















Syllabus Content:






2.2.1 Algorithms

-  show understanding that an algorithm is a solution to a problem expressed as a sequence of defined steps
-  use suitable identifier names for the representation of data used by a problem
 - summarise identifier names using an identifier table
-  show understanding that many algorithms are expressed using the four basic constructs of assignment, sequence, selection and repetition
-  show understanding that simple algorithms consist of input, process, output at various stages
-  document a simple algorithm using:
 - **Structured English**
 - **pseudocode** (on the examination paper, any given pseudocode will be presented using the Courier New font)
 - **program flowchart**
-  derive pseudocode or a program flowchart from a structured English description of a problem
-  derive pseudocode from a given program flowchart or vice versa
-  use the process of stepwise refinement to express an algorithm to a level of detail from which the task may be programmed
-  decompose a problem into sub-tasks leading to the concept of a program module (procedure/ function)
-  show an appreciation of why logic statements are used to define parts of an algorithm solution
-  use logic statements to define parts of an algorithm solution

2.2.1 Data types




-  select appropriate data types for a problem solution
-  use in practical programming the data types that are common to procedural high-level languages: integer, real, char, string, Boolean, date (pseudocode will use the following data types: **INTEGER, REAL, CHAR, STRING, BOOLEAN, DATE, ARRAY, FILE**)
-  show understanding of how character and string data are represented by software including the ASCII and Unicode character sets

2.2.2 Arrays



-  use the technical terms associated with arrays including upper and lower bound
-  select a suitable data structure (1D or 2D array) to use for a given task
-  use pseudocode for 1D and 2D arrays (pseudocode will use square brackets to contain the array subscript, for example a 1D array as A[1:n] and a 2D array as C[1:m, 1:n])
-  write program code using 1D and 2D arrays
-  write algorithms/program code to process array data including:

Syllabus Content:

2.3.1 Programming basics

-  write a program in a high-level language (The nature of the language should be procedural and will be chosen by the Centre from the following: Python, Visual Basic (console mode), Pascal/Delphi (console mode))
-  implement and write a program from a given design presented as either a program flowchart or pseudocode
-  write program statements for:
 - the declaration of variables and constants
 - the assignment of values to variables and constants
 - expressions involving any of the arithmetic or logical operators
 - input from the keyboard and output to the console given pseudocode will use the following structures:
 - DECLARE <identifier> : <data type> // declaration
 - CONSTANT <identifier> = <value>
 - <identifier> ← <value> or <expression> // assignment
 - INPUT <identifier>
 - OUTPUT <string> , OUTPUT <identifier(s)>




2.3.2 Transferable skills

-  recognise the basic control structures in a high-level language other than the one chosen to be studied in depth
-  appreciate that program coding is a transferable skill

2.3.3 Selection

-  use an 'IF' structure including the 'ELSE' clause and nested IF statements
-  use a 'CASE' structure

2.3.4 Iteration

-  use a 'count-controlled' loop:
 - FOR <identifier> ← <value1> TO <value2> <statement(s)> ENDFOR
 - alternatively: FOR <identifier> ← <value1> TO <value2> STEP <value3> <statement(s)> ENDFOR
-  use a 'post-condition' loop:
 - REPEAT <statement(s)> UNTIL <condition>
-  use a 'pre-condition' loop
 - WHILE <condition> <statement(s)> ENDWHILE • justify why one loop structure may be better suited to a problem than the others

Algorithms:

An algorithm is a sequence of steps done to perform some task.

- 🌱 The essential aim of an algorithm is to get a specific output,
- 🌱 An algorithm involves with several continuous steps,
- 🌱 The output comes after the algorithm finished the whole process.

So basically, all algorithms perform logically while following the steps to get an output for a given input.

Types of Algorithms:

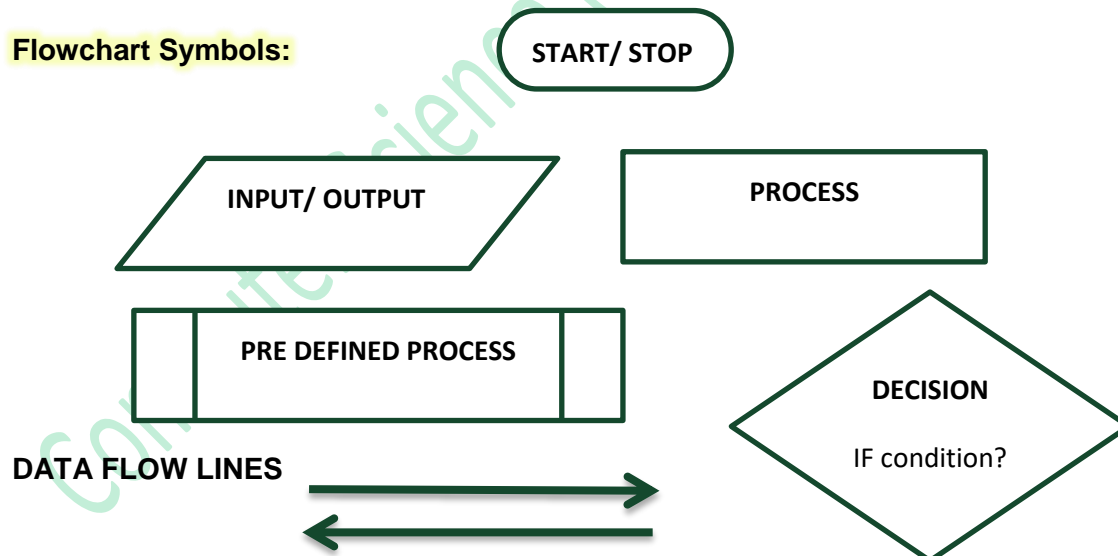
- 🌱 Structured English
- 🌱 Flowcharts
- 🌱 Pseudo codes
- 🌱 Program Code

STRUCTURED ENGLISH:

Structured English provides a more formal way of documenting the stages of the algorithm. Structured English is a subset of English language that consists of command statements used to describe an algorithm.

FLOWCHARTS:

Flow chart is a graphical representation of a program. Flowcharts use different symbols containing information about steps or a sequence of events.

Flowchart Symbols:**PSEUDOCODE:**

Pseudo code is an outline of a program, written as a series of instruction using simple English sentences.

Pseudo code uses keywords commonly found in high-level *languages* and mathematical notation. It

describes an algorithm's steps like program statements, without being bound by the strict rules of vocabulary and syntax of any particular language, together with ordinary English.

Variable:

Variable is memory location where a value can be stored.

Constants:

Just like variables, constants are "dataholders". They can be used to store data that is needed at runtime.

