***Visual Basic***

**Console.** A console program has no graphics. It is text. Easy to develop, it uses few resources and is efficient.   
**These programs,** however, will readily accomplish an analytical or processing \task. We invoke WriteLine and Write. We can use ReadLine for input. Numbers and strings are handled.  
**An example.** This program uses Console.WriteLine. It prints "Hello world" to the screen. The program is contained in a module named Module1. The Sub Main is the entry point of the program.

Imports System

Module Module1

'This program will display Hello World

Sub Main()

Console.WriteLine("Hello World")

Console.ReadKey()

End Sub

End Module

Data Types Available in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Storage Allocation** | **Value Range** |
| Boolean | Depends on implementing platform | **True** or **False** |
| Byte | 1 byte | 0 through 255 (unsigned) |
| Char | 2 bytes | 0 through 65535 (unsigned) |
| Date | 8 bytes | 0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999 |
| Decimal | 16 bytes | 0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal |
| Double | 8 bytes | -1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values  4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values |
| Integer | 4 bytes | -2,147,483,648 through 2,147,483,647 (signed) |
| Long | 8 bytes | -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed) |
| Object | 4 bytes on 32-bit platform  8 bytes on 64-bit platform | Any type can be stored in a variable of type Object |
|  |  |  |
|  |  |  |
| Single | 4 bytes | -3.4028235E+38 through -1.401298E-45 for negative values;  1.401298E-45 through 3.4028235E+38 for positive values |
| String | Depends on implementing platform | 0 to approximately 2 billion Unicode characters |

Module DataTypes

Sub Main()

Dim b As Byte

Dim n As Integer

Dim si As Single

Dim d As Double

Dim da As Date

Dim c As Char

Dim s As String

Dim bl As Boolean

b = 1

n = 1234567

si = 0.12345678901234566

d = 0.12345678901234566

da = Today

c = "U"c

s = "Me"

If ScriptEngine = "VB" Then

bl = True

Else

bl = False

End If

If bl Then

'the oath taking

Console.Write(c & " and," & s & vbCrLf)

Console.WriteLine("declaring on the day of: {0}", da)

Console.WriteLine("We will learn VB.Net seriously")

Console.WriteLine("Lets see what happens to the floating point variables:")

Console.WriteLine("The Single: {0}, The Double: {1}", si, d)

End If

Console.ReadKey()

End Sub

End Module

# Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

|  |  |
| --- | --- |
| **Type** | **Example** |
| Integral types | SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char |
| Floating point types | Single and Double |
| Decimal types | Decimal |
| Boolean types | True or False values, as assigned |
| Date types | Date |

## Variable Declaration in VB.Net

The **Dim** statement is used for variable declaration and storage allocation for one or more variables.

Some valid variable declarations along with their definition are shown here:

Dim StudentID As Integer

Dim StudentName As String

Dim Salary As Double

Dim count1, count2 As Integer

Dim status As Boolean

Dim exitButton As New System.Windows.Forms.Button

Dim lastTime, nextTime As Date

## Variable Initialization in VB.Net

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

variable\_name = value;

for example,

Dim pi As Double

pi = 3.14159

You can initialize a variable at the time of declaration as follows:

Dim StudentID As Integer = 100

Dim StudentName As String = "Bill Smith"

## Example

Try the following example which makes use of various types of variables:

Module variablesNdataypes

Sub Main()

Dim a As Short

Dim b As Integer

Dim c As Double

a = 10

b = 20

c = a + b

Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c)

Console.ReadLine()

End Sub

End Module

## Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

Dim message As String

message = Console.ReadLine

The following example demonstrates it:

Module variablesNdataypes

Sub Main()

Dim message As String

Console.Write("Enter message: ")

message = Console.ReadLine

Console.WriteLine()

Console.WriteLine("Your Message: {0}", message)

Console.ReadLine()

End Sub

End Module

# Constants

## Declaring Constants

In VB.Net, constants are declared using the **Const** statement.

For example,

'The following statements declare constants.'

