should be used for the calculations?
- ◆ Which values can be referenced by using absolute addresses in functions and formulae?



- ♦ For any advanced filter or pivot table tasks, what criteria should be used, and where should the resulting data be placed?
- ♦ How should textual data be formatted (for example, features such as bold, italic, use of borders and merging cells)?
- ♦ How should numerical data be formatted (for example, accounting or currency and number of decimal places)?
- ♦ What type of chart(s) should be used, what data is necessary to create them, and where should they be placed (on the same sheet as the data or as a separate sheet)?

### *Database management*

If the assignment of your SBA requires that you first create a database, then this data will be used in the other sections of the assignment. Alternatively, you may be required to import data from the spreadsheet part of the SBA to the database. However, you will need to demonstrate your ability to construct tables using appropriate field names, suitable data types and field sizes, or to modify the design of the imported tables.

Primary keys are necessary and should be assigned to appropriate fields in all tables. Every table created for your database should then form a part of a relationship, ultimately for the creation of form, queries and reports.

The design of data entry forms and sub-forms should emphasise good design features and clarity for the benefit of anyone who will make use of it (your teacher or the SBA marker). Consider the form colour, layout of fields, the font type and size of the labels and data being displayed.

Every query required for the database should produce a result, and every query should be saved using an appropriate name.

Like queries, reports should display a result. Attention to the layout of the information in a report is important since some field headings or narrowed columns may not be displayed properly. Adjustments

to the placement of headings and orientation of the report should be clearly seen. That includes all titles, subtitles, fields and calculations required for the report.

### *Problem solving and program design*

A collaborative approach has its advantages in this component of the SBA. However, if you are working alone, it certainly helps to seek the advice of others whilst managing this component. Try to approach the solution through discussions with your teacher to ensure that you and/or your group understands the scope and requirements of the problem. Create your algorithm. It may not be perfect but at least have one for your teacher to review. Your teacher can guide you towards a more accurate solution based on what you and/or your group would have submitted.

Implement your solution with the language selected by your teacher – test your solution thoroughly and record your test data and output in the form of screenshots. Fix any errors arising from your tests until the working solution and suitable output is reached.

### **Final submissions**

Note that all of your work must be submitted to your teacher in soft copy. You may need to save some files in PDF format, or place them in folders according to the requirements for submitting your SBA.

Create an IT SBA folder with your name or the name of the group leader. Inside this folder create four subfolders – Word processing, Spreadsheet, Database, Problem solving and program design. Save the files into the folders.

- ♦ *Word processing:* If you have mail merge as a requirement, then you should have the primary and secondary document (even if it is a spreadsheet or database file), web page and final merged letters in this folder.
- ♦ *Spreadsheet:* All spreadsheet files in the Spreadsheet folder. If it is the same file that was used for the mail merge, then copy the file in this folder as well.



- ♦ *Database:* should contain your database file. Again, if it is the same file that was used for the mail merge, then copy the file in this folder as well.
- ♦ *Problem solving and program design:* You could have a number of different files in this folder, as shown in the list below. However, most of the information can be included in one word-processed document. The program with the code can be submitted as a separate file.

|                                    |                                                                                                                |
|------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>Cover sheet</b>                 | The information supplied here is essential since it is used to identify your submissions                       |
| <b>Problem definition</b>          | This can be a Word document that provides the statement of that part of the problem that was chosen for coding |
| <b>Algorithm</b>                   | You are expected to include flowcharts or pseudocode for the segment of code that will be written              |
| <b>Source code</b>                 | A copy of the code written in the programming language chosen by the teacher                                   |
| <b>Trace table using test data</b> | Supply the test data that produced the output to determine whether the tasks have been performed correctly     |
| <b>Program execution</b>           | Submit screenshots of the working program showing data entry and results produced                              |

Your teacher should inform you of the process for submitting your IT SBA. You may be required to email your SBA, in which case you can zip the folder before attaching it to your email message. Be guided by your teacher for the submission of your SBA.

Again, it is very important to meet your teacher's deadline(s) for submission. Time must be allocated for processing received SBAs before uploading the final marks to CXC.

## Appendix 2

### Teacher guidelines for the School-Based Assessment

Teachers are encouraged to use the following guidelines to make sure that all the requirements for the students'

SBA are provided. Since the SBAs are uploaded to a portal for moderation, there is no need to print any documents for submission. It is advisable to create a specific email address where students can email their submissions for review. Then folders created on a memory stick or hard drive can be created with the name of each student. These folders will store their submissions for grading and safekeeping until the samples have been chosen for moderation. A spreadsheet can also be used to track their progress for each component during the completion of their SBA.

The folder for each student should contain the submissions from the requirements of the SBA. You may wish to type out each requirement, similar to the list, so that you do not omit anything. Every mark counts for your students! The following list provides an overview of the requirements for the SBA.

## Requirements for the SBA

|                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Word-processing task(s) with table of contents</b> | <ul style="list-style-type: none"> <li>♦ Documents which require formatting of text with subscript, superscript, tables, fonts and different line spacing, page layout</li> <li>♦ Any two of: inserting/importing files, columns and/or tables</li> <li>♦ Any two of: table of contents, mail merge and/or fillable forms</li> </ul>                                                                                                                                                 |
| <b>Web page design</b>                                | <ul style="list-style-type: none"> <li>♦ A logo depicting the concept of the project</li> <li>♦ Defined areas on the page for navigational links and content</li> <li>♦ At minimum, two hyperlinks of the following:               <ul style="list-style-type: none"> <li>♦ link to a location within the web page</li> <li>♦ link to an email address</li> <li>♦ link to another web page (which may or may not exist)</li> <li>♦ link to user-created files</li> </ul> </li> </ul> |



|                               |                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Spreadsheet/s</b>          | <ul style="list-style-type: none"> <li>◆ A maximum of two major tasks (e.g. create the spreadsheet; modify the spreadsheet)</li> <li>◆ No more than THREE requirements (e.g. sorting of data, use of functions and formulae on data)</li> <li>◆ Creation of no more than TWO types of charts</li> </ul>                                                                                                      |
| <b>Databases</b>              | <ul style="list-style-type: none"> <li>◆ THREE tables or files</li> <li>◆ TWO queries (using criteria from one table, and more than one table)</li> <li>◆ ONE calculation within queries</li> <li>◆ ONE form, showing a main and sub-form (e.g. to search for a record, or to move to the next or previous record)</li> <li>◆ ONE report, with grouping and sorting involving TWO or THREE tables</li> </ul> |
| <b>Algorithm development</b>  | <ul style="list-style-type: none"> <li>◆ A statement that defines the problem (e.g. you may want to calculate the average salary of a company's employees)</li> <li>◆ IPO, flowchart and/or pseudocode of solution</li> <li>◆ Trace table</li> </ul>                                                                                                                                                         |
| <b>Program implementation</b> | <ul style="list-style-type: none"> <li>◆ Program listing showing your code compiled free of errors</li> <li>◆ If necessary, the test data that you used from the trace table to produce the output should be provided as a printout</li> </ul>                                                                                                                                                               |

## SBA mark scheme

In devising your mark scheme, remember that there are certain tasks that you cannot verify, such as confirming that the candidate used Find and Replace instead of manually correcting words! Here is a sample list of skills that cannot be verified by moderator, and therefore should not be included in your mark scheme.

Avoid allocating marks in your candidate's SBA for:

- ◆ use of a spellchecker
- ◆ Find and Replace

- ◆ block operations such as cut, copy and paste
- ◆ adding or deleting records
- ◆ changing field definitions.

Also try not to award more than 6 marks for layout and document formatting features such as justification; bold, underline and italics; single and/or double spacing; superscript and/or subscript; font and font size; bullets and numbering.

## Appendix 3

### Guidelines for problem solving and program design

The general-proficiency SBA consists of an individual project worth 90 marks. Students should therefore follow the steps below in developing the problem solving and program design section of their project.

Teachers are advised to use the Assessment Criteria found in the CSEC syllabus as a guide for allocating marks for the project. The copy containing the student's mark alongside CSEC criteria should be uploaded with each student's project when submitting the sample projects to the online portal provided by the Caribbean Examination Council (CXC®).

## Define the problem

Defining the problem requires six steps:

- 1 *Specify objectives and user:* Explain what you are trying to accomplish and state the type of person who will be using this program.
- 2 *Specify the desired output:* What kind of output are you expecting from this program? You can write meaningful variable names and a brief description of each one.
- 3 *Specify the desired input:* Since you know what the output is, you can specify what the input should be. Again, write meaningful variable names and a brief description of each one.
- 4 *Specify the desired processing:* What processing should the input go through in order for it to provide the necessary output?