In contrast to variable, the content of a constant can't change at runtime, it has a constant value. Before the program can be executed (or compiled) the value for a constant must be known.

Arithmetic

Use the arithmetic operators.

Assignment

Assignment is the process of writing a value into a variable (a named memory location). For example, $\text{Count} \leftarrow 1$ can be read as 'Count is assigned the value 1', 'Count is made equal to 1' or 'Count becomes 1'.

Initialization:

If an algorithm needs to read the value of a variable *before* it assigns input data or a calculated value to the variable, the algorithm should assign an appropriate initial value to the variable, known as Initialization.

Input

We indicate input by words such as **INPUT**, **READ** or **ENTER**, followed by the name of a variable to which we wish to assign the input value.

Output:

We indicate output by words such as **OUTPUT**, **WRITE** or **PRINT**, followed by a comma-separated list of expressions.

Totaling

To keep a running total, we can use a variable such as Total or Sum to hold the running total and assignment statements such as:

$\text{Total} \leftarrow \text{Total} + \text{Number}$

ADD Number to Total

Counting

It is sometimes necessary to count how many times something happens.

To count up or increment by 1, we can use statements such as:

$\text{Count} \leftarrow \text{Count} + 1$

INCREMENT Count by 1

Structured statements

In the sequence structure the processing steps are carried out one after the other. The instructions are carried out in sequence, unless a selection or loop is encountered.

Operator	Use
\wedge	Exponentiation
-	Negation (used to reverse the sign of the given value, exp -intValue)
*	Multiplication
/	Division
\	Integer Division
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Operator	Comparison
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
=	Equals to
<>	Not equal
()	Group
AND	And
OR	Or
NOT	not

Data types

The following table shows the Visual Basic data types, their supporting common language runtime types, their nominal storage allocation, and their value ranges.

Basic Data Types

A variable can store one type of data. The most used data types are:

Type	Description
Integer	Stores a whole number, e.g. 78
double	Stores a decimal number, e.g. 74.23754
Char	Stores one character, e.g. A
string	Stores text, e.g. Hello
Boolean	Stores True or False

Declaration of Variables and Constant:

The process of creating a variable is called declaring a variable. Variables must be created or declared where users enter their data.

Pseudo code

BEGIN

DECLARE variable As Datatype

Variable ← 0 //initialization

OUTPUT ("What is your Email address")

INPUT variable value

IF valid email address?

Then ...

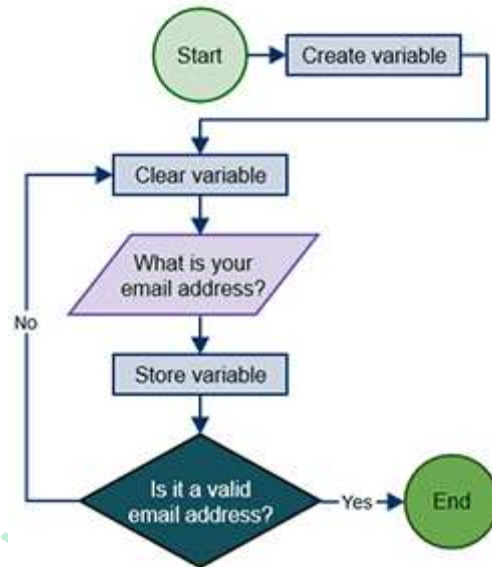
END

Each declaration needs 4 things:

Pseudo code

- **DECLARE** keyword
- Variable name
- **AS** keyword
- Variable data type

DECLARE variable As Datatype



VB code example:

- **DIM** keyword
- Variable name
- **AS** keyword
- Variable data type

Dim mark As Integer

Declaring Multiple Variables:

Pseudocodes

DECLARE index As Integer

DECLARE grade As Integer

DECLARE counter As Integer

VB Code Console Mode

Dim index As Integer

Dim grade As Integer

Dim counter As Integer

The three declarations above can be rewritten as one declaration if same data type is used:

DECLARE index, grade, counter As Integer

Dim index, grade, counter As Integer

Constants

Creating Constants in Pseudocode is just writing constant name and value with it. In contrast to variable, the content of a constant can't change at runtime, it has a constant value.

CONSTANT <identifier> = <Value>

CONSTANT Pi ← 3.1415 or **CONSTANT** Pi = 3.14

```

Const pi As Double = 3.1415
'create a constant called pi with a value 3.1415
Dim radius As Double = 10
'creates a constant called radius with a value 10
Dim circumference As Double = radius * 2 * pi
'Creates a constant with a calculation
Dim area As Double = radius ^ 2 * pi
'creating a constant with a calculation
Console.WriteLine("Circle Circumference : " & circumference)
Console.WriteLine("Circle Area : " & area)

Console.ReadLine()

```

Type of Programs:

- Sequence
- Selection
- Repetitions/Loops

Sequence

Statements are followed in sequence so the order of the statements in a program is important.

Assignment statements rely on the variables used in the expression on the right-hand side of the statement all having been given values. Input statements often provide values for assignment statements. Output statements often use the results from assignment statements.

PSEUDOCODE**BEGIN**

```

DECLARE number1 As Integer
DECLARE number2 As Integer
DECLARE sum As Integer
DECLARE product As Integer

```

```

PRINT ("Enter number 1")
INPUT number1

```

```

PRINT ("Enter number 2")
INPUT number2

```

```

Sum ← number1 + number2
product ← number1 * number2

```

```

PRINT ("the sum is")
PRINT (sum)
PRINT ("the product is")
PRINT (product)

```

END**VB code example**

```

Sub Main()
    Dim number1 As Integer
    Dim number2 As Integer
    Dim sum As Integer
    Dim product As Integer

    Console.WriteLine("Enter number 1")
    number1 = Console.ReadLine()

    Console.WriteLine("Enter number 2")
    number2 = Console.ReadLine()

    sum = number1 + number2
    product = number1 * number2

    Console.Write("the sum is ")
    Console.WriteLine(sum)

    Console.Write("the product is ")
    Console.WriteLine(product)

    Console.ReadLine()

End Sub

```

STRUCTURED ENGLISH**WORKED EXAMPLE 11.01****Using input, output, assignment and sequence constructs**

The problem to be solved: Convert a distance in miles and output the equivalent distance in km.

