# Cambridge

# OL IGCSE

# Computer science

CODE: (0478)

# Chapter 07

# Algorithm design and problem solving

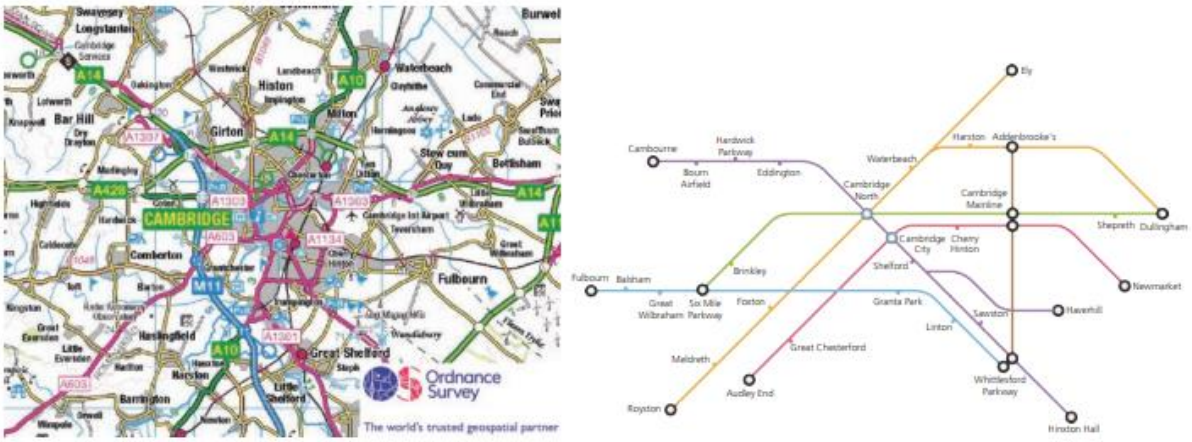
# 7.1 The program development life cycle

The program development life cycle is divided into five stages: analysis, design, coding, testing and maintenance. This chapter and Chapter 8 will discuss the four stages listed below:

» Analysis
» Design
» Coding
» Testing

## 7.1.1 Analysis

**Abstraction** and **decomposition** tools are essential in the analysis stage of problem-solving. Abstraction focuses on key elements, discarding unnecessary details. Decomposition tools help identify program requirements, ensuring a clear understanding of the problem's requirements and addressing any overlapping information.




▲ **Figure 7.1** Road map and rail map

## 7.1.2 Design

The program specification from the analysis stage is used to show to how the program should be developed. When the design stage is complete, the programmer should know what is to be done, this can be formally documented using **structure charts, flowcharts** and **pseudocode**.

## 7.1.3 Coding and iterative testing

**Iterative testing** involves developing and modifying a program's modules through modular tests, ensuring they meet the required performance standards.

## 7.1.4 Testing

The completed program or set of programs is run many times with different sets of test data. This ensures that all the tasks completed work together as specified in the program design.

# 7.2 Computer systems, sub-systems and decomposition

**Computer systems** consist of software, data, hardware, communications, and people, and can be divided into sub-systems for a single action. These systems can be large or small, and people often interact with multiple computer systems in their daily lives. For example, an alarm program on a smartphone can be accessed from a large computer system.

## 7.2.1 The computer system and its sub-systems

Computer systems are often divided into sub-systems, demonstrating their modular construction through top-down design and **flowcharts** or **pseudocode**. Programmers develop these sub-routines as sub-routines.

**Top-down design** is the decomposition of a computer system into a set of sub-systems, then breaking each sub-system down into a set of smaller sub-systems, until each sub-system just performs a single action.

## 7.2.2 Decomposing a problem

Any problem that uses a computer system for its solution needs to be decomposed into its component parts. The component parts of any computer system are: » inputs – the data used by the system that needs to be entered while the system is active
» Processes – the tasks that need to be performed using the input data and any other previously stored data
» Outputs – information that needs to be displayed or printed for the users of the system
» Storage – data that needs to be stored in files on an appropriate medium for use in the future.

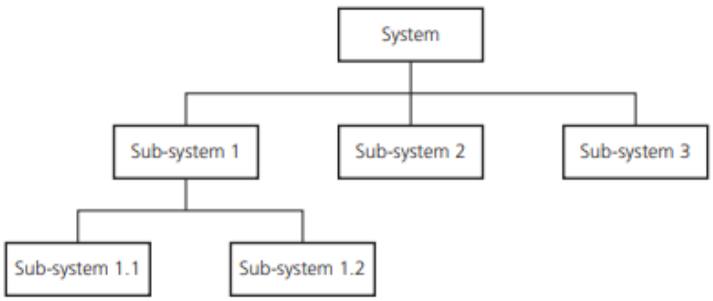
## 7.2.3 Methods used to design and construct a solution to a problem

Solutions to problems need to be designed and developed rigorously. The use of formal methods enables the process to be clearly shown for others to understand the proposed solution. The following methods need to be used by IGCSE Computer Science students:
» Structure diagrams
» Flowcharts
» Pseudocode.

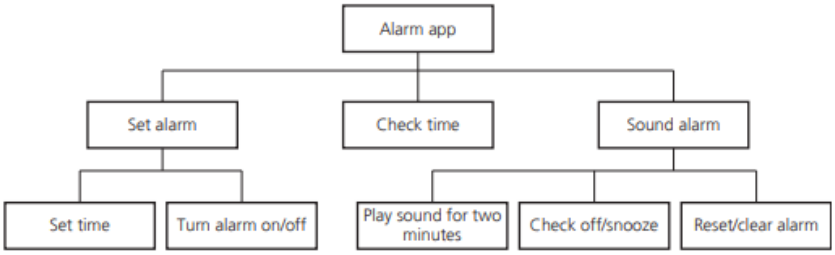
### Structure diagrams

Structure diagrams can be used to show top-down design in a diagrammatic form. **Structure diagrams** are hierarchical, showing how a computer system solution can be divided into sub-systems with each level giving a more detailed breakdown. If necessary, each sub-system can be further divided.

Consider the alarm app computer system for a smart phone; this could be divided into three sub-systems, setting the alarm, checking for the alarm time, sounding the alarm. These sub-systems could then be further sub-divided; a structure diagram makes the process clearer.




▲ **Figure 7.2** Basic structure diagram




▲ **Figure 7.3** Structure diagram for alarm app

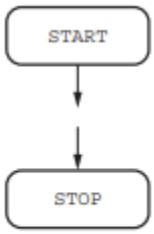
## Flowcharts

A **flowchart** shows diagrammatically the steps required to complete a task and the order that they are to be performed. These steps, together with the order, are called an algorithm. Flowcharts are an effective way to communicate how the **algorithm** that makes up a system or sub-system works.

Flowcharts are drawn using standard flowchart symbols.