## Design the program

This second stage consists of two steps:

- 1 *Design details using IPO charts or flowcharts:* Then begin to write your algorithm using pseudocode. This will be used as a guide when you start writing your program.
- 2 *Do a structured walkthrough:* You should go through the design and see if anything has been omitted or needs to be corrected. You can also use a trace table to check the logic of your algorithm.

## Code and document the program

This stage is where the actual writing of the program starts. It consists of three main steps:

- 1 *Select the appropriate programming language:* For your SBA you will be using the programming language chosen by your teacher.
- 2 *Follow the syntax:* You can start writing the program from the flowchart or algorithm. Most programming languages have a particular syntax which you must follow.
- 3 *Put comments in your code:* First, write in programming code the purpose of your program. This is when you write comments to explain what

your code is doing. For example, a comment such as `{Prompt for a number}` can be typed on a line above the code that actually prompts the user for a number.

Then, as you write your code, it is best to include comments before conditional statements and loops to explain the purpose of the code. However, comments can be included anywhere in the code to clarify the logic of some sections.

## Test the program

In order to test a program there are three things you must do:

- 1 *Perform desk-checking:* Manually go through the program making sure its logic works.
- 2 *Debug the program:* Use a compiler or interpreter to detect any programming errors. Then locate and correct these errors.
- 3 *Test the program with real-world data:* After you make sure that the program is correct, you must test the program by providing it with input and seeing what the output is.

Make sure that you take screenshots of the output of your program to include with your submissions.

## Answers to end of topic questions

### Topic 1.1

- input, output, processing and storage *or* accept data, manipulate/process data, produce results and storing data and results
- the IPOS cycle
- hardware
- central processing unit
  - control unit
  - arithmetic and logic unit.
- An input device gets data into a computer while an output device gets processed information out of the computer.
- an operating system
- software
  - communication technology.

### Topic 1.2

- input device
  - touchpad or touchscreen
  - biometrics
  - scanner
  - direct data entry (DDE) device
  - optical mark recognition (OMR)
  - turnaround document.
- manual data entry
  - remote control.

### Topic 1.3

- display devices, printing devices and audio devices
- pixel
  - printer
  - impact or dot-matrix printer
  - thermal printer
  - 3D printer
  - earbuds.

### Topic 1.4

- to complete processing instructions as quickly as possible
- random-access memory (RAM)
  - read-only memory (ROM)
  - hybrid memory.
- bit

### Topic 1.5

- secondary storage
- magnetic media (hard drive), optical disks, flash memory drives
- as backup storage
- flash memory cards
- Advantages: multiple users can access on document at the same time to make updates; users can access their data from anywhere and at any time once they have Internet access. Concerns: risk of data being accessed, deleted, stolen or corrupted.

### Topic 1.6

- to control the hardware and how all other software works

- computer operating systems: LINUX, UNIX, Windows, Mac OS; mobile operating systems: Apple iOS, Android
- booting
  - virtual memory.

### Topic 1.7

- batch
  - online *or* real-time
  - online.
- import *or* download
  - file compression.
- JPEG *or* PNG
  - MPEG
  - MP3.

### Topic 1.8

- custom-written software
  - specialised software
  - integrated software.
- Google Docs, Sheets *or* Slides, Microsoft 365
- presentation software, PowerPoint *or* Google Slides
  - custom-written software.

### Topic 1.9

- hardware
  - hardware
  - both.
- too many options and sub-menus
- windows, icons, menus and pointers
  - graphical user interface.

### Topic 1.10

- personal computer
  - mainframes
  - an embedded system *or* embedded device.

### Topic 1.11

- The battery could be dead after prolonged use.
- The cartridge may not be the correct one for the printer, the cartridge may not be installed properly *or* the ink may be dry in the cartridge.

### Topic 2.1

- The data is applicable *or* appropriate for the user.
- accurate, timely, complete, appropriate *or* cost-effective
- information commodity *or* information as a commodity.

### Topic 2.2

- data validation: to prevent errors. data verification: to detect when errors occur.
- bad sectors on a hard disk *or* bad memory
- 10am (*or* any time between 8am and 4pm)
  - 300 *or* any number greater than 250
  - 3, 4 *or* 5 days
  - 100233.

### Topic 2.3

- hard copy
  - soft copy.
- turnaround document, multiple choice sheet, lottery sheet, handheld computer *or* microfilm.

### Topic 2.4

- A data file is a collection of records and a record is a collection of related data fields *or* file → records → fields *or* fields → records → file
- to help businesses store and update *or* modify large amounts of files containing data
- primary key
- serial
  - sequential
  - direct access
  - index sequential.

### Topic 2.5

- if the balance on the due date is greater than 0
- temperature, pressure, liquid flow rate, proximity *or* amount of light.

### Topic 3.1

- upload
  - download.
- paging systems, mobile (or cellular) networks *or* personal communication services
- printers and computers
- wide area network
  - local area network
  - wireless local area network
  - metropolitan area network.
- peer-to-peer network
- star, bus and ring
- hotspot
- simplex
  - half-duplex
  - full duplex.
- cable
  - wireless.
- intranet

### Topic 3.2

- electronic (email), uploading data, downloading data, internet relay chat, browsing the web, file transfer protocol *or* newsgroups.
- converts analogue and digital signals between the telephone and the Internet
  - sends data from the modem to different devices connected to it
  - expands the number of devices that can be connected to a router
  - provides the computer with a dedicated connection to a network.
- broadband
  - 2G, 3G *or* 4G-LTE
- blog: posts of text or picture entries; vlog: posts of video entries
- Hypertext Markup Language (HTML)
- Voice over IP (VoIP)



**Topic 4.1**

- 1 external: natural disasters: floods and other natural phenomena (hurricanes, earthquakes, volcanoes), power surges and spikes *or* terrorist activities that target buildings or rooms with computer systems.  
internal: employee error, no backup procedures, no protection of hardware and/or software, no anti-virus programs *or* ex-employee data not removed from systems
- 2 a data security  
b computer security  
c security threat.
- 3 a Deliberate  
b Accidental  
c Cyber threat  
d Cyber security.

**Topic 4.2**

- 1 a Computer fraud is unauthorised access and modification to financial accounts; propaganda is the use of computer systems to distribute information.  
b Phishing is the use of websites and email messages that trick users into entering personal information; identity theft is the use of computerised systems to steal credit card information and other personal details.
- 2 a unauthorised access, single  
b industrial espionage, multiple  
c electronic eavesdropping, multiple  
d denial-of-service (DOS) attack, multiple.

**Topic 4.3**

- 1 a Files can become damaged, corrupted or even lost.  
b Saves and updates files that are no longer needed on a regular basis.
- 2 a false  
b false  
c false  
d true  
e true.
- 3 a fireproof cabinets  
b uninterruptible power supply (UPS)  
c lock computers to the desks *or* limit access to authorised persons and maintain records and logs of computer usage  
d surveillance.

**Topic 4.4**

- 1 automation of jobs, retraining for different skills *or* the creation of new jobs
- 2 a poor *or* incorrect posture when using furniture or equipment  
b using computer screens for long periods, poor lighting, glare *or* being too close to (or too far from) the screen.

**Topic 4.5**

- 1 database, network and systems administrators
- 2 Computer support specialists provide assistance directly to computer users; social media specialists communicate with the

public *or* computer support specialists focus on troubleshooting and resolving computer problems; social media specialists share content with others.

**Topic 4.6**

- 1 Barcode readers used at point-of-sale (POS) linked to computer terminals automatically update inventory; credit/debit cards can be swiped so that money is automatically transferred to the business's account.
- 2 checking bank balance, transferring funds, paying bills online *or* viewing/printing bank statements
- 3 training simulations and interactive games

**Topic 5.1**

- 1 a Open retrieves a saved document; New creates a blank document.  
b Save gives your document a filename for the first time, or resaves an existing document; Save As allows you to create a duplicate document with another filename.
- 2 Highlight the text – usually by using the mouse to drag over the required text.
- 3 a 3  
b 2  
c 3  
d 2  
e 2
- 4 CTRL + 1, CTRL + 2, CTRL + 5
- 5 quickly apply the formatting from one set of text to another
- 6 Advantage: it saves time searching manually for the word to replace it. Disadvantage: you may replace parts of other words that contain the characters you want to replace elsewhere.