Step 1: Write the problem as a series of structured English statements:

```

INPUT number of miles
Calculate number of km
OUTPUT calculated result as km

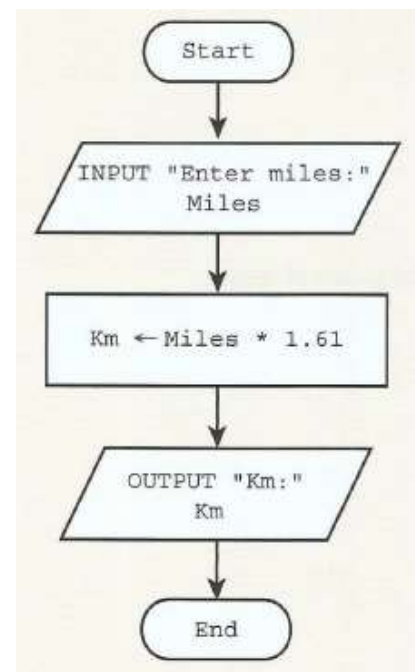
```

Step 2: Analyse the data values that are needed.

We need a variable to store the original distance in miles and a variable to store the result of multiplying the number of miles by 1.61. It is helpful to construct an **identifier table** to list the variables.

Identifier	Explanation
Miles	Distance as a whole number of miles
Km	The result from using the given formula: $Km = Miles * 1.61$

Table 11.02 Identifier table for miles to km conversion

FLOWCHART

Pseudocode

```

INPUT "Enter miles:" Miles
Km ← Miles * 1.61
OUTPUT "km:" Km

```

```

BEGIN
DECLARE miles, km As REAL

OUTPUT ("Enter miles")
INPUT miles

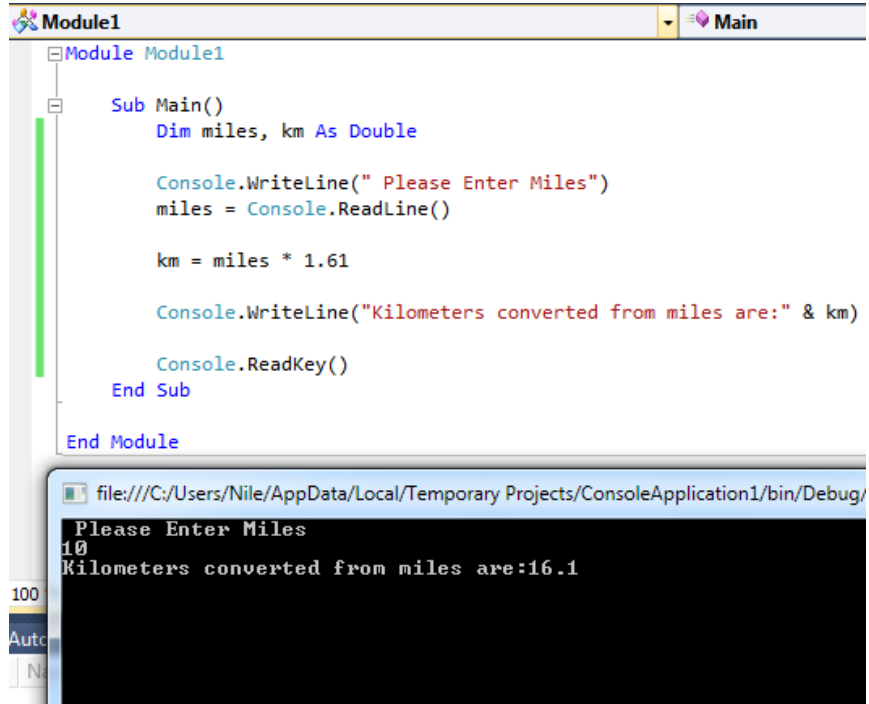
km ← miles * 1.61

OUTPUT ("Km are : " & km)

END

```

VB Code



```

Module1
Module Module1

Sub Main()
Dim miles, km As Double

Console.WriteLine(" Please Enter Miles")
miles = Console.ReadLine()

km = miles * 1.61

Console.WriteLine("Kilometers converted from miles are:" & km)

Console.ReadKey()
End Sub

End Module

```

file:///C:/Users/Nile/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/
Please Enter Miles
10
Kilometers converted from miles are:16.1

Structured statements for selection (conditional statements)

These statements are used to select alternative routes through an algorithm; selection's logical expressions often involve comparisons, which can operate on text strings as well as numbers.

- IF...THEN...ELSE...ENDIF
- CASE...OF...OTHERWISE...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.

A condition can be set up in different ways:

```

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

```

CASE ... OF ... OTHERWISE ... ENDCASE

For a CASE condition 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.

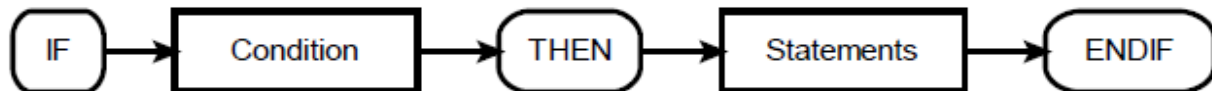
The algorithm below specifies what happens if the value of Choice is 1, 2, 3 or 4.

CASE Choice OF

- 1: Answer \leftarrow Num1 + Num2
- 2: Answer \leftarrow Num1 - Num2
- 3: Answer \leftarrow Num1 * Num2
- 4: Answer \leftarrow Num1 / Num2

OTHERWISE PRINT "Please enter a valid choice"
ENDCASE

The IF THEN statement



PSEUDOCODE

```

BEGIN
DECLARE grade As Integer

PRINT ("Enter your grade")
INPUT grade

IF grade > 50
THEN PRINT ("You have passed")
ELSE PRINT ("You have failed")
END IF

END
  
```

VB Code

```

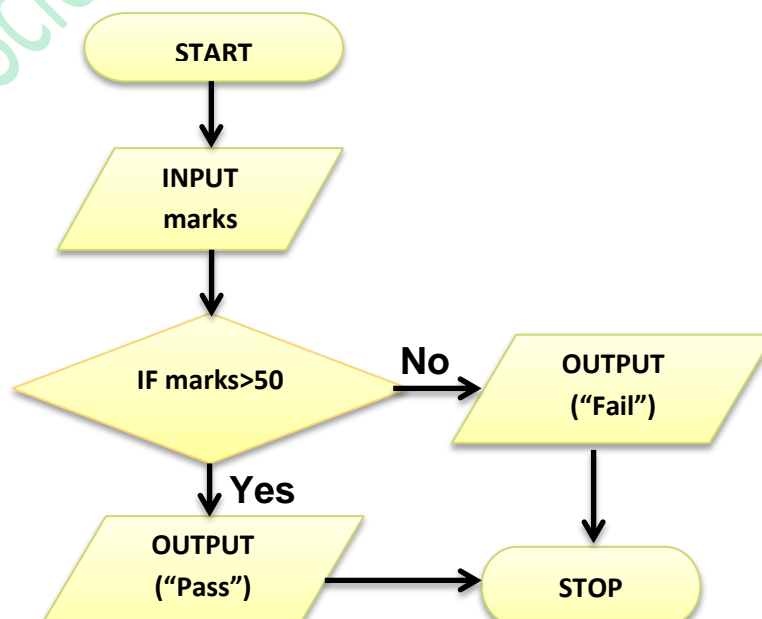
Sub Main()
    Dim grade As Integer

    Console.WriteLine("Enter your grade")
    grade = Console.ReadLine()

    If grade > 50 Then
        Console.WriteLine("You have passed")
    Else
        Console.WriteLine("You have failed")
    End If

    Console.ReadLine()
End Sub
  
