AS and A LEVEL
Information Technology
9626

Chapter 4

Algorithms and Flowcharts

Faisal Chughtai
info.sirfaisal@gmail.com
www.faisalchughtai.com

Algorithms

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.

Flowcharts

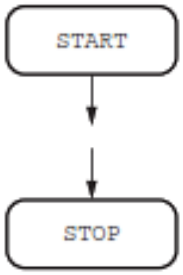
A flowchart shows diagrammatically the steps required to complete a task and the order that they are to be performed.

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.

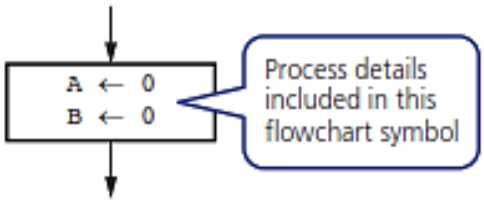
Begin/End

Terminator flowchart symbols are used at the beginning and end of each flowchart.



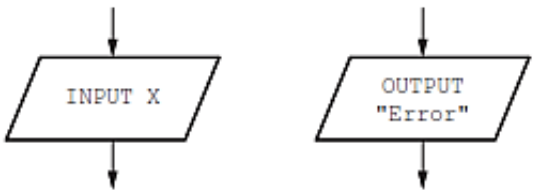
Process

Process flowchart symbols are used to show actions, for example, when values are assigned to variables.



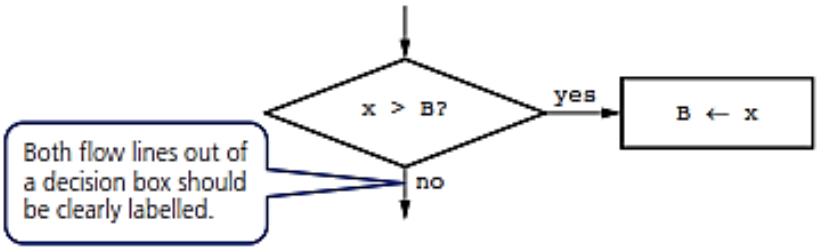
Input and output

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



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.



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.

Keywords
Input: putting data into an algorithm **Output:** displaying data from an algorithm to the user **Process:** an action performed to some data to make a change **Decision:** a comparison is used to decide if code is run, or not

Input and output

Worked example

An algorithm needs to ask a user to enter their name. It should take their name as an input and then welcome them by name, for example, "Welcome Sasha".

Draw a flowchart for the algorithm.

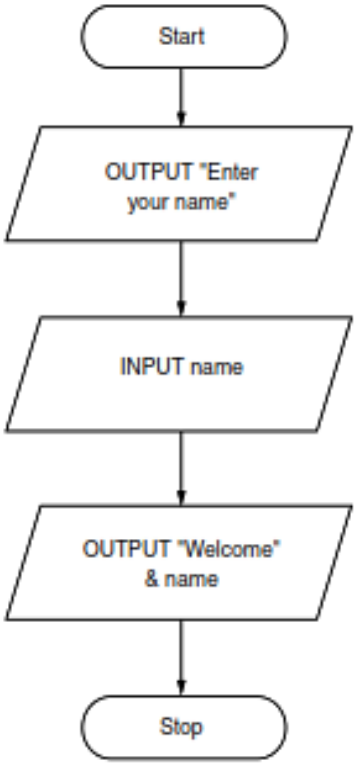
Start by identifying the steps required.

Step 1: Ask them to enter their name.
Step 2: Take their name as input.
Step 3: Welcome them by their name.

