

**University of Basrah**

**Electrical Department**

**First Class**

**Visual Basic 6.0**

## 1-Introduction

Hundreds of high-level languages have been developed, but only a few have achieved broad acceptance, for example (**QBASIC, FORTAN and Pascal**). Visual Basic is an example of a graphical-based language. A graphical-based language allows the user to work directly with graphics.

Visual Basic is derived from the “visual” term refers to the method used to create the graphical user interface (**GUI**), Simply drag and drop prebuilt objects into place on screen without having to learn an additional software package. The “Basic” term refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the windows **GUI**. Beginners can create useful applications by professionals to accomplish anything that can be accomplished using any other windows programming language.

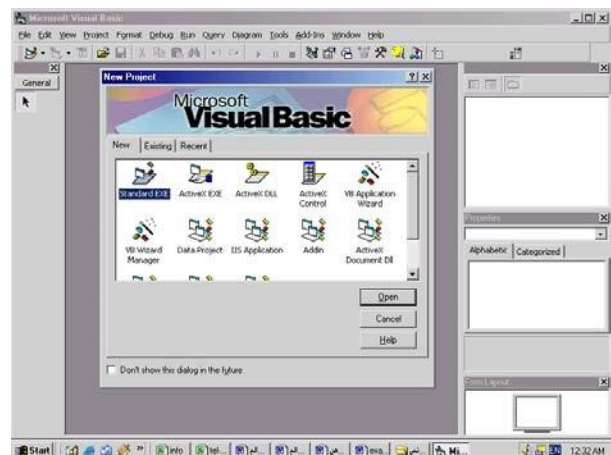
Visual Basic is a Microsoft Windows Programming language. Visual Basic programs are created in an Integrated Development Environmental (**IDE**). The **IDE** allows the programmer to create, run and debug Visual Basic programs conveniently. **IDEs** allows a programmer to create working programs in a fraction of the time that it would normally take to code programs without using **IDEs**.

## 2-Structure of a Visual Basic Application

To run Visual Basic program select, Start> Programs > Microsoft Visual Studio 6.0> Microsoft Visual Basic 6.0 as shown in Fig.(2-1). When Visual Basic is loaded, the **New Project** dialog shown in Fig.(2-2) is displayed.



**Fig.(2-1) Computer screen**



**Fig.(2-2) New Project dialog.**

The New Project dialog allows the programmer to choose what type of Visual Basic program to create. Standard EXE, which is highlighted by default, allows the programmer to create a standard executable. Each type listed in Fig.(2-2) describes a group of related files called a Project.

## 2-1 Project (VBP)

Project is a program designed to user application that may be simple (like calculator program) or complex (like word program). The project types listed in Fig.(2-3) are the “Visual” in Visual Basic, because they contain predefined features for designing Windows programs. The project is a collection of files that makes the user program. They may consist of form, modules, active x controls. The new project dialog contains three tabs

- New: creating new project.
- Existing: opening an existing project.
- Recent: opening a project that has been previously loaded into the IDE.

## 2-2 Elements of Integrated Development Environmental (IDE).

Figure (2-3) shows The IDE after Standard EXE is selected. The top of the IDE window (the title bar) displays “Project1-Microsoft Visual Basic [design]”. The environment consists of various windows when Visual Basic is started (by default):

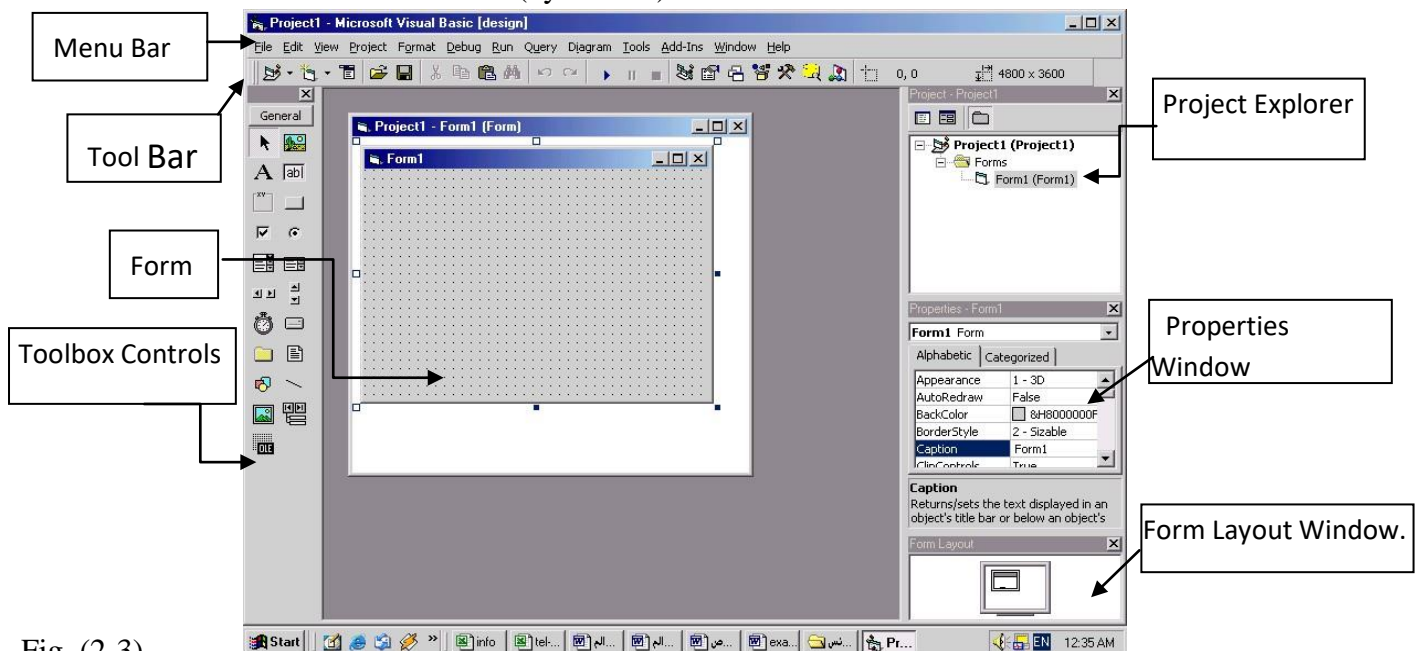


Fig. (2-3)

**2-2-1 Project1-Form1 (Form):** window contains a form named Form1, which is where the program’s Graphical User Interface (GUI) will be displayed. A GUI is the visual portion of the program, this is where the user enters data (called inputs) to the program and where the program displays its results (called outputs). We refer to the Form1 window simply as “**the form**”. Forms are the foundation for creating the interface of an application. You can use the forms to add windows and dialog boxes to your application. You can also use them as container for items that are not a visible part of the application’s interface. For example, you might have a form in your application that serves as a container for graphics that you plan to display in other forms.

**2-2-2 Toolbox Controls:** Contains a collection of tools that are needed for project design as shown in Fig. (2-4). To show the toolbox press View> toolbox icon. The user can place the tool on form, and then work with the tool. To place the tool on form: click on tool>draw tool to form > the tool

appears on form or double click on tool then the tool appears on form. Table (1) summarizes the toolbox

controls.