```

FLOWCHART:



IF THEN, ELSE-IF statements

VB code example

```

BEGIN
DECLARE grade As Integer
PRINT ("Enter a grade")
INPUT grade
IF grade > 80
    THEN PRINT ("Grade A")
    ELSE IF grade > 60
        THEN PRINT ("Grade B")
        ELSE IF grade > 50
            THEN PRINT ("Grade C")
            ELSE PRINT ("Grade U")
        END IF
    END IF
END IF
END

```

```

Sub Main()
    Dim grade As Integer

    Console.WriteLine("Enter a grade")
    grade = Console.ReadLine

    If grade > 80 Then
        Console.WriteLine("Grade A")
    ElseIf grade > 60 Then
        Console.WriteLine("Grade B")
    ElseIf grade > 50 Then
        Console.WriteLine("Grade C")
    Else
        Console.WriteLine("Grade U")
    End If

    Console.ReadLine()
End Sub

```

The IF statement is useful, but can get clumsy if you want to consider "multi-way selections"

CASE OF OTHERWISE...

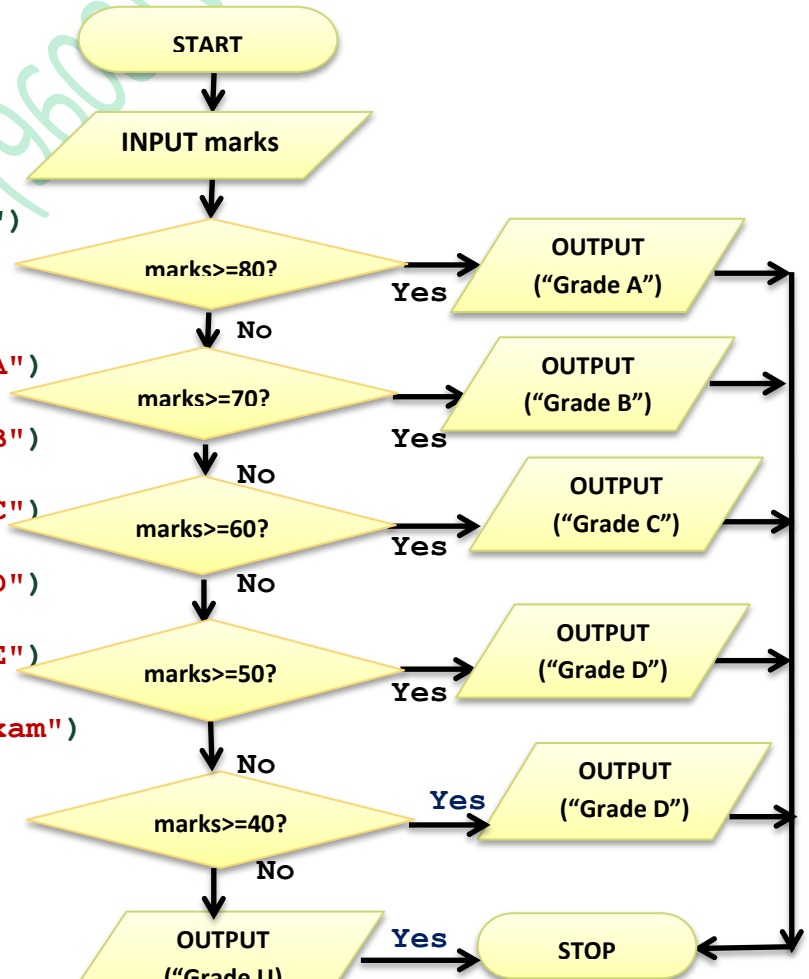
FLOWCHART

Pseudo code

```

BEGIN
DECLARE grade As Integer
PRINT ("Enter your grade")
INPUT grade
CASE grade OF
    grade >= 80
        PRINT ("Grade A")
    grade >= 70
        PRINT ("Grade B")
    grade >= 60
        PRINT ("Grade C")
    grade >= 50
        PRINT ("Grade D")
    grade >= 40
        PRINT ("Grade E")
    OTHERWISE
        PRINT ("Grade U, Repeat Exam")
END CASE
END

```



Program Code in Visual Basic Console Mode:

```

Sub Main()
    Dim marks As Integer
    Console.WriteLine("Please Input your marks")
    marks = Console.ReadLine()

    While marks > 100 Or marks < 0
        Console.WriteLine("Wrong Entry, Please Enter Between 0 and 100")
        Console.WriteLine("Please Input your marks")
        marks = Console.ReadLine()
    End While

    Select Case marks 'NOTES by Sir Majid Tahir
        Case Is >= 90 'Download notes at www.majidtahir.com
            Console.WriteLine("Your Grade is A* ")
        Case Is >= 80
            Console.WriteLine("Your Grade is A ")
        Case Is >= 70
            Console.WriteLine("Your Grade is B ")
        Case Is >= 60
            Console.WriteLine("Your Grade is C ")
        Case Is >= 50
            Console.WriteLine("Your Grade is D ")
        Case Else
            Console.WriteLine("Your Grade is U, Please Repeat the Exam")
    End Select
    Console.Read()
End Sub
End Module

```

file:///C:/Users/Majid/AppData/Local/Temporary Projec...
 Please Input your marks
 -1
 Wrong Entry, Please Enter Between 0 and 100
 Please Input your marks
 120
 Wrong Entry, Please Enter Between 0 and 100
 Please Input your marks
 92
 Your Grade is A*

LOOPS (Structured statements for iteration (repetition))

Many problems involve repeating one or more statements, so it is useful to have structured statements for controlling these iterations or repetitions. Exit conditions consist of logical expressions whose truth can be tested, such as Count = 10 or Score < 0. At a particular time, a logical expression is either **True** or **False**.