## Begin/End

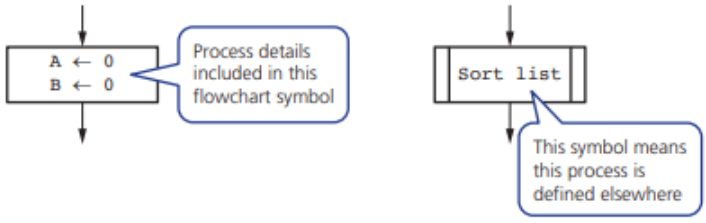
**Terminator flowchart symbols** are used at the beginning and end of each flowchart. (Figure 7.5)




▲ **Figure 7.5** Terminator symbols

## Process

**Process flowchart symbols** are used to show actions, for example, when values are assigned to variables. If a process has been defined elsewhere then the name of that process is shown. (figure 7.6)




▲ **Figure 7.6** Process symbols

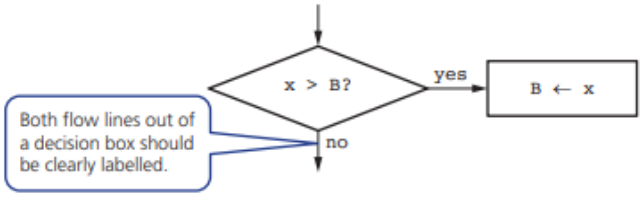
## Input and output

The same flowchart symbol is used to show the input of data and output of information. (figure 7.7)


▲ **Figure 7.7** Symbol used to show input and symbol used to show output

## Decision

Decision flowchart symbols are used to decide which action is to be taken next; these can be used for selection and repetition/iteration. There are always two outputs from a decision flowchart symbol. (figure 7.8)




▲ **Figure 7.8** Decision symbol

## Flow lines

Flowchart flow lines use arrows to show the direction of flow, which is usually, but not always, top to bottom and left to right. (figure 7.9)



▲ **Figure 7.9** Flow line

## Pseudocode

Pseudocode is a method of describing an algorithm using English key words like high-level programming language. It provides meaningful names for data items but is not bound by strict syntax rules. Consistency in writing is essential for easy understanding.

The pseudocode in this book is written in the following way to match the pseudocode given in the IGCSE Computer Science syllabus and to help you understand the algorithms more easily:

» A non-proportional font is used throughout
» All keywords are written in capital letters
» All names given to data items and subroutines start with a capital letter
» Where conditional and loop statements are used, repeated or selected statements are indented by two spaces.

## The pseudocode for an assignment statement

A value is assigned to an item/variable using the ← operator. The variable on the left of the ← is assigned the value of the expression on the right. The expression on the right can be a single value or several values combined with any of the following mathematical operators.

▼ **Table 7.1** Mathematical operators

| Operator | Action |
|---|---|
| + | Add |
| – | Subtract |
| * | Multiply |
| / | Divide |
| ^ | Raise to the power |
| ( ) | Group |

## The pseudocode for conditional statements

When different actions are performed by an algorithm according to the values of the variables, conditional statements can be used to decide which action should be taken. There are two types of conditional statement:

1 a condition that can be true or false such as: **IF** … **THEN** … **ELSE** … **ENDIF**

```
IF Age < 18
  THEN
    OUTPUT "Child"
  ELSE
    OUTPUT "Adult"
ENDIF
```

2 a choice between several different values, such as: **CASE OF** … **OTHERWISE** … **ENDCASE**

```
CASE OF Grade
  "A" : OUTPUT "Excellent"
  "B" : OUTPUT "Good"
  "C" : OUTPUT "Average"
  OTHERWISE OUTPUT "Improvement is needed"
ENDCASE
```

**IF … THEN … ELSE … ENDIF**
For an **IF** condition the **THEN** path is followed if the condition is true and the **ELSE** path is followed if the condition is false. There may or may not be an **ELSE** path. The end of the statement is shown by **ENDIF**.

There are different ways that an **IF** condition can be set up:

» use of a Boolean variable that can have the value **TRUE** or **FALSE** (see Chapter 8 for details of Boolean variables). For example:

```
IF Found
  THEN
    OUTPUT "Your search was successful"
  ELSE
    OUTPUT "Your search was unsuccessful"
ENDIF
```

» comparisons made by using comparison operators, where comparisons are made from left to right, for example: **A > B** means '**A** is greater than **B**' Comparisons can be simple or more complicated, for example:

```
IF ((Height > 1) OR (Weight > 20)) AND (Age < 70) AND
  (Age > 5)
  THEN
    OUTPUT "You can ride"
  ELSE
    OUTPUT "Too small, too young or too old"
ENDIF
```

Have a look at the algorithm below that checks if a percentage mark is valid and whether it is a pass or a fail. This makes use of two IF statements; the second **IF** statement is part of the first **ELSE** path. This is called a **nested IF.**

```
OUTPUT "Please enter a mark "
INPUT PercentageMark
IF PercentageMark < 0 OR PercentageMark > 100
  THEN
    OUTPUT "Invalid Mark"
  ELSE
    IF PercentageMark > 49
      THEN
        OUTPUT "Pass"
      ELSE
        OUTPUT "Fail"
    ENDIF
ENDIF
```

A rejected percentage mark must be either less than zero or greater than 100

This is a nested IF statement, shown clearly by the use of a second level of indentation. The percentage mark is only tested if it is in the correct range

**Table 7.2** Comparison operators

| Operator | Comparison |
|---|---|
| > | Greater than |
| < | Less than |
| = | Equal |
| >= | Greater than or equal |
| <= | Less than or equal |
| <> | Not equal |
| AND | Both |
| OR | Either |
| NOT | Not |

## CASE OF … OTHERWISE … ENDCASE

For a CASE statement the value of the variable decides the path to be taken. Several values are usually specified. **OTHERWISE**, is the path taken for all other values. The end of the statement is shown by **ENDCASE**

Have a look at the algorithm below that specifies what happens if the value of **Choice** is **1**, **2**, **3** or **4**.

```
CASE OF Choice
   1 : Answer ← Num1 + Num2
   2 : Answer ← Num1 - Num2
   3 : Answer ← Num1 * Num2
   4 : Answer ← Num1 / Num2
  OTHERWISE OUTPUT "Please enter a valid choice"
ENDCASE
```

## The pseudocode for iteration

When some actions performed as part of an algorithm need repeating this is called iteration. Loop structures are used to perform the iteration.

Pseudocode includes these three different types of loop structure:

| | |
|---|---|
| A set number of repetitions | **FOR … TO … NEXT** |
| A repetition, where the number of repeats is not known, that is completed at least once: | **REPEAT … UNTIL** |
| A repetition, where the number of repeats is not known, that may never be completed: | **WHILE … DO … ENDWHILE** |

All types of loops can all perform the same task, for example displaying ten stars:

```
FOR Counter ← 1 TO 10
    OUTPUT "*"
NEXT Counter
```

A FOR … NEXT loop

```
Counter ← 0
REPEAT
    OUTPUT "*"
    Counter ← Counter + 1
UNTIL Counter > 9
```

A REPEAT … UNTIL loop

```
Counter ← 0
WHILE Counter < 10 DO
    OUTPUT "*"
    Counter ← Counter + 1
ENDWHILE
```

A WHILE … DO … ENDWHILE loop

As you can see, the **FOR … TO … NEXT** loop is the most efficient way for a programmer to write this type of task as the loop counter is automatically managed.