**Topic 5.2**

- 1 a mark which shows the end of formatting applied to part of the document
- 2 Pages 17 and 35 will use odd section breaks and page 22 will use an even section break.
- 3 Section breaks can be used if you have a page of your document in portrait orientation and the next page is to be in landscape orientation *or* if you have different margins set for multiple pages.
- 4 A footnote appears at the bottom of a page to give more information about text in the document.
- 5 Similarities: footers and footnotes are both found at the bottom of a page. Differences: a footer is the same text repeated while a footnote can be a reference and varies from page to page.

**Topic 5.3**

- 1 A table is useful for organising and aligning text.
- 2 A column is added to the left or right of the current column; a row is added above or below another row.
- 3 4 rows; 3 columns

**Topic 5.4**

- 1 a spellcheck  
b Track Changes  
c Comment  
d Word Count.
- 2 A spellcheck will ensure that words are spelt correctly according to a given dictionary; a thesaurus will suggest alternative words.
- 3 create a copy of the document; save as read-only; mark as final; use a password; restrict editing; or apply a digital signature.

**Topic 5.5**

- 1 copy or cut, and paste *or* Combine Documents
- 2 to combine information into one document

**Topic 5.6**

- 1 Select headings using Heading 1 or 2, place the cursor where the table of contents is to be placed and generate contents list.
- 2 update the page numbers only *or* the entire list to include the new heading

**Topic 5.7**

- 1 personalises letters to be sent to many people; saves time by not having to retype letters
- 2 letters, labels *or* envelopes
- 3 Selective mail merge allows you to choose specific data to merge, saving unnecessary printing.
- 4 You do not have to retype the data and possibly introduce typing errors.
- 5 a blank document, printer or email message
- 6 In the data source, select the box next to the record, select the category (field or heading at the top) or select all records.

**Topic 5.8**

- 1 margins, paper size, paper source *or* layout
- 2 Print creates a hard copy of your document, while Print Preview shows on screen how the document would look if it were printed.
- 3 entire document, single page or selected pages
- 4 collated (prints the document in order) or grouped (prints all copies of page 1, then all of page 2 and so on)

**Topic 5.9**

- 1 Paper-based forms:  
Advantages: don't need Internet access to complete, can be distributed in a face-to-face environment like a class. Disadvantages: users may write incorrect information, need to re-enter the data to analyse it.  
Online forms:  
Advantages: saves paper, can be distributed in soft copy, data can be downloaded and analysed, captures specific type of data for fields. Disadvantage: not everyone is online to access the form.
- 2 a Rich text: users can type multiple paragraphs. Plain text: users type limited amount of text.



## Answers to end of topic questions

- b Combo box: select from list of choices or type in information. Drop-down list: only select from list of choices.
- 3 iv, v, ii, vi, iii, i

### Topic 6.1

- web browser
- HTML (Hypertext Markup Language)
- Google Chrome, Microsoft Edge, Microsoft Internet Explorer, Mozilla Firefox, Opera or Apple Safari
- type the URL (web address) or use a search engine
- A website consists of one or more web pages.
- dynamic website
- a shopping or e-commerce  
b mobile  
c community building or social networking.
- content, organisation, number of web pages or security features

### Topic 6.2

- name of the website, web page, blog name, logo or company name
- a Homepage > Flowers > Garden > Seedlings  
b Sensible suggested arrangement.
- A bookmark is a hyperlink that jumps to a different part of the web page.
- You are either directed to another web page or a location within the current web page, or to open a document.

### Topic 6.3

- email address and password
- web page designs, options to change font types, sizes and text alignment; the ability to add your own graphics, pictures or video; option to edit the background colour or upload a background image; integrate Google Maps to navigate to the location of your business; add a contact form or subscription page, etc.
- to make sure that it is working well, e.g. that all graphics displaying correctly, text is in the correct position and background colours are not too bright
- check each hyperlink; check that animations, audio and video are playing correctly; check that email addresses are valid; make sure that all pages link to the homepage or check page titles are appropriate
- orphan page

### Topic 7.1

- budgets, financial statements, invoices or payroll
- a spreadsheet  
b column/row  
c column/row  
d =A1+A2  
e =A1-A2  
f WHAT-IF, forecasting  
g Microsoft Excel
- a =B4+C4  
b =B4\*2  
c =C4/3  
d =C4-B4

- a column  
b cell  
c row.

### Topic 7.2

- A spreadsheet is also a workbook that consists of multiple worksheets.
- Functions: =SUM(B1:G4), =MAX(E4:E6), =MIN(C1:C10), =AVERAGE(B1:C9).  
Formulae: =(E6-E4)\*1, =E3\*5%, =H2/3
- to save time having to select the cells repeatedly
- a Merge and Center  
b C5  
c D, E or F  
d 1 or 2  
e i left  
ii right  
f C3:C6  
g SUM  
h i MIN  
ii MAX  
iii The data in the column is too narrow for the width of the cell
- j i yes  
ii yes  
iii yes  
iv yes  
v yes

### Topic 7.3

- the name box is in the top left-hand corner that shows the active cell I8
- =D4\*E4 or =E4\*D4
- =F4\*10%
- F6\*CHARITY
- right-click on cell E2, select Insert and click Entire column or Select column E and select Insert
- a =MIN(D3:D6)  
b =AVG(E3:E6)  
c =COUNTA(A3:A6) – use COUNTA for the same range in columns A, B or C or =COUNT(D3:D6) – use Count for the same range in columns D to G.
- =IF(G3<30, "CANCEL", "")
- cells I3 and I6
- =IF(G4<E4, A4, G4)
- 25, 81, 61, Latin
- a E5  
b D6  
c D5  
d D5

### Topic 7.4

- a Currency, percentage, comma, increase/decrease decimal places  
b wrap text, merge and centre cells, orientation, increase/decrease indent; horizontal/vertical alignment in the cell  
c font style, size, bold, italics, underline, super/subscript; colour cell; colour text; border pattern

### Topic 7.5

- a sorting  
b AutoFilter  
c Advanced Filter  
d pivot table.
- a i

| Day       | Amount  | Location  |
|-----------|---------|-----------|
| Wednesday | \$10.00 | Bench     |
| Thursday  | \$13.00 | Café      |
| Friday    | \$14.00 | Desk      |
| Monday    | \$15.00 | Lunchroom |
| Tuesday   | \$25.00 | Desk      |

ii

| Day       | Amount  | Location  |
|-----------|---------|-----------|
| Friday    | \$14.00 | Desk      |
| Monday    | \$15.00 | Lunchroom |
| Thursday  | \$13.00 | Café      |
| Tuesday   | \$25.00 | Desk      |
| Wednesday | \$10.00 | Bench     |

iii

| Day       | Amount  | Location  |
|-----------|---------|-----------|
| Wednesday | \$10.00 | Bench     |
| Thursday  | \$13.00 | Café      |
| Tuesday   | \$25.00 | Desk      |
| Friday    | \$14.00 | Desk      |
| Monday    | \$15.00 | Lunchroom |

b i

| Day       | Amount  | Location |
|-----------|---------|----------|
| Wednesday | \$10.00 | Bench    |
| Thursday  | \$13.00 | Café     |
| Friday    | \$14.00 | Desk     |

ii

| Day       | Amount  | Location |
|-----------|---------|----------|
| Wednesday | \$10.00 | Bench    |
| Thursday  | \$13.00 | Café     |
| Tuesday   | \$25.00 | Desk     |
| Friday    | \$14.00 | Desk     |

### Topic 7.6

- A chart can make a greater visual impact than numbers.
- An embedded chart is located on the same page as the data. A chart sheet contains a chart that is attached to your data but exists on a separate worksheet.
- Chart title: a heading which is shown at the top of the chart. Category axis: identifies the data being charted on the horizontal x-axis; examples include dates and salespersons. Value axis: identifies the data being charted on the vertical y-axis; examples include numbers and years. Data series: the range of data selected to plot the chart.
- A column (or bar) chart is useful for comparing two or more similar items. A pie chart shows the percentage of the wedges in the chart.
- A legend is not necessary since there is only one set of data being plotted.



**Topic 7.7**

- 1 Advantage: You should print your data or charts so that you can view them away from the computer or distribute them to others *or* so that you have a hard copy in case you lose the data or the soft copy. Disadvantages: printing uses paper, which costs money and contributes to environmental damage *or* if you are not going to use the data afterwards then there may be no need to print it.