- FOR...TO...NEXT
- WHILE...DO...ENDWHILE
- REPEAT...UNTIL

FOR ... NEXT LOOP

This is to be used when loop is to be repeated a known fixed number of times. The counter is automatically increased each time the loop is performed.

```

FOR count = 1 to 10
    INPUT number
    total = total + number
NEXT count

```

WHILE ... Do LOOP

This loop is used when we don't know how many times the loop is to be performed. The Loop is ended when a certain condition is true.

This condition is checked before starting the loop.

```
While COUNT < 10 DO
    Input NUMBER
    TOTAL = TOTAL + NUMBER
    COUNT = COUNT + 1
Endwhile
Output TOTAL
```

REPEAT ... UNTIL LOOP

REPEAT UNTIL Loop is used when we do not know how many times loop will be performed. The Loop is ended when a certain condition is true.

The Condition is checked at the end of the Loop and so a REPEAT Loop always has to be performed at least once.

```
REPEAT
    Input NUMBER
    TOTAL = TOTAL + NUMBER
    COUNT = COUNT + 1
Until COUNT = 10
Output Total
```

FOR loop

The for loop repeats statements a set number of times. It uses a variable to count how many times it goes round the loop and stops when it reaches its limit.



BEGIN

DECLARE index As Integer

FOR index = 1 To 20

PRINT (index & "times 5 is" & index * 5)

NEXT

VB code example:

```
Sub Main()
    Dim index As Integer

    For index = 1 To 20
        Console.WriteLine(index & " times 5 is " & index * 5)
    Next
    Console.ReadLine()
End Sub

End Module
```

Other examples of FOR loop

BEGIN

DECLARE BiggestSoFar, NextNumber, Counter **As Integer**

INPUT BiggestSoFar

FOR Counter ← 1 **TO** 5

INPUT NextNumber

IF NextNumber > BiggestSoFar

THEN

BiggestSoFar ← NextNumber

ENDIF

END FOR

OUTPUT ("The biggest number so far is" & BiggestSoFar)

END

Sample VB Code of above Pseudocode:

```

Module Module1

    Sub Main()
        Dim biggestSoFar, NextNum, counter As Integer

        Console.WriteLine("Enter Biggest number")
        biggestSoFar = Console.ReadLine()

        For counter = 1 To 5

            Console.WriteLine("Enter Next biggest number")
            NextNum = Console.ReadLine()

            If NextNum > biggestSoFar Then
                biggestSoFar = NextNum
            End If

        Next

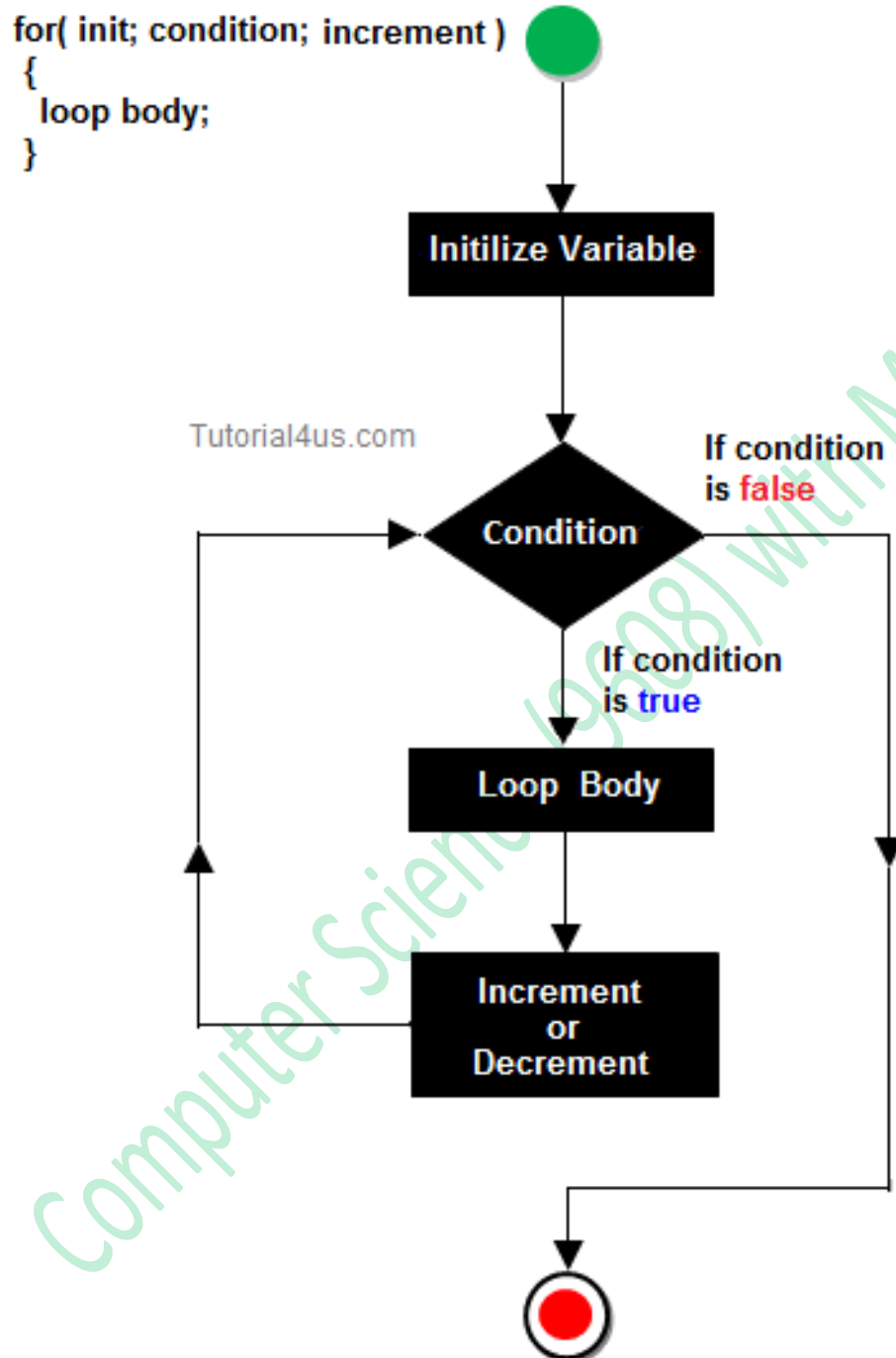
        Console.WriteLine("The biggest number entered is" & biggestSoFar)
        Console.ReadLine()

    End Sub

End Module

```


FLOWCHART FOR LOOP



WHILE DO ENDWHILE loop

The while loop is known as a **test before loop**. The condition is tested before entering the loop, but tested each time it goes round the loop. The number of times the statements within the loop are executed varies. The test before loop goes round 0 or more times.