### FOR … TO … NEXT loops

A variable is set up, with a start value and an end value, this variable is incremented in steps of one until the end value is reached and the iteration finishes. The variable can be used within the loop so long as its value is not changed. This type of loop is very useful for reading values into lists with a known length.

```
FOR Counter ← 1 TO 10
    OUTPUT "Enter Name of Student "
    INPUT StudentName[Counter]
NEXT
```

Counter starts at 1 and finishes at 10

Array (see Chapter 8) items `StudentName[1]` to `StudentName[10]` have data input

### REPEAT … UNTIL loop

This loop structure is used when the number of repetitions/iterations is not known, and the actions are repeated **UNTIL** a given condition becomes true. The actions in this loop are always completed at least once. This is a post-condition loop as the test for exiting the loop is at the end of the loop.

```
Total ← 0
Mark ← 0
REPEAT
    Total ← Total + Mark
    OUTPUT "Enter value for mark, -1 to finish "
    INPUT Mark
UNTIL Mark = -1
```

Variables Total and Mark are both initialised to zero.

At least one mark is entered.

## WHILE … DO … ENDWHILE loop

This loop structure is used when the number of repetitions/iterations is not known, and the actions are only repeated **WHILE** a given condition is true. If the WHILE condition is untrue then the actions in this loop are never performed. This is a pre-condition loop as the test for exiting the loop is at the beginning of the loop.

```
Total ← 0
OUTPUT "Enter value for mark, -1 to finish "
INPUT Mark
WHILE Mark <> -1 DO
    Total ← Total + Mark
    OUTPUT "Enter value for mark, -1 to finish"
    INPUT Mark
ENDWHILE
```

Only the variable Total is initialised to zero

Condition tested at start of loop

## The pseudocode for input and output statements

**INPUT** and **OUTPUT** are used for the entry of data and display of information. Sometimes **READ** can be used instead of **INPUT** but this is usually used for reading from files – see Chapter 8. Also, PRINT can be used instead of **OUTPUT** if a hard copy is required.

**INPUT** is used for data entry; it is usually followed by a variable where the data input is stored, for example:

```
INPUT Name
```

```
INPUT StudentMark
```

**OUTPUT** is used to display information either on a screen or printed on paper; it is usually followed by a single value that is a string or a variable, or a list of values separated by commas, for example:

```
OUTPUT Name
```

```
OUTPUT "Your name is ", Name
```

```
OUTPUT Name1, "Ali", Name3
```

## 7.3 Explaining the purpose of an algorithm

An algorithm sets out the steps to complete a given task. This is usually shown as a flowchart or pseudocode, so that the purpose of the task and the processes needed to complete it are clear to those who study it. You will be able to practise this skill as you become more familiar with writing and finding and correcting errors in algorithms.

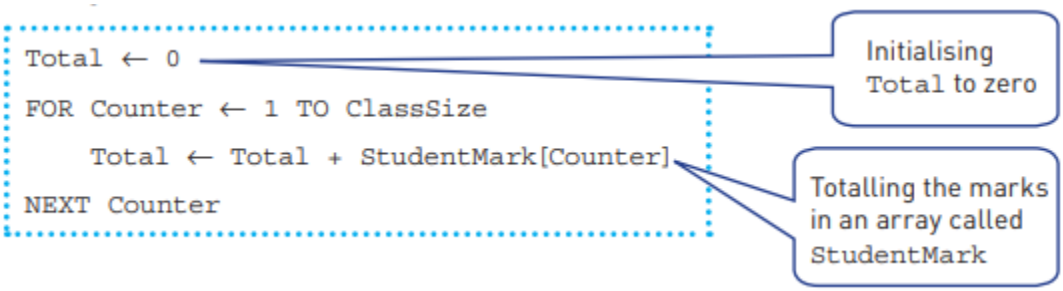
## 7.4 Standard methods of solution

The ability to repeat existing methods is very important in the design of algorithms; when an algorithm is turned into a program the same methods may be repeated many thousands of times.

You need to be able to use and understand these standard methods used in algorithms:
» Totaling
» Counting
» Finding maximum, minimum, and average (mean) values
» Searching using a linear search
» Sorting using a bubble sort.

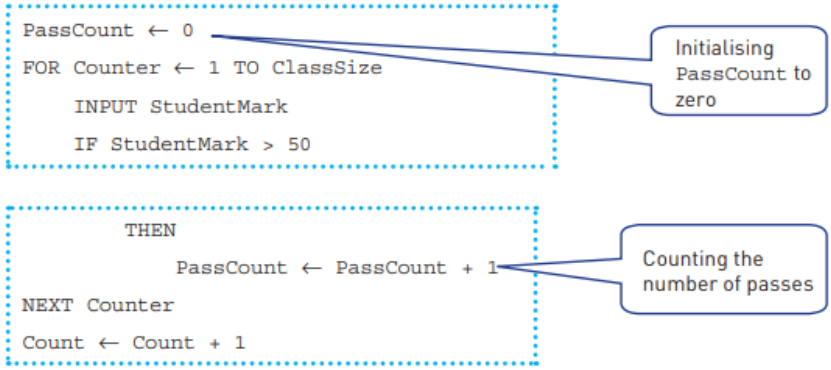
### 7.4.1 Totaling

Totaling means keeping a total that values are added to.




### 7.4.2 Counting

Keeping a count of the number of times an action is performed is another standard method. For example, counting the number of students that were awarded a pass mark.




Counting is also used to count down until a certain value is reached, for example, checking the number of items in stock in a supermarket:

```
:
NumberInStock ← NumberInStock - 1
IF NumberInStock < 20
    THEN
        CALL Reorder()
:
```

Counting down items in stock

## 7.4.3 Maximum, minimum and average

Finding the largest and smallest values in a list are two standard methods that are frequently found in algorithms, for example, finding the highest and lowest mark awarded to a class of students.

```
MaximumMark ← 0
MinimumMark ← 100
FOR Counter ← 1 TO ClassSize
  IF StudentMark[Counter] > MaximumMark
    THEN
      MaximumMark ← StudentMark[Counter]
  ENDIF
  IF StudentMark[Counter] < MinimumMark
    THEN
      MinimumMark ← StudentMark[Counter]
  ENDIF
NEXT Counter
```

Initialising maximum to the lowest mark possible

Initialising minimum to the highest possible

Calls data from an array (see Chapter 8) called StudentMark)

Replacing the maximum mark with a higher mark

Replacing the minimum mark with a lower mark

If the largest and smallest values are not known, an alternative method is to set the maximum and minimum values to the first item in the list. For example, using this method to find the highest and lowest mark awarded to a class of students.

```
MaximumMark ← StudentMark[1]
MinimumMark ← StudentMark[1]
FOR Counter ← 2 TO ClassSize
  IF StudentMark[Counter] > MaximumMark
    THEN
      MaximumMark ← StudentMark[Counter]
  ENDIF
  IF StudentMark[Counter] < MinimumMark
    THEN
      MinimumMark ← StudentMark[Counter]
  ENDIF
NEXT Counter
```

Initialising minimum and maximum to the first mark

Starting the loop at the second position in the list.