Const maxval As Long = 4999

Public Const message As String = "HELLO"

Private Const piValue As Double = 3.1415

## Example

The following example demonstrates declaration and use of a constant value:

Module constantsNenum

Sub Main()

Const PI = 3.14149

Dim radius, area As Single

radius = 7

area = PI \* radius \* radius

Console.WriteLine("Area = " & Str(area))

Console.ReadKey()

End Sub

End Module

# Statements

A **statement** is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as:

* **Declaration statements** - these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
* **Executable statements** - these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.
* Declaration Statements
* The declaration statements are used to name and define procedures, variables, properties, arrays, and constants. When you declare a programming element, you can also define its data type, access level, and scope.
* The programming elements you may declare include variables, constants, enumerations, classes, structures, modules, interfaces, procedures, procedure parameters, function returns, external procedure references, operators, properties, events, and delegates.
* Following are the declaration statements in VB.Net:

|  |  |  |
| --- | --- | --- |
| **S.N** | **Statements and Description** | **Example** |
| 1 | **Dim Statement**  Declares and allocates storage space for one or more variables. | Dim number As Integer  Dim quantity As Integer = 100  Dim message As String = "Hello!" |
| 2 | **Const Statement**  Declares and defines one or more constants. | Const maximum As Long = 1000  Const naturalLogBase As Object  = CDec(2.7182818284) |
| 3 | **Enum Statement**  Declares an enumeration and defines the values of its members. | Enum CoffeeMugSize  Jumbo  ExtraLarge  Large  Medium  Small  End Enum |
| 4 | **Class Statement**  Declares the name of a class and introduces the definition of the variables, properties, events, and procedures that the class comprises. | Class Box  Public length As Double  Public breadth As Double  Public height As Double  End Class |
| 5 | **Structure Statement**  Declares the name of a structure and introduces the definition of the variables, properties, events, and procedures that the structure comprises. | Structure Box  Public length As Double  Public breadth As Double  Public height As Double  End Structure |
| 6 | **Module Statement**  Declares the name of a module and introduces the definition of the variables, properties, events, and procedures that the module comprises. | Public Module myModule  Sub Main()  Dim user As String =  InputBox("What is your name?")  MsgBox("User name is" & user)  End Sub  End Module |
| 7 | **Interface Statement**  Declares the name of an interface and introduces the definitions of the members that the interface comprises. | Public Interface MyInterface  Sub doSomething()  End Interface |
| 8 | **Function Statement**  Declares the name, parameters, and code that define a Function procedure. | Function myFunction  (ByVal n As Integer) As Double  Return 5.87 \* n  End Function |
| 9 | **Sub Statement**  Declares the name, parameters, and code that define a Sub procedure. | Sub mySub(ByVal s As String)  Return  End Sub |
| 10 | **Declare Statement**  Declares a reference to a procedure implemented in an external file. | Declare Function getUserName  Lib "advapi32.dll"  Alias "GetUserNameA"  (  ByVal lpBuffer As String,  ByRef nSize As Integer) As Integer |
| 11 | **Operator Statement**  Declares the operator symbol, operands, and code that define an operator procedure on a class or structure. | Public Shared Operator +  (ByVal x As obj, ByVal y As obj) As obj  Dim r As New obj  ' implemention code for r = x + y  Return r  End Operator |
| 12 | **Property Statement**  Declares the name of a property, and the property procedures used to store and retrieve the value of the property. | ReadOnly Property quote() As String  Get  Return quoteString  End Get  End Property |
| 13 | **Event Statement**  Declares a user-defined event. | Public Event Finished() |
| 14 | **Delegate Statement**  Used to declare a delegate. | Delegate Function MathOperator(  ByVal x As Double,  ByVal y As Double  ) As Double |

* Executable Statements
* An executable statement performs an action. Statements calling a procedure, branching to another place in the code, looping through several statements, or evaluating an expression are executable statements. An assignment statement is a special case of an executable statement.
* **Example**
* The following example demonstrates a decision making statement:
* Module decisions
* Sub Main()
* 'local variable definition '
* Dim a As Integer = 10
* ' check the boolean condition using if statement '
* If (a < 20) Then
* ' if condition is true then print the following '
* Console.WriteLine("a is less than 20")
* End If
* Console.WriteLine("value of a is : {0}", a)
* Console.ReadLine()
* End Sub
* End Module
* When the above code is compiled and executed, it produces the following result:
* a is less than 20;
* value of a is : 10
* Arithmetic Operators
* Following table shows all the arithmetic operators supported by VB.Net. Assume variable **A** holds 2 and variable **B** holds 7, then:
* [Show Examples](http://www.tutorialspoint.com/vb.net/vb.net_arithmetic_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| ^ | Raises one operand to the power of another | B^A will give 49 |
| + | Adds two operands | A + B will give 9 |
| - | Subtracts second operand from the first | A - B will give -5 |
| \* | Multiplies both operands | A \* B will give 14 |
| / | Divides one operand by another and returns a floating point result | B / A will give 3.5 |
| \ | Divides one operand by another and returns an integer result | B \ A will give 3 |
| MOD | Modulus Operator and remainder of after an integer division |  |

