CS608 Lecture Notes

Visual Basic.NET Programming

Introduction to visual Basic.NET

Review of VB.NET Basic Language Components

(Part II of IV)

(Lecture Notes 1B)

Prof. Abel Angel Rodriguez

Chapter 3 Programming Fundamentals

3.1 Code Writing Basics (Syntax)

3.1.1 Visual Basics Code Statements

- A Statement is simply an executable program instruction.
- A program is simply a list of Executable Statements that are created using the language code syntax

Assignment Statement (=)

- Assignment statement is used to assign values to a property of an object or variables
- The assignment statement operates from right to left.
- That is the Item appearing on the right-side of the equal sign is assigned to the Item on the left-side of the equal sign.
- Syntax:

Item = Item

Example:

- **Graphical Controls using Dot Operator (.)**

- Assigning a value to a Control Property (Run Time):
  Object.Property = value
  Example: *lblTitle.text = "Terminator 3"*

- Retrieving value from a Control Property (Run Time):

  value = Object.Property
  Example: *value = lblTitle.text*

- Assigning content of one control to another:

  Object2.Property = Object1.Property
  Example: *lblTitle.text = txtName.text*

Remark or Comment Statement (')

- The Remark or Comment statement is used for documentation only. Comments are not executable statements.
- Good programming practice dictates that programmers use remarks to clarify and explain their code.
- Syntax:

'Comment or Remark

Example:

- **Comments**

  'This Project was written by Joe Smith

  'This section of code repeats itself

3.1.2 Introduction to Object-Oriented Related Statements

- Lets look at some additional object related statemet:

Class Statement Declaration

- A ***Class*** is a template or blueprint that define what object of the class look like
- A ***Class*** is a plan or template that specifies what ***Properties***, ***Methods*** and ***Events*** that will reside in ***objects***
- We will go into details on creating a class in the future, for now lets just look at the basic Syntax for declaring a class:

```
Public Class ClassName

'Class Body
'Class code is entered inside class body
'Properties, Methods & Event-Procedures here

End Class
```

Example:

- **Creating a Classes:**

- Example 1 - Creating a Video Class:

```
Public Class Video
'Properties, Methods & Event-Procedures here

End Class
```

- Example 2 - Creating a class for a Form Object:

```
Public Class Form1
'Properties, Methods & Event-Procedures here

End Class
```

- **Note that inside the body of a Class is where you find all properties, methods & event-procedures for that object.**

- **If the class happens to be a Form Class, then all Control Objects placed in the Form will be members of the Class and their code will reside inside the body as well.**

Object Statement Declaration

- Once we create a class, we need to then create objects of the class.
- We will cover creating class objects in details in later sections. For now we will look at on method as an introduction.
- Note that creating an object of the class is simply declaring a variable in addition to using the keyword ***New***.
- Syntax:

Dim| Public | Private *ObjectName* **As** *ClassName* **= New** *ClassName***()**

Example:

- Assuming we have previously defined a Class named clsCustomer. Creating a Customer object of the class clsCustomer is as follows:

  Dim objCustomer **As** clsCustomer **= New** clsCustomer

Property & Method Statement Declaration

- Syntax for using an object **Properties** is based on the dot operator:

Object. *Property*

Example:

- Assuming we have previously created a Customer Object:

  objCustomer*.FirstName = "Joe"*

- Syntax for using an object **Methods** uses the dot operator as well:

Object. *Method()*

Example:

- Assuming we have previously created a Customer Object:

  objCustomer*.PurchaseProduct()*

With Block Statement

- There are times when several code statements are being applied to the same Object.
- As we have already learned, the basic mechanism for accessing and setting Objects is the Dot (.) Operator. Using the Dot Operator we listed the basic Object statement as follows:

```
'Object basic Statements Using Dot Operator
Object.Property
Object.Method()
Object.Event()
```

- When several statements are being applied to the same object, the Object's name will be repeated for every statement and it can become redundant.
- For example, supposed we wanted to set the following properties to a Button Control named btnExit: text, Enable, TabStop, and TabIndex. The code would be as follows:

```
btnExit.Text = "E&xit"
btnExit.Enable = True
btnExit.TabStop = True
btnExit.TabIndex = 1
```

- Note that the Control btnExit is repeated for every statement. In this case is not big deal, but when you need to populate many properties on an Object as well as Method() etc., repeating the name of the control in each statement can become tedious.
- Visual Basics provides a statement that allows us to group of block a set of code that pertains to an Object.
- This statement is called the **With/End With** Statement. This statement works in conjunction with the Dot Operator as well.
- The syntax is as follows:

```
'With Block Statement
With ObjectName
        .Property or Method
       .Property or Method
      .etc…

End With
```

- All statements residing in the body of the With Statement before the End With relate to the Object named on the With header.

Example:

- **Using the With/End With Statement:**

  - Example 1 – *Assigning value to button*:

```
With btnExit
    .Text = “E&xit”
    .Enable = True
    .TabStop = True
    .TabIndex = 1
End With
```

  - Example 2 – *Assigning value to a Customer Object and execute methods*:

```
With Customer
    .FirstName = “Joe”
    .LastName = “Smith”
    .IDNumber = 111
    .BirthDate = 12/12/65
    .RentVideo()    ‘Calling Method of Object
End With
```

  - Example 3 – *Retrieving values from Customer Object*:

```
With Customer
    t = .FirstName
    Value2 = .LastName”
    .IDNumber
    .BirthDate
    .RentVideo()    ‘Calling Method of Object
End With
```

3.2 VB.NET Language Components

3.2.1 Introduction

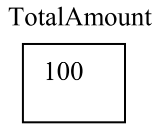
- Programming Language tools and components refers to the items and components that make up the language syntax (rules) required for us to write code.
- Program code is usually composed of :
  1) **Data:** The ***variables*** or data structures that a program requires to perform the processing
  2) **Methods:** Actions taken by an Object
  3) **Events:** Actions taken upon the Object by the User
  4) **Controls:** Controls Objects such as Text boxes, Buttons, Listboxex, Label, etc.
  5) **Programming Language Elements:** Decision-making statement, looping & branching.