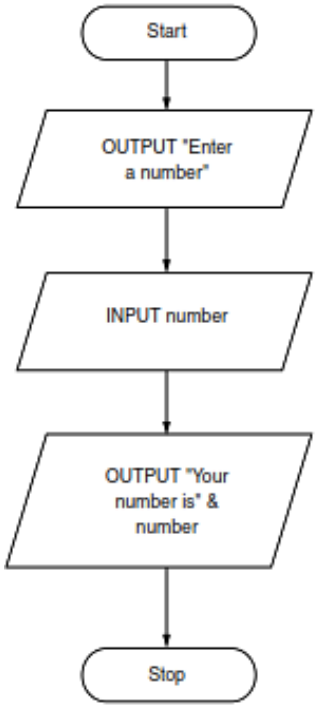
Then identify what type of symbols these steps need.

Step 1: Ask them to enter their name.
Output
Step 2: Take their name as input.
Input and store in variable
Step 3: Welcome them by their name.
Output including variable

Draw the flowchart following the steps identified.



Flowchart example: Enter a number



Processes

A process is an action that is performed. In a program it usually means that a change is being made to something.

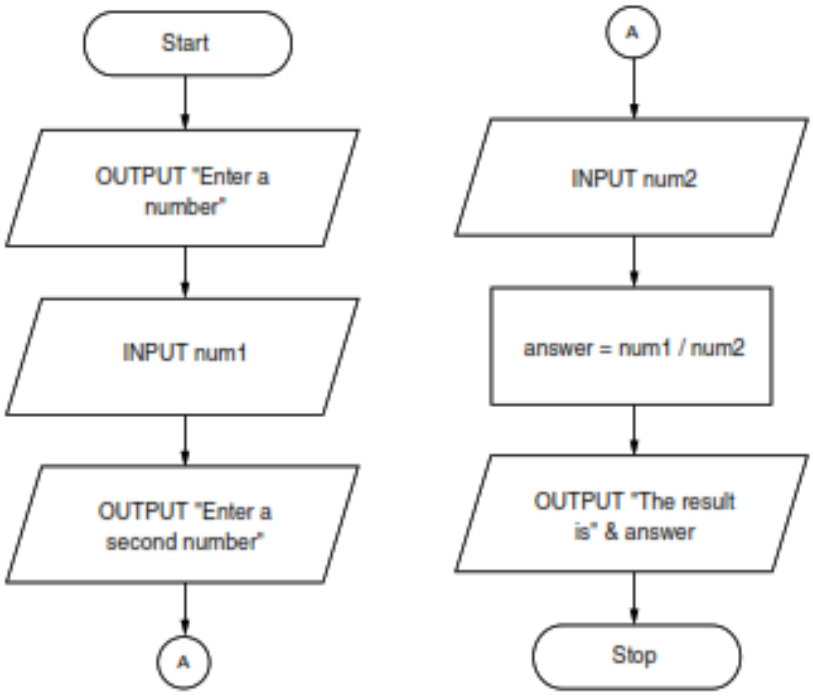
If you have a number stored in a variable, you could change this; you might add something to it, or subtract from it.

There are different types of processes.

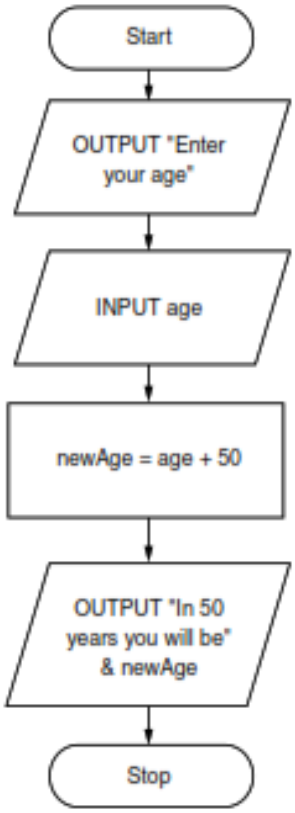
The table shows the different mathematical processes that be performed.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power of
MOD	Modulus division (gives the remainder)
DIV	Integer division (gives the whole number)

Flowchart example: Arithmetic calculation



Flowchart example: New age calculator



Decisions

A decision is also known as comparison and selection. It is a statement that involves a comparison and is either true or false.

If the result is true, then some code is run, if it is false then either different code is run or some code is skipped.