This method is useful when processing files and using "read ahead" data



VB Code example

```

BEGIN
DECLARE name As String

INPUT name

WHILE name <> "x"
PRINT ("Your name is: "name)
INPUT name
END WHILE

END
  
```

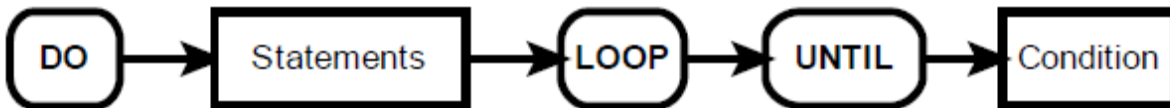
```

Sub Main()
    Dim name As String

    name = Console.ReadLine()
    'Test before loop -
    'only enter the loop is name not equal "X"
    While name <> "X"
        Console.WriteLine(name)
        name = Console.ReadLine()
    End While
End Sub
  
```

REPEAT UNTIL loop

The repeat loop is similar to the while loop, but it tests the condition after the statements have been executed once. This means that this test after loop goes round 1 or more times.



VB code example

```

BEGIN
DECLARE name As String

REPEAT
INPUT name
PRINT ("Your name is:" name)
UNTIL name = "x"

END
  
```

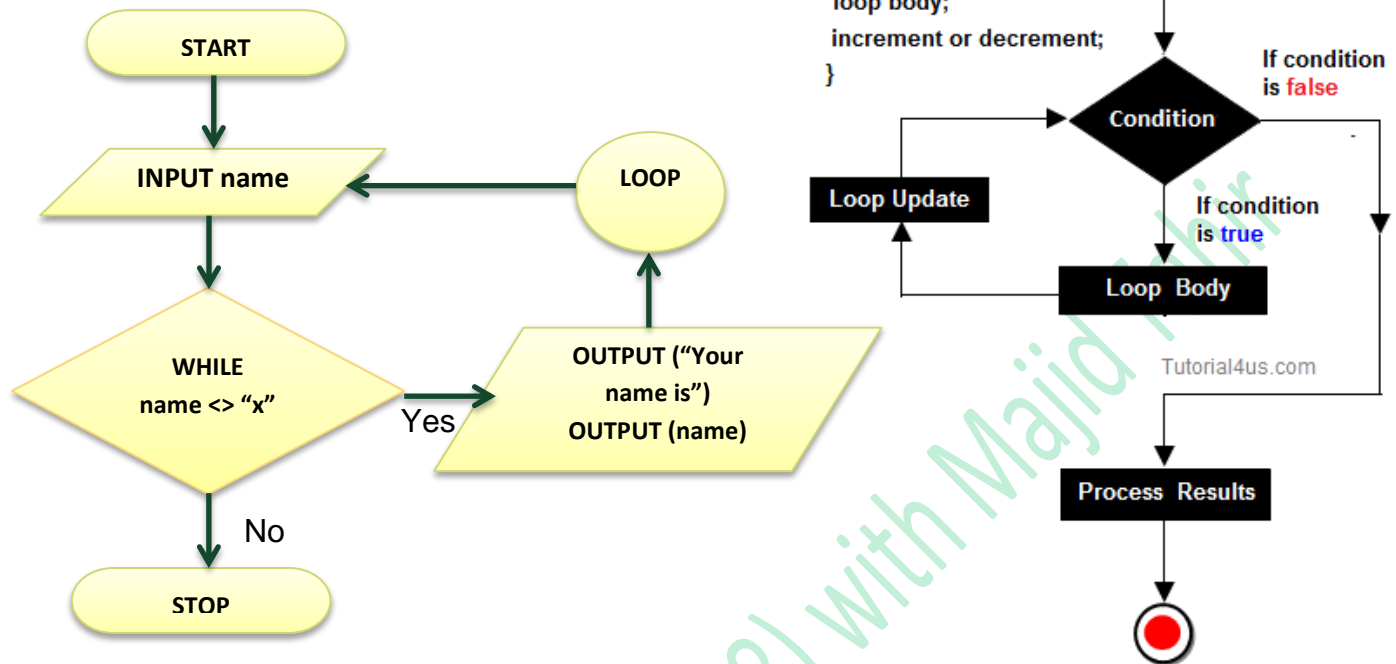
```

Sub Main()
    Dim name As String

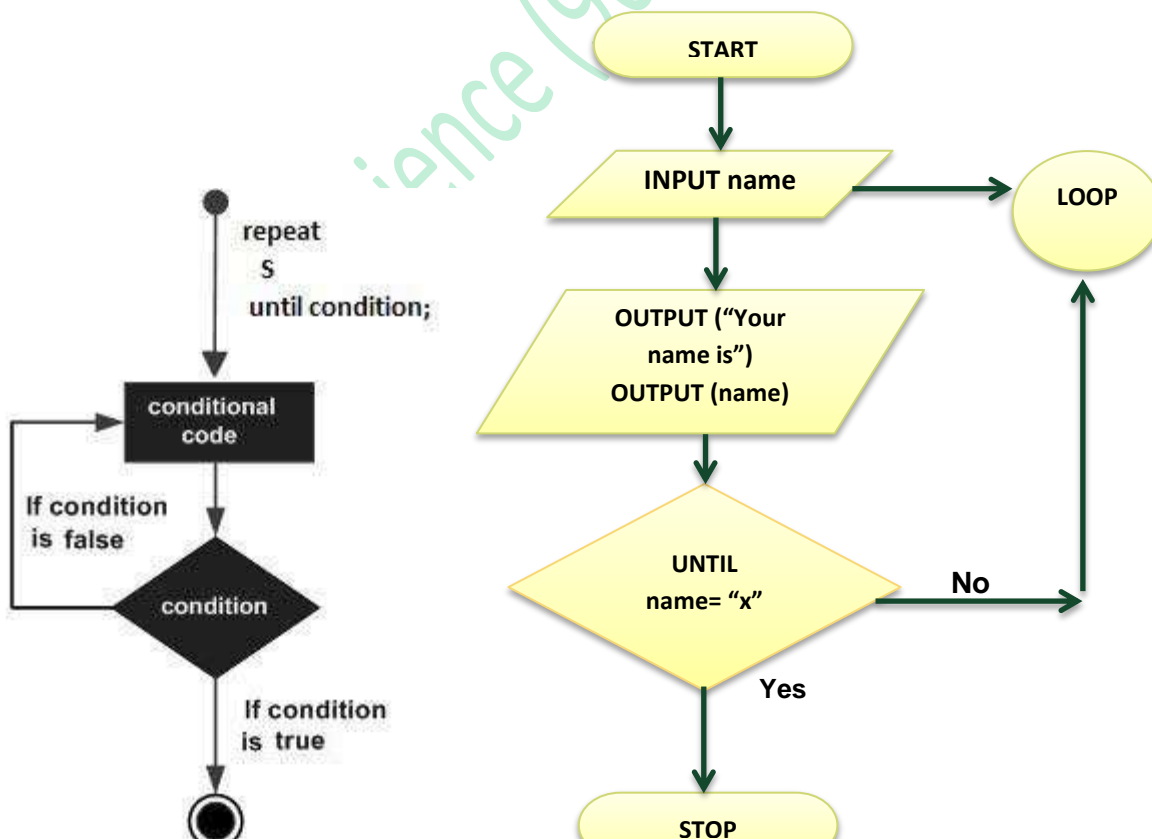
    Do
        name = Console.ReadLine()
        Console.WriteLine(name)
    Loop Until name = "X"
    'Test after loop
End Sub
  