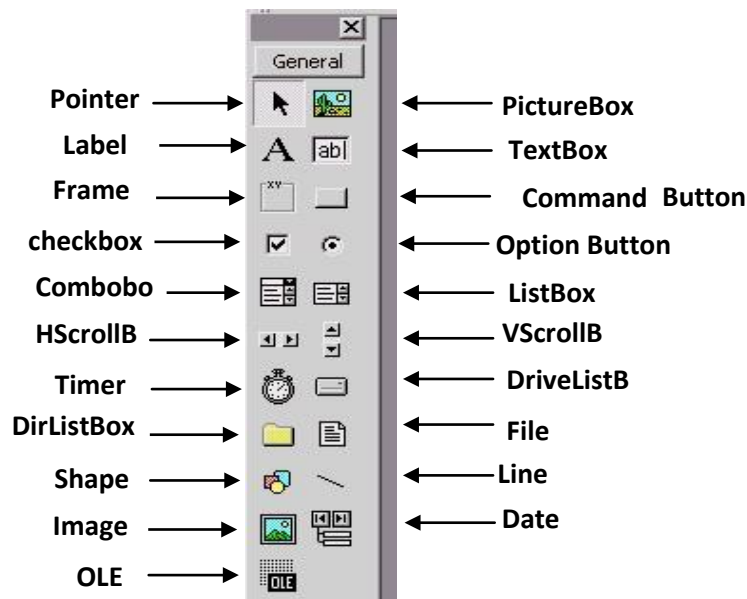


Fig.(2-4) Toolbox Window.

**Table (1): Toolbox controls summary.**

Control	Description
Pointer	Used to interact with controls on the form(resize them, move them, etc.). The pointer is not a control
PictureBox	A control that display images or print the result.
Label	A control that displays uneditable text to the user.
TextBox	A control for accepting user input. Textbox can also display text.
Frame	A control for grouping other controls.
CommandButton	A control that represents a button. The user presses or clicks to initiate an action.
CheckBox	A control that provides the user with a toggle choice (checked or unchecked)
OptionButton	Option buttons are used in groups where only one at a time can be true.
ListBox	A control that provides a list of items.
ComboBox	A control that provides a short list of items.
HscrollBar	A horizontal scrollbar.
VscrollBar	A vertical scrollbar.
Shape	A control for drawing circles, rectangles, squares or ellipse
Line	A control for drawing line.
DrivelistBox	A control accessing the system disk drivers.
DirlistBox	A control accessing directories on a system
Filelistbox	A control accessing file in a directory
Image	A control for displaying images. The images control does not provide as many capabilities as a picturebox.
OLE	A control for interacting with other window applications.
Timer	A control that performs a task at programmer specified intervals. A timer is not visible to the user.

**2-2-3 Properties Window:** The properties window displays the properties for a form or control. Properties are attributes such as size, position, etc. like a form; each control type has its own set of properties. Some properties, like width and height, such as, are common to both forms and controls, while other properties are unique to form or control. Controls often differ in the number

and type of properties. Properties are listed either alphabetically (by selecting the alphabetic tab) or categorically (by selecting the categorized tab). The most important properties of the objects in general are listed in the following table. To show the properties window press View> properties window icon.

Properties name	Objective
Name	Used to represent name of object in code.
Caption	Name appears on object.
Back color	Background color for object.
Fore color	Color of text written on object.
Font	Font style type and size
Visible	The tool is visible or invisible.
Enable	The tool enable or disable
Height	Length of object
Width	Width of object
Top	Coordinates of top of object on screen
Left	Coordinates of left of object on screen
Text	Allows inputting and editing text in object.

**2-2-4 Project Explorer Window:** The window titled Project-Project1 is called the Project Explorer and contains the project files. The project explorer window's tool bar contains three buttons, namely view code, view object and toggle folders. When pressed, the view code button displays a window for writing Visual Basic code. View object, when pressed, displays the form. Double-clicking form1 (form1) also displays the form. The toggle folders button toggles (i.e., alternately hides or shows) the forms folder. The forms folder contains a listing of all forms in the current project. To show the Project Explorer window press View> Project Explorer window icon

**2-2-5 Form Layout Window:** The Form Layout window specifies a form's position on the screen at runtime. The Form Layout window consists of an image representing the screen and the form's relative position on the screen. With the mouse pointer positioned over the form image, drag the form to a new location.

**2-2-6 Menu Bar:** Contains a standard command and specific command like (File, Edit, View, Project, Format, Debug, Run, etc.)

**2-2-7 Tool Bar:** Contains several icons that provide quick access to commonly used features

**2-3 Code Form:** Each standard form has one code form for code. The user can write code in this code form (as a work sheet) in the design stage. This code will be applied at run time.

The code is written in code form and it will be edited quickly by code editor. The codes are of two categories:

- Declaration is written before any procedure in the code.
- Statements. The user selects the required event then code statements are written in side these event procedures.

**2-3-1 Sub Procedures:** A Sub Procedure is a block of code that is executed in response to an event. By breaking the code in a module into Sub procedures, it becomes much easier to find or modify the code in your application. The syntax for a Sub procedure is:

[**Private Sub** procedure name (arguments)

Statements

**End Sub**

**2-3-2 Events:** Events are like electrical switches. The electrical switches are of many types, so are the events. The form and controls support events (generation, interaction with mouse and keyboard). The most important events in Visual Basic are described in the following table.

Event	Action taken when	It provide the following integers
Click	Single click on object.	
DbClick	Double click on object.	
Mouse move	Mouse pointer move object.	Button ,shift ,X,Y
Key press	Pressing a key of the key board.	Key Ascii
DragDrop	Move object to another place.	Source, X, Y

## **2-4 Steps in Developing Application:**

There are three primary steps involved in building a Visual Basic application:

1-**Draw** the user interface

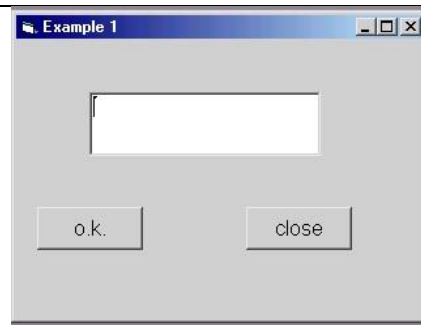
2- **Assign** properties to controls

3- **Attach** code to control

To see how this is done, use the steps in the following procedures to create a simple application for the following example