Calculating the average (mean) of all the values in a list is an extension of the totalling method, for example, calculating the average mark for a class of students

```
Total ← 0
FOR Counter ← 1 TO ClassSize
  Total ← Total + StudentMark[Counter]
NEXT Counter
Average ← Total / ClassSize
```

Calculating the average from the total after the loop has been completed

## 7.4.4 Linear search

A search is used to check if a value is stored in a list, performed by systematically working through the items in the list. There are several standard search methods, but you only need to understand one method for IGCSE Computer Science. This is called a **linear search.**

For example, searching for a name in a class list of student names, where all the names stored are different:

```
OUTPUT "Please enter name to find "
INPUT Name
Found ← FALSE
Counter ← 1
REPEAT
  IF Name = StudentName[Counter]
    THEN
      Found ← TRUE
    ELSE
      Counter ← Counter + 1
  ENDIF
UNTIL Found OR Counter > ClassSize
IF Found
  THEN
    OUTPUT Name, " found at position ",
Counter, " in the list."
  ELSE
    OUTPUT Name, " not found."
ENDIF
```

Setting a variable, Found, as a flag, using TRUE and FALSE to indicate if the name has been found or not

Checking if the name input matches a name in the list

Setting the flag to TRUE when a match is found

Stopping the search when a match is found or the whole list has been searched

Outputting the position in the list when a match is found

In this example, the search checks how many people chose ice cream as their favourite dessert, where several values in the list can be the same.

```
ChoiceCount ← 0
FOR Counter ← 1 TO Length
IF "ice cream" = Dessert[Counter]
  THEN
    ChoiceCount ← ChoiceCount + 1
NEXT Counter
OUTPUT ChoiceCount, " chose ice cream
as their favourite dessert."
```

Checking ice cream has been chosen

Checking every item in the list

## 7.4.5 Bubble sort

Sorting is a crucial process in organizing lists in a meaningful order, such as alphabetical or ascending/descending order, which can enhance their usefulness, especially in IGCSE Computer Science. This method of sorting is called a **bubble sort**.

## 7.5 Validation and verification

Computer systems use two methods for data entry: validation and coding. Validation ensures reasonable data input and prevents data changes during entry, while coding ensures accurate input.

## 7.5.1 Validation

Validation is the automated checking by a program that data is reasonable before it is accepted into a computer system. When data is validated by a computer system, if the data is rejected a message should be output explaining why the data was rejected and another opportunity given to enter the data.

There are many different types of validation checks including:
» Range checks
» Length checks
» Type checks
» Presence checks
» Format checks
» Check digits.

### Range check

A **range check** checks that the value of a number is between an upper value and a lower value. For example, checking that percentage marks are between 0 and 100 inclusive:

```
OUTPUT "Please enter the student's mark "
REPEAT
INPUT StudentMark
IF StudentMark < 0 OR StudentMark > 100
  THEN
    OUTPUT "The student's mark should be in the range
            0 to 100, please re-enter the mark "
ENDIF
UNTIL StudentMark >= 0 AND StudentMark <= 100
```

### Length check

A length check checks either:

» That data contains an exact number of characters, for example that a password must be exactly eight characters in length so that passwords with seven or fewer characters or nine or more characters would be rejected, for instance:

Password has a data type of string and LENGTH is the pseudocode operation that returns a whole number showing the number of characters in the string

```
OUTPUT "Please enter your password of eight
characters "
REPEAT
    INPUT Password
    IF LENGTH(Password) <> 8
      THEN
        OUTPUT "Your password must be exactly eight
                characters, please re-enter "
    ENDIF
UNTIL LENGTH(Password) = 8
```

» or that the data entered is a reasonable number of characters, for example, a family name could be between two and thirty characters inclusive so that names with one character or thirty-one or more characters would be rejected.

### Type check

A type check checks that the data entered is of a given data type, for example, that the number of brothers or sisters would be an integer (whole number).

```
OUTPUT "How many brothers do you have? "
REPEAT
INPUT NumberOfBrothers
IF NumberOfBrothers <> DIV(NumberOfBrothers, 1)
  THEN
    OUTPUT "This must be a whole number, please re-enter"
ENDIF
UNTIL NumberOfBrothers = DIV(NumberOfBrothers, 1)
```

### Presence check

A presence check checks to ensure that some data has been entered and the value has not been left blank, for example, an email address for an online transaction must be completed.

```
OUTPUT "Please enter your email address "
REPEAT
INPUT EmailAddress
IF EmailAddress = ""
  THEN
    OUTPUT "*=Required "
ENDIF
UNTIL EmailAddress <> ""
```

### Format check and check digit

A format check checks that the characters entered conform to a pre-defined pattern,

A check digit is the final digit included in a code; it is calculated from all the other digits in the code. Check digits are used for barcodes, product codes, International Standard Book Numbers (ISBN) and Vehicle Identification Numbers (VIN).

Check digits are used to identify errors in data entry caused by mistyping or mis-scanning a barcode. They can usually detect the following types of error:

» An incorrect digit entered, for example, 5327 entered instead of 5307
» Transposition errors where two numbers have changed order for example 5037 instead of 5307
» Omitted or extra digits, for example, 537 instead of 5307 or 53107 instead of 5307
» Phonetic errors, for example, 13, thirteen, instead of 30, thirty.

## 7.5.2 Verification

**Verification** is checking that data has been accurately copied from one source to another – for instance, input into a computer or transferred from one part of a computer system to another. Verification methods for input data include:
» Double entry
» Screen/visual check.
For **double entry** the data is entered twice, sometimes by different operators

A **screen/visual check** is a manual check completed by the user who is entering the data. When the data entry is complete the data is displayed on the screen and the user is asked to confirm that it is correct before continuing.

# 7.6 Test data

## 7.6.1 How to suggest and apply suitable test data

Test data is essential for ensuring a solution's functionality. A **set of test data** includes all necessary data items for a solution to function. Different sets of test data may be needed to thoroughly test a solution, as demonstrated in Activity 7.6 using 7 and 18 items.

**Normal data** is a set of test data used to verify program or algorithm solutions' functionality, ensuring they work as expected and produce expected results.

Solutions also need to be tested to prove that they do not do what they are supposed not to do. In order to do this, test data should be chosen that would be rejected by the solution as not suitable, if the solution is working properly. This type of test data is called **abnormal test data**. (It is also sometimes called **erroneous test data.)**

When testing algorithms with numerical values, sometimes only a given range of values should be allowed. For example, percentage marks should only be in the range 0 to 100. Our algorithm above should be tested with **extreme data**

There is another type of test data called **boundary data**. This is used to establish where the largest and smallest values occur.

## 7.7 Trace tables to document dry runs of algorithms

A thorough structured approach is required to find out the purpose of an algorithm. This involves recording and studying the results from each step in the algorithm and requires the use of test data.