```

Keeps inputting name and keeps printing name until user enters "X"

FLOWCHART...WHILE-ENDWHILE



FLOWCHART...REPEAT-UNTIL



Array Data Type

An array is a special variable that has one name, but can store multiple values. Each value is stored in an element pointed to by an index.

The first element in the array has index value 0, the second has index 1, etc

One Dimensional Arrays

A one dimensional array can be thought as a list. An array with 10 elements, called names, can store 10 names and could be visualized as this:

index	Element
0	Fred
1	James
2	Tom
3	Robert
4	Jonah
5	Chris
6	Jon
7	Matthew
8	Mikey
9	Jack

Arrays (One-dimensional arrays)

In order to use a one-dimensional array in a computer program, you need to consider:

- What the array is going to be used for, so it can be given a meaningful name
- How many items are going to be stored, so the size of the array can be determined.
- What sort of data is to be stored, so that the array can be the appropriate data type.

This array would be created by:

```
DECLARE names (9) As String
```

```
PRINT (names (1))
```

will display James

```
PRINT (names (7))
```

Will display Mathew

VB code example:

```
Dim names(9) As String
```

Elements indexed from 0 to 9

The statement:

```
Console.WriteLine(names(1))
```

Will display James

```
Console.WriteLine(names(7))
```

Will display Matthew

Entering Values in One-Dimension Array

BEGIN

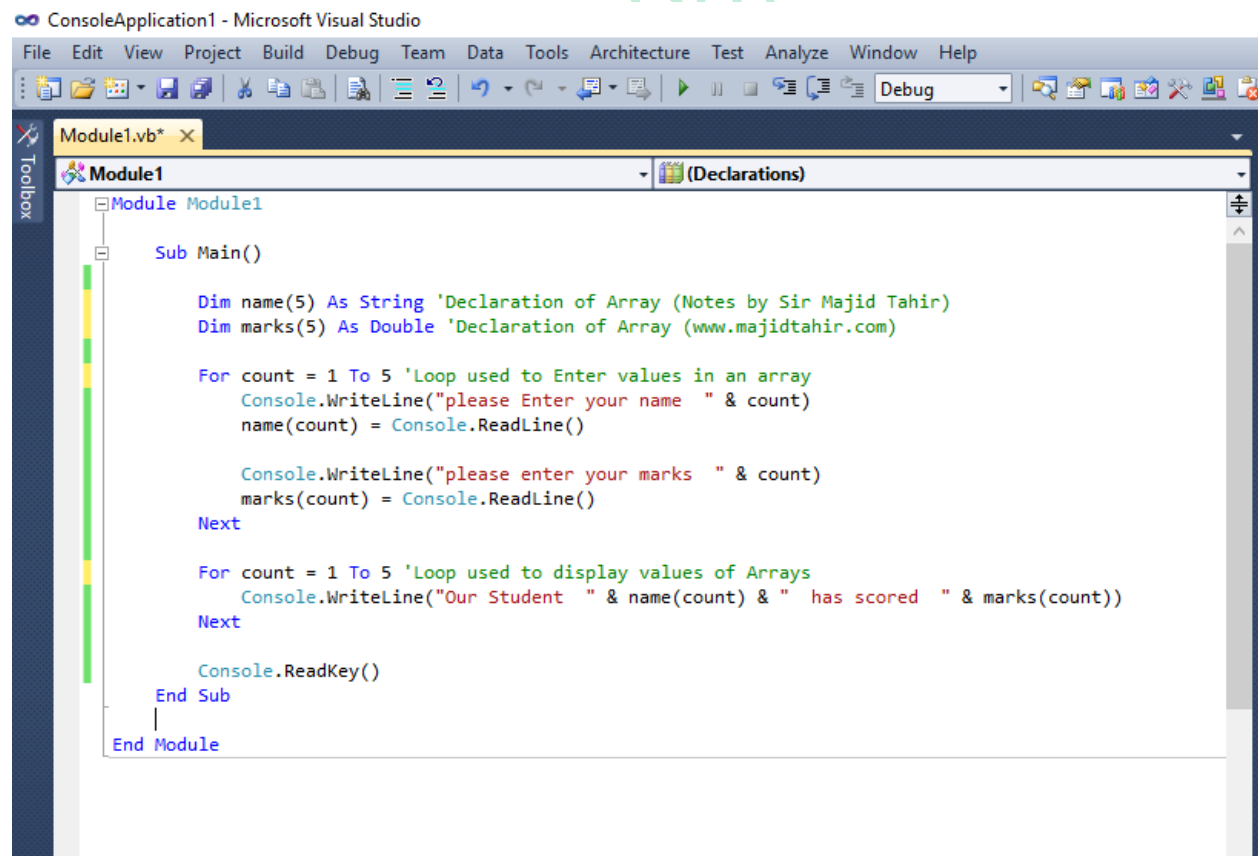
```
DECLARE count As Integer
DECLARE name (5) As String    // for declaring 10 elements in ARRAY
DECLARE marks (5) As Integer
```

```
FOR count = 1 to 5           // for inputting 10 names and grades
PRINT ("Enter Name "& count)
INPUT name (count)
PRINT ("Enter grade for "& name (count))
INPUT grades (count)
NEXT count
```

```
                                // for displaying 10 names and grades
FOR count 1 to 5
PRINT (name (count) & "has grade " & grades (count))
NEXT count
```