**Topic 7.8**

- 1 a MAX  
b PRICES  
c A4:A12  
d The exclamation mark is used to separate the name of the worksheet from the cell.

**Topic 8.1**

- 1 telephone book, dictionary *or* recipe book.
- 2 an electronic library, travel agency *or* online store
- 3 a a database  
b a spreadsheet  
c a word processor.

**Topic 8.2**

- 1 database, table, record, field
- 2 last and first could refer to names but it would be more appropriate to use LastName, FirstName or lastname, firstname. ID can refer to 'identification'; but it is better to use more meaningful names such as ProductID for PR-ID and CustomerID for CID.
- 3 primary key: **a, b, c** and **e**; candidate key: **f** and **g**; composite key: **h**; foreign key: **d**
- 4 Numeric data types are mainly used in fields where calculations are performed. Unless a primary key is created based on a calculation then it should be a text field.
- 5 a text  
b text  
c text  
d number  
e Y/N (Boolean)  
f text  
g date  
h text  
i text or AutoNumber  
j number.
- 6 a, b LastName and FirstName: text, 15; Relation: text, 15; Month of birth: number, 2 (using the numbers of the months as 1 to 12); Male: Y/N, size of 1 character (Boolean); Adult: Y/N or Boolean (Y says you are an adult)  
c Since you may have more than one person in the family with the same first name, it may be best to let the database create a primary key.

**Topic 8.3**

- 1 a Customer and Order (1:M); Product and Order (1:M)

- b Customer: CID; Product: PR\_ID; Order: CID+PR-ID  
c Lou and Ann  
d Pocket Diary  
e 4
- a Athlete and Division  
b 4  
c 2  
d Athlete table: AthleteID, Division table: Code  
e Athlete table: Code  
f Code  
g 1:M; one division has many athletes.  
h It cannot be deleted since there are athletes linked to this category. The athletes must be removed/deleted first.  
i Yes, this athlete can be deleted from the table.  
j Figman McJig  
k Under 13

**Topic 8.4**

- 1 Datasheet view allows you to enter raw data into each field of your database table. Design view allows you to enter field names, data types and descriptions into your database table.
- 2 form
- 3 check box and value list

**Topic 8.5**

- 1 ascending *or* ordering top to bottom
- 2 a Product  
b Description and Cost  
c Cost < 3  
d Cost <= 3
- 3 a Product and Order  
b Description and Discount  
c Discount = 'Y'

**Topic 8.6**

- 1 a [Firstname] & " " & [Lastname]  
b [Lastname] & " " & [Firstname]  
c [description] & " cost \$" & [cost]
- 2 incorrectly spelling a field name
- 3 a Total cost: [QTY]\*[Cost]  
b Deduction:[Cost]\*.10  
c NewCost:[Cost]\*1.10
- 4 a SUM  
b MIN  
c MAX  
d COUNT.

**Topic 8.7**

- 1 Tabular; since it is formatted like a table with rows and columns.  
a First, Last and Description are text; Cost, Quantity and TOTAL COST are numbers (Cost and TOTAL COST can also be currency); Discount is Yes/No *or* Boolean.  
b First and Last are the grouping fields.  
c First (sorted automatically with the grouping); Cost in descending order  
d TOTAL COST  
e Order Report – February  
f 3

- 3 a Department, LastName, FirstName and DaysWorked are text fields; Fees is a numeric *or* currency field.  
b Department  
c LastName  
d Fees  
e Payroll for September  
f SUM

**Topic 8.8**

- 1 Exporting is saving a copy of data from the current application which can be opened in another application. Importing creates a copy of data from another application in an existing database.
- 2 Use the copy and paste method.

**Topic 8.9**

- 1 To make sure you have included all the information that is required.
- 2 Have adequate field lengths; break down names and addresses to appropriate fields; think how tables can be linked; give the tables meaningful names.
- 3 To make sure the design stores and retrieves data correctly.
- 4 Tables that contain unrelated data; blank fields that should contain data; tables that contain the many duplicate fields of data.

**Topic 9.1**

- 1 algorithm phase and implementation phase
- 2 I = input, P = processing, O = output
- 3 IPO charts identify the data needed to convert the input into the desired outputs.
- 4 iii, v, i, ii, iv

**Topic 9.2**

- 1 a true  
b false  
c false  
d true  
e true
- 2 A variable stores data for the program to work with.
- 3 The values in a variable can change while the values do not change for a constant.
- 4 a integer  
b real  
c string or text  
d Boolean  
e string or text.
- 5 a year  
b height  
c mobile\_num  
d day (or night)  
e blood\_type

**Topic 9.3**

- 1 a REPEAT (count = count - 2) UNTIL count = 25  
b FOR (count = 1 to 20) DO count = count \* 5  
c IF total - count \* 5 THEN display count

## Answers to end of topic questions

- 2 a Loop A: REPEAT  
b Loop B: WHILE.
- 3 Left:  
a i Num  
ii Blu and Met  
iii BEGIN, INPUT, DISPLAY, END  
iv + and -  
v no Boolean operator
- b i no  
ii no  
iii yes  
iv yes.
- Right:  
i word  
ii no constant  
iii BEGIN WHILE, IF, THEN, ELSE, DISPLAY, ENDWHILE  
iv no arithmetic operator  
v <>
- 4 a i yes  
ii yes  
iii no  
iv yes.
- 5 a input hours, total = hours \* 10, Output total  
b input cost, amount, cash = amount - cost, output cash

5 a

```

INPUT mark1, mark2, mark3,
mark4
total = mark1 + mark2 +
mark3 + mark4
OUTPUT total;

```

b

```

mark = 0
total = 0
WHILE (number <= 4) DO
BEGIN
 INPUT mark
 Total = Total + mark
ENDWHILE
OUTPUT Total
END

```

c

```

mark = 0
Total = 0
REPEAT
 INPUT mark
 Total = Total + mark
UNTIL (number = 4)
OUTPUT Total
END

```

d

```

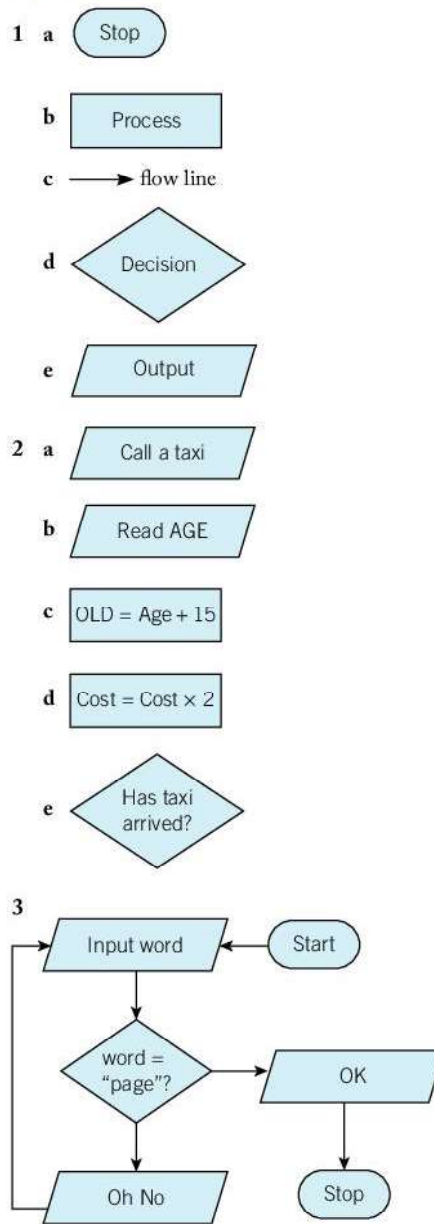
FOR Count = 1 to 4 DO
 INPUT mark
 Total = Total + mark
ENDFOR
OUTPUT Total

```

7 (Topic 9.5) T = TRUE, F = FALSE

| A | B | NOT A | NOT B | A AND B | A OR B | (NOT A) OR B | NOT (A OR B) | NOT (A AND B) | A AND (NOT B) |
|---|---|-------|-------|---------|--------|--------------|--------------|---------------|---------------|
| T | T | F     | F     | T       | T      | T (garden)   | F (town)     | F (town)      | F (town)      |
| T | F | F     | T     | F       | T      | F (town)     | F (town)     | T (garden)    | T (garden)    |
| F | T | T     | F     | F       | T      | T (garden)   | F (town)     | T (garden)    | F (town)      |
| F | F | T     | T     | F       | F      | T (garden)   | T (garden)   | T (garden)    | F (town)      |