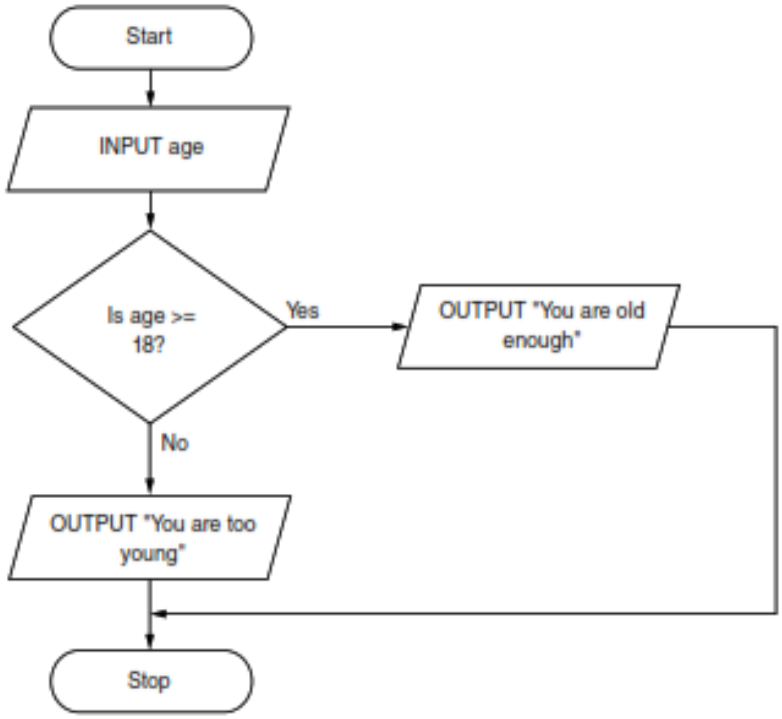
A comparison needs two sides of the argument, and a comparison symbol.

The following table shows common comparison operators.

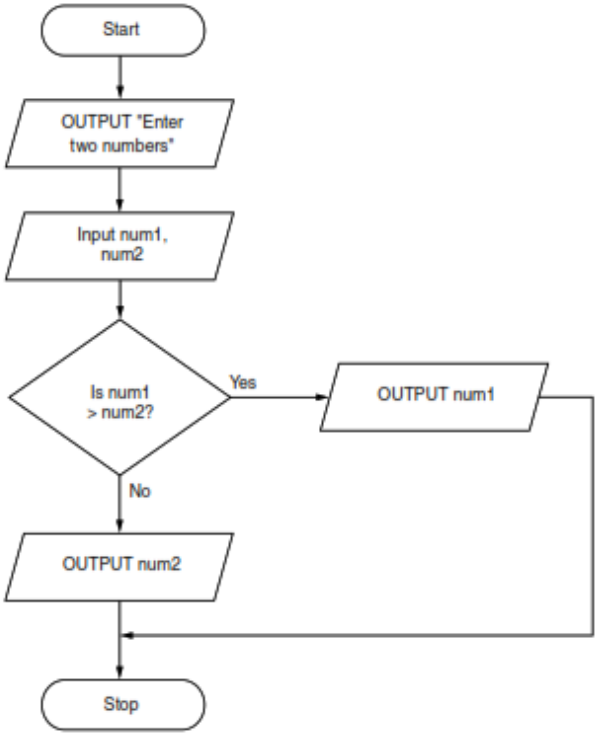
Operator	Description	Example
>	Greater than	10 > 5 is true 5 > 10 is false
<	Less than	10 < 5 is false 5 < 10 is true
>=	Greater than or equal to	10 >= 5 is true 10 >= 10 is true 5 >= 10 is false
<=	Less than or equal to	10 <= 5 is false 10 <= 10 is true 5 <= 10 is true
=	Equals to	10 = 5 is false