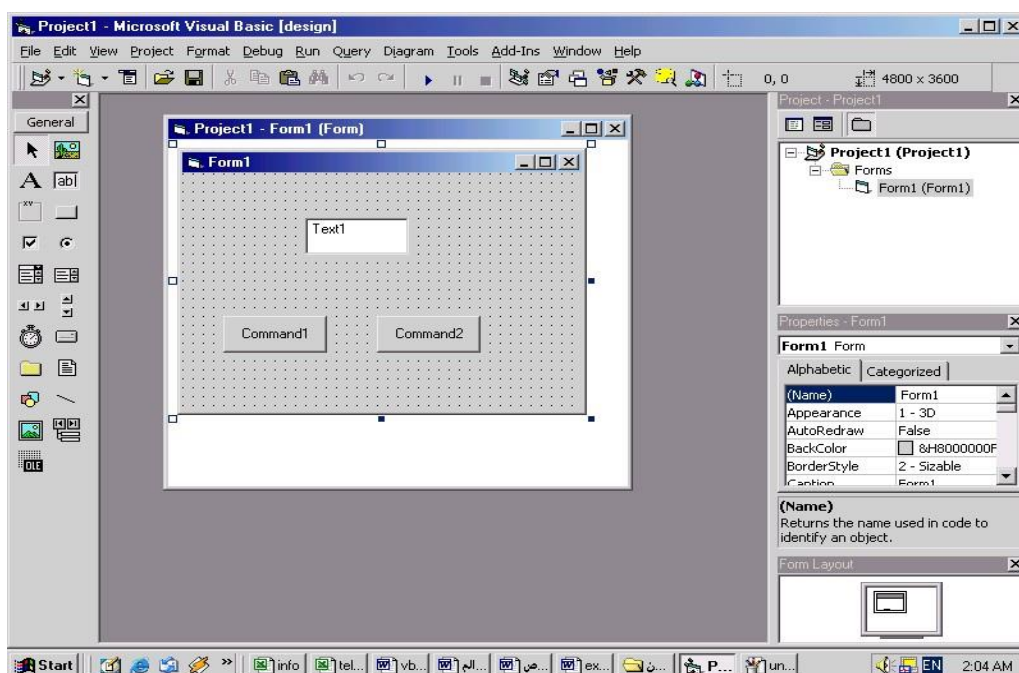
**Example 2-1:** Design a form with one text box and two Commands button. Write a code so when run project and click on command1 (**O.k.**) replace the word (**Welcome**) in text box, and when click on Command2 (**Close**) terminate the program and return back to the form interface.

**Solution:**



❖ **Creating the Interface.**: The first step in building a Visual Basic application is to create the forms that will be the basis for your application's interface. Then you draw the objects that make up the interface on the forms you create.

1. Adding a text box to the form. Double-click the toolbox's textbox to create a text box with sizing handles in the center of the form.
2. Adding a Command Button1 to the form. Click on button and draw button1 to form then the button appears on form.
3. Repeat step 2 to add a Command Button2 to the form.



### ❖ **Setting Properties**

The next step is to set properties for the objects. The properties window provides an easy way to set properties for all objects on a form. For the Example 1, you'll need to change three property setting. Use the default setting for all other properties.

**Note:**

- The **Caption** property determines what is displayed in the form's title bar or what text the controls displays on a form.
- The TextBox's **Text** Property determines what text (if any) the TextBox displays.
- The **Name** property identifies a form or control. It's necessary only for writing code.

Object	Property	Setting
<b>Form1</b>	Name	Form1
	Caption	Example1
	Font	Bold and size 12
<b>Command Button1</b>	Name	Command1
	Caption	O.k.
	Font	Bold and size 12
<b>Command Button2</b>	Name	Command 2
	Caption	Close
	Font	Bold and size 12
<b>TextBox</b>	Name	Text1
	<b>Text</b>	Empty

#### ❖ Writing Code:

The code editor window is where you write Visual Basic code for your application. Code consists of language statements, constants, and declarations. To open the code window, double-click the form or control for which you choose to write code, or from the Project Explorer window, select the name of a form and choose the View code button.

- In the Object list box, select the name of an object in the active form. Or double click of an object.
- In the procedure list box, select the name of an event for the selected object. The Click procedure is the default procedure for a command button and the Load is default procedure for a form.
- An event procedure for a control combines the control's actual name (specified in the name property), an underscore ( \_ ), and the event name. For example (Command1\_click).
- Type the code between the **Sub** and the **End Sub** statements.

Choose the command1 and type the following code:

```
Private Sub Command1_click ( )
```

```
Text1.text="Welcome"
```

```
End Sub
```

Choose the command2 and type the following code:

```
Private Sub Command2_click ( )
```




End

**End Sub**

**Note:** The statement **END** used to close the program runtime.

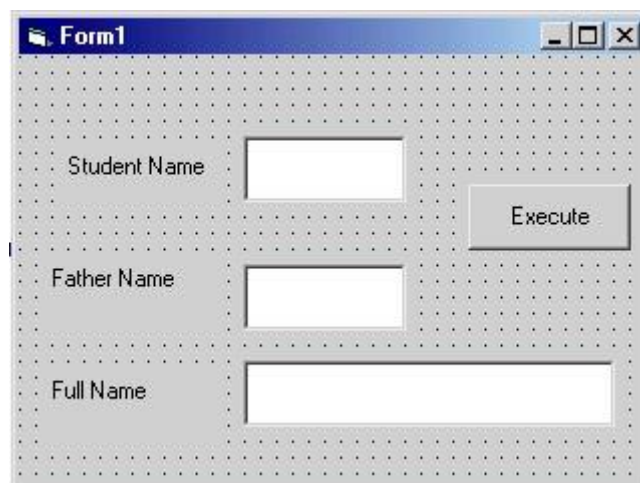
### ❖ Running the Application

To run the application, choose start from the run menu, or click the start button on the toolbar , or **F5** Click the command button (O.k.) and see the “Welcome” displayed in the text box. Click the command button (close) the end the program and return to the form window.

### ❖ Saving a Project

Choosing save project from the file menu. Visual Basic will prompt you separately to save the form and then the project.

**Example 2-2:** Design a form shown in figure below, with three text boxes and one Command Button. Write code in the Command1 (**Execute**). So when run project enter the Student Name in TextBox (Txt1) and the Father Name in TextBox (Txt2). When click on Command1 (**Execute**) replace the Full Name in the TextBox(Txt3).

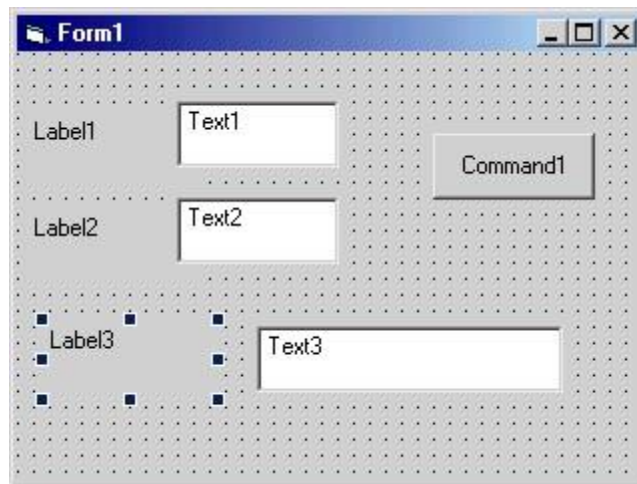


### Solution:

#### ❖ Creating the Interface.:

1. Adding a Label to the form1. Double-click the Label's Label to create a Label with sizing handles in the center of the form1.
2. Repeat step 1 to add Label2 and Label3.

3. Adding a TextBox to the form1. Double-click the toolbox's textbox to create a text box with sizing handles in the center of the form1.
4. Repeat step 3 to add Text2 and Text3.
5. Adding a Command Button1 to the form. Click on button and draw Button to form then the Button1 appears on form1.