### Topic 9.4



### Topic 9.5

- 1 a =  
b -  
c +  
d >=
- 2 a sisters = 3  
b age <= 18  
c passes = 7  
d weeks <= 4  
e pension >= 65  
f cheque <> cash

- 3 a false  
b false  
c false  
d false  
e true
- 4 a +  
b =, <, <=  
c AND
- 5 a i (word <> password), (count <= 3)  
ii 'Access' and 'Forgot password'  
iii 4  
b

| word = password | count <= 3 | (word <> password) AND (count <= 3) |
|-----------------|------------|-------------------------------------|
| TRUE            | TRUE       | Access                              |
| TRUE            | FALSE      | Forgot password                     |
| FALSE           | TRUE       | Forgot password                     |
| FALSE           | FALSE      | Forgot password                     |

- 6  $2^3 = 2 \times 2 \times 2 = 8$  conditions
- 7 See below

### Topic 9.6

1

| Mark | Total | Count | Average |
|------|-------|-------|---------|
| 25   | 25    | 1     | 25.00   |
| 30   | 55    | 2     | 27.50   |
| 12   | 67    | 3     | 22.33   |
| 10   | 77    | 4     | 19.25   |
| -1   |       |       |         |

- 2 mark = 65 await end of term report;  
mark = 71 Satisfactory results!; await end of term report;  
mark = 82 Excellent results!; see your teacher; await end of term report

3

| Sum | Count |
|-----|-------|
| 0   | 2     |
| 2   | 3     |
| 2   | 4     |
| 6   | 5     |
| 6   | 6     |

### Topic 10.1

- 1 Low-level languages are machine-dependent and not easy to understand by humans. High-level languages must be converted to machine language and are easier to understand by humans.

### Topic 10.2

- 1 Syntax is a set of rules when writing a programming language. Semantics is the meaning associated with the words, symbols and punctuation that make up the language.
- 2 Variable declaration means giving a new name and a data type for the variable.
- 3 so that they are easy to remember and understand.



**Topic 10.3**

- 1 program listing
- 2 A translator converts program code to machine code. Examples are compilers, interpreters and assemblers.
- 3 Debugging finds errors in program code and corrects them.

**Topic 10.4**

- 1 a Testing tries to find problems in your code. Debugging isolates and fixes the problems.  
b Black box testing is used to check that the correct output of a module is produced from the input. White box testing checks the accuracy of processing from input, through every possible path through to the output.
- 2 Input values, recording expected and actual output. Every conditional branch and loop must be tested at least once, and follow as many paths within each module and function as possible.
- 3 reliable, stable, easy to maintain, easy to use or portable
- 4 Examples of values: 4, 5; 2, 2; -2, 4; 0, 2.5.

**Topic 10.5**

- 1 to allow others to maintain the application if the original programmer is not available; to allow others to see internal and user details of the application
- 2 if the program needs to be debugged or updated, to understand what the code is doing or supposed to do
- 3 a curly brackets {}  
b single quote '
- 4 System documentation is written by technical persons for technical persons and is concerned with how the program works. User documentation is concerned with what the program does and how the end user makes the program do what it is supposed to do.
- 5 a overview of the process and tasks, name(s) of the authors of the program, date that program was created or modified or reviewed  
b overall system specification showing how the requirements are broken into a set of interacting programs; test plans describing how each program unit is tested; acceptance test plan describing the tests which must be satisfied before the system is accepted  
c functional description explaining what the system can and cannot do; installation document explaining how to install the application and giving suggestions how to recover from errors and basic problems when things go wrong; introductory manual explaining in simple terms how to get started with the system; reference manual describing system facilities available to the user and how these facilities can be used.

**Topic 11.1**

- 1 a invalid  
b invalid  
c invalid  
d valid  
e valid
- 2 identify and correct each error individually; save the program; compile the program again

**Topic 11.2**

- 1 a A compound statement: comprises two or more statements within Begin and End keywords. A simple statement comprises a single instruction.  
b write places the cursor immediately after the text that is written on the screen. writeln prints the text on the screen then places the cursor at the beginning of the next line.  
c := is used in an assignment statement. = is an operator to check equality.

2 a

| Output with line = 85 |            |
|-----------------------|------------|
| Fragment 1            | Fragment 2 |
| Extra line            | Extra line |
| Same line 5           |            |

b

| Output with line = 55 |              |
|-----------------------|--------------|
| Fragment 1            | Fragment 2   |
| Same line 30          | Same line 85 |

- c and d are practical work to observe the same answers output as in parts a and b  
e Fragment 2

**Topic 11.3**

- 1
 

```
{Name, date and purpose of
program}
Program Calculate_age;
Var thisyear, birthyear, age :
integer;
Begin
 {Prompt to enter two dates}
 Write ('Enter this year:');
 Readln(thisyear);
 Write('Enter a birthyear:');
 Readln(birthyear);
 {Calculate the age}
 age := thisyear -
 birthyear;
 If age < 13
 Then Writeln('A teen')
 Else Writeln ('Not a
 teen');
End.
```

- 2
 

```
{Name, date and purpose of
program}
Program Q2;
Var number, count: integer;
Begin
 {initialise the variable
 number}
 number := 1;
 While (number <= 3) Do
 Begin
 {Increment number by 1}
 count := count + 1;
 {Output the number}
 Writeln('You have entered',
 number);
 End;
 Writeln('out of loop');
 End.
```

- 3
 

```
{Name, date and purpose of
program}
Program Q3;
Var age: integer;
Begin
 For age := 13 to 19 Do
 Writeln('You are a
 teenager');
 End.
```

- 4
 

```
{Name, date and purpose of
program}
Program Q4;
Var total, count, mark:
integer;
average : real;
Begin
 {Initialise variables}
 total := 0;
 count := 0;
 mark := 0;
 {Prompt to enter a mark}
 Write('enter mark');
 Readln(mark);
 While (mark <> -1) Do
 Begin
 {Add the mark and increment
 the count}
 total := total + mark;
 count := count + 1;
 Writeln('enter mark');
 Read(mark);
 End;
 {If count = 0 then no marks
 were entered}
 If count = 0
 Then writeln('no marks
 entered')
 {Else calculate the average}
 Else begin
 average:= total/count;
 Writeln('The average is',
 average);
 End;
 End.
```

## Answers to end of topic questions

### Topic 12.1

- 1 add the Developer tab
- 2 submit, OK, Cancel or Close
- 3 activeX Controls group
- 4 make sure Design Mode is deselected by clicking it by remove the highlighting.
- 5 as a macro-enabled workbook

### Topic 12.2

1

```
Private Sub CommandButton1_
Click()
Dim message As String
'Type a message in cell J5
MsgBox Range("J5").Value
'Click command button to
output message
End Sub
```

2

```
Private Sub CommandButton1_
Click()
Dim info As Integer, message
As String
info = Range("D5").Value
'Type number in cell D5
If info = 10 Then
message = "Correct number!"
Else:
message = "Sorry!"
End If
Range("J15").Value =
message
'Click command button to
output message
End Sub
```

3

```
If eligible > 4 Then
result = "Well done!"

If eligible < 3 Then
result = "Good work"

Private Sub CommandButton1_
Click()
Dim total As Integer, message
As String
total = Range("J8").
Value + Range("J10").
Value + Range("J12").Value
' add numbers in cells J8,
J10, J12
If total > 6 Then
message = "Prize and
Plaque"
Else:
message = "Sorry!"
End If
Range("J15").Value =
message
'Put answer in cell J15
End Sub
```

4

```
Private Sub CommandButton1_
Click()
Dim age As Integer, message As
String
age = Range("G10").Value
'Place age in cell G10
Select Case age
Case Is >= 18
message = "Tertiary"
Case Is >= 16
message = "Secondary"
Case Is >= 11
message = "Primary"
Case Else
message = "Nursery"
End Select
Range("G12").Value =
message
' Output message in G12
End Sub
```

5

```
'Type names of drinks in Cells
A2 to A6
'Type cost of drings in cells
B2 to B6
Private Sub CommandButton1_
Click()
Dim cost As Integer
cost = 2
Do While Cells(cost, 2) <>
""
Cells(cost, 3).Value =
Cells(cost,2).Value * 1.15
cost = cost + 1
Loop
End Sub
```