Comparison Operators

Following table shows all the comparison operators supported by VB.Net. Assume variable **A** holds 10 and variable **B** holds 20, then:

[Show Examples](http://www.tutorialspoint.com/vb.net/vb.net_comparison_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Checks if the values of two operands are equal or not; if yes, then condition becomes true. | (A = B) is not true. |
| <> | Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true. | (A <> B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true. | (A <= B) is true. |

Assignment Operators

There are following assignment operators supported by VB.Net:

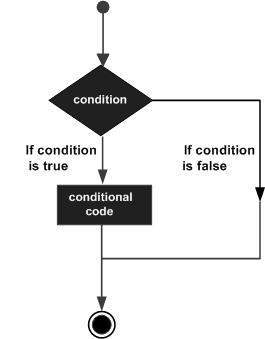
[Show Examples](http://www.tutorialspoint.com/vb.net/vb.net_assignment_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand | C -= A is equivalent to C = C - A |
| \*= | Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division) | C /= A is equivalent to C = C / A |
| \= | Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (Integer division) | C \= A is equivalent to C = C \A |
| ^= | Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand. | C^=A is equivalent to C = C ^ A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| &= | Concatenates a String expression to a String variable or property and assigns the result to the variable or property. | Str1 &= Str2 is same as  Str1 = Str1 & Str2 |

# Decision Making

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



VB.Net provides the following types of decision making statements. Click the following links to check their details.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| [**If ... Then statement**](http://www.tutorialspoint.com/vb.net/vb.net_if_statements.htm) | An **If...Then statement** consists of a boolean expression followed by one or more statements. |
| [**If...Then...Else statement**](http://www.tutorialspoint.com/vb.net/vb.net_if_else_statements.htm) | An **If...Then statement** can be followed by an optional **Else statement**, which executes when the boolean expression is false. |
| [**nested If statements**](http://www.tutorialspoint.com/vb.net/vb.net_nested_if_statements.htm) | You can use one **If** or **Else if** statement inside another **If** or **Else if** statement(s). |
| [**Select Case statement**](http://www.tutorialspoint.com/vb.net/vb.net_select_case_statements.htm) | A **Select Case** statement allows a variable to be tested for equality against a list of values. |
| [**nested Select Case statements**](http://www.tutorialspoint.com/vb.net/vb.net_nested_select_case_statements.htm) | You can use one **select case** statement inside another **select case** statement(s). |

[**If Then**](https://www.dotnetperls.com/if-vbnet) Else

**Info:**We see if the user typed "1" or "2" and pressed return. We also display the output.

**VB.NET program that uses ReadLine**

Module Module1

Sub Main()

While True

' Read value.

Dim s As String = **Console.ReadLine**()

' Test the value.

If s = "1" Then

Console.WriteLine("One")

ElseIf s = "2" Then

Console.WriteLine("Two")

End If

' Write the value.

Console.WriteLine("You typed " + s)

End While

End Sub

End Module

**Output**

1

One

You typed 1

2

Two

You typed 2

3

You typed 3

## Example:

Module decisions

Sub Main()

Dim a As Integer = 100

If (a = 10) Then

Console.WriteLine("Value of a is 10")

ElseIf (a = 20) Then

Console.WriteLine("Value of a is 20")

ElseIf (a = 30) Then

Console.WriteLine("Value of a is 30")

Else

Console.WriteLine("None of the values is matching")

End If

Console.WriteLine("Exact value of a is: {0}", a)

Console.ReadLine()

End Sub

End Module

When the above code is compiled and executed, it produces the following result:

None of the values is matching

Exact value of a is: 100

# VB.Net - Select Case Statement

A **Select Case** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.

## Syntax:

The syntax for a Select Case statement in VB.Net is as follows:

Select [ Case ] expression

[ Case expressionlist

[ statements ] ]

[ Case Else

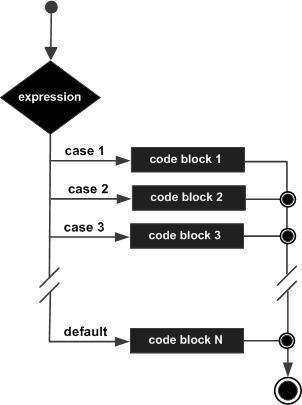
[ elsestatements ] ]

End Select

Where,

* ***expression***: is an expression that must evaluate to any of the elementary data type in VB.Net, i.e., Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort.
* ***expressionlist***: List of expression clauses representing match values for *expression*. Multiple expression clauses are separated by commas.
* ***statements***: statements following Case that run if the select expression matches any clause in *expressionlist*.
* ***elsestatements***: statements following Case Else that run if the select expression does not match any clause in the *expressionlist* of any of the Case statements.