#### ❖ Setting Properties

Object	Property	Setting
<b>Form1</b>	Name	Form1
	Caption	Example1
	Font	Bold and size 12
<b>Command Button1</b>	Name	Command1
	Caption	Execute
	Font	Bold and size 12
<b>TextBox1</b>	Name	Txt1
	Text	Empty
<b>TextBox2</b>	Name	Txt2
	Text	Empty
<b>TextBox3</b>	Name	Txt3
	Text	Empty
<b>Label1</b>	Name	Label1
	Caption	Student Name
	Font	Bold and size 12
<b>Label2</b>	Name	Label2
	Caption	Student Name
	Font	Bold and size 12

<b>Label13</b>	Name	Label3
	Caption	Full Name
	Font	Bold and size 12

❖ **Writing Code:**


Choose the command1 and type the following code:

```
Private Sub Command1_click ( )
```

```
txt3.text=txt1.text+ " "+txt2.text
```

```
End Sub
```

❖ **Running the Application**

To run the application, choose start from the run menu, or click the start button on the toolbar ,

or **F5** Click the command button1 (Execute) and see the Full Name displayed in the TextBox3. ❖

**Saving a Project**

Choosing save project from the file menu. Visual Basic will prompt you separately to save the form and then the project.

**3. Fundamentals of programming in Visual Basic****3.1 Data Types (Constant and Variable):**

Data types control of internal storage of data in Visual Basic. There are a number of variable data types that allow you to optimize your code for speed and size.

**1- Boolean:** A variable of type Boolean requires 2 bytes of memory and holds either the value True or False. If boolVar is a Boolean variable, then the statement Print boolVar displays(1) when the value is True and displays (0) when the value is False.

**2- Currency:** The currency data type is extremely useful for calculations involving money. A variable of type Currency requires 8 bytes of memory and can hold any number from  $-9 \times 10^{14}$  to  $9 \times 10^{14}$ .

**3- Date:** A variable of type Date requires 8 bytes of memory and holds numbers representing dates from January 1<sup>st</sup> 100 To December 31<sup>st</sup> 9999. Values of dateVar are displayed in the form month/day/year (for example, 5/12/1996).

**4-Single:** A variable of type Single requires 4 bytes of memory and can hold 0, the numbers from  $1.40129 \times 10^{-45}$  to  $3.40283 \times 10^{38}$  with the most seven significant digits, and the negatives of these numbers.

**5-Double:** A variable of type Double requires 8 bytes of memory and can hold 0, the numbers from  $4.94065 \times 10^{-324}$  to  $1.7976 \times 10^{308}$  with at most 14 significant digits and the negatives of these numbers.

**6-Integer:** A variable of type integer requires 2 bytes of memory and can hold the whole numbers from -32,768 to 32,767.

**7-Long:** A variable of type Long requires 4 bytes of memory and can hold the whole numbers from  $2 \times 10^9$  to  $2 \times 10^9$ .

**8-String:** A variable of type string requires 1 byte of memory per character and can hold a string of up to 32,767 characters, string values are enclosed in quotes. A variable of type String\*n holds a string of n characters, where n is a whole number from 1 to 32,767.

**9-Variant:** A variable of type variant can be assigned numbers, Strings and several other types of data. A variable of type variant requires 16 bytes of memory and can hold any type of data. When values are assigned to a variant variable, Visual Basic keeps track of the “type “of data that has been sorted 9for example, type 2 for integer). By default, Visual Basic uses the variant data type.

### **3.2 Variables:**

In Visual Basic, uses variable for storage values. must start with character and maximum length 255 character and not contain any point.

### **3.3 Declaration of a variable**

The declaration means defining the variable type. The variable has to be declared with the Dim Statement, supplying a name for the variable:

#### **Dim variable name [As type]**

Variables declared with the **Dim** statement within a procedure exist only as long as the procedure is executing. When the procedure finishes the value of the variable disappears. In addition, the value of a variable in a procedure is local to that procedure can't access a variable in one procedure from another procedure.

#### **A variables name:**

- Must begin with letter.
- Can't contain an embedded period or embedded type-declaration character.
- Must not exceed 255 characters. The names of controls, forms, and modules must not exceed 40 characters.
- They can't be the same as restricted keywords (a restricted keyword is a word that Visual Basic uses as part of its language. This includes predefined statements such as “If and Loop”, functions such as “Len and Abs”, and operators such as “Or and Mod”).

The optional as type clause in the Dim statement allows you to define the data type or object type of the variable you are declaring (see sec.3.1).

#### **Examples:**

Dim X As Integer

Dim Balance As Currency

Dim Y As Long

Dim A AS Double, B AS Double

Dim Month AS Date

Dim Max AS Single

Dim Name AS String

Dim Z,V,C

### **Error examples:**

Dim x AS string : Dim A, B, C, X (Repeat the variable name at the same time in two Dim statement)

Dim 1<sup>st</sup> AS date (first character is number)

Dim (Ad#1) AS string (symbol)

Dim MyName.is AS string (point)

Dim Num one AS long (space)

**Note:** The types of variables are used the corresponding suffix shown below in the data type table.

Variable Type	Suffix	Example
Boolean, Variant, and Date	None	---
Integer	%	Dim A %
Long(Integer)	&	Dim Ab&
Single	!	Dim AC!
Double	#	Dim ACC#
Currency	@	Dim AB1 @
String	\$	Dim AA\$

### **3.4 Scoping Variables:**

The Scope of variable defines within parts of program code are aware of its existence. Depending on how it is declared, a variable is scoped as either a procedure-level (local) or module-level variable.

- **Variables used within a procedure:** Procedure-level variables are recognized only in the procedure in which they're declared. These are also known as local variables. You declare them with the Dim or Static keywords. For example:

#### **Dim S as integer Or Static SR as integer**

Values in local variables declared with static exist the entire time your application is running while variables declared with Dim exist only as long as the procedure is executing.

Local variables are a good choice for any kind of temporary calculation. For example, you can create numbers of different procedures containing a variable called (valu1). As long as each valu1 is declared as a local variable, each procedure recognizes only its own version of valu1. Any one procedure can alter the value in its local valu1 without affecting valu1 variables in other procedures.

- **Variables Used within a Module:** By default, a module-level variable is available to all the procedures in that module, but not to code in other modules. You create module-level variables by declaring them with the private keyword in the declaration section at the top of the module. For example:

#### **Private valu1 as integer**

At the module level, there is no difference between private and Dim, but private is preferred because it readily contrasts with public and makes your code easier to understand.

- **Variables used by all modules:** To make a module-level variable available to other modules, use the public keyword to declare the variable. The values in public variables are available to all procedures in your application. Like all module-level variables, public variables are declared in the declarations section at the top module. For example: **Public valu1 as integer**

**Note:** You can't declare public variables within a procedure, only within the declarations section of a module.

- **Declaring All Local Variables as static:** To make all local variables in a procedure static, place the Static keyword at the beginning of a procedure heading. For Example: **Static Function total (num)**  
This makes all the variables in the procedure static regardless of whether they are declared with Static, Dim, and Private. You can place Static in front of any Sub or Function Procedure heading, including event procedures and those declared as Private.