6 a

```
Private Sub CommandButton1_
Click()
Dim thisyear, birthyear, age
As Integer
MsgBox "Enter two dates,
this year and a birth day"
thisyear = InputBox("Enter
this year", "Information
Required", 2018)
birthyear = InputBox("Enter
your birth year",
"Information Required", 0)
'Calculate the age
age = thisyear - birthyear
If age < 13 Then
MsgBox "A teen!"
Else
MsgBox "Not a teen"
End If
End Sub
```

b

```
'Name, date and purpose of
program
Private Sub CommandButton1_
Click()
Dim number, count As Integer
number = 1
Do While number <= 3
number = InputBox("Enter
an integer", "Information
Required", 0)
'Increment number by 1
count = count + 1
'Output the count
MsgBox "You have entered "
& number
Loop
MsgBox "out of loop"
End Sub
```

c

```
Private Sub CommandButton1_
Click()
Dim age As Integer
For age = 13 To 19
MsgBox "You are a teenager"
Next age
End Sub
```

d

```
Private Sub CommandButton3_
Click()
Dim Total As Integer, Count As
Integer, Average As Double,
mark As Integer, Message As
String
Total = 0
Count = 0
Average = 0
mark = 0
Do While mark <> -1
mark = InputBox("Enter mark",
"Information Required", 0)
If mark = -1 Then
Message = "End of marks"
Else
Total = Total + mark
Count = Count + 1
End If
Loop
If Count = 0 Then
Message = " No marks entered"
Else
Average = Total / Count
Range("C4").Value = Average
MsgBox Average
End If
End Sub
```

7 a amount, interest, years, payment

b C2, C3, C4, C5

c PMT

d Cell C2 contains 50,000, C3 contains 8.5%, C4 contains 7, C4 contains =PMT(C3/12,C4\*12,-C2).



**A**

adding text 105  
 algorithms 231, 233  
   algorithm phase 231  
   basic data types 233  
   conditional statements 234  
   implementation phase 231  
   loops 234  
   statements and keywords 234  
   subroutines 234  
   testing algorithms 256–7  
   variables and constants 233  
 alignment 107, 155, 170–1  
 AND operator 253  
 anti-virus software 85–6  
 application software 7, 29  
   customised and custom-written  
     software 29  
   general-purpose software 29  
   integrated software 30  
   popular application programs 30  
   software packages 30  
   specialised software 29  
 archives 86  
 arithmetic operators 156, 252  
 asynchronous transmission 66  
 audio devices 17

**B**

back problems 91  
 backups 86  
 banking 55, 97  
 barcode readers 10, 13  
 batch processing 27  
 battery problems 37  
 biometric systems 10  
 bistable devices 20  
 bits 20  
 blogging 74  
 Blu-ray Disks (BDs) 22, 25  
 Bluetooth 65  
 bookmarks 142–3  
 breadcrumbs 140  
 broadband 67, 71  
 browsers 71, 72  
 bugs 84  
 bus networks 64  
 business 95–6  
 bytes 20

**C**

cabled media 67, 71  
 CD-ROMs 22, 25  
 cells 154  
   active cell 154  
   cell ranges 157–8  
   cell references 154  
 central processing units (CPUs) 6  
 charts 180–4  
   changing charted data 182  
   creating charts 180–1  
   deleting charts 182  
   moving charts 181  
   sizing charts 181–2  
 cloud-based storage 23–4  
 coaxial cables 67  
 columns 112–15  
 command-line interfaces 31  
 commerce 95  
 communication channels 66  
   cabled media 67  
   direction of data flow 66–7  
   transmission media 67  
   transmission modes 66  
   wireless transmission 68  
 Communications Technology  
   (CT) 7  
 computer fraud 81–3  
 computer programs 7  
   algorithm design 233–4  
   flowcharts 246–51  
   input-processing-output (IPO)  
     charts 232  
   operators 252–4  
   problem definition 231  
   problem solving 231  
   program documentation 270–1  
   programming languages 260–1  
   programming with Pascal  
     274–82  
   programming with Visual Basic  
     for Applications (VBA) 283–91  
   pseudocode 235–44  
   running a program 266–7  
   testing algorithms 256–7  
   testing and debugging techniques  
     268–9  
   writing a program 262–5  
 computer–user interfaces 31–3

**computers 6**

  advantages and disadvantages  
     of networked and stand-alone  
     systems 66  
   broadcast configurations 62  
   computer networks 62–6  
   computer security 79–80  
   computer systems 34–5  
   computer vulnerability 79  
   features of a computer screen 15  
   point-to-point configurations 62  
   troubleshooting basic hardware  
     problems 36–7  
 conditional branching 236  
   IF-THEN statements 236–8  
   IF-THEN-ELSE statements  
     238–9  
 conditional statements 234, 264,  
   288–9  
   IF-THEN structure 248  
   IF-THEN-ELSE structure  
     248–9  
 contents list 122  
   generating the contents  
     list 122–3  
   linking to a page 123  
   selecting headings 122  
   updating the contents list 123  
 control systems 56–7  
 copying text 105–6  
 copyright 87–8  
 cyber security 79–80, 86–7

**D**

damage 80  
 data 8, 15, 19, 40–1  
   data logging 47–8  
   data processing 27–8  
   data security 80  
   data transfer 28  
   data validation 42–4  
   data verification 44  
   data-capture forms 46–7  
   direction of data flow 66–7  
   how data is represented 20–1  
   importing data 120–1  
   interpretation of coded data 44  
   problems associated with shared  
     data 44–5



- proprietary data 81
  - spreadsheets 154–5
  - storage 21–5
  - data protection 84
    - computer viruses 85–6
    - copyright and piracy 87–8
    - network and cyber security 86–7
    - protecting files and databases 86
    - protection from nature 84–5
    - protection from theft 85
    - spreadsheets 171
    - surveillance 84
  - databases 86, 97, 99, 191, 193, 227
    - advantages and disadvantages 192
    - calculated fields 217–19
    - calculated fields with numbers 217
    - calculated fields with numbers 218–19
    - candidate, composite and foreign keys 196
    - capturing and entering data 203–8
    - common data types found in tables 194
    - common design problems 227
    - creating a report 220–4
    - creating a table 194–9
    - data entry 203–4
    - data types 194
    - database management systems (DBMS) 191
    - database terms and definitions 193
    - errors in queries 219
    - exporting from a database 225
    - field length, description and properties 195
    - field name 194
    - field options 203–4
    - fields 193
    - importing to a database 225, 226
    - joining multiple database tables 200–2
    - mathematical functions 217–18
    - operators used in searching databases 211
    - primary key field 195–6
    - printing a report 223
    - property types for fields in a database table 195
    - queries 209
    - queries using more than one field 212–13
    - records 193
    - relationships 200–1
    - report formats 220
    - reversing queries 213–14
    - searching 209–10, 216
    - searching for specific records 210–11
    - sorting 214–15, 216
  - debugging 267, 268–9
  - decryption 86–7
  - deleting text 105
  - denial-of-service attacks 82
  - desktop systems 34–5
  - dictionary 116
  - digitisers 9
  - direct access file organisation 51–2
  - direct data entry (DDE) devices 10–12
  - display devices 15
    - features of a computer screen 15
  - documents 103
    - adding comments 118
    - combining files 120
    - creating tables and columns 112–15
    - footnotes and endnotes 110
    - headers and footers 110
    - mail merge 124–8
    - margins 104
    - orientation 104
    - paper size 104
    - printing 129
    - proofreading 116–19
    - protecting a document 118
    - section breaks 110–11
    - table of contents 122–3
    - typefaces 103
  - dot-matrix printers 16
  - double data entry 44
  - downloading 28, 62
  - driver licensing databases 99
  - drivers 26
  - drives 6, 21, 24
  - DVDs 22, 25
- ## E
- e-mail 70, 74–5
  - editing text 104–5
    - adding, deleting and retyping text 105
    - alignment 107
    - fonts 106–7
    - Format Painter 107–8
    - indenting 108
    - keyboard commands used to move the cursor 106
    - line spacing 108
    - moving and copying text 105–6
    - paragraph formatting 108
    - save vs. save as 106
    - undo and redo commands 106
    - working with tabs 107
  - education 97
    - teaching and instruction 97–8
  - EFT (electronic funds transfer) 55
  - electronic eavesdropping 82–3
  - electronic point of sale (EPOS) 13
  - embedded controllers 57
  - embedded systems 35
  - encryption 86–7
  - endnotes 110
  - entertainment 99–100
  - environmental concerns 92
  - errors 42, 50, 219
    - data entry errors 42
    - programming errors 266–7, 286
    - software and hardware errors 42
    - transmission errors 42
  - ethernet cables 67
  - extranets 69
  - eye problems 91
- ## F
- fibre optic cables 67
  - file organisation 49
    - direct access file organisation 51–2
    - index sequential file organisation 52–3
    - processing errors 50
    - record matching 49–50
    - serial and sequential file organisation 50–1
  - file servers 63
  - files 7