		10 = 10 is true
!= Or <>	Nor equal to	10 != 10 is false 5 <> 10 is true

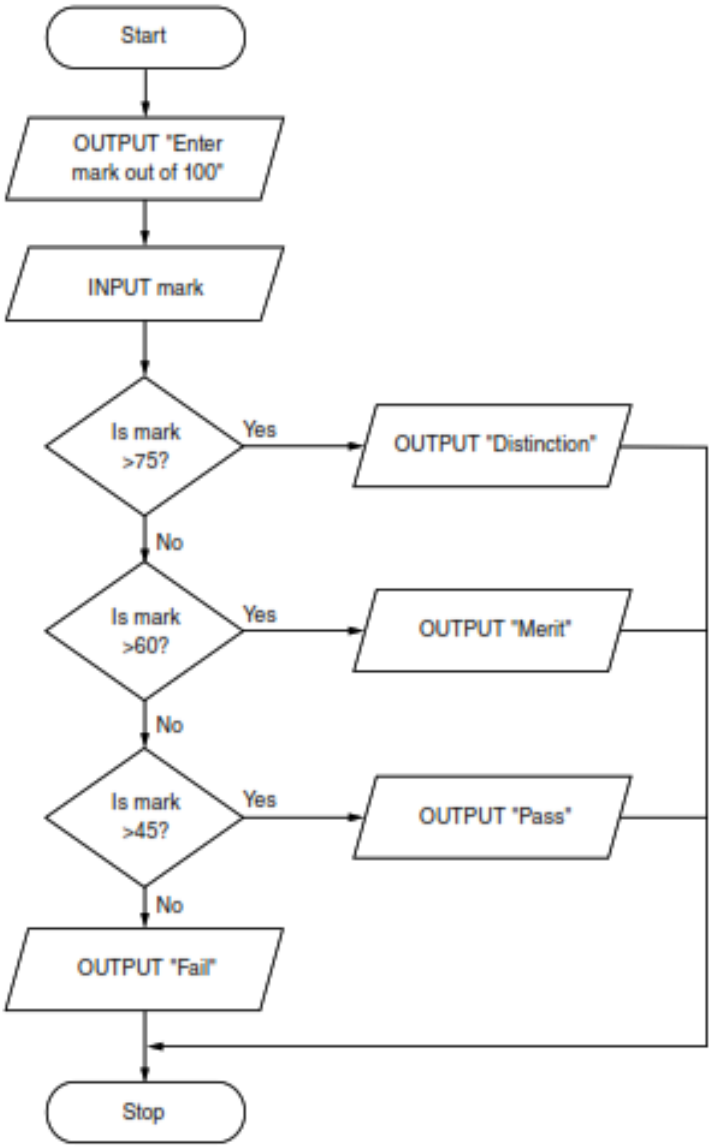
Flowchart example: Comparison and selection



Flowchart example: Compare two numbers



Flowchart example: Multiple selection



Loops

A loop means repetition; doing the same thing several times. It is also called iteration.

Iteration: a loop to repeat a section of code for a fixed number of times or until a required outcome is achieved.

Count-controlled loop: a loop where you know the number of times it will run.

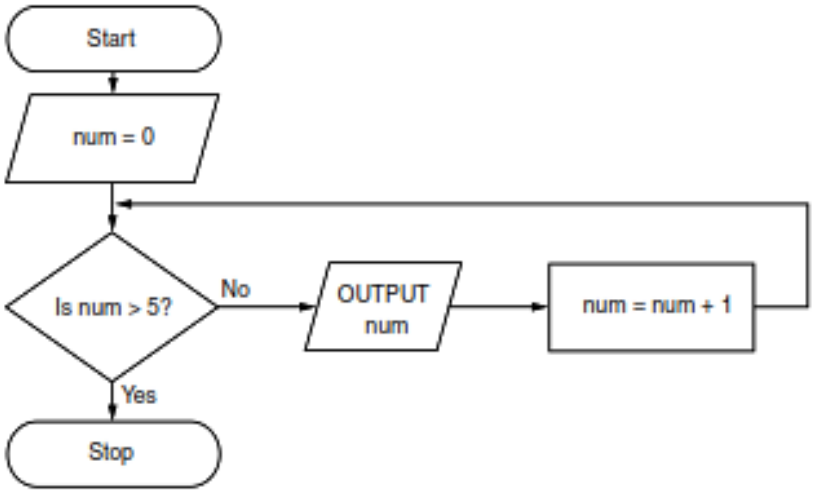
Condition-controlled loop: a loop that runs based on a condition, not the number of times it will run.