A **trace table** can be used to record the results from each step in an algorithm; it is used to record the value of an item (variable) each time that it changes. The manual exercise of working through an algorithm step by step is called a **dry run.**

## 7.8 Identifying errors in algorithms

Trace tables and test data can be used to identify and correct errors. Your completed trace table for Activity 7.14 should look like this:

▼ **Table 7.6** Completed trace table for flowchart

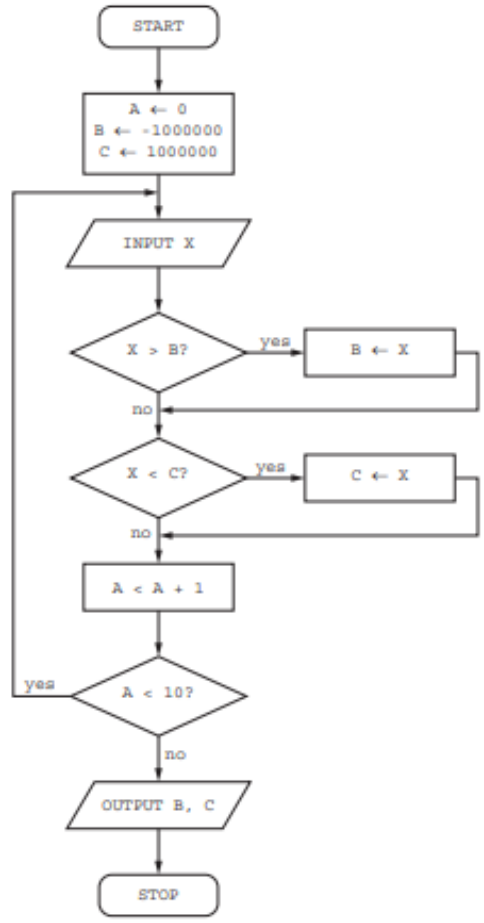
| A | B | C | X | OUTPUT |
|---|---|---|---|---|
| 0 | 0 | 100 | | |
| 1 | 400 | | 400 | |
| 2 | 800 | | 800 | |
| 3 | | | 190 | |
| 4 | | | 170 | |
| 5 | | | 300 | |
| 6 | | | 110 | |
| 7 | | | 600 | |
| 8 | | | 150 | |
| 9 | | | 130 | |
| 10 | 900 | | 900 | |
| | | | | 900 100 |

There is an error as the smallest number, 110, has not been identified.

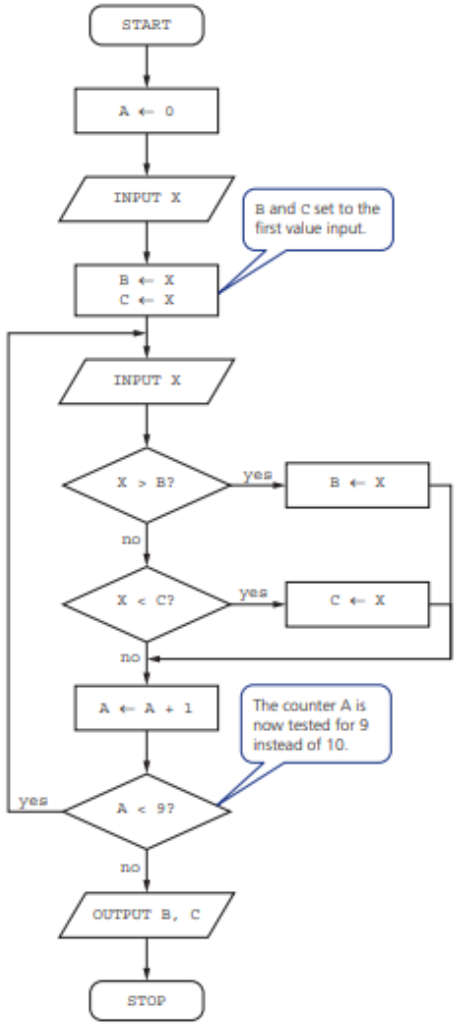
As this algorithm only works for numbers between 0 and 100; a better algorithm could look like this:

This algorithm is very similar and works for a much larger range of numbers, but it still does not work for every set of numbers

In order to work for any set of numbers, the algorithm needs to be re-written to allow the largest and smallest numbers to be tested against numbers that appear in any list provided. The provisional values set at the start of the algorithm need to be chosen from the list. A standard method is to set both these provisional values to the value of the first item input.




▲ **Figure 7.16** A better algorithm




▲ **Figure 7.17** A much better algorithm

## 7.9 Writing and amending algorithms

There are a number of stages when producing an algorithm for a given problem:

1. Make sure that the problem is clearly specified – the purpose of the algorithm and the tasks to be completed by the algorithm

2. Break the problem down in to sub-problems; if it is complex, you may want to consider writing an algorithm for each sub-problem.

3. Decide on how any data is to be obtained and stored, what is going to happen to the data and how any results are going to be displayed.

4. Design the structure of your algorithm using a structure diagram.

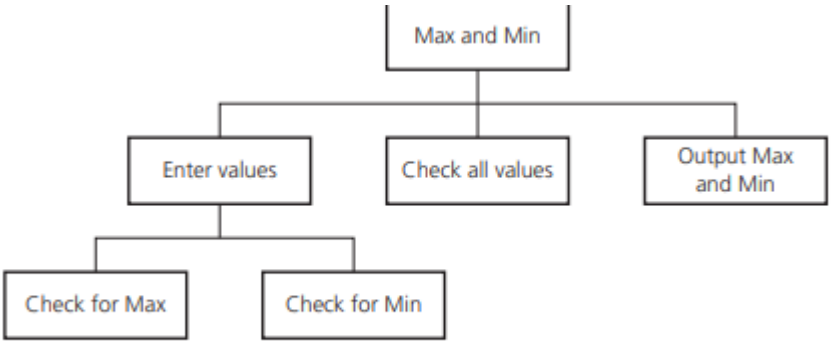
5. Decide on how you are going to construct your algorithm, either using a flowchart or pseudocode.

6. The text emphasizes the importance of creating an algorithm that is easily read and understood, requiring precision in setting out conditions, using meaningful names for data stores, and considering the problem being solved, rather than just focusing on readability.

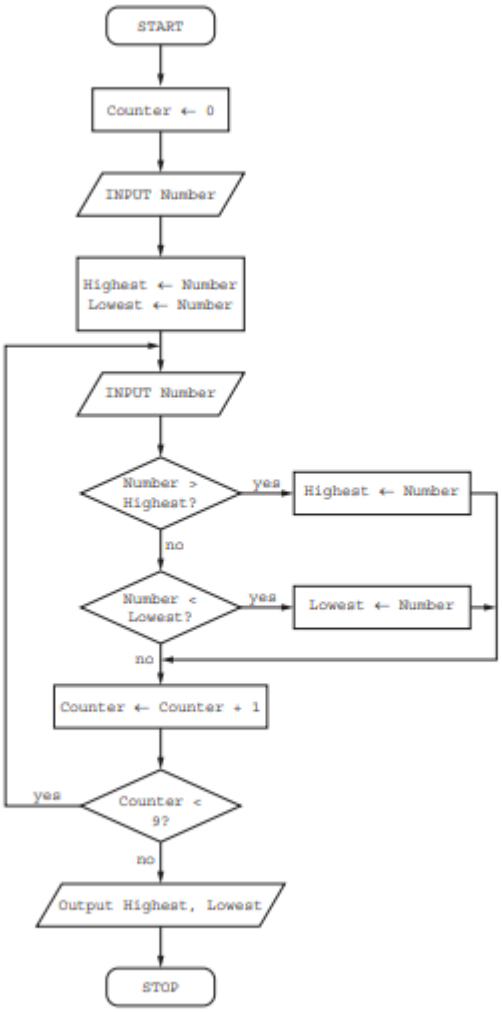
7. Use several sets of test data (Normal, Abnormal and Boundary) to dry run your algorithm and show the results in trace tables, to enable you to find any errors