3.2.2 Variables & Memory

- All programming languages provide mechanism for a programmer to store a variety of different types of ***data*** or ***information*** as well as the different format this data can be as.
- In addition programming languages provide more sophisticated data storage mechanism for more complex data storage. This is known as a ***Data Structures***.
  - ***Data Structure:*** A mechanism or structure for storing simple and complex data
  - The simplest Mechanism provided by a programming language for data storage is the ***Variable***
- Variables are **memory** locations that are assigned a **name**.
- Every variable has:
  1. **Name:** *A variable can be any name, as long as is not a reserved word* or special words used by the language
  2. **Type:** *The type of data the variable holds, for example integer, character, decimal point numbers, etc,.*
  3. **Size:** *The number of bytes or binary numbers that it hold in memory.*
  4. **Value:** *The actual data value assigned to the variable.*

Memory Concepts

- Variables are stored in particular places in the computer's memory.
- The data stored in memory is stored in a unit called Byte. A Byte is an 8 bit binary number. Example : 10011100
- When a variable is given a value that value is actually stored in the memory location set aside for that variable name.
- Values placed in a memory location, replaces the previous value that was stored there. The previous value is destroyed.



Declarations and Definitions

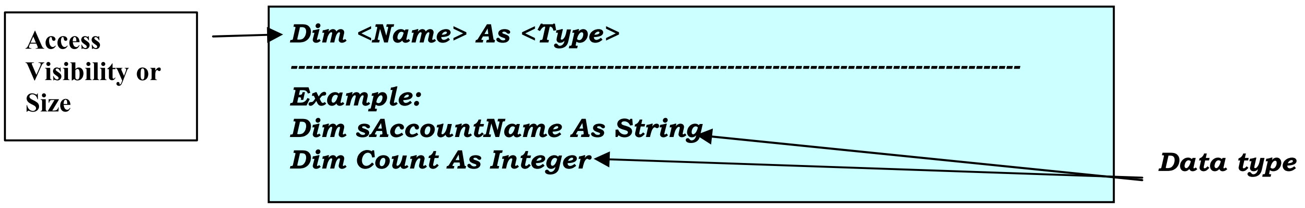
- Using Variables is a two step process:
  1. ***Declaration:*** Introduces the variable name into a program
  2. ***Definition:*** Memory is actually created or reserved for the variable
  3. ***Initialize:*** Populate a variable with data

- Variables are declared in the following locations:
  - Forms header or top of form code section
  - Inside Class Modules & Standard Modules
  - Inside Methods & Event-Handlers

- The declaring of a variable includes the ***name***, ***type*** of data that is stored, ***size*** of the data stored and its ***access visibility*** or who can see it (*Scope*).
- The basic variable Declaration Syntax:

```
'Basic variable declaration syntax
Access_Visibility  name As Type

Where:
- Access_Visibility:  Dim, Public or Private
- Name: Name of variable or memory location
- Type:  The type of data stored in this memory location
```



```
Dim <Name> As <Type>
-----------------------------------------------------------------------------------------------
Example:
Dim sAccountName As String
Dim Count As Integer
```

VB.NET Primitive Data Types

- When variables are declared and created, you need to specify the type of data that the variable will store.
- VB.NET contains nine Primitive Data Types. They are called primitive because they are the basic data types and are part of the VB.NET language NOT the Framework Library (More on this later)
- The basic Primitive Data Types for VB are listed in the table below:

	Data Type	Storage in Bytes	Used for
Numeric with no Decimals	Byte	1	0 to 255 binary data
	Short	2	Small integers in the range of -32,768 to 32,767
	Integer	4	Whole numbers in the range -2,147,483,648 to 2,147,483,647
	Long	8	Large whole numbers
Numeric with Decimals	Single	4	Single-Precision floating point numbers with 6 digits of accuracy
	Double	8	Double-Precision floating point numbers with 14 digits of accuracy
	Decimal	16	Decimal Fractions, such as dollars and cents
Other	Boolean	2	True or False values
	Char	2	Single character

- **IMPORTANT!** Note that with the primitive Data Types when you declare a variables you are also defining it. In other words, with primitive data types, the declaration & definitions are done in one step during the declaration:

Access *Name* ***As*** *Type*

Naming Conventions for Primitive Data Types

- There is a naming convention that has been adopted for naming variables in Visual Basics.NET
- The rule is that the variable name should be prefixed by a three letter characters of the variables data type
- The table below lists the prefixes for the variable naming convention:

Prefix	Data Type
bln	Boolean
dec	Decimal
dbl	Double-Precision floating point
sng	Single-Precision floating point
int	Integer
lng	Long Integer

Examples of Declaring and Assigning Variables

- The following are examples or declaring, initializing and assigning variables:

Example:

- Example 1: Declaring a Integer variable
  Dim *intCount* As **Integer**

- Example 2: Declaring a Decimal variable
  Dim *decTotalAmount* As **Decimal**

- Example 3: Declaring a Boolean variable
  Dim *blnAcccessGranted* As **Boolean**

- You can assign or retrieve values to variables:

Example:

- Example 1: Declaring and assigning values to string variables
  Dim *intCount* As **Integer**
  Dim *decTotalAmount* As **Decimal**
  Dim *intTotalCount* As **Integer**

  intCount = 150
  decTotalAmount = 350.00

- Example 2: Accessing value from one variable to another

  intTotalCount = intCount

- Example 1: Declaring and assigning values in one step
  Dim *intCount* As **Integer** = 150

Constant Variables

- ❑ Constant variables hold their value **indefinitely**. Constant variable ***cannot be changed***.
- ❑ Constant variable syntax:

```
'Basic variable declaration syntax
Conts  name As Type = value
```

```
Const <Name> As Datatype = Value
==================================================
Example:
Const SIZE As Integer = 10
```

- ❑ Examples:

Example:

- Example 1: Declaring a Constant Interger variable
  Const *inMaxCount* As **Integer** = 10

- Example 2: Declaring a Constant Decimal variable
  Dim *decSales_Tax_Rate* As **double** = 8.25

Scope (Visibility) & Lifetime

- **Lifetime:** refers to how long the variable lives while the program is running.
- **Visibility or Scope:** refers to who can see and access the variable.
- The scope of the variables declared in Visual Basics.NET fall in the following category:

  - **Global variables –** Accessible and visible by the entire project
    - These variables are declared at the declaration section of Forms and Module & Classes. (more on this in future lectures)

  - **Modular-Level variables –** Accessible from all procedures of a Forms (Can be seen by all code inside a Form including Methods & Event-Procedures)
    - Variables that need to be seen by all code in a Form or Standard Module or Class Module.
    - Module-Level variables are declared using **Dim**, **Const**, or **Private**
    - These variables are declared in the Declaration Section of a Form
    - Naming convention for Module Level variables is that you append the prefix letter **m** or **m_** before the name of the variable.