Count-controlled

- This type of loop needs a counter to keep track of the number of repetitions you have done (the number of loops).
- The counter will be a variable.
- Before you start the loop, the counter needs to be set to the starting value (usually 0 or 1).
- Then inside the code that loops, you need to increment the counter (this means add 1 to it).

Flowchart example: Count-controlled loop

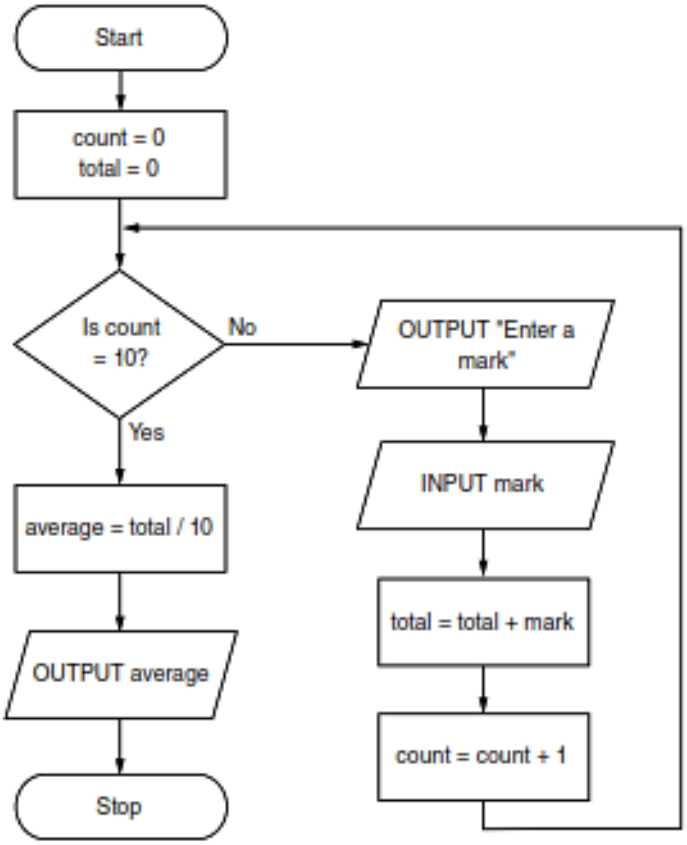
In this example, the counter variable is num. num is set to start at 0. Following the arrows, it keeps increasing by 1 until num is greater than 5, at which point the algorithm Stops.



Flowchart example: Calculating average

An algorithm needs to ask the user to input the 10 marks of its students, and then output the average.

Step 1: Repeat 10 times.
Step 2: Ask the user to input a mark.
Step 3: Input the mark.
Step 4: Add the mark to a total.
Step 5: After all the marks are entered, calculate the average.
Step 6: Output the average.