## Flow Diagram:



## Example:

Module Module1

Sub Main()

Dim grade As Char

Console.writeline(“Enter Grade”)

grade = CChar(console.readline())

Select grade

Case "A"

Console.WriteLine("Excellent!")

Case "B", "C"

Console.WriteLine("Well done")

Case "D"

Console.WriteLine("You passed")

Case "F"

Console.WriteLine("Better try again")

Case Else

Console.WriteLine("Invalid grade")

End Select

Console.WriteLine("Your grade is " & grade)

Console.ReadLine()

End Sub

End Module

When the above code is compiled and executed, it produces the following result:

Well done

Your grade is B

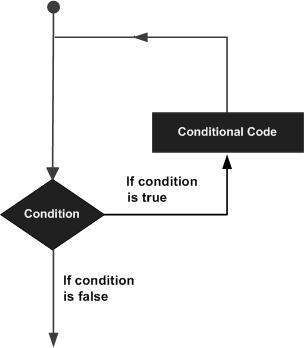
# Example Program - Boolean operators *- NOT, AND, OR* Module Module1 Sub Main() Dim age, points As Integer Console.WriteLine("What is your age?") age = Int(Console.ReadLine()) Console.WriteLine("How many points do you have on your licence?") points = Int(Console.ReadLine()) If age > 16 And points < 9 Then Console.WriteLine("You can drive!") Else Console.WriteLine("You are not eligable for a driving licence") End If Console.ReadKey() End Sub End Modul

# Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



VB.Net provides following types of loops to handle looping requirements. Click the following links to check their details.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| [**Do Loop**](http://www.tutorialspoint.com/vb.net/vb.net_do_loops.htm) | It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement. |
| [**For...Next**](http://www.tutorialspoint.com/vb.net/vb.net_fornext_loops.htm) | It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes. |
| [**For Each...Next**](http://www.tutorialspoint.com/vb.net/vb.net_foreachnext_loops.htm) | It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection. |
| [**While... End While**](http://www.tutorialspoint.com/vb.net/vb.net_while_loops.htm) | It executes a series of statements as long as a given condition is True. |
| [**With... End With**](http://www.tutorialspoint.com/vb.net/vb.net_with_statement.htm) | It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure. |
| [**Nested loops**](http://www.tutorialspoint.com/vb.net/vb.net_nested_loops.htm) | You can use one or more loops inside any another While, For or Do loop. |

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

VB.Net provides the following control statements. Click the following links to check their details.

|  |  |
| --- | --- |
| **Control Statement** | **Description** |
| [**Exit statement**](http://www.tutorialspoint.com/vb.net/vb.net_exit_statement.htm) | Terminates the **loop** or **select case** statement and transfers execution to the statement immediately following the loop or select case. |
| [**Continue statement**](http://www.tutorialspoint.com/vb.net/vb.net_continue_statement.htm) | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| [**GoTo statement**](http://www.tutorialspoint.com/vb.net/vb.net_goto_statement.htm) | Transfers control to the labeled statement. Though it is not advised to use GoTo statement in your program. |

Module loops

Sub Main()

' local variable definition

Dim a As Integer = 10

'do loop execution

Do

Console.WriteLine("value of a: {0}", a)

a = a + 1

Loop Until (a = 20)

Console.ReadLine()

End Sub

End Module

Module loops

Sub Main()

Dim a As Byte

' for loop execution

For a = 10 To 20

Console.WriteLine("value of a: {0}", a)

Next

Console.ReadLine()

End Sub

End Module

Module loops

Sub Main()

Dim a As Integer = 10

' while loop execution '

While a < 20

Console.WriteLine("value of a: {0}", a)

a = a + 1

End While

Console.ReadLine()

End Sub

End Module

# Nested Loops

Example:

The following program uses a nested for loop to find the prime numbers from 2 to 100:

Module loops

Sub Main()

' local variable definition

Dim i, j As Integer

For i = 2 To 100

For j = 2 To i

' if factor found, not prime

If ((i Mod j) = 0) Then

Exit For

End If

Next j

If (j > (i \ j)) Then

Console.WriteLine("{0} is prime", i)

End If

Next i

Console.ReadLine()

End Sub

End Module

When the above code is compiled and executed, it produces the following result:

2 is prime

3 is prime

5 is prime

7 is prime

11 is prime

13 is prime

17 is prime

19 is prime

23 is prime

29 is prime

31 is prime

37 is prime

41 is prime

43 is prime

47 is prime

53 is prime

59 is prime

61 is prime

67 is prime

71 is prime

73 is prime

79 is prime

83 is prime

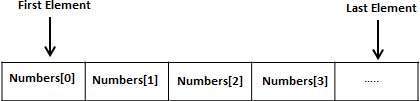
89 is prime

97 is prime

# Arrays

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



## Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

Dim intData(30) ' an array of 31 elements

Dim strData(20) As String ' an array of 21 strings

Dim twoDarray(10, 20) As Integer 'a two dimensional array of integers

Dim ranges(10, 100) 'a two dimensional array

You can also initialize the array elements while declaring the array. For example,

Dim intData() As Integer = {12, 16, 20, 24, 28, 32}

Dim names() As String = {"Karthik", "Sandhya", \_

"Shivangi", "Ashwitha", "Somnath"}

Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

Module arrayApl

Sub Main()

Dim n(10) As Integer ' n is an array of 11 integers '

Dim i, j As Integer

' initialize elements of array n '

For i = 0 To 10

n(i) = i + 100 ' set element at location i to i + 100

Next i

' output each array element's value '

For j = 0 To 10

Console.WriteLine("Element({0}) = {1}", j, n(j))

Next j

Console.ReadKey()

End Sub

End Module

When the above code is compiled and executed, it produces the following result:

Element(0) = 100

Element(1) = 101

Element(2) = 102

Element(3) = 103

Element(4) = 104

Element(5) = 105

Element(6) = 106

Element(7) = 107

Element(8) = 108

Element(9) = 109

Element(10) = 110

## Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim**statement.

Syntax for ReDim statement:

ReDim [Preserve] arrayname(subscripts)

Where,

* The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
* **arrayname** is the name of the array to re-dimension.
* **subscripts** specifies the new dimension.

Module arrayApl

Sub Main()

Dim marks() As Integer

ReDim marks(2)

marks(0) = 85

marks(1) = 75

marks(2) = 90

ReDim Preserve marks(10)

marks(3) = 80

marks(4) = 76

marks(5) = 92

marks(6) = 99

marks(7) = 79

marks(8) = 75

For i = 0 To 10

Console.WriteLine(i & vbTab & marks(i))

Next i

Console.ReadKey()

End Sub

End Module

When the above code is compiled and executed, it produces the following result:

0 85

1 75

2 90

3 80

4 76

5 92

6 99

7 79

8 75

9 0

10 0

re-release material May/June 2015  
Here is a copy of the pre-release material  
Task1  
A school keep records of the weights of each pupil. The weight in kilograms of each  
pupil is recorded on the first day of term. Input and store the weights and names  
recorded for a class of 30 pupils. You must store the weights in a one-dimensional array  
and the names in another one-dimensional array. All the weights must be validated on  
entry and any invalid weights rejected. You must decide your own validation rules. You  
must assumethatthepupilsnamesareunique.Outputthenamesandweightsofthepupilsin the  
class.  
Task2  
The weight in kilograms of each pupil is recorded again  
onthelastdayofterm.Calculateandstorethedifferenceinweightforeachpupil.  
Task3  
Forthosepupilswhohaveadifferenceinweightofmorethan2.5kilograms,output,  
withasuitablemessage,thepupil’sname,thedifferenceinweightandwhetherthisis rise or a  
fall. Your program must include appropriate prompts for the entry of data.  
Errormessagesandotheroutputsneedtobesetoutclearlyandunderstandably.All variables,  
constants and other identifiers must have meaningful names. Each task must be fully  
tested.

Coding for the given tasks  
Module Module1  
Sub Main()  
Dim Name(30) As String  
Dim Count As Integer  
Dim Weight1(30) As Single  
Const Upper\_Limit As Single = 500  
Const Lower\_Limit As Single = 5  
'Task 1  
 For Count = 1 To 30

Console.WriteLine("Student No. : "& Count)  
 Console.Write("Enter name : ")  
 Name(Count) = Console.ReadLine()  
 Console.Write("Enter Weight at day 1 of term ")  
 Weight1(Count) = Console.ReadLine()

'Validation Check for Weight  
 While Weight1(Count) < Lower\_Limit Or Weight1(Count) > Upper\_Limit  
 Console.WriteLine("Error: Invalid weight. It must be between 5 and 500")  
 Console.Write("Re-enter weight on first day ")  
 Weight1(Count) = Console.ReadLine()  
 EndWhile

Next

'For Displaying list of name and weight of students  
 For Count = 1 To 5  
 Console.WriteLine(Name(Count) &" "& Weight1(Count))  
 Next