      Example: Dim m_decTotalPay As Decimal
      or
      Example: Dim mdecTotalPay As Decimal

  - **Local variables –** Only seen within the a Method or Procedure (Method Procedure or Event-Procedure) in which it was declared
    - Variables declared inside Procedures are Local
    - Local variables are declared using **Dim**

Variable Access/Visibility Indicator

- When declaring variables you can use the specified **Dim**, **Public** or **Private**
- **Dim:** Means Dimension or specifies that memory is to be reserved.   Use this statement when creating most local or module level variables that should only be seen from with Forms & Procedures.

- **Public:** is code that is accessible throughout the entire program or application. A variable declared to be public can be seen by the entire project. Declare variable using public when you want the variable to be accessible from anywhere within the project

- **Private Code:** is code that is accessible only within the Object (Form or Module) in which it was created.   Declare variables as Private only when you don't want the entire project to see the variable.

- The table below shows the lifetime and visibility of variables:

Declaration used	Where it is declared	Lifetime (how long it lives)	Visibility or Scope (who can access)
Dim	Header of Form	As long as program runs	Any code on the form only
Private	Header of Form	As long as program runs	Any code on the form only
Public	Header of Form	As long as program runs	Any code anywhere on the project, provided the variable is prefixed with the form name: ex: Form.variablename
Dim	Header of Module	As long as program runs	Any code on the Module
Private	Header of Module	As long as program runs	Any code on the Module
Public	Header of Module	As long as program runs	Any code anywhere on the project
Dim	Procedure, function or Event-Handler	As long as routine runs	Only code in routine

Converting Data Types

- When using variables, there are situations where we need to convert from one data type to another.
- For example supposed we are performing calculations with variables whose data type is Integer, but we want the result of the calculation to be a string data type. In this case we need to convert the result of the calculation from Integer to String.
- The process of converting one data type to another is also known as *Casting*.

Using Built-In Conversion Functions

- A **Function** is a Method that performs an action and returns a value.
- The main idea of a Function is that it perform some process but it RETURNS A VALUE or answer.
- We will be covering methods and functions in more details later in the course.
- Visual Basics provides a set of conversion function to convert the various data types:

Function	Conversion	
CInt(*ExpressionToConvert***)**	Convert expression to Integer	
CStr(*ExpressionToConvert***)**	Convert expression to String	
CDec(*ExpressionToConvert***)**	Convert expression to Decimal	
CDate(*ExpressionToConvert***)**	Convert expression to Date	
CShort(*ExpressionToConvert***)**	Convert expression to Short	
CBool(*ExpressionToConvert***)**	Convert expression to Boolean	
CDbl(*ExpressionToConvert***)**	Convert expression to Double	

Example:

- Example 1: *Converting string value into integer*

  intQuantity = CInt("30") *'note that "30" is a string and not a number*

- Example 2: *Converting string value in TextBox into integer(Usually the case)*

  intQuantity = CInt(txtQuantity.Text) *'The content inside the textbox, a string is converted to an integer number*

- Example 3: *Converting string value in TextBox into Decimal*

  decPrice = CDec(txtPrice.Text) *'The content inside the textbox, a string is converted to a Decimal number*

- Example 4: *Converting Integer into Decimal*

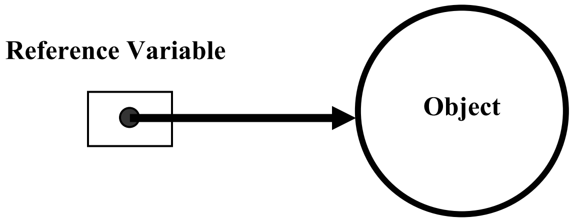
  decDollars = CDec(intDollars) *'The content of the integer variable intDollars is converted to Decimal*

3.2.3 Reference Variables (Important Topic)

- There are actually two types of variables, **Primitive** and **Reference** Variables.
- Reference variables are declared with the data type of a Class.

***Access** Name **As** ClassName*

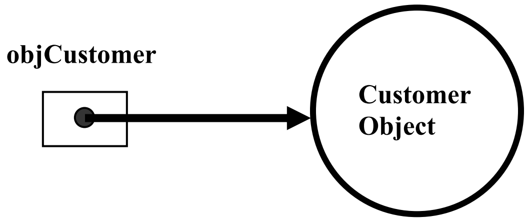
- Reference variables <u>*Point*</u> to an instance or ***Object*** of the class.
- The keyword here is Point!
- A Reference Variable contains no data it is actually what is called a POINTER or Reference Pointer to a Class Object. In reality what is stored in a Reference Variable is a ***Memory Address*** of the Object it is pointing to.
- The Object itself has no name; the name of the object is represented by the reference variable that points to it.
- This is sometimes a difficult concept to grasp. This is best understood by seeing a diagram.
- Reference variable its represented as follows:



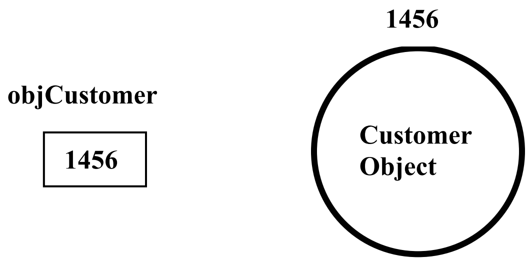
- Note that the Reference Variable contains a Pointer or a memory address to the object it points to.
- So what we have here in reality is a memory location that has a name and contains an address to a memory location of an Object.
- For a better understanding lets look at an example, suppose we have the following reference variable declaration of an Object of the Class *clsCustomer*:

Dim *objCustomer* **As** *clsCustomer*

  - The pictorial diagram is as follows:



  - For those who still have problem understanding this concept, here is a more realistic representation, here we assume that the number 1456 is an address of the starting memory location of an object:



  - Note that the content of the variable *objCustomer* is an address 1456 or a pointer to that location. What ever operations you perform on the variable *objCustomer*, you are actually performing to the object whose address is 1456. This is called indirect access.

❖ **IMPORTANT**! Note that when creating reference variables, the Declaration & Definition are two separate steps

3.2.4 Option Explicit & Option Strict

- VB provides two options that can significantly change the behavior of the editor and the compiler. These two options are **Option Explicit** & **Option Strict**.

Option Explicit

- When **Option Explicit** is **OFF**, you can create and use variables without declaring them. In other words you simply add your variables to a program without declaring them:
- When **Option Explicit** is **ON**, you MUST declare every variable using the declaration syntax prior to using them.