**3.5 Constants:** Constant also store values, but as the name implies, those values remain constant throughout the execution of an application. Using constants can make your code more readable by providing meaningful names instead of numbers. There are a number of built-in constants in Visual Basic. There are two sources for constants:

- System-defined constants are provided by applications and controls. Visual Basic constants are listed in the Visual Basic (VB).
- User-defined constants are declared using the Const statement. It is a space in memory filled with fixed value that will not be changed. For example:

Const X=3.14156	Constant for procedure
Private Const X=3.14156	Constant for form and all procedure
Public Const X=3.14156	Constant for all forms

**3.6 Comment Statement (Rem any information or ' any information)** : Only to explain the code, while being changed to green color.

**3.7 Assignment Statement:** (Variable-Name=Expression): Expression may include constant, character, variable (or variables), operators and functions. For example

City="Baghdad"

Age=29



Note: Logical operators follow arithmetic operators in precedence.

### Examples:

```
Private Sub Command1_click ( )
```

```
Picture1.Print 7\3
```

```
Picture1.Print 7 Mod 3
```

```
Picture1.Print "My"&" Name"
```

```
Picture1.Print 10/3*15/3*3/2-9/3/2*4*3
```

```
Picture1.Print 4E3-3E2/5/3E1
```

```
Picture1.Print 4E-8/2*5E8/6E16*4E14*3
```

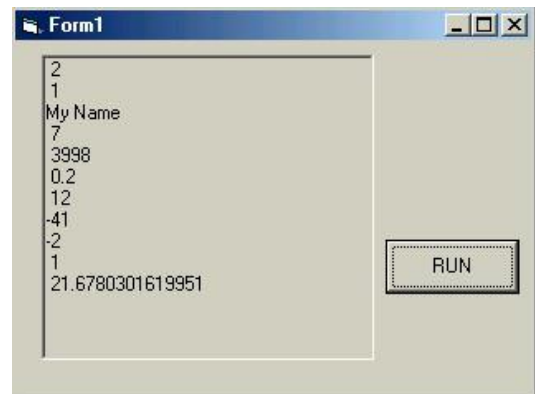
```
Picture1.Print 4/3^3/4^2*3^4*2^4
```

```
Picture1.Print 27^1/3-2E2^3*4E-4/4^3
```

```
Picture1.Print (3-3^3)/((3^2+3^3)/3^5)/3^4
```

```
Picture1.Print (14+2^5/2^4)^(1/4)+((15-5*4)/(3^2-2^3/2))
```

```
Picture1.Print (((3^(3^3)/3^3)^(1/3)+3^4)^(1/3)* 5^2)^(1/2)
```



### Exercise (3-1):

```
15/3*8/3*9/2-15/3/12*4*3 =
```

```
6E3-3E2/5/3E1 =
```

```
2E-7/2*5E7/6E7*4E7*3 =
```

```
6/3^3/4^2*3^4*2^4 =
```

```
16^1/4+2E2^3*2E-4/4^3 =
```

```
(3+3^3)/((3-3^3)/3^7)/3^5 =
```

```
(23+2^6/2^4)^(1/3)+((10-5*4)/(3^2-2^3/2)) =
```

```
((3^(3^3)/3^3)^(1/3)-1)^(1/3)* 2^3)^(1/2) =
```

**3.10 Visual Basic Functions:** Visual Basic offers a rich assortment of built-in functions. The numeric and string variables are the most common used variables in programming. Therefore Visual Basic provides the user with many functions to be used with a variable to perform certain operations or type conversion. Detailed description of the function in general will be discussed in the following functions section. The most common functions for (numeric or string) variable **X** are stated in the following table.

Function	Description
----------	-------------



Numerical Function	
X= RND	Create random number value between 0 and 1
Y=ABS(X)	Absolute of X, $ X $
Y=SQR(X)	Square root of X, $\sqrt{X}$
Y=SGN(X)	$-(-1 \text{ or } 0 \text{ or } 1)$ for $(X < 0 \text{ or } X = 0 \text{ or } X > 0)$
Y=EXP(X)	$e^x$
Y=LOG(X)	Natural logarithms, $\ln X$
Y=LOG(X) / LOG(10)	$\log X$
Y=sin (X) Y=cos (X) Y=tan (X)	Trigonometric functions
Y=ATN(X)	Is arc= $\tan^{-1}(X)$ (Where X angle in radian).
Y=INT(X)	Integer of X
Y= FIX(X)	Take the integer part
Function of String Variable	
Y=Len(x)	Number of characters of Variable
Y=LCase (x)	Change to small letters
Y=UCase (x)	Change to capital letters
Y=Left (X,L)	Take L character from left
Y=Right (X,L)	Take L character from right
Y=Mid (X,S,L)	Take only characters between S and R

**3.11 Converting Data Types:** Visual Basic provides several conversion functions can used to convert values into a specific data type. The following table describes the convert function.

Function	Description
<b>CDbl</b>	The function CDbl converts, integer, long integer, and single- precision numbers to double-precision numbers. If x is any number, then the value of CDbl(x) is the double precision number determined by x.
<b>CInt</b>	The function CInt converts long integer, single-precision, and double precision numbers to integer numbers. If x is any number, the value of CInt(x) is the (possibly rounded) integer constant that x determines.

<b>CLng</b>	The function CLng converts integer, single precision and double-precision numbers to long integer numbers. If x is any number, the value of CLng(x) is the (possibly rounded) long integer that x determines.
<b>CSng</b>	The function CSng converts integer, long integer, and double-precision numbers to single-precision numbers. If x is any number, the value of CSng(x) is the singleprecision number that x determines.
<b>CStr</b>	The function CStr converts integer, long integer, single-precision, double-precision, and variant numbers to strings. If x is any number, the value of CStr(x) is the string determined by x. unlike the Str function, CStr does not place a space in front of positive numbers.[variant]
<b>Str</b>	The Str function converts numbers to strings. The value of the function Str(n) is the string consisting of the number n in the form normally displayed by a print statement.
<b>Val</b>	The Val function is used to convert string to double-precision numbers.

**Note:** The following function values for different X are given for comparison.

<b>X=</b>	<b>10.999</b>	<b>- 10.999</b>	<b>10.123</b>	<b>-10.123</b>
<b>FIX(X)</b>	<b>10</b>	<b>-10</b>	<b>10</b>	<b>-10</b>
<b>INT(X)</b>	<b>10</b>	<b>-11</b>	<b>10</b>	<b>-11</b>
<b>CINT(X)</b>	<b>11</b>	<b>-11</b>	<b>10</b>	<b>-10</b>

**Examples:**

A=Lcase ("My Name Is")	→	A= my name is
A=Ucase ("My Name Is")	→	A=MY NAME IS
A=" My Name Is": B=Left (A,7)	→	B=My Name
C=Right(A,7) :	→	C=Name Is
D= Mid (A,3,5)	→	D=Name
E=Mid(A,3)	→	E=Name Is

**Examples:**

```
Print INT(4.1)
Print INT(-4.1)
Print INT(-4.8)
Print INT(2.34567*100+0.5)/100

A=3.14159/180: Print SIN (45*A)/COS(60*A)^2/COS(45*A)/SIN(30*A)^3
Print INT (-4E-6/2)*INT(5E8/6E15*1.2E10)
```

```
Print SGN (INT(4/3^8/4^3*3^5*2^5))
```