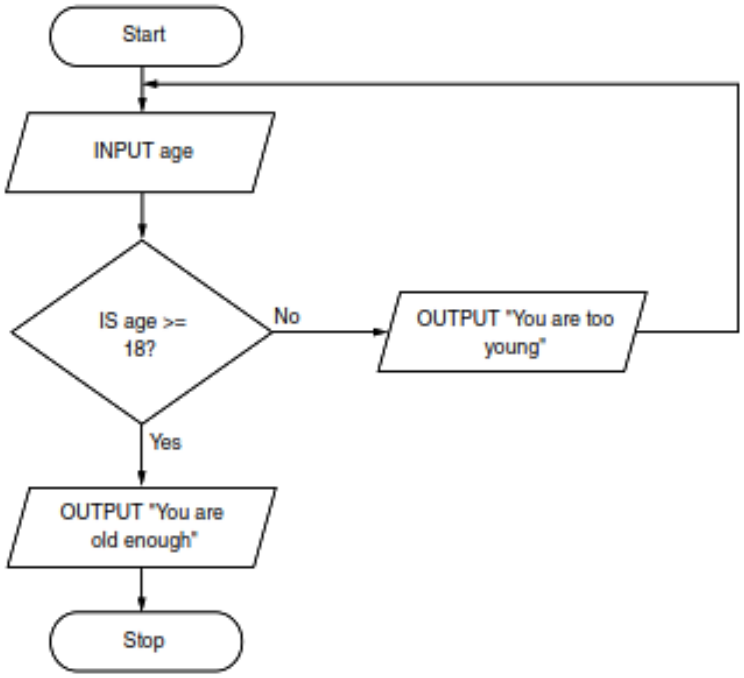
Condition-controlled

A condition-controlled loop has the same structure as a count-controlled loop, but it doesn't need a counting variable.

This could be controlled by an input from the user (you could loop until it is a valid input), or until a specific number is greater than another.

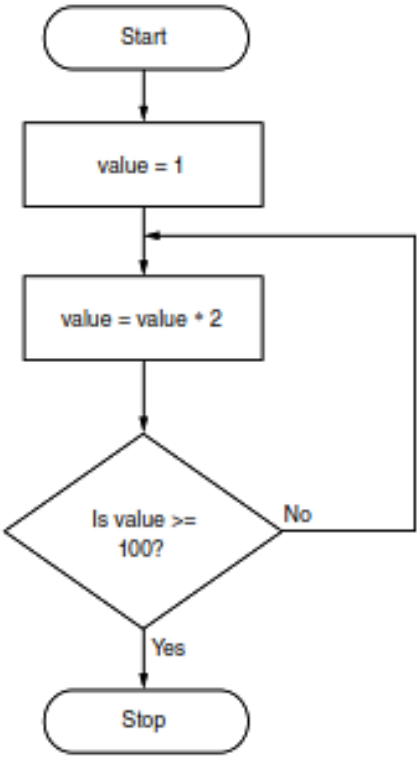
The flowchart looks similar to a count controlled due to the condition, but it would be missing the variable counter and does not run a set number of times.

Flowchart example: Condition-controlled loop 1



Flowchart example: Condition-controlled loop 2

The following flowchart is an example of a condition-controlled loop. The code inside the loop will always run once because the condition is at the end of the code. So, the statement value = value * 2 repeats until the value is >= 100.



String manipulation

Strings are collections of letters, characters and numbers. They are represented by speech marks; this is because the variable value is different from the word “value”.

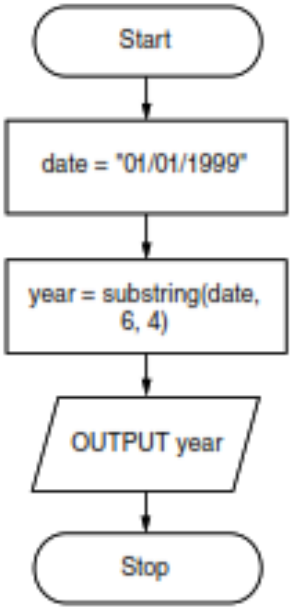
You may need to manipulate strings, for example get one or more characters from it or find its length.

The table below shows common string manipulators.

Command	Description	Example
length(string)	Returns the length of the string	length("Hello") would give 5
char(string, characterNum)	Returns the character at characterNum in the string.	char("Hello",0) would give "H"
mid(string, startingChar, numChars) or substring(string, startingChar, numChars)	Returns numChars number of characters from startingChar in the string	mid("Hello",1,2) would give "el"
upper(string)	Converts string to capitals	upper("Hello") would give "HELLO"
lower(string)	Converts string to lowercase	lower("Hello") would give "hello"

Flowchart example: String manipulation

In this example, the four digits of the year (1999) are extracted from the string and output.