Example with Option Explicit ON:
```
Dim Z As Integer
Dim X As Integer
Dim Y As Integer

Z = X + Y
```
- ❖ With Option Explicit ON, you need to declare the variables before using them

Example with Option Explicit OFF:
```
Z = X + Y
```

- ❖ With Option Explicit OFF, you don't need to declare the variables before using them, simply use variables as is. No need to declare them.
- ❖ Option Explicit OFF is not good programming practice and is inefficient since the compiler does not know the type of size of the variables and simple chooses more memory than required for variables.

Option Strict

- When **Option Strict** is **ON**, the VB.NET language becomes a strong-typed language like C++, Java & C#. Strong-Typed means that the compiler is very strict on how you use the data type. Integer variables must be manipulated with integer variables etc.
- When **Option Strict** is **OFF**, the compiler is not as strict when it comes to manipulating an assigning variable of different data types.

Example with Option Strict OFF:
```
Dim intValue As Integer
Dim decValue As Decimal

decValue = 100.50

intValue = decValue
```

- ❖ Integer variable is assigned to a decimal variable. With Option Strict OFF this is allowed.

Example with Option Strict ON:
```
Dim intValue As Integer
Dim strValue As String

Dim intValue As Integer
Dim decValue As Decimal

decValue = 100.50

intValue = decValue '"ERROR" - Compiler Error. This will not be allowed

'Correct code:
IntValue = CInt(decValue)   ' Decimal value must first be converted to integer
```

- ❖ Note that with Option Strict ON, you cannot assign variables of different data types. You will need to convert these variables to the correct data type prior to assigning.

3.3 Mathematical Calculation

3.3.1 Arithmetic Operators

- Visual Basics.Net provides several Mathematical language components to enable you to program mathematical processes.
- The basic arithmetic operators available in VB are summarize in the following table:

Operator	Name	Example	Behavior
+	Addition	X + Y	Adds x and y
-	Subtraction	X - Y	Subtracts y from x
*	Multiplication	X * Y	Multiplies x and y
/	Division	X / Y	Divides x by y and returns a floating-point result
Mod	Modulo	X Mod Y	Divides x by y and returns the remainder
^	Exponetiation	X ^ Y	Raises x to the power of y
-	Negation	-X	Negates y

Example:

- Example 1: Adding three Integer variables and assigning it to another variable

```
Dim intX  As Integer
Dim intY  As Integer
Dim intZ  As Integer
Dim intResults  As Integer

intResults = intX + intY + intZ
```

- Example 2: Multiplying the three numbers declared in example 1

```
intResults = intX *  intY * intZ
```

Arithmetic Math Class

- In addition to the mathematical operators, VB.NET includes a Math Class in the Standard Library which contain methods to accomplish task such as exponentiation, rounding, and numerous other tasks.
- The following table lists some of the Methods supplied by the Math Class:

Class	Method	Description
Math	**Pow(x,y)**	Returns the value of x raised to the power of y
	Round(x, n)	Returns x rounded to the n decimal
	Sqrt(x)	Returns the square root of x

3.3.2 Comparison or Relational Operators

- A relational operator **compares** two variables. The two variables can be any of the built-in data types such as Integer, Decimal, Single, Double etc.
- The comparison involves relationships such as equal to, less than, and greater than. The result of the comparison is either **TRUE** or **FALSE**.
- Again, the results of comparing these operators are True & False, for example if you compare the following statement: 5 > 8, the result is either True or False. Also the comparison of the following variables and value, **Sum1 = 5** is either True or False, not assigning the value 5 to Sum1, but comparing the variable Sum1 with the number 5.
- Comparison operators are used in if/else and loop statements as we will see later.

- The complete list of VB Comparison operators:

Operator	*Meaning*	*Examples*
>	Greater than	decAmount > decBalance
<	Less than	intSales < 1000
=	Equality	decIncomeTax = decSalesTax
<>	Not equal to	intAge <> 40
>=	Greater than or equal to	intQuantity >=500
<=	Less than or equal to	CInt(txtQuantity.Text) <= 500

3.3.3 Boolean Operators

- Logical operators allow you to combine Boolean (true/false) expressions.
- The basic logical operators in VB.NET are:

Operator	**Syntax**	**Behavior**	**Examples & Explanation**
And	Expr1 And Expr2	Returns TRUE only and only if <u>both</u> expressions are TRUE	*chkUnder18.Checked =True* **AND** *strMovie = "Adult"* *Explanation:* - Only when the checkbox is checked or True and the strMovie variable is equal to "Adult" will this entire statement be TRUE: True And True results in TRUE. Any other combination will yield FALSE
Or	Expr1 Or Expr2	Returns TRUE if Expr1 or Expr2 (or <u>both)</u> are TRUE	*intSum = 100* **OR** *decTax = 8.25* *Explanation:* - When either of the statement are true the results of the entire expression will be True. Only when both of these expressions False will the result be False.
NOt	Not Expr	Returns the inverse of the Expression. Returns TRUE if Expr is FALSE and FALSE if the Expr is TRUE	**Dim** *blnResults* **As Boolean** *blnResults* **= True** **NOT** *Results* *Explanation:* - The result of the *NOT Results* statement is False, since the NOT inverts it and the inverse of a True is False and visa versa.

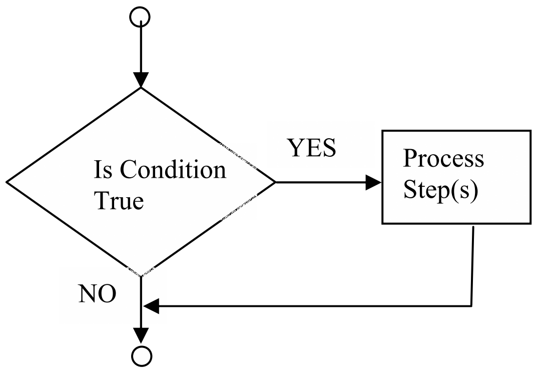
3.4 Decision Making Statements

3.4.1 Introduction

- In programming decision making statements allow you to take different action when a condition is either true of false.
- You use decision making statements to evaluate a condition and if the condition is true a single or group of statements are executed otherwise another single or group of statement is executed.