8. If any errors are found, correct them and repeat the process until you think that your algorithm works perfectly.

Have a look at this structure diagram and flowchart for the algorithm to select the largest, Max, and smallest, Min, numbers from a list of ten numbers. This time the flowchart is more easily readable than the structure chart




▲ **Figure 7.18** Structure chart for Max and Min




▲ **Figure 7.19** A more easily understandable flowchart for Max and Min

## Key terms used throughout this chapter

**analysis** – part of the program development life cycle; a process of investigation, leading to the specification of what a program is required to do

**design** – part of the program development life cycle; uses the program specification from the analysis stage to show to how the program should be developed

**coding** – part of the program development life cycle; the writing of the program or suite of programs

**testing** – part of the program development life cycle; systematic checks done on a program to make sure that it works under all conditions

**abstraction** – a method used in the analysis stage of the program development life cycle; the key elements required for the solution to the problem are kept and any unnecessary details and information that are not required are discarded

**decomposition** – a method used in the analysis stage of the program development life cycle; a complex problem is broken down into smaller parts, which can then be sub divided into even smaller parts that can be solved more easily

**top-down design** – the breaking down of a computer system into a set of sub-systems, then breaking each sub-system down into a set of smaller sub-systems, until each sub-system just performs a single action

**inputs** – the data used by the system that needs to be entered while the system is active

**processes** – the tasks that need to be performed by a program using the input data and any other previously stored data

**output** – information that needs to be displayed or printed for the users of the system

**storage** – data that needs to be stored in files on an appropriate media for use in the future

**structure diagram** – a diagram that shows the design of a computer system in a hierarchical way, with each level giving a more detailed breakdown of the system into sub-systems

**flowchart** – a diagram that shows the steps required for a task (sub-system) and the order in which the steps are to be performed

**algorithm** – an ordered set of steps to solve a problem

**pseudocode** – a simple method of showing an algorithm; it describes what the algorithm does by using English key words that are very similar to those used in a high-level programming language but without the strict syntax rules

**linear search** – an algorithm that inspects each item in a list in turn to see if the item matches the value searched for

**bubble sort** – an algorithm that makes multiple passes through a list comparing each element with the next element and swapping them. This continues until there is a pass where no more swaps are made

**validation** – automated checks carried out by a program that data is reasonable before it is accepted into a computer system

**verification** – checking that data has been accurately copied from another source and input into a computer or transferred from one part of a computer system to another

**set of test data** – all the items of data required to work through a solution

**normal data** – data that is accepted by a program

**abnormal data** – data that is rejected by a program

**extreme data** – the largest/smallest data value that is accepted by a program

**boundary data** – the largest/smallest data value that is accepted by a program and the corresponding smallest/largest rejected data value

**range check** – a check that the value of a number is between an upper value and a lower value

**length check** – a method used to check that the data entered is a specific number of characters long or that the number of characters is between an upper value and a lower value

**type check** – a check that the data entered is of a specific type

**presence check** – a check that a data item has been entered

**format check** – a check that the characters entered conform to a pre-defined pattern

**check digit** – an additional digit appended to a number to check if the entered number is error-free; check digit is a data entry check and not a data transmission check

# Revision questions

1). (2023 Nov: 6)

```
01 DECLARE A[1:10] : STRING
02 DECLARE T : STRING
03 DECLARE C, L : INTEGER
04 L ← 10
05 FOR C ← 1 TO L
06    OUTPUT "Please enter name "
07    INPUT A[C]
08 NEXT C
09 FOR C ← 1 TO L
10    FOR L ← 1 TO 9
11       IF A[L] > A[L + 1]
12          THEN
13             T ← A[L]
14             A[L] ← A[L + 1]
15             A[L + 1] ← T
16          ENDIF
17    NEXT L
18 NEXT C
19 FOR C ← 1 TO L
20    OUTPUT "Name ", C, " is ", A[C]
21 NEXT C
```

(a) State the purpose of this pseudocode algorithm. [1]

(b) State four processes in this algorithm. [4]

(c) Meaningful identifiers have not been used in this algorithm.

Suggest suitable meaningful identifiers for:

The array:

A

The variables:

T

C

L [3]

(d) State two other ways the algorithm can be made easier to understand and maintain.

[2]

2). (2023 Nov: 8)

8 A programmer is designing an algorithm to calculate the cost of a length of rope.

- The program requirements are:
- input two values: the length of rope in metres Length and the cost of one metre Cost
- perform a validation check on the length to ensure that the value is between 0.5 and 6.0 inclusive
- calculate the price Price
- output the price rounded to two decimal places.

Use the variable names given.

(a) State the name of the validation check. [1]
(b) Complete the flowchart for this algorithm.



[6] (c) Give two different sets of test data for this algorithm and state the purpose of each set.
Set 1
Purpose
Set 2
Purpose [4]
(d) Complete the headings for the trace table to show a dry-run for this algorithm.
You do not need to trace the algorithm.

| ..... | ..... | ..... | ..... |
|---|---|---|---|

[3]
(e) Describe an improvement that should be made to the requirements for this algorithm.
[2]

3). (2023 Nov: 10)
Drama students put on a performance of a play for one evening. Seats in a small theatre can be booked for this performance.

The theatre has 10 rows of 20 seats. The status of the seat bookings for the evening is held in the two-dimensional (2D) Boolean array Evening[ ]

Each element contains FALSE if the seat is available and TRUE if the seat is booked.

Up to and including four seats can be booked at one time. Seats are allocated in order from those available. A row or seat number cannot be requested.

The array Evening[ ] has already been set up and some data stored.

Write a program that meets the following requirements:
- counts and outputs the number of seats already booked for the evening
- allows the user to input the number of seats required
- validates the input
- checks if enough seats are available:
  - if they are available:

- changes the status of the seats
- outputs the row number and seat number for each seat booked

- if they are not available:
- outputs a message giving the number of seats left or 'House full' if the theatre is fully booked.

You must use pseudocode or program code and add comments to explain how your code works. You do not need to declare any arrays or variables; you may assume that this has already been done.

You do *not* need to initialise the data in the array Evening[ ]

All inputs and outputs must contain suitable messages. [15]

4) (2022 may: Paper 2:1) case study
In preparation for the examination candidates should attempt the following practical tasks by writing and testing a program or programs.

Friends of Seaview Pier is an organisation devoted to the restoration and upkeep of a pier in the town. A pier is a wooden structure that provides a walkway over the sea. The pier requires regular maintenance and the friends of the pier need to raise money for this purpose.