- combining files 120
- file compression 28
- file types 120–1
- protecting files and databases 86
- switching between applications 121
- fillable forms 130–5
  - add content to create the form 131
  - add the Developer tab to display the content controls 130
  - content controls 131
  - customise the labels of the form 131
  - open a template or create a blank document to design the form 131
  - protect the form 131
  - set or change properties for content controls 131
- financial abuse 81
- find and replace 108–9
- flash memory 22–3, 24
  - flash memory cards 23
- flowcharts 233, 246
  - conditional statements 248–9
  - flowchart symbols 246
  - nested conditions 249–51
  - sequence statements 247
- folders 32
- fonts 106–7
- footers 110, 141
- footnotes 110
- Format Painter 107–8
- formatting 103, 108
  - formatting the output of Pascal programs 280–2
  - spreadsheets 155–6, 170–1
- forms 130
  - creating a fillable form 130–5
  - Google Forms 131–2
- formulae 152, 156–9
  - arithmetic formulae 160
  - common arithmetic operators used in formulae 156
- full-duplex data flow 66
- functions 156–9, 160
  - AVERAGE 160–1
  - COUNTA and COUNT 161
  - COUNTIF 161
  - DATE 162
  - IF 162
  - MAX 162
  - MIN 162
  - PMT 162–3
  - RANK 163
  - SUM 163–4
  - VLOOKUP 164
- G**
  - gigabytes (GB) 20
  - global village 90
  - Google Forms 131–2
  - graphical user interfaces (GUIs) 32
    - icons 32–3
    - menus 33
    - pointers 33
    - windows 32
  - graphics tablets 13
- H**
  - hacking 80, 83
  - half-duplex data flow 66
  - hard disks 21–2
  - hard drives 21, 24
  - hardware 6, 26, 42
    - commercial applications 96
    - hardware interfaces 31
    - monitoring with hardware devices 84
  - headers 110, 141
  - health concerns 90–2
  - home pages 73
  - hotspots 65
  - hubs 63
  - hyperlinks 140, 142–3
- I**
  - icons 32–3
  - identity theft 81
  - IF-THEN statements 236–8
  - IF-THEN-ELSE statements 238–9
  - impact printers 16
  - importing 120–1
  - indenting 108
  - index sequential file organisation 52–3
    - indexed files 52
  - industry 57
    - industrial espionage 82
  - information 40–1
    - information as a commodity 41
    - misuse of information 81–3
  - information processing 54
    - advantages and disadvantages 54
    - banking 55
    - control systems 56–7
    - health care 54
    - industry 57
    - libraries 55–6
    - payrolls 55
    - setting up an information processing system 54
    - supermarket stock control 58–9
    - weather forecasting 57–8
  - information systems 40, 95
    - advantages and disadvantages 95
    - banking 97
    - business 95–6
    - education 97–8
    - law enforcement 99
    - legal, ethical and moral effects 93
    - medicine 98–9
    - recreation and entertainment 99–100
  - Information Technology (IT) 7
    - effects of IT in the workplace 89
    - IT skills required in the workplace 92–3
  - inkjet printers 16
  - input devices 6, 8
    - advantages and disadvantages 12–14
    - biometric systems 10
    - direct data entry (DDE) devices 10–12
    - manual input devices 8–9
    - pointing devices 9
    - remote-control devices 9–10
    - touch-sensitive devices 9
  - input-processing-output (IPO) charts 232
  - interfaces 31–3
    - improving interfaces 33
  - Internet 70
    - 2G, 3G and 4G-LTE 71, 72
    - advantages and disadvantages 76
    - blogs, vlogs and podcasts 74

broadband 71  
 cable 71  
 connecting to the Internet 70–1  
 creating web pages 74  
 dial-up 71  
 Internet caches 74  
 Internet protocols 72  
 web browsers 71, 72  
 Internet addresses 72–4  
   understanding Internet  
   addresses 73  
 intranets 68–9

**J**

job losses 89–90  
 joysticks 13

**K**

keyboards 8, 12  
   keyboard commands used to  
   move the cursor 106  
 keypads 8  
 keywords 234  
 kilobytes (kB) 20

**L**

landscape orientation 104  
 laser printers 16  
 law enforcement 99  
   driver licensing databases 99  
 libraries 55–6  
 line printers 16  
 line spacing 108  
 local storage 21–3  
 logic errors 267  
 logic operators 252  
 loops 234, 239–44, 250–1, 264–5,  
   289–90

**M**

magnetic tape 21, 24  
 mail merge 124–5, 127–8  
   selective mail merge 125  
   using Microsoft Word to perform  
   a mail merge 125–6  
 mainframes 34  
 manual input devices 8–9  
 margins 104  
 marketing and distribution 96  
 master files 49

medicine 98  
   expert systems 98–9  
   medical information systems 98  
 megabytes (MB) 20  
 memory 6, 19  
   hybrid memory 19  
   main memory 19  
   memory sticks 22–3  
   RAM (random-access  
   memory) 19  
   ROM (read-only memory) 19  
 menus 33, 103  
   menu-driven interfaces 31–2  
 MICR (magnetic ink character  
 recognition) 12, 14  
 microfilm 48  
 microphones 9  
 mobile devices 35, 36  
 mobile phones  
   2G, 3G and 4G-LTE 71, 72  
 mouse 8–9, 12  
 moving text 105–6  
 music piracy 83  
 musical instrument digital interface  
 (MIDI) 14

**N**

nested conditions 239, 249  
   loops 239–44, 250–1  
 network configurations 62–3  
   hubs 63  
   peer-to-peer networks 64  
 networks 6, 62  
   advantages and disadvantages  
   of networked and stand-alone  
   systems 66  
   network and cyber security 86–7  
   network layouts 64–5  
 non-impact printers 16  
 NOT operator 252–3

**O**

online processing 27–8  
 operating system 7, 26  
   booting 26  
   hardware control 26  
   software control 26  
 operators 252–5  
   combinations of operators 254  
   operators used in searching  
   databases 211

optical character recognition (OCR)  
   11–12, 13, 47  
 optical disks 22  
 optical mark recognition (OMR)  
   11, 14  
 OR operator 253  
 orientation 104  
 output devices 6, 15–18  
   audio devices 17  
   display devices 15  
   printing devices 16–17  
 outsourcing 90

**P**

packet sniffing 84  
 page numbering 108  
 paper sizes 104  
 paragraph formatting 108  
 Pascal 274  
   compiling the program 275–6  
   dividing two numbers 280–1  
   finding a Pascal compiler 274  
   formatting the output 280–2  
   punctuation 278  
   running the program 276  
   saving the program 275  
   statements in Pascal 278–9  
   structure of a Pascal program 277  
   writing your first program 274–5  
 paths 140  
 payrolls 55  
 peer-to-peer networks 64  
 peripheral devices 6  
 personal computers 34–5  
 personnel 94–5  
 phishing attacks 82  
 piracy 83, 87–8  
 pivot tables 174–5  
   frequency distribution 176  
   modifying pivot tables 177–8  
   one-dimensional pivot tables 175  
   pivot charts 177  
   two-dimensional pivot tables 176  
 pixels 15  
 plotters 17  
 podcasting 74  
 pointers 33  
 pointing devices 9  
 portrait orientation 104  
 printers 16–17