3.4.2 If…Then Statement

- An If Statement a question or condition is asked, if the answer is Yes or TRUE, one of more instructions are executed, otherwise if the answer is NO or FALSE, the flow of the program continues as normal with the next sequential statement.
- The ***flow chart*** to such a structure looks as follows:



- The ***Pseudo-Code*** to the If/Else Decision Structure is as follows:

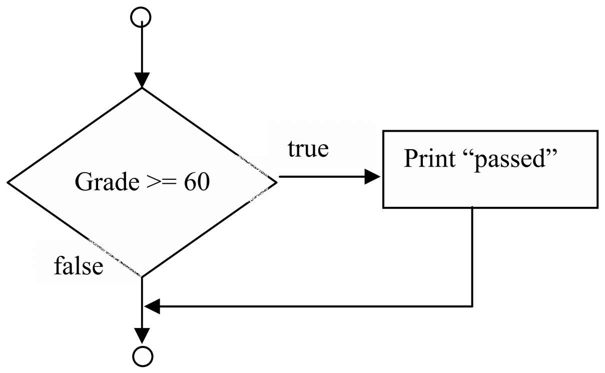
```
If condition is True Then
        Process Step(s) 1
End If
```

Example

- For example, suppose the passing grade on an exam is 60, and we needed a program to determine and print this fact. Part of the algorithm can say:
- Part of the algorithm ***pseudo-code*** would say:

```
If student's grade is greater than or equal to 60 Then
    Print "passed"
End If
```

- The ***flow chart*** illustrating this concept is:

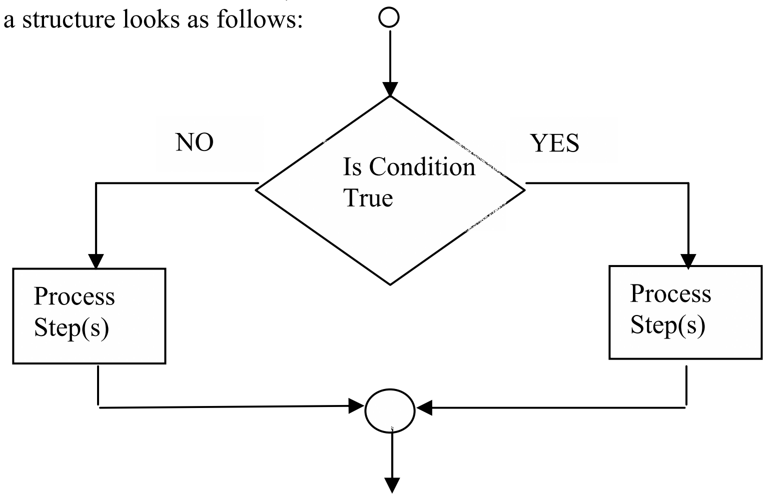


- VB.NET Syntax:

```
Declaration:
If  <Boolean expression> Then
...
...
End If
================================================
Example:
If Password.Text = 111 Then
   MsgBox "Access Granted"
   Unload Form1
End If
```

3.4.3 If…Else Statement

- In an If/Else Decision Structure, a question is asked, if the answer is Yes or TRUE, one of more instructions are executed,otherwise if the answer is NO or FALSE, then another set of instructions are executed.
- The ***flow chart*** to such a structure looks as follows:



- The ***Pseudo-Code*** to the If/Else Decision Structure is as follows:

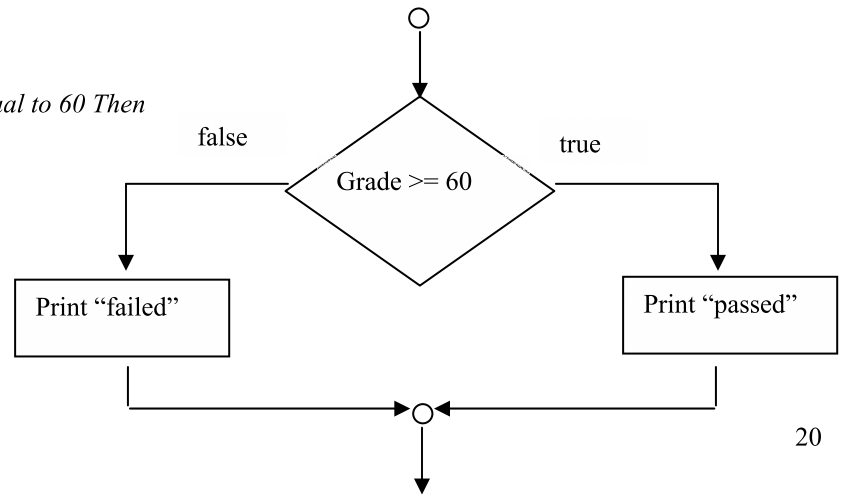
```
If condition is True Then
        Process Step(s) 1
Else
        Process Step(s) 2
End If
```

Example

- Suppose we needed a program that determined whether a passing grade on an exam is a 60 or grater, and we wanted to print whether an exam passed or failed.
- Part of the algorithm ***pseudo-code*** would say:

```
If student's grade is greater than or equal to 60 Then
        Print "passed"
Else
        Print "Failed"
End If
```

- The ***flow chart*** illustrating this concept is:

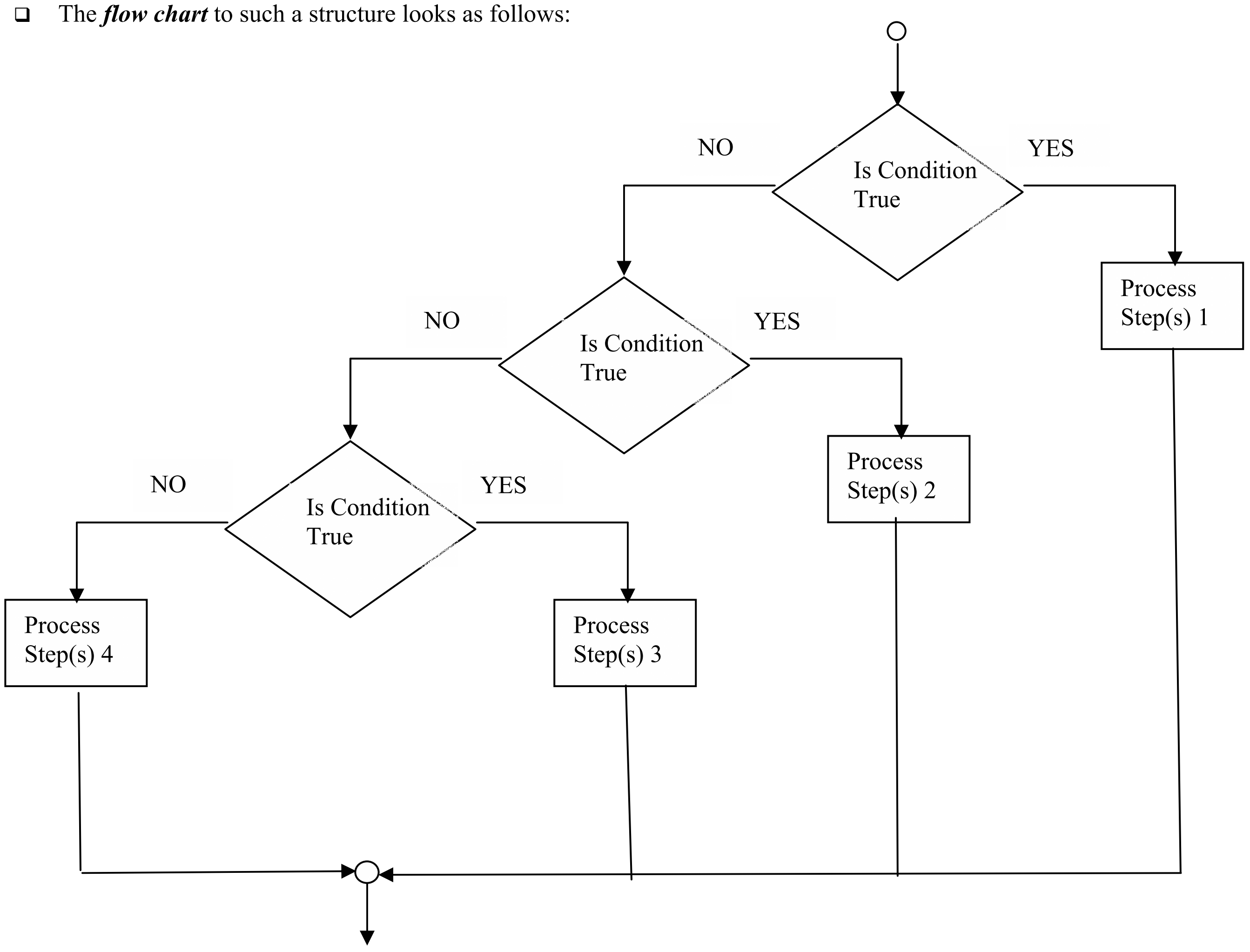


- VB.NET Syntax:

```
Declaration:
If  <Boolean expression> Then
...
...
Else
...
...
End If
====================================================
Example:
If Password.Text = 111 Then
   MsgBox "Access Granted"
  Unload Form1
Else
   MsgBox "Access Denied - Please try again"
End If
```

3.4.4 Nested If…Else Statement

- The **if/else** statements can be nested, in other words we can place if/else statements within if/else statements.
- The ***flow chart*** to such a structure looks as follows:



- The ***Pseudo-Code*** to the If/Else Decision Structure is as follows:

```
If condition is True Then
        Process Step(s) 1

Else if Condition is True Then
        Process Step(s) 2

Else if Condition is True Then
        Process Step(s) 3

Else
        Process Step(s) 4

End If
```

- Note If.. Then.. Else statements can be **nested**.

```
Declaration:
If  <Boolean expression> Then
...
ElseIf
...
ElseIf
..
Else
..
End If
====================================================
Example:
If Password.Text = 111 Then
   MsgBox "Access Granted"
   Unload Form1
ElseIf Password.Text = 222 Then
  MsgBox "Access Granted"
  Unload Form1
ElseIf Password.Text = 333 Then
  MsgBox "Access Granted"
  Unload Form1
Else
  MsgBox "Access Denied - Please try again"
End If
```

3.4.5 Select Case Statement

- Case Statement is another method of implementing nested if/else. But is a more readable and flexible method. Syntax:

```
Declaration:
Select Case <expression>
    Case <expression>
          [Statements]
   Case <expression>
          [Statements]
   Case <expression>
          [Statements]
   Case Else
          [Statements]
End Select
====================================================
Example:
Select Case Password
   Case 111
          MsgBox "Access Granted"
          Unload Form1
  Case 222
          MsgBox "Access Granted"
          Unload Form1
    Case 333, 123, 456
          MsgBox "Access Granted"
          Unload Form1
  Case 444 To 600
          MsgBox "Access Granted"
          Unload Form1
  Case Else
           MsgBox "Access Denied - Please try again"
End Select
```

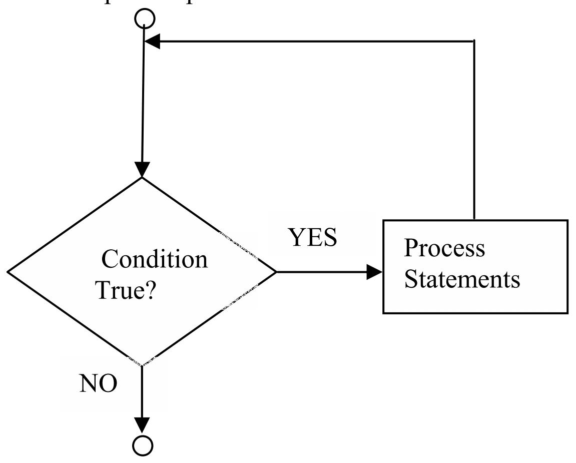
3.5 Loops

3.5.1 Introduction

- Loop structures allow you to repeat a block of code.
- A repetition structure or **loop**, causes a section of your program to be repeated a certain number of times. The repetition continues while a condition is **TRUE**. When the condition is **FALSE** the loop ends and **control passes to the statement following the loop**

    - **One very important factor in a loop structure is that there must be some code and value within the processing Step(s) that must set the condition to the loop to FALSE otherwise the loop will loop forever (Infinite Loop). This is most cases in undesirable**.
    - There are situation in programming where we do want the loop to loop forever. But such situations will not be covered in this course.

- The ***flow chart*** to a Loop Structure looks as follows:



3.5.2 For..Next Loop

- The ***For..Next*** loop executes a section of code (body of loop) a fixed number of times.
- It is usually used when you know, before entering the loop, how many times you want to execute the code.
- Syntax:

```
Declaration:
For CounterVariable = Start To End
        [Statements]
        [Statements]
Next CounterVariable
-----------------------------------------------------------------------------------------------------
For CounterVariable = Start To End [ Step step ]
        [ Statements ]
        [ Exit For ]            'Causes loop to end prematurely
        [ Statements ]
Next CounterVariable

=====================================================
Example:
For nIndex = 1 To 10
        Form1.Print " This Text is repeated 10 Times"
Next nIndex
-----------------------------------------------------------------------------------------------------
For nIndex = 1 To 100 Step 50
        Sum = Sum + 1
Next nIndex
-----------------------------------------------------------------------------------------------------
For nIndex = 1 To 100
        Sum = Sum + 1
        If nIndex = 50 Then
                Exit For
        End If
Next nIndex
```