END

VB Code in Console Mode



```
ConsoleApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Module1.vb* x
Module1 (Declarations)
Module Module1
    Sub Main()
        Dim name(5) As String 'Declaration of Array (Notes by Sir Majid Tahir)
        Dim marks(5) As Double 'Declaration of Array (www.majidtahir.com)

        For count = 1 To 5 'Loop used to Enter values in an array
            Console.WriteLine("please Enter your name " & count)
            name(count) = Console.ReadLine()

            Console.WriteLine("please enter your marks " & count)
            marks(count) = Console.ReadLine()
        Next

        For count = 1 To 5 'Loop used to display values of Arrays
            Console.WriteLine("Our Student " & name(count) & " has scored " & marks(count))
        Next

        Console.ReadKey()
    End Sub
End Module
```

Output of VB code displayed above

The screenshot shows a Visual Basic IDE with a code window on the left and a console window on the right. The code window displays the following code:

```

Module1.vb
Module1
Sub Main()
    Dim name(5) As String 'Declaration of Array (Notes by Sir Majid Tahir)
    Dim marks(5) As Double 'Declaration of Array (www.majidtahir.com)

    For count = 1 To 5 'Loop used to Enter values in an array
        Console.WriteLine("please Enter your name " & count)
        name(count) = Console.ReadLine()

        Console.WriteLine("please enter your marks " & count)
        marks(count) = Console.ReadLine()
    Next

    For count = 1 To 5 'Loop used to display values of Arrays
        Console.WriteLine("Our Student " & name(count) & " has scored ")
    Next

    Console.ReadKey()
End Sub
End Module

```

The console window shows the following output:

```

please Enter your name 1
Majid
please enter your marks 1
99
please Enter your name 2
Sajid
please enter your marks 2
88
please Enter your name 3
Tahir
please enter your marks 3
90
please Enter your name 4
Waris
please enter your marks 4
78
please Enter your name 5
Mustafa
please enter your marks 5
11
Our Student Majid has scored 99
Our Student Sajid has scored 88
Our Student Tahir has scored 90
Our Student Waris has scored 78
Our Student Mustafa has scored 11

```

Another example of One-Dimensional Array

```

Module Module1
Sub Main()
    Dim count As Integer
    Dim name(4) As String
    Dim marks(4) As Integer
    Dim gender(4) As String
    For count = 0 To 4
        Console.WriteLine("please enter your name" & count)
        name(count) = Console.ReadLine()
        Console.WriteLine("please enter your gender" & count)
        gender(count) = Console.ReadLine()
        Console.WriteLine("please enter your marks" & count)
        marks(count) = Console.ReadLine()
    Next count
    For count = 0 To 4
        Console.WriteLine("your name is : " & name(count))
        Console.WriteLine("your gender is : " & gender(count))
        Console.WriteLine("your marks are : " & marks(count))
    Next count
    Console.ReadKey()
End Sub
End Module

```


Two Dimensional Arrays (2-D Arrays)

Using pseudocode, the algorithm to set each element of array `ThisTable` to zero is:

```
FOR Row ← 1 TO MaxRows
  FOR Column ← 1 TO MaxColumns
    ThisTable[Row, Column] ← 0
  ENDFOR
ENDFOR
```

When we want to output the contents of a 2D array, we again need nested loops. We want to output all the values in one row of the array on the same line. At the end of the row, we want to output a new line.

```
FOR Row ← 1 TO MaxRows
  FOR Column ← 1 TO MaxColumns
    OUTPUT ThisTable[Row, Column] // stay on same line
  ENDFOR
  OUTPUT Newline // move to next line for next row
ENDFOR
```

VB Code Example of Two-Dimension Array

```
Sub Main()

    Dim thistable(3, 5) As Integer
    Dim maxrow, maxcolumn, row, column As Integer

    maxrow = 3
    maxcolumn = 5

    For row = 1 To maxrow
        For column = 1 To maxcolumn
            thistable(row, column) = (maxrow) & (maxcolumn)
        Next
    Next

    For row = 1 To maxrow
        For column = 1 To maxcolumn
            Console.WriteLine("This table has values" & thistable(row, column))
        Next
    Next
    Console.ReadKey()

End Sub
```

Multi-Dimensional Arrays:

A multi-dimensional array can be thought of as a table, each element has a row and column index. Following example declares a two-dimensional array called `matrix` and would be declared by

```
Dim matrix(2,3) As Integer
```

Usually we refer to the first dimension as being the rows, and the second dimension as being the columns.

index	0	1	2	3
0	A	B	C	D
1	E	F	G	H
2	I	J	K	L

The following statements would generate the following

```
Console.WriteLine(matrix(0, 0))
```

Would display A

```
Console.WriteLine(matrix(2, 1))
```

Would display J

```
Console.WriteLine("first row, first column : " & matrix(2, 3))
```

Would display first row, first column : L

VB Code for 2-D Array is:

```
Sub Main()
```

```
Dim matrix(2, 3) As Integer
```

```
matrix(0, 0) = 10
```

```
matrix(1, 0) = 20
```

```
matrix(1, 2) = 30
```

```
Console.WriteLine("first row, first column : " & matrix(0, 0))
```

```
Console.WriteLine("second row, first column : " & matrix(1, 0))
```

```
Console.WriteLine("second row, second column : " & matrix(1, 1))
```

```
Console.WriteLine("third row, third column : " & matrix(1, 2))
```

```
Console.ReadLine()
```

```
End Sub
```

Using Pre-Release Material in Programming

In order to answer practical questions based on pre-release material, you will need to practise the skills you have learnt so far. The pre-release material will arrive a few months before your examination; you can discuss it with your teacher and your fellow students.

You need to practise applying your skills to the tasks mentioned in the scenario, which is different for each examination series.

Here is a checklist of useful things to do:

1. Read through the pre-release material several times. Check with your teacher if there is anything at all that you do not understand.
2. For each task, write an algorithm using both pseudocode and a flowchart to show what is required.
3. Choose sets of **test data** that you will need to use, and work out the expected results. Remember to use normal, boundary and erroneous data. Be able to give reasons for your choice of test data.
4. Complete trace tables to test your pseudocode and flowcharts. This will enable you to ensure that both the pseudocode and the flowcharts work properly. It is a good idea to get another student to trace your algorithms as well.
5. Decide which works best for each task, pseudocode or a flowchart, and why.
6. Before starting to write your program for each task:
 - a. decide the **variables**, including **any arrays**, and **constants** you will need
 - b. decide the **data types** required for these
 - c. decide the **meaningful names** you will use
 - d. be able to explain your decisions.
7. If you are asked to repeat the same thing many times, for example adding up totals, complete the task for one and check it works before repeating it many times.
8. Write and test each task. You can use the same test data as you used for your **pseudocode** and **flowcharts**.