‘Task 2  
 Dim weight2(30), Weight\_Difference(30) AsSingle  
 For Count = 1 To 30  
 Console.WriteLine(Count & " " & Name(Count) & " " & Weight1(Count))  
 Console.Write("Enter weight on last day ")  
 weight2(Count) = Console.ReadLine()  
 'Validation Check for Weight  
 While weight2(Count) < Lower\_Limit Or weight2(Count) > Upper\_Limit  
 Console.WriteLine("Error: Invalid weight. It must be between 5 and 500")  
 Console.Write("Re-enter weight on lastt day ")  
 weight2(Count) = Console.ReadLine()  
 EndWhile  
 Weight\_Difference(Count) = weight2(Count) - Weight1(Count)

Next  
'Task 3  
 For Count = 1 To 30  
 If Weight\_Difference(Count) > 2.5 Then  
 Console.WriteLine(Name(Count) & " has a rise in weight of " & Weight\_Difference(Count) & " kg")  
 ElseIf Weight\_Difference(Count) < -2.5 Then  
 Console.WriteLine(Name(Count) &" has a fall in weight of "& Weight\_Difference(Count) &" kg")  
 EndIf  
 Next  
Console.ReadKey()  
EndSub  
EndModule

Pre-release material  
A teacher needs a program to record marks for a class of 30 students who have sat three computer  
science tests.  
Write and test a program for the teacher.  
• Your program must include appropriate prompts for the entry of data.  
• 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 – Set up arrays  
Set-up one dimensional arrays to store:  
• Student names  
• Student marks for Test 1, Test 2 and Test 3  
o Test 1 is out of 20 marks  
o Test 2 is out of 25 marks  
o Test 3 is out of 35 marks  
• Total score for each student  
Input and store the names for 30 students. You may assume that the students’ names are unique.  
Input and store the students’ marks for Test 1, Test 2 and Test 3. All the marks must be validated on  
entry and any invalid marks rejected.  
 TASK 2 – Calculate  
Calculate the total score for each student and store in the array.  
Calculate the average total score for the whole class.  
Output each student’s name followed by their total score.  
Output the average total score for the class.  
 TASK 3 – Select  
Select the student with the highest total score and output their name and total score.

Following are the questions and pseudocodes from chapter 10 of your book, make their Visual Basic programs. ( A sample VB program is given for question no. 8)

Q1: Show two ways of selecting different actions using Pseudocode.

Ans:

Pseudocode

**If Condition**

Begin

Input marks

If marks >= 60

Then Print "passed"

Else Print "failed"

End If

End

**Case Statement**

Begin

Input Marks  
  
Case Marks of  
    Case =100  
        Print “Perfect Score”  
    Case > 89  
       Print “Grade = A”  
    Case > 79  
       Print “Grade = B”  
    Case > 69  
       Print “Grade = C”  
    Case > 59  
    Print “Grade = D”  
    Otherwise  
       Print “Grade = F”  
End Case

End

Or

Begin

Input grade

CASE  grade  OF  
                “A”       : points = 4  
                “B”       : points = 3  
                “C”       : points = 2  
                “D”       : points = 1  
                “F”       : points = 0

Otherwise : print “ invalid grade”

ENDCASE

Output points

End

**Q: Input two numbers and a mathematical symbol from user and output total.**

* **Total should be the sum of two numbers if the mathematical symbol is “+”**
* **Total should be the difference of two numbers if the mathematical symbol is “-”**
* **Total should be the product of two numbers if the mathematical symbol is “\*”**
* **Total should be the coefficient of two numbers if the mathematical symbol is “/”**
* **Else display “invalid operator”**

Pseudocode

Begin

Input num1, num2, symbol

CASE  symbol  OF  
                “+”       : total = num1 + num2  
                “-”       : total = num1 - num2

“\*”       : total = num1 \* num2

“/”       : total = num1 / num2  
 otherwise : print “Invalid symbol”

ENDCASE

Output total

End

Q2: You have been asked to choose the correct routine from the menu shown below.

1. Decide which type of conditional statement are you going to you use.
2. Explain your choice.
3. Write the Pseudocode
4. Select your test data and explain why you choose each value.

Answer:

1. I am using Case Statement
2. because it is very simple and relevant to use in the given scenario.
3. Pseudocode:

Pseudocode

Begin

Input Choice

Case Choice of

1 : SetUpNewAccount;

2 : MakeChangesToAnExistingAccount;

3: CloseAnAccount;

4 : ViewMyOrders;

5 : PlaceANewOrder;

6 : AlterAnExistingOrder;

0 : Exit;

H : Help;

End Case

End

Q3: Show three ways to use a loop to add up five numbers and print out the total can be set up using Pseudocode. Explain which loop is the most efficient to use.