Members of Friends of Seaview Pier each pay $75 per year, as a contribution to the pier's running costs. This entitles them to free admission to the pier throughout the year. They can also volunteer to help run the pier, by working at the pier entrance gate, working in the gift shop, or painting and decorating.

To provide additional income, the pier's wooden planks can be sponsored. A brass plaque, which contains a short message of the sponsor's choice, is fitted to a plank on the pier, for a donation of $200.

Write and test a program or programs for the Friends of Seaview Pier:
- Your program or programs must include appropriate prompts for the entry of data. Data must be validated on entry.
- All outputs, including error messages, need to be set out clearly and understandably.
- All variables, constants and other identifiers must have meaningful names.

You will need to complete these three tasks. Each task must be fully tested.

**Task 1** - becoming a member of Friends of Seaview Pier

Set up a system to enable people to become members of Friends of Seaview Pier and for each new member enter:
- their first name and last name
- whether or not they wish to work as a volunteer
- if they choose to volunteer, identify the area from:
  - the pier entrance gate
  - the gift shop
  - painting and decorating
- the date of joining
- whether or not they have paid the $75 fee.

All of this information needs to be stored using suitable data structures.

**Task 2** - using the membership data

Extend the program in **Task 1** so that a list of the first and last names of members can be output in any of the following categories:
- Members who have chosen to work as volunteers.
- Volunteers who would like to work at the pier entrance gate.
- Volunteers who would like to work in the gift shop.
- Volunteers who would like to help with painting and decorating tasks.
- Members whose membership has expired (they have not re-joined this year).
- Members who have not yet paid their $75 fee.

**Task 3** - sponsoring a wooden plank

Add an additional option to the program in **Task 1** to enable the pier's wooden planks to be sponsored. Separate data structures should be used to store the names of the individuals and the short messages they would like to have written on their brass plaque. An output would display everything that was input for the sponsor to confirm. If errors are found, the program should allow data to be re-entered. Once complete, the data is stored and the sponsor is charged $200.

a). Describe the data structures you could have used in **Task 1**. Your description should include types of data structure, names used for data structures, their uses and examples of sample Data. [5]
b). Explain how you could change your program in **Task 1** to total all the money collected from new members who have paid. [3]
c). Describe how data input in **Task 1** could be validated to find out if a new member wants to work as a volunteer. [3]
d). Write an algorithm to show how your program completes Task 3 assuming the option to sponsor a wooden plank has been chosen, using pseudocode, programming statements or a flowchart. Details completed in Task 1 are not required. [5]
e). Explain how your program allows any one of the member or volunteer lists to be selected and displayed (part of Task 2). Any programming statements used in your answer must be fully Explained. [4]

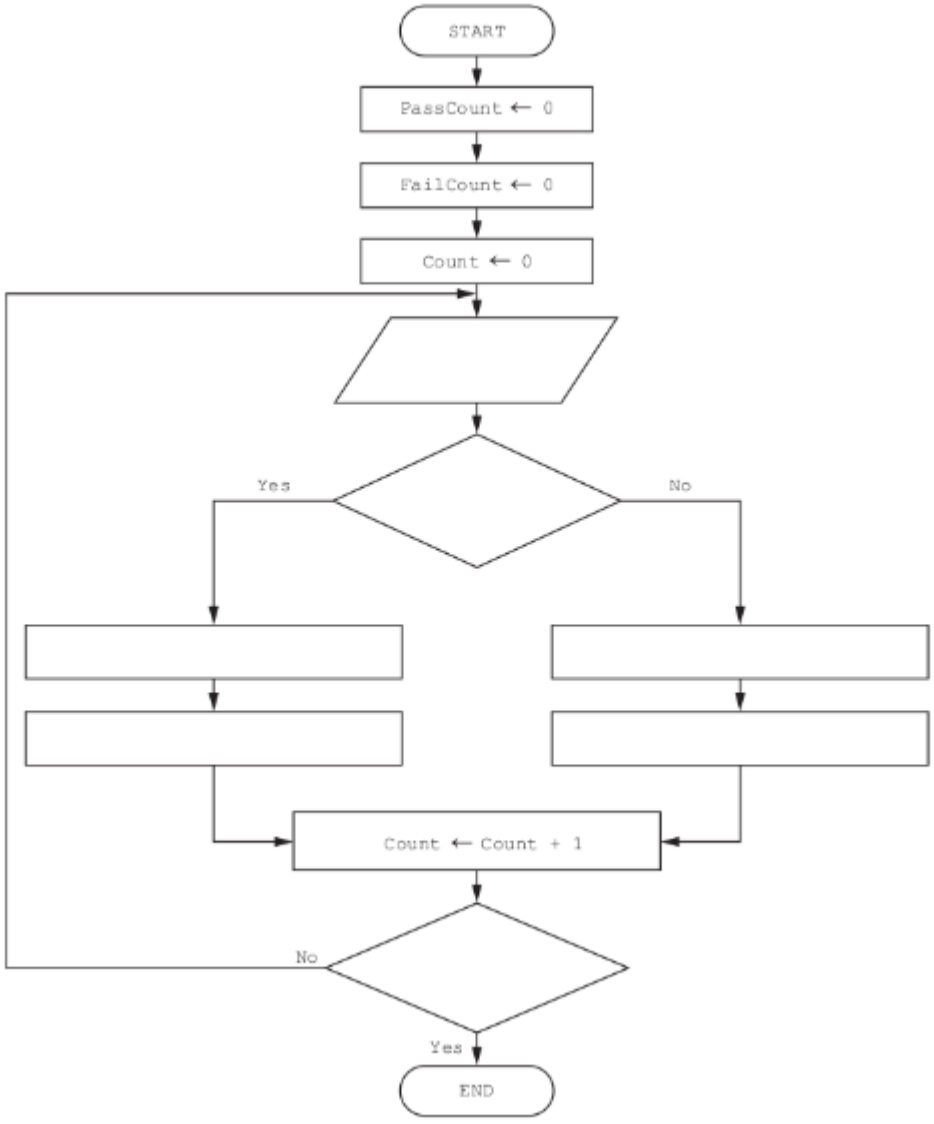
5). (2022 may:2)
Tick (✓) one box in each row to identify the most appropriate data type for each description. Only one tick (✓) per column
[4]

| Description | Data Type | | | | |
|---|---|---|---|---|---|
| | Boolean | char | Integer | Real | String |
| a single character from the keyboard | | | | | |
| multiple characters from the keyboard | | | | | |
| only one of two possible values | | | | | |
| only whole numbers | | | | | |
| any number | | | | | |

6). (2022 may:3)
Give one piece of normal test data and one piece of erroneous test data that could be used to

validate the input of an email address.
State the reason for your choice in each case.
Normal test data……
Reason……
Erroneous test data……
Reason……
[4]

7). (2022 may:4)
The flowchart shows an algorithm that should allow 60 test results to be entered into the variable Score. Each test result is checked to see if it is 50 or more. If it is, the test result is assigned to the Pass array. Otherwise, it is assigned to the Fail array.
(a) Complete this flowchart:




[6]
(b) Write a pseudocode routine that will check that each test result entered into the algorithm is

between 0 and 100 inclusive.