3.5.3 Do Loops

Do..Loop, Do..Loop While & Do..Loop Until

- The Do-While loops repeatedly executes a section of code (body of loop) until some condition within the body of the loop is satisfied.
- How do we know when to use a Do-While loop in our program? **Ans:** When you don't know how many repetitions are required or how many times to loop.
  - There are 3 types of Do-While loops
    1) **Do..Loop:** Those that loop forever
    2) **Do While… Loop** or **Do…Loop While:** Those that run while a condition is being met
    3) **Do..Until Loop** or **Do…Loop Until:** Those that run until a condition is met

Do..Loop

- The **Do..Loop** loops forever
- Syntax:

```
Declaration:
Do
        [Statements]
        [ Exit Do ]     ''Optional Causes loop to end prematurely
        [Statements]
Loop
=============================================================
Example:
Do
        Form1.Print " This Text is repeated Forever"
Loop
-------------------------------------------------------------------------------------------------------------------
--
Do
        Form1.Print " This Text is repeated Forever"
        ....
        If value = 50 Then     'loop will terminate when condition is met
                Exit Do
        End If

_
```

Do While… Loop

- The **DoWhile… Loop** executes a section of code (body of loop) while a condition is being met.
- The condition is a Boolean expression that **first** is tested, and if the condition is **TRUE** the body of the loop executed.
- NOTE that there must be a condition inside the body of the loop that needs to satisfy the Boolean expression to FALSE in order to terminate the loop.
- Syntax:

```
Declaration:
Do While <Boolean Expression>
        [Statements]
        [ Exit Do ]     'Optional Causes loop to end prematurely
        [Statements]
Loop
=============================================================
Example:
Do While Sum < 100
        Sum = Sum + 1          'Condition terminates the loop when sum =100
Loop
-------------------------------------------------------------------------------------------------------------------
--
Do While Sum < 100
        Sum = Sum + 1
        If Sum = 50 Then       'loop will terminate when condition is met
                Exit Do
        End If
```

Do Loop… While

- The **Do Loop… While** is the same as the Do While… Loop with the exception that the test condition is done after the body of the loop not at the beginning.
- This guarantees that the body of the loop is executed at least ONCE!
- Syntax:

```
Declaration:
Do
        [Statements]
        [ Exit Do ]      'Optional Causes loop to end prematurely
        [Statements]
Loop While <Boolean Expression>
=================================================================
Example:
Do
        Sum = Sum + 1         'Condition terminates the loop when sum =100
Loop While Sum < 100
---------------------------------------------------------------------------------------------------------------------
--
Do
        Sum = Sum + 1
        If Sum = 50 Then       'loop will terminate when condition is met
                Exit Do
        End If
```

Do Until… Loop

- The **DoWhile… Loop** executes a section of code (body of loop) until a condition or Boolean expression is met or TRUE. The loop will execute indefinitely and stop only until the condition is met.
- Again, NOTE that there must be a condition inside the body of the loop that needs to satisfy the Boolean expression to FALSE in order to terminate the loop.
- Syntax:

```
Declaration:
Do Until <Boolean Expression>
        [Statements]
        [ Exit Do ]      'Optional Causes loop to end prematurely
        [Statements]
Loop
=================================================================
Example:
Do Until sPassword = "slick"
        sPassword = InputBox$ ("Enter the password and click OK")
Loop
---------------------------------------------------------------------------------------------------------------------
--
Do Until sPassword = "slick"
        sPassword = InputBox$ ("Enter the password and click OK")

        If sPassword = "Quit" Then
                Exit Do                    'loop will terminate when condition is met
        End If
Loop
```

Do Loop… Until

- The **Do Loop… Until** is the same as the Do Until… Loop with the exception that the test condition is done after the body of the loop not at the beginning.
- This guarantees that the body of the loop is executed at least ONCE!
- Syntax:

```
Declaration:
Do
        [Statements]
        [ Exit Do ]     'Optional Causes loop to end prematurely
        [Statements]
Loop Until <Boolean Expression>
===============================================================
Example:
Do
        sPassword = InputBox$ ("Enter the password and click OK")

Loop Until sPassword = "slick"
-----------------------------------------------------------------------------------------------------------------------------
--
Do
        sPassword = InputBox$ ("Enter the password and click OK")

        If sPassword = "Quit"  Then
                Exit Do                 'loop will terminate when condition is met
        End If
Loop Until sPassword = "slick"
```

3.5.4 BRANCHING:

- Branching is when the program code jumps from one statement to another statement.

GoTo Statement, On Error & Labels

- The ***GoTo*** statement is normally used with an associated ***label*** or a name you assign to a section of code.
- ***GoTo*** allows you to *branch* or *Jump* to a different location within a procedure or method, which happens to be where the **label** is located.
- The Problems with ***GoTo*** is that if used too much, your code could start looking like a real mess with branches all over the place. This is known as "spaghetti code".
- In VB, the ***GoTo*** statement is commonly used with the ***On Error*** Statement when coding error handlers or routines in your program. Error handling routines are code you add to your program to branch or jump on errors to another location in the program identified by a label.
- Use ***GoTo*** with care!
  - Syntax:

```
Declaration:
GoTo <Label>
...
...]
<Label>:
...                'Section of code that GoTo Statement will jump to
----------------------------------------------------------------------------------------------------------------------------------
--
On Error GoTo <Label>
...
...]
<Label>:
...


================================================================
Example:
Private Sub A_SubProcedure()

GoTo AccountingSection
...
...]
AccountingSection:
...                'Section of code that GoTo Statement will jump to
...
End Sub
----------------------------------------------------------------------------------------------------------------------------------
--
Private Sub A_SubProcedure()

On Error GoTo ErrorHandler
...
...]
ErrorHandler:
...                'Section of code that GoTo Statement will jump to
...
```

3.6 More Powerful Data Storage

- So far we have learned the simplest data structure which is the variable.
- But the variable simply is one memory location. Supposed we needed to store data in multiple memory locations?
- Now let's look at some more sophisticated mechanism provided by VB.NET to store complex data formats.

3.6.1 Arrays of Data

- An Arrays is a list data of a single data type.
- The data items grouped in an array can be of any of the data type listed.
- Arrays are a simply a consecutive group of *memory* locations that have the same **name, type** and the following characteristics:
- Each item in an array is called an **element**.
- An array has a fix **size**, which determines the number of *elements*
- Arrays are accessed by **index** numbers, which specifies the location of each *element* in the array.