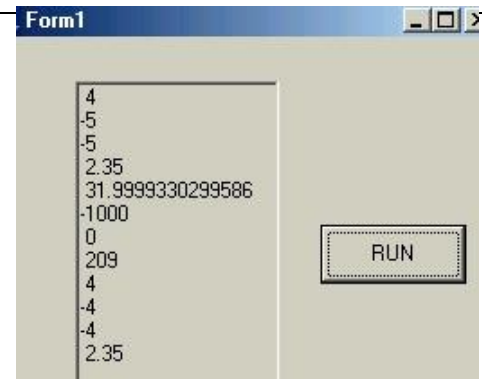
```
Print EXP (LOG(27^1/3+2E2^3*4E-4/4^2))
```

```
Print FIX (4.1)
```

```
Print FIX (-4.1)
```

```
Print FIX(-4.8)
```

```
Print FIX (2.34567*100+0.5) / 100
```



**Example (3-1):** Convert the following arithmetic formula to visual Basic language.

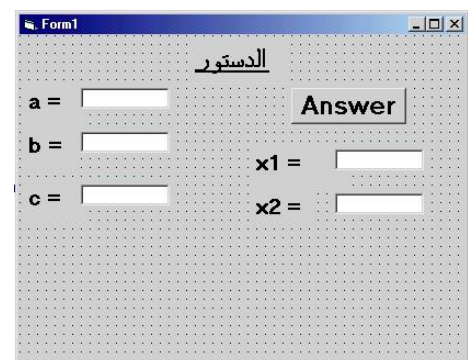
Arithmetic formula	Visual Basic language
$\sqrt[3]{\frac{e^5 + \sin 30}{\log(2) - \tan(35)}}$	<code>((exp(5)+sin(30*3.14159/180))/(log(2)/log(10)-tan(35*3.14159/180)) ^ (1/3))</code>
$\frac{\pi}{4} \left( \frac{U_{av}}{100} \right)^2$	<code>3.14159/4*(Uav /100)^2</code>
$\frac{\pi}{4} \left( \frac{U_{av}}{100} \right)^2 \frac{1}{\left[ 1 - \left( \frac{U_{av}}{100} \right)^{5.63} \right]^{0.533}}$	<code>3.14159/4*(Uav/100)^2/(1-(Uav/100)^5.63)^0.533</code>
$\frac{-b + \sqrt{b^2 - 4 * a * c}}{2 * a}$	<code>(-b+sqr(b^2-4*a*c))/(2*a)</code>

**Example(3-2):**

Create a Visual Basic project to solve for the roots of the quadratic equation  $aX^2 + bX + c = 0$ ,

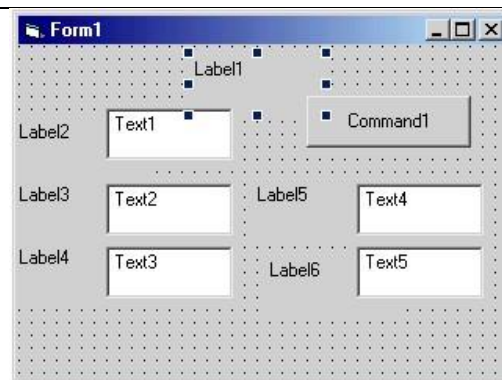
using quadratic formula as:

$X_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ . Design the program so that the values of a, b, and c are entered into separate (labeled) text boxes and display  $X_1$  and  $X_2$  in separate (labeled) text boxes?



**Solution:**

- 1- Place six labels, five text boxes, and one command button on the form. The form should appear similar to this.



2- Set the form and object properties:

Object	Property	Setting
<b>Form1</b>	Name	Form1
	Caption	Form1
<b>Command Button1</b>	Name	Cmd1
	Caption	Answer
<b>TextBox1</b>	Name	Txt1
	Text	Empty
<b>TextBox2</b>	Name	Txt2
	Text	Empty
<b>TextBox3</b>	Name	Txt3
	Text	Empty
<b>TextBox4</b>	Name	Txt4
	Text	Empty
<b>TextBox5</b>	Name	Txt5
	Text	Empty
<b>Label1</b>	Name	Label1
	Caption	الدستور
<b>Labe12</b>	Name	Label2
	Caption	a =
<b>Labe13</b>	Name	Label3
	Caption	b =
<b>Labe14</b>	Name	Label4
	Caption	c =
<b>Labe15</b>	Name	Label5
	Caption	X1=
<b>Labe16</b>	Name	Label6
	Caption	X2=

3- Attach this code to the command1 button (Answer)

```

Private Sub Cmd1_click ( )
Dim a , b , c , X1 , X2
a= Val (Txt1.text)
b= Val (Txt2.text)
c= Val (Txt3.text)
X1=Cdbl ( - b + Sqr ( b ^ 2 - 4 * a * c ) ) / ( 2 * a)
X2= Cdbl ( - b - Sqr ( b ^ 2 - 4 * a * c ) ) / ( 2 * a)
Txt4.text = CStr (X1)
Txt5.text = CStr (X2)
End Sub

```

4- **Running the Application:** press F5 or icon



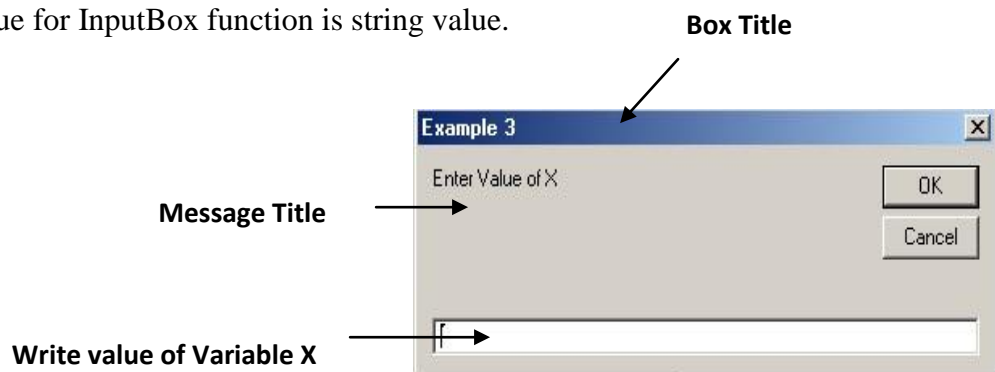
**3.12 InputBox Function:** InputBox function is used to input a value or character for one variable from keyboard at running stage.