- troubleshooting printer problems 36–7
- printing 129
  - copies 129
  - Print Preview 129
  - print range 129
  - spreadsheets 185
- program documentation 270
  - documenting programming code 270
  - user documentation 270–1
- programming languages 260
  - choosing the programming language 260–1
  - implementation phase 260
- proofing language 116–17
- proofreading 116–19
- propaganda 81
- pseudocode 235–6
  - conditional branching 236–9
  - nested conditions 239–44
  - sequential statements 236
- R**
- RAM (random-access memory) 19
- real-time processing 27–8
- records 49–50
  - primary key 49
  - searching for a record 50–1, 52–3
- recreation 99–100
- redo command 106
- relational operators 252
- remote-control devices 9–10
- repetitive strain injury (RSI) 91
- research and development 95
- retraining 89–90
- retyping text 105
- ring networks 64
- ROM (read-only memory) 19
- routers 70
- running a program 266
  - debugging 267
  - from source code to object code 266
  - programming errors 266–7
- runtime errors 267
- S**
- sales 96
- save vs. save as 106
- scanners 9, 13
- SBA guidelines 292
  - final submissions 294–5
  - group or individual SBA 292
  - preparing your data 292–3
  - working with your teacher 293–4
- SBA guidelines for problem solving and program design 296
  - code and document the program 297
  - define the problem 296
  - design the program 297
  - test the program 297
- SBA guidelines for teachers 295
  - requirements for the SBA 295–6
  - SBA mark scheme 296
- screens 15
  - monitor problems 37
- ScreenTip 142
- search engines 73–4
- section breaks 110–11
- security 79–80
  - computer fraud 81–3
  - data security 80
  - deliberate and accidental damage 80
  - proprietary data and software 81
  - websites 140
- semantics 262
- sensors 12, 58
- sequence statements 247
- sequential statements 236
- serial and sequential file organisation 50–1
- sidebars 141
- SIM cards 23
- simplex data flow 66
- smart cards 10–11, 14
- smartboards 15
- social concerns 90
- software 6–7, 26, 42
  - anti-virus software 85–6
  - application software 7, 29–30
  - commercial applications 96
  - computer programs 7
  - proprietary software 79, 81
  - software interfaces 31
  - software piracy 83
  - system software 7, 26
- source code 260
- spellcheckers 116–17
- spreadsheets 152–3, 154, 160, 167–9, 179
  - Advanced Filter 173–4
  - align data 170–1
  - alignment 155
  - arithmetic formulae 160
  - AutoFill 157
  - AutoFilter 172–3
  - cells 154, 157–8
  - charts 180–4
  - column widths and row heights 155–6
  - cut, copy and paste 155
  - filtering records 172–4
  - format numbers 170
  - formatting features 170–1
  - formatting values and dates 155–6
  - formulae and functions 156–9
  - functions 160–4
  - importing files into Excel 186–7
  - labels 154
  - linking files in Excel 186–7
  - manipulating rows and columns 165–6
  - pivot tables 174–8
  - printing 185
  - protection 171
  - relative and absolute addressing 164–5
  - sorting 171–2
  - sorting a list by more than one field 172
  - sorting a list by one field 172
  - title locking 158–9
  - types of data 154–5
  - values 155
- stand-alone computers 63
- star networks 64
- statements 234, 236–9, 247–9, 264
  - statements in Pascal 278–9
  - statements in Visual Basic for Applications (VBA) 287
- stock management 96
- storage devices and media 6, 21
  - advantages and disadvantages 24–5
  - cloud-based storage 23–4
  - local storage 21–3

stress 92  
 stylus 9  
 subroutines 234  
 supermarket stock control 58–9  
 surveillance 84  
   monitoring with hardware devices 84  
   monitoring with utility software 84  
 synchronous transmission 66  
 syntax 262  
   syntax errors 266–7  
 system software 7, 26

## T

tables 112–15  
 tabs 107  
 telecommuting 90  
 teleworking 92  
 terabytes (TB) 20  
 testing algorithms 256–7  
 testing programs 268–9, 286  
 theft 85  
 thermal printers 16  
 three-dimensional (3D) printers 16–17  
 time-sharing 27  
 touch pads 9  
 touch screens 9, 13  
 touch-sensitive devices 9  
 Track Changes 116, 117  
 transaction files 49  
 transmission media 67  
 transmission modes 66  
 troubleshooting 36  
   battery problems 37  
   computer, laptop or mobile device does not respond when power is turned on 36  
   monitor problems 37  
   printer problems 36–7  
 turnaround documents 11–12, 46  
   alternatives 47  
 typefaces 103

## U

unauthorised access 80, 83  
 undo command 106  
 uninterruptible power supply (UPS) 84–5

uploading 28, 62  
 USBs 22–3  
 user interfaces 31–3  
 utility software 7, 26, 84

## V

validation 42  
   check digit 43–4  
   consistency check 43  
   data type check 42–3  
   format check 43  
   length check 43  
   presence check 43  
   range check 42  
   reasonableness check 42  
 variables 233  
 verification 44  
   visual checks 44  
 virtual memory 26  
 virtual reality 99  
 viruses 80, 85  
   preventing viruses 85–6  
 Visual Basic for Applications (VBA) 283  
   adding the Developer tab 283–4  
   Command Button 284, 285–6  
   conditional statements 288–9  
   correcting errors 286  
   declaring variables and data types 287–91  
   displaying information 287–8  
   edit the Command Button 285–6  
   loops 289–90  
   saving the program 286  
   statements 287  
   testing your program 286  
   using the VBA editor 284–6  
 vlogs 74  
 voice-recognition systems 13  
 voiceband 67

## W

weather forecasting 57–8  
 web pages 73, 74, 138, 141  
   backgrounds and themes 142  
   content 139, 141, 147  
   designing a web page 141–6  
   economy 140  
   footers 141

general layout 141  
 getting started 147  
 headers 141  
 hyperlinks and bookmarks 142–3  
 navigation 140  
 organisation 139  
 planning structure 141  
 security 140  
 sidebars 141  
 text and images 142  
 web technology concepts 72–4  
 webcams 9  
 websites 73, 138  
   categories of websites 138  
   creating a web page 147–9  
   description of websites and their use 139  
   finalising your web pages 147–8  
   planning your web pages 139–40  
 Wi-Fi 65  
 WIMP (Windows, Icons, Menus and Pointers) 32  
 windows 32  
 wireless networks 63  
 wireless transmission 68  
 word 19  
 Word Count 117  
 word processing 103  
   editing text 104–8  
   find and replace 108–9  
   key features 103–4  
   page numbering 108  
 workplace 89  
   effects of IT in the workplace 89  
   environmental concerns 92  
   health concerns 90–2  
   IT skills required in the workplace 92–3  
   loss of jobs and retraining 89–90  
   social concerns 90  
   stress 92  
 worksheets 154, 160  
 worms 85  
 writing a program 262  
   conditional statements 264  
   looping 264–5  
   program statements 264  
   program structure 262  
   sample program 262–4





# Information Technology

THIRD EDITION

*Oxford Information Technology for CSEC®*, the market leading textbook, has been updated to meet the requirements of the latest Caribbean Secondary Examination Certificate (CSEC®) syllabus. As well as comprehensive coverage, it provides essential support for the School-Based Assessment. With a focus on the development and application of problem solving, this course provides students with the knowledge and skills for the examination and beyond.

## *Oxford Information Technology for CSEC®:*

- ▶ Explains the key concepts
- ▶ Contains practice exam-style questions
- ▶ Contains relevant coverage of the most recent developments in School-Based Assessment (SBA) and provides guidelines for students and teachers
- ▶ Online support packed with extra practice material, along with answers
- ▶ Is in full colour with many useful illustrations to help
- ▶ Is clear and easy to understand
- ▶ Has fresh and up-to-date content with Caribbean examples

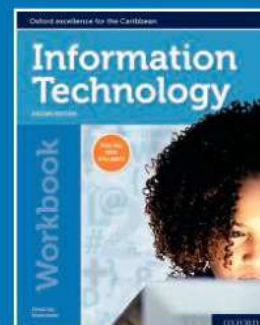


Further learning with online support at:  
[www.oxfordsecondary.com/9780198437161](http://www.oxfordsecondary.com/9780198437161)

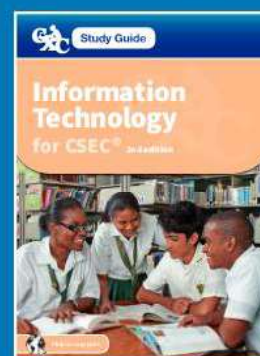


FOR THE  
NEW  
SYLLABUS

CSEC®



9780198437208



9780198437215

OXFORD  
UNIVERSITY PRESS

## How to get in touch:

**web** [www.oxfordsecondary.co.uk](http://www.oxfordsecondary.co.uk)  
**email** [schools.enquiries.uk@oup.com](mailto:schools.enquiries.uk@oup.com)  
**tel** +44 (0)1536 452620  
**fax** +44 (0)1865 313472

ISBN 978-0-19-843716-1



9 780198 437161