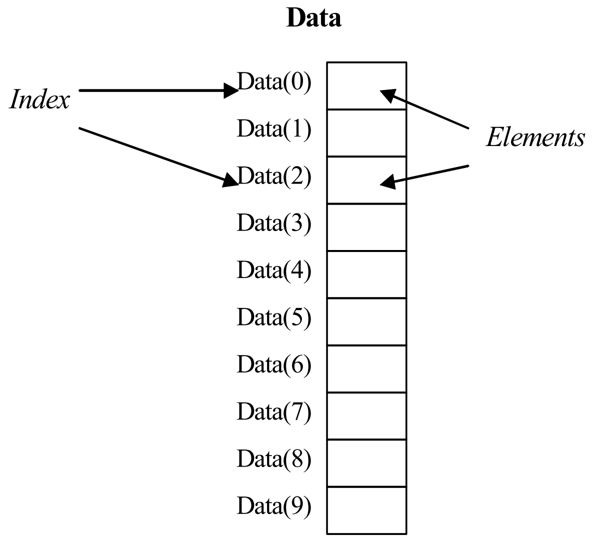
- Visual Basic gives you the ability to create two types of arrays:
- **Static Arrays:** an array of fixed size.
- **Dynamic Arrays:** An array of variable length that is can be changed during run time.

Static One-Dimensional Arrays

- Static Array declaration syntax:

```
 Array Declaration:
Dim <Name> (Size) As <Type>
-------------------------------------------------------------------------------------------
Public <Name>(Size) As <Type>
Private <Name>(Size) As <Type>
====================================================
Example:
Dim Data (9) As Integer  '10 element array
Public data (9) As Single  '10 element array
Private MyArray (10) As Integer '11 elements array
```

- Array Diagram:



- Note that the number within the parentheses is 1-SIZE of the array
- Note how the array has 10 memory cells and begins with an index = 0.

Populating Individual Array Elements

- When you populate data to an array you do via the name of the array and the index. Syntax:

  ArrayName(*index***) =** *value*

- Assuming we have the following array declaration:

  Dim *data(9) As* **Integer**

- Here are some examples of populating elements of the array:

```
Data(0) = 31       ‘adds the integer 2 to the location of index 0
Data(5) = 6        ‘adds the integer 6 to the location of index 5

Data(j) = 31;      ‘j is variable storing index value, if j=2, data[2] will be assigned 31

Data(j + k) =12;   ‘if j=2 and k=4, data[6] is assign 12;
```

Accessing Individual Array Elements

- When you access or retrieve data from an array you do via the name of the array and the index. Syntax:

  value = **ArrayName(***index***)**

- Assuming we have the following array declaration and that the array is populated with the data shown in diagram below:

  Dim *data(9) As* **Integer**

	data
data[0]	31
data[1]	28
data[2]	31
data[3]	30
data[4]	30
data[5]	6
data[6]	12
data[7]	8
data[8]	20
data[9]	15

- Here are some examples of accessing data from the array assuming we have a variable declared named *myIntVar*:

```
myIntVar = data(0)       ‘retrieves the item in location 0; value 31 is assign to variable
myIntVar = data(j)       ‘j is variable storing index value, if j=2, variable = 31
myIntVar  = data(j + k)  ‘if j=2 and k=4, data[6], variable is assign 12
```

Initializing Arrays

- You can give values to each array element when the array is created as follows
- Syntax:

Dim <Name> () As <Type> = { value1, value2, ……}

- Example declaration and results:

int data() As Integer = {31, 28, 31, 30, 30, 6,12, 8, 20,15 }

	data
data[0]	31
data[1]	28
data[2]	31
data[3]	30
data[4]	30
data[5]	6
data[6]	12
data[7]	8
data[8]	20
data[9]	15

Populating All the Array Elements

- To populate all values of an array in one pass, you need to assign the values in each element consecutively. This is done usually by using a loop.
- The For..Next Loop is a excellent mechanism to use with arrays since we know the number of iterations or size of array:

```
For i = 0 to MAXSIZE

     Data(i) = intTestScore

Next
```

Accessing All Array Elements

- You can extract or access all elements of array also using the For loop as follows:

```
For i = 0 to MAXSIZE

     intTestScore = data(i)

Next
```

3.6.2 Multidimensional Arrays

- In VB.Net arrays can have multiple dimensions.
- Multidimensional arrays are used to represent tables of values arranged in rows and columns.

Columns

Rows

	0	1	2	3	4
0	31	28	31	30	30
1	6	12	8	20	15
2	22	16	45	37	20

- In multidimensional arrays we need to specify two subscripts in order to identify an element of the array, one for the row and the other for the column. For example in the above table element (1, 2) contains the value 8, where row =1 and column = 2.
- The diagram above is and example of a two-dimensional array since it has only two dimensions or subscripts.

- Syntax for declaring a two dimensional array:

Dim <Name> (row, column) As <Type>

- Example:

```
int scoreboard (3)(5)     'This declaration is represented by the table above
```

- Multidimensional arrays can also be initialized when defined by using braces to group each Row

int   data (,) = { {31, 28, 31, 30, 30},{ 6, 12, 8, 20, 15}, {22, 16, 45, 37,20} }

- Multidimensional arrays must be displayed or populated using the same methods as single dimensional arrays, except that we must use nested loops, one outer loop for the Column and the inner loop for the Rows
- The syntax, assuming we have declared a variable named Col and a variable name Row, the following declaration:

Dim <Name> (COL_SIZE, ROW_SIZE) As <Type>

```
For Col = 0 to COL_SIZE

      For Row = 0 to ROW_SIZE

      value = Name(Row, Col)  'perform required operation here!


      Next

Next
```

Homework # B

- ❑ Problem statement.

  Develop a class-averaging program that will process the average of a list of grades stored in a list.

Algorithm:

1. ***Problem:*** Calculate and display the grade-point average for a class. Grades are kept on a list.
2. ***Discussion:*** The grade-point average equals the sum of all the grades divided by the number of students. Therefore we need to get each grade and as we do so, add the value to a running total or accumulation of all the grades. This means repetition or loop. Also we need to keep count of the total number of students. At the end we should have a Total Grade and a Total count of students. At this point all we need to do is divide the Total Grade by the Total Student Count and we get the grade point average.
3. ***Input:*** Student Grades obtained from a list of Grades. Values in list are as follow: **80, 79, 100, 95, 45, 55, 75, 93, 100, 100**.
4. ***Processing:*** 1) The Sum of all the Grades; 2) The count or number of grades; 3) Calculate the average = Total Sum of Grade/Grade Count
5. ***Output:*** The Grade-Point Average

Required Results:

1. Create an Algorithm for this problem
2. Create a Console Application and implement this problem. Write a user friendly application. This means the output should display information to the user that makes sense.