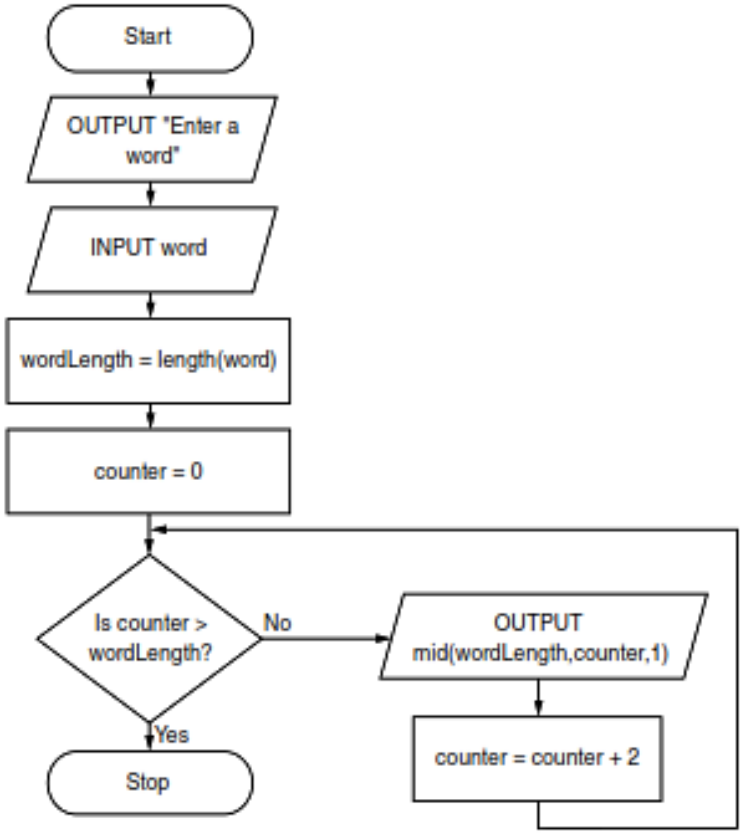


Flowchart example: String manipulation with mid command

A program is needed to input a word from the user, and then output alternate letters starting with the first letter.

Start by identifying the steps required.

Step 1: Ask the user to input a word.
Step 2: Input a word and store it in a variable.
Step 3: Count the number of letters in the word.
Step 4: Set a counter to start with the letter at position 0.
Step 5: Increase the counter by 2 each time through the loop.
Step 6: Output the letter of the counter in the loop.
Step 7: Loop until the counter is greater than or equal to the number of letters.



Pseudocode

Pseudocode is 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.

Data items to be processed by the algorithm are given meaningful names in the same way that variables and constants are in a high-level programming language. However, pseudocode is not bound by the strict syntax rules of a programming language.

Pseudocode for an assignment operator

- 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.

Mathematical operators

Operator	Action
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power

Examples of pseudocode assignment statements:

Cost ← 10	Cost has the value 10
Price ← Cost * 2	Price has the value 20
Tax ← Price * 0.12	Tax has the value 2.4
SellingPrice ← Price + Tax	SellingPrice has the value 22.4
Gender ← "M"	Gender has the value M
Chosen ← False	Chosen has the value False

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

Sometimes, IF statements do not have an ELSE part, such as in this sequence of instructions.

```
IF <condition>
   THEN
   <statement or list of statements to be carried out>
ENDIF
```

An example of this type is.

```
INPUT number1, number2
X ← number1/number2
      IF number2 > number1
      THEN
            X ← number2/number1
      ENDIF
OUTPUT X
```

IF statements with an ELSE clause are written as follows.

```
IF <condition>
   THEN
        <statement or list of statements to be carried out>
   ELSE
        <alternative statement or list of alternative statements to be carried out>
ENDIF
```

Using the example above, a better way of writing the algorithm would be.

```
INPUT number1, number2
    IF number2 > number1
      THEN
          X ← number2/number1
      ELSE
          X ← number1/number2
    ENDIF
OUTPUT X
```

Sometimes it is necessary to have what are called nested IF statements. This is an IF condition within another IF.