**Variable-Name=InputBox (“Message”,”Title”)**

For Example

`X=InputBox(“Enter Value of X”,”Example 3”)`

**Note:** The type value for InputBox function is string value.



**Example(3-3):** Repeat Example(3-2). Using Input Box function to input value of a, b, and c.

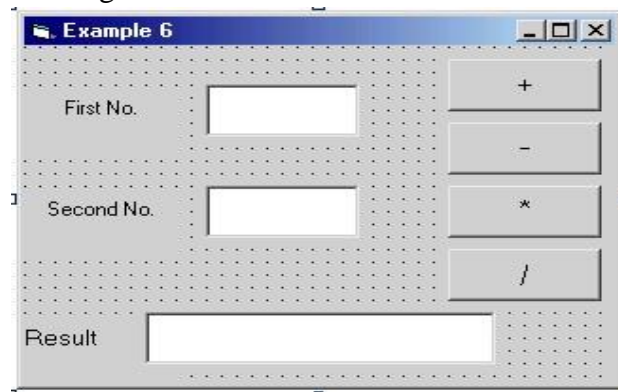
**Exercise (3-1):** Create a Visual Basic project to find the Perimeter and area of any triangular using the equation formula as shown below. Design the program so that the values of a, b, and c are entered into separate (labeled) text boxes and display Perimeter and area in separate (labeled) text boxes?

$$\text{Perimeter} = a + b + c \quad : \quad s = (a + b + c) / 2$$

$$\text{area} = \sqrt{s * (s - a) * (s - b) * (s - c)}$$



**Exercise (3-2):** To design a simply calculator, design a form with three text boxes and four command buttons. The integer value of the first and second number is entered into separate (labeled) text boxes. Write codes to perform add, subtract, multiply, and divide where pressing on buttons. Display the result operation in separate TextBox by using the following formula.  $4 + 5 = 9$



**Exercise (3-3):** Create a Visual Basic project to find the average value of three positive and integer variables (X1, X2, and X3). Find the deviation  $\{(\text{average} - X1), (\text{average} - X2), \text{and } (\text{average} - X3)\}$ . Print the value of Average and Deviation in PictureBox.

**Exercise (3-4):** Create a Visual Basic project to enter an angle value (Degree, Minutes, and Seconds) into separate text boxes. Design the program to find the value of angle (in degree only) as the following equation. Display Angle in separate text box.

$$\text{Angle} = \text{Degree} + (\text{Minutes}/60) + (\text{Seconds}/3600)$$

**Exercise (3-4):** Create a Visual Basic project to enter an angle value (used InputBox statement). Design the program to find the value of angle (in Degrees, Minutes, and Seconds). Display Degrees, Minutes, and Seconds in PictureBox. Pointer the control objects are used on the form window.




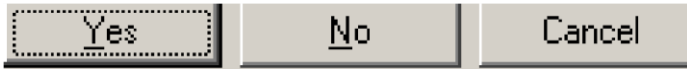


#### **Message Boxes (MsgBox Function):**

The objective of MsgBox is to produce a pop-up message box and prompt to click on a command button before can continue. This format is as follows:

**MsgBox “Prompt”, Style Value, “Title”**





The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command buttons appear on the message box. The Title argument will display the title of the message board. The Style values are listed below.

**Table 1-13.4: Style values**

Name Constant	Style Value	Buttons Displayed
VBOKOnly	0	
VBOKCancel	1	
VBAabortRetryIgnor	2	
VBYesNoCancel	3	
VBYesNo	4	
VBRetryCancel	5	

To make the message box looks more sophisticated, you can add an icon besides the message. There are four types of icons available in VB6 as shown in Table 2-13.4

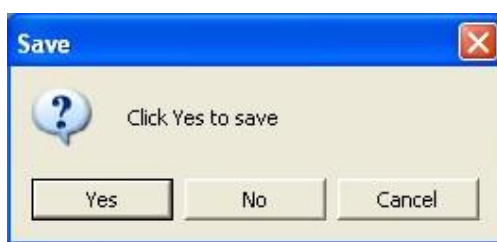
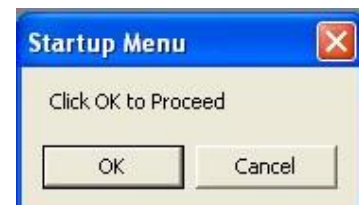
**Table 2-13.4:Types of Icons**

Value	Named Constant	Icon
16	vbCritical	
32	vbQuestion	
48	vbExclamation	
64	vbInformation	

We can use named constants in place of integers for the second argument to make the programs more readable. In fact, VB6 will automatically shows up a list of named constants where you can select one of them.

**For example:** MsgBox "Click OK to Proceed", 1, "Startup Menu" or, MsgBox "Click OK to Proceed". vbOkCancel,"Startup Menu"

Msgbox "Click Yes to save", 35, "Save"



**Note:** The statement “**Exit Sub**” is defined to stop the program without close the form window. While the statement “**End**” is stopped the program return to IDE window.

#### 4. Control Structures

In this chapter, you will learn how to write VB6 code that can make decision when it process input from the users, and control the program flow in the process. Decision making process is an important part of programming because it will help solve practical problems intelligently so that it can provide useful output or feedback to the user. For example, we can write a VB6 program that can ask the computer to perform certain task until a certain condition is met, or a program that will reject nonnumeric data. In order to control the program flow and to make decisions, we need to use the **conditional operators** and the **logical operators** together (see section 3.9) with the If control structure. To effectively control the VB6 program flow, we shall use the If control structure together with the conditional operators and logical operators. There are basically three types of If control structures, namely:

- **If .....Then**
- **If – Then –Else**
- **Select Case**

##### 4.1 If....Then Statement:

This is the simplest control structure which ask the computer to perform a certain action specified by the VB expression if the condition is true. However, when the condition is false, no action will be performed. The general format for the (If- Then) statement is

##### **4.1-1 If *Condition* Then *Statement***

Where, Condition is usually a comparison, but it can be any expression that evaluates to a numeric value, this value as true or false. If condition is True, Visual Basic executes all the statements following the Then keyword.

**Example 4-1:** Write a program to enter the value of two variables (X and Y). Find and print the maximum value for two variables. Design form window and select all the control objects are used.

##### **Solution(1):**

```
Private Sub Command1_Click
Dim X , Y , Max
X =Val (Text1.Text )
Y =Val (Text2.Text)
Max=X
If Y> X Then Max= Y
Text3.Text= Cstr (Max)
End Sub
```

or

##### **Solution(2):**

```
Private Sub Command1_Click
Dim X , Y , Max
X =Val (Text1.Text )
Y =Val (Text2.Text)
If X> Y Then Max= X
If Y> X Then Max= Y
Text3.Text= Cstr (Max)
End Sub
```

##### **4.1-2 If *condition* Then *Goto n***

Where *n* : number of statement ( must be Positive Integer value) for example:



Goto 5 , Goto 16 , Goto 2010

**Example 4.2:** Used (If-Then Goto) condition to write a program for the previous Example 4.1

**Solution(1):**

```
Dim X , Y , Max
X =Val (Text1.Text )
Y =Val (Text2.Text)
Max=X
If X> Y Then Text3.Text= Cstr (Max): Exit Sub
Max=Y
Text3.Text= Cstr (Max)
End Sub
```

**Solution(2):**

```
Dim X , Y , Max
X =Val (Text1.Text )
Y =Val (Text2.Text)
If X> Y Then Max=X : Goto 10
Max=Y
10 Text3.Text= Cstr (Max)
End Sub
```

**Note:** The statement **Exit Sub** used to stop the program without return to the project window.

## **4.2 If - Block Statement:**

**4.2.1 (If – Then – EndIf) statement:** The If...Then – EndIf Statement performs an indicated action only when the condition is True; otherwise the action is skipped.

```
If condition Then
VB Expression
End If
```

**For example:**

```
Dim X , Y , Max
X =Val (Text1.Text ) : Y =Val (Text2.Text)
Max=X
If X< Y Then
Max=Y
EndIf
Text3.Text= Cstr (Max)
End Sub
```