Pseudocode

Answer:

There are three different loop structures that we can use to add five numbers.

1. By Using For Loop

Begin

Sum=0

For Count = 1 to 5

Input Num

Sum = Sum + Num

Next Count

Output “Total = ”, Sum

End

1. By Using Repeat Until Loop

Begin

Sum=0

Count = 0

Repeat

Input Num

Sum = Sum + Num

Count = Count + 1

Until Count = 5

Output “Total = ”, Sum

End

1. By Using While Do EndWhile Loop

Begin

Sum=0

Count = 0

While Count < 5 Do

Input Num

Sum = Sum + Num

Count = Count + 1

EndWhile

Output “Total = ”, Sum

End

Q4: A sweets shop sells five hundred different types of sweets. Each sort of sweet is identified by a different four digit code. All sweets that start with 1 are Chocolates, All sweets that start with 2 are toffees, All sweets that start with 3 are jellies and all other sweets are miscellaneous and can start with any other digit except zero.

1. Write an algorithm, using a flowchart or Pseudocode which input the four digit code for all 500 items and output the number of chocolates, toffees and jellies.
2. Explain how you would test your flow chart.
3. Decide the test data to use and complete a trace table showing a dry run of your flow chart.

Answer:

Pseudocode

Begin

TotalChocolate = 0

TotalToffees = 0

TotalJellies = 0

For Count = 1 to 500

Input Code

If Code >= 1000 And Code <=1999

Then TotalChocolate = TotalChocolate + 1

Else

If Code >= 2000 And Code <=2999

Then TotalToffees = TotalToffees + 1

Else

If Code >= 3000 And Code <=3999

Then TotalJellies = TotalJellies + 1

End If

End If

End If

Next Count

Output “Total Number Of Chocolates :” , TotalChocolate

Output “Total Number Of Toffees :” , TotalToffees

Output “Total Number Of Jellies :” , TotalJellies

End

Q5: The temperature in an apartment must be kept between 18⁰C and 20⁰C. If the temperature reaches 22⁰C then the fan is switched On; If the temperature reaches 16⁰C then the heater is switched On; otherwise the fan and the heater are switched Off. The following library routines are available:

* GetTemperature
* FanOn
* FanOff
* HeaterOn
* HeaterOff

Write an algorithm using Pseudocode or flow chart, to keep the temperature at the right level.

Pseudocode

Begin

Temperature = GetTemperature;

If Temperature >= 22

Then FanOn;

Else

If Temperature <= 16

Then HeaterOn;

Else

FanOff;

HeaterOff;

End If

End If

End

Q6: Daniel lives in Italy and travels to Mexico, India and New Zealand. The time difference are:

|  |  |  |
| --- | --- | --- |
| Country | Hours | Minutes |
| Mexico | -7 | 0 |
| India | +4 | +30 |
| New Zealand | +11 | 0 |

Thus, If it is 10:15 in Italy it will be 14:45 in India.

1. Write an algorithm which:

* Inputs the name of the country
* Inputs the time in Italy in hours and in minutes
* Calculate the time in the country input using the data from the table
* Output the country and the time in hours and in minutes.

1. Describe with examples two sets of test data you would use to test your algorithm.

a)

Begin

Input Country, Hours, Minutes

If Country = “Mexico”

Then Hours = Hours - 7

Else

If Country = “India”

Then Hours = Hours + 4

Minutes = Minutes + 30

If Minutes > = 60

Minutes = Minutes – 60

Hours = Hours + 1

End If

Else

If Country = “New Zealand”

Then Hours = Hours + 11

End If

End If

End If

End

Q7: A school is doing a check on the heights and weights of the students. The school has 1000 students. Write a Pseudocode and program in VB, which:

* Input height and weight of all 1000 students
* Output the average height and weight
* Include any necessary error traps for the input

Pseudocode

Begin

TotalWeight =0

TotalHeight =0

For x= 1 to 1000

Repeat

Input height, weight

Until (height > 30) and (height < 80) and (weight > 30 ) and ( weight < 100)

TotalWeight = TotalWeight + weight

TotalHeight = TotalHeight + height

Next

AverageHeight = TotalHeight / 1000

AverageWeight = TotalWeight / 1000

Output “ Average height of the students is : ”, AverageHeight

Output “ Average weight of the students is : ”, AverageWeight

End

Q8: A small café sells five types of items:

Bun $0.50

Coffee $1.20

Cake $1.50

Sandwich $2.10

Dessert $4.00

Write a program, which

* Input every item sold during the day
* Uses an item called “end” to finish the day’s input
* Adds up the daily amount taken for each type of item
* Outputs the total takings ( for all items added together ) at the end of the day
* Output the item that had the highest takings at the end of the day

Pseudocode

Begin

Tbun =0

Tcoffee =0

Tcake =0

Tsandwich = 0

Tdessert =0

HighestTaking = 0

Repeat

Input Item, quantity

Case Item of

“bun” : Tbun = Tbun + quantity

“coffee” : Tcoffee = Tcoffee + quantity

“cake” : Tcake = Tcake + quantity

“sandwich” : Tsandwich = Tsandwich + quantity

“dessert” : Tdessert = Tdessert + quantity

Otherwise Output “ Enter relevant product ”

End Case

Until Item = “End”

TotalTakings = Tbun + Tcoffee + Tcake + Tsandwich + Tdessert

Output “The total takings of the whole day” , TotalTakings

If (Tbun > HighestTaking) Then

HighestTaking = Tbun

Item = “Bun”

End If

If (Tcoffee > HighestTaking) Then

HighestTaking = Tcoffee

Item = “Coffee”

End If

If ( Tcake > HighestTaking) Then

HighestTaking = Tcake

Item = “Cake”

End If

If ( Tsandwich > HighestTaking) Then

HighestTaking = Tsandwich

Item = “Sandwich”

End If

If (Tdessert > HighestTaking) Then

HighestTaking = Tdessert

Item = “Dessert”

End If

Output “The item which has the highest sales today is : ” , Item

End

VB program

Module Module1

Sub Main( )

Dim Tbun, Tcoffee, Tcake, Tsandwich, Tdessert, quantity, TotalTakings, HighestTaking As Integer

Tbun =0

Tcoffee =0

Tcake =0

Tsandwich = 0

Tdessert =0

Dim Item As String

Do

Console.writeline ( “Enter the item in lower case only”)

Item = console.readline( )

Console.writeline ( “Enter its quantity”)

quantity = Int(console.readline( ))

Select Item

Case “bun”

Tbun = Tbun + quantity

Case “coffee”

Tcoffee = Tcoffee + quantity

Case “cake”

Tcake = Tcake + quantity

Case “sandwich”

Tsandwich = Tsandwich + quantity

Case “dessert”

Tdessert = Tdessert + quantity

Case Else

Console.writeline(“ Enter relevant product ”)

End Select

Loop Until ( Item = “End” )

TotalTakings = Tbun + Tcoffee + Tcake + Tsandwich + Tdessert

Console.writeline(“The total takings of the whole day” & TotalTakings)

If (Tbun > HighestTaking) Then

HighestTaking = Tbun

Item = “Bun”

End If

If (Tcoffee > HighestTaking) Then

HighestTaking = Tcoffee

Item = “Coffee”

End If

If ( Tcake > HighestTaking) Then

HighestTaking = Tcake

Item = “Cake”

End If

If ( Tsandwich > HighestTaking) Then

HighestTaking = Tsandwich

Item = “Sandwich”

End If

If (Tdessert > HighestTaking) Then

HighestTaking = Tdessert

Item = “Dessert”

End If

Console.writeline(“The item which has the highest sales today is : ” & Item)

Console.readkey( )

End Sub

End Module

Q9: 5000 numbers are being input which should have either one digit, two digits, three digits or four digits. Write an algorithm which:

* Input 5000 numbers
* Output how many numbers have one digit, two digits, three digits and four digits.
* Output the percentage of numbers which were outside the range.

Pseudocode

Begin

OneDigit = 0

TwoDigit = 0

ThreeDigit = 0

FourDigit = 0

OutSide = 0

For Count = 1 to 500

Input Number

If Number >= 0 And Number <=9

Then OneDigit = OneDigit + 1

Else

If Number >= 10 And Number <=99

Then TwoDigit = TwoDigit + 1

Else

If Number >= 100 And Number <=999

Then ThreeDigit = ThreeDigit + 1

Else

If Number >= 1000 And Number <=9999

Then FourDigit = FourDigit + 1

Else

OutSide = OutSide + 1

End If

End If

End If

End If

Next Count

Percentage = OutSide / 5000 \* 100

Output “Total Number Of One Digit Numbers :” , OneDigit

Output “Total Number Of Two Digit Numbers :” , TwoDigit

Output “Total Number Of Three Digit Numbers :” , ThreeDigit

Output “Total Number Of Four Digit Numbers :” , FourDigit

Output “Percentage of numbers outside the range” , Percentage

End