Following is an example of a problem which requires nested IF statements to solve it.

```
INPUT Mark
      IF Mark >=75
            THEN
                OUTPUT “A Grade”
            ELSE
               IF Mark >=65
                     THEN
                          OUTPUT “B Grade”
                     ELSE
                        IF Mark >=55
                              THEN
                                   OUTPUT “C Grade”
                              ELSE
                                 IF Mark >=45
                                       THEN
                                            OUTPUT “D Grade”
                                       ELSE
                                            OUTPUT “YOU HAVE FAILED”
                                  ENDIF
                          ENDIF
               ENDIF
      ENDIF
```

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.

```
CASE <identifier> OF
    <value 1> : <statement or list of statements>
    <value 2> : <statement or list of statements>
    OTHERWISE : <statement or list of statements>
ENDCASE
```

The algorithm below that specifies what happens if the value of Choice is 1, 2, 3 or 4.

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

Loops

When referring to the writing of computer programs or algorithms, a loop is a sequence of repeated statements that are carried out a number of times.

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
```

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

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

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.

In this example, the variable counter starts at 0 and continues until it is 9.

```
FOR count = 1 to 9
   OUTPUT count
NEXT count
```

Example: Calculating average

```
total = 0
FOR count = 0 TO 10
    OUTPUT "Enter a mark"
    INPUT mark
    total = total + mark
NEXT count
average = total / 10
OUTPUT average
```

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.

Example 1: REPEAT…UNTIL loop

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

Example 2: REPEAT…UNTIL loop

```
count ← 0
INPUT number
REPEAT
      count ← count + 1
      OUTPUT count, "x 10 = ", count*10
UNTIL count = number
```

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.

Example 1: WHILE…DO…ENDWHILE 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
```

Example 2: WHILE…DO…ENDWHILE loop

```
count ← 0
INPUT number
WHILE count < number
      count ← count + 1
      OUTPUT count, "x 10 = ", count*10
ENDWHILE
```

String manipulation

Strings are collections of letters, characters and numbers. They are represented by speech marks.

You may need to manipulate strings, for example get one or more characters from it or find its length.

Example: String manipulation

```
UTPUT "Enter a word"
INPUT word
wordLength = length(word)
FOR counter = 0 TO wordLength
        OUTPUT mid(wordLength, counter, 1)
        counter = counter + 1
NEXT counter
```

Subroutines

A subroutine is a set of instructions that are independent. They have an identifier (a name) and can be called from other parts of the program. There are two main types of subroutine, a **procedure** and a **function**.

Procedure: a type of subroutine that does not return a value to the main program.

Function: a separate piece of code that has an identifier and performs a task; it can be called from elsewhere in the code and returns a value.

In pseudocode, a subroutine still has an identifier. It starts with the command PROCEDURE and ends with calculate(value) ENDPROCEDURE.

Example:

In this example, the procedure is called outputMessage(). The brackets identify it as a subroutine. The procedure outputs the words "Hello World" and then returns control.

```
PROCEDURE outputMessage()
      OUTPUT "Hello World"
ENDPROCEDURE
```

A procedure is called by using the name of the procedure, for example,

```
outputMessage()
OUTPUT "Finished"
```

Worked example:

```
PROCEDURE multiply(number)
    result = number * number
    OUTPUT result
ENDPROCEDURE

PROCEDURE powerOf(number)
    result = number ^ number
    OUTPUT result
ENDPROCEDURE

INPUT number
Counter = 1
WHILE counter < number
     IF counter MOD 2 = 0 THEN
          multiply(number)
     ELSE
          powerOf(number)
     ENDIF
ENDWHILE
```