**4.2.2 (If – Then – Else) statement:** The If – Then - Else statement allows the programmer to specify that a different action is to be performed when a certain action specified by the VB expression if the condition is True than when the condition is false, an alternative action will be executed. The general format for the **If - Then - Else** statement is

```
If condition Then
VB expression
Else
VB expression
End If
```

**For example:**

```

Dim X , Y , Max
X =Val (Text1.Text ) : Y =Val (Text2.Text)
If X> Y Then
Max=X
Else
Max=Y
EndIf
Text3.Text= Cstr (Max)
End Sub

```

**4.2.3 Nested (If – Then – Else) statement:** If there are more than two alternative choices, using just If – Then - Else statement will not be enough. In order to provide more choices, we can use If...Then...Else statement inside If...Then...Else structures. The general format for the Nested If...Then.. Else statement is

<b><u>Method 1</u></b>	<b><u>Method 2</u></b>
<pre> ┌ If <i>condition 1</i> Then │   <i>VB expression</i> │ Else │   ┌ If <i>condition 2</i> Then │   │   <i>VB expression</i> │   │   Else │   │   ┌ If <i>condition 3</i> Then │   │   │   <i>VB expression</i> │   │   │   Else │   │   │   ┌ <i>VB expression</i> │   │   │   └ End If │   │   └ End If │   └ End If └ EndIf </pre>	<pre> If <i>condition 1</i> Then <i>VB expression</i> ElseIf <i>condition 2</i> Then <i>VB expression</i> ElseIf <i>condition 3</i> Then <i>VB expression</i> Else <i>VB expression</i> End If </pre>

**Example 4.3:** Write a program to enter the value of variable (Mark). Find the grade using If – Block statement and display the value of grade in a text box. When the value of variable (Mark) exceed 100, write a Message Box (Wrong entry, please Re-enter the Mark). Design form window and select all the control objects are used.

**Solution:**

```

Private Sub Command1_click()
Dim Mark As Single , Grade as String
Mark = Val (Text1.Text)
If Mark >100 Then
Msgbox "Wrong entry, please Re-enter the mark", Vbcritical , " Error"
Text1.Text=" " : Text2.Text=" " : Exit Sub
ElseIf Mark >= 90 and Mark <=100 Then
Grade="Excellent"
ElseIf Mark >= 80 Then
Grade="Very Good"
ElseIf Mark>=70 Then
Grade="Good"
ElseIf Mark>=60 Then
Grade="Medium"
ElseIf Mark>=50 Then
Grade="Pass"
Else
Grade="Fail"
End If
Text2.Text=Grade
End Sub

```

**4.3 Select- Case statement:** Select - Case structure is an alternative to If – Then - ElseIf for selectively executing a single block of statements from among multiple block of statements. The Select Case control structure is slightly different from the If - ElseIf control structure. The difference is that the Select Case control structure basically only makes decision on one expression or dimension while the If - ElseIf statement control structure may evaluate only one expression, each If - ElseIf statement may also compute entirely different dimensions. Select- Case is more convenient to use than the If- Else - End If. The format of the Select Case control structure is as follows:

**Select Case** *test expression*

**Case** expression list 1

VB statements

**Case** expression list 2

VB Statements

**Case** expression list 3

VB statements

**Case** expression list 4

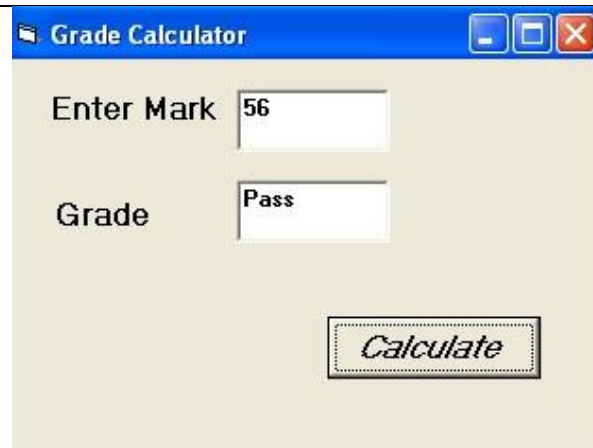
-

**Case Else**

VB Statements

**End Select**

**Example 4.4:** Example 4.3 can be rewritten as follows:

**Solution1**

```
Private Sub Command1_click()
Dim Mark As Single , Grade as String
Mark = Val (Text1.Text)
Select Case Mark
Case 0 To 49
Grade="Fail"
Case 50 To 59
Grade="Pass"
Case 60 To 69
Grade="Medium"
Case 70 to 79
Grade="Good
Case 80 To 89
Grade="Very Good"
Case 90 To 99
Grade="Excellent"
Case Else
Msgbox "Wrong entry, please Re-enter the mark", 16
, " Error"
Text1.Text=" " : Text2.Text=" " : Exit Sub
End Select
Text2.Text=Grade
End Sub
```

**Solution2**

```
Dim Mark As Single , Grade as String
Mark = Val (Text1.Text)
Select Case Mark
Case Is >100 , Is < 0
Msgbox "Wrong entry, please Re-enter the mark", 16
, " Error"
Text1.Text=" " : Text2.Text=" " : Exit Sub Case
Is >= 90
Grade="Excellent"
Case Is >= 80
Grade="Very Good"
Case Is >= 70
Grade="Good
Case Is >= 60
Grade="Medium" Case
Is >=50
Grade="Pass"
Case Else
Grade="Fail"
End Select
Text2.Text=Grade
End Sub
```

**Examples:**

- **Select Case X**  
Case 3, 5, 8 : Print X                      Value of X ( 3 or 5 or 8 ) only.  
End Select
- **Select Case X**  
Case 3, 5, 8 To 20: print X                      Value of X ( 3 or 5 or 8,9,10,....20 ) only.  
End Select

- Select Case X  
 Case 3:  $X = X + 1$ : Print X      Value of X (3) then print ( $X = 4$ ).  
 Case 3,8 To 20 : Print X      Ignore statement when value of  $X = 3$   
 End Select

**Example 4.5:** Design a form with four text boxes and three commands. Design the program so that the values of num1, num2, and Symbol are entered into separate three text boxes. Write a code to perform (add, subtract, multiply and divide) when pressing on command (Calculate). Display the result in separate text box. The command (Clear) used to clear values in text boxes. Click command (Exit) to end the program and return to the project window.

**Solution:**

```
Private Sub Calculate_Click()
```

```
    Dim x As Double, y As Double, z As Double
```

```
    Dim symbol As String
```

```
    x = Cdbl(Text1.Text)
```

```
    Symbol = Text2.Text
```

```
    y = Cdbl(Text3.Text)
```

```
    Select Case Symbol
```

```
        Case "+" : z = x + y
```

```
        Case "-" : z = x - y
```

```
        Case "*" : z = x * y
```

```
        Case "/"
```

```
        If y = 0 Then MsgBox "division by zero": Text3.Text = " " : GoTo 10
```

```
        z = x / y
```

```
    Case Else
```

```
        MsgBox "select any symbol(+,-,*,/)"
```

```
        GoTo 10
```

```
    End Select
```

```
    Text4.Text = Str(z)
```

```
10 End Sub
```

---

```
Private Sub Clear_Click()
```

```