7). (2022 may:5)
The pseudocode represents an algorithm.
The pre-defined function DIV gives the value of the result of integer division.
For example, Y = 9 DIV 4 gives the value Y = 2
The pre-defined function MOD gives the value of the remainder of integer division.
For example, R = 9 MOD 4 gives the value R = 1

```
First ← 0
Last ← 0
INPUT Limit
FOR Counter ← 1 TO Limit
  INPUT Value
  IF Value >= 100
    THEN
      IF Value < 1000
        THEN
          First ← Value DIV 100
          Last ← Value MOD 10
            IF First = Last
              THEN
                OUTPUT Value
            ENDIF
      ENDIF
  ENDIF
NEXT Counter
```

(a) Complete the trace table for the algorithm using this input data:
8, 66, 606, 6226, 8448, 642, 747, 77, 121

| Counter | Value | First | Last | Limit | OUTPUT |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |

[5]
(b) Describe the purpose of the algorithm.[2]

8). (2021 Paper2: 1) Case Study
A program is needed for a quiz to help younger students to practise their multiplication tables. There needs to be two ways of using the quiz; testing and learning.
Testing: the student is given one attempt at answering each question and the score is calculated for the whole test.
Learning: the student is given up to three attempts to get their answer to each question correct. There is no scoring.
A student can choose which multiplication table, from 2 to 12, to use for the quiz. There are five questions in each quiz, each question must use the chosen multiplication table and a different whole number (from 1 to 12) as the multiplier.
Write and test a program or programs for a multiplication tables quiz.
* Your program or programs must include appropriate prompts for the entry of data; data must be validated on entry.

* Error messages and other output need to be set out clearly and understandably.
* All variables, constants and other identifiers must have meaningful names.

You will need to complete these three tasks. Each task must be fully tested.

Task 1 - Testing a student

Students enter their name and choice of multiplication table. Each question is displayed on the screen one at a time, for example:

```
Question 1

2 X 7 =
```

Students enter their answer and move on to the next question. A running total of correct answers (score) is kept. At the end of the quiz the student's name and score are displayed with a personalised message related to the score, for example:

```
Aarav your score is 5/5

Well done full marks
```

```
Diya your score is 3/5

Have another practice
```

,

Task 2 - Student learning

Students enter their name and choice of multiplication table. Each question is displayed on the screen as in Task 1. If an answer is correct, a personalised message containing the student's name confirms this, the quiz then moves to the next question. If an answer is incorrect, a personalised message containing the student's name and a hint is displayed, for example:

```
Aarav your answer is too large
```

Up to three attempts are offered to get each answer correct. After the third incorrect attempt, the correct answer is displayed and the quiz moves on to the next question.

Task 3 - Varying the quiz

Modify Task 1 to allow students to choose how many questions they would like in the test and if they would like a 'mixed' set of questions. A 'mixed' set means that each question can be from a different multiplication table; from 2 to 12.

All variables, constants and other identifiers must have meaningful names.

(a) Identify the variable that you used to store the student's answer in Task 1. Give the most appropriate data type for this variable. Explain how your program ensured that any data entered for the answer was valid.

Variable ___

Data type _____

Validation _____

[4]

(b) Identify and give the data type of a different variable, that you could have used in Task 2. State the use of this variable in Task 2.

Variable _____

Data type _____

Use _____

[3]

(c) Write an algorithm for Task 1, using either pseudocode, programming statements or a Flowchart. [6]

(d) Explain how your program completed Task 3.
Include any programming statements that you have added to Task 1 and fully explain the purpose of each statement. [4]

(e) Explain how you could alter Task 1 to change the quiz to:

- display three alternative answers for each question
- allow the student to choose one of these answers

[3]

9). (2021 Paper2: 2)
An algorithm has been written in pseudocode to:
- input 25 positive whole numbers less than 100
- find and output the largest number
- find and output the average of all the numbers

```
01  A ← 0
02  B ← 0
03  C ← 0
04  REPEAT
05    REPEAT
06      INPUT D
07    UNTIL D > 0 AND D < 100 AND D = INT(D)
08    IF D > B
09      THEN
10        B ← D
11    ENDIF
12    C ← C + D
13    A ← A + 1
14  UNTIL A >= 25
15  E ← C / A
16  OUTPUT "Largest number is ", B
17  OUTPUT "Average is ", E
```

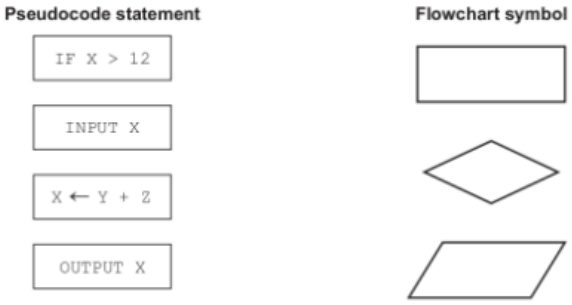
(a) Give the line number for the statements showing:
Totalling _____
Counting _____
Range check _____
Calculating the average _____ [4]

(b) State an example for each type of test data needed to test the input of the number:
Normal test data example _____
Erroneous/abnormal test data example _____
Extreme test data example _____
[3]

(c) The algorithm needs to be changed to include finding and outputting the smallest number input.
Describe how you would change the algorithm.

10). (2021 Paper2: 3)
Four pseudocode statements and three flowchart symbols are shown.

Draw a line from each pseudocode statement to its correct flowchart symbol.




11). (2021 Paper2: 4)

This algorithm accepts weights of bags of cookies. Any cookie bag weighing between 0.9 and 1.1 kilograms inclusive is acceptable. Underweight bags weigh less than 0.9 kilograms and overweight bags weigh more than 1.1 kilograms. An input of a negative number stops the process. Then the total number of bags, the number of overweight bags and the number of underweight bags weighed are output.

```
Accept ← 0
Over ← 0
Under ← 0
OUTPUT "Enter weight of first cookie bag"
INPUT BagWeight
WHILE BagWeight > 0
    IF BagWeight > 1.1
      THEN
        Error ← 1
      ELSE
        IF BagWeight < 0.9
          THEN
            Error ← 2
          ELSE
            Error ← 0
        ENDIF
    ENDIF
    CASE Error OF
      0 : Accept ← Accept + 1
      1 : Over ← Over + 1
      2 : Under ← Under + 1
    ENDCASE
  OUTPUT "Weight of next bag?"
  INPUT BagWeight
ENDWHILE
Total ← Accept - Over - Under
OUTPUT "Number of bags weighed ", Total
OUTPUT "Number overweight ", Over
OUTPUT "Number underweight ", Under
```

(a) Complete a trace table for the given algorithm using this input data: [7]

1.05, 0.99, 1.2, 0.85, 1.1, 0.9, 1.5, 0.95, 1.05, 1.00, 1.07, 0.89, –10

| BagWeight | Accept | Over | Under | Error | Total | OUTPUT |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |

(b) There is an error in this algorithm.

Identify the error and write the corrected pseudocode statement.

Error .....

Correction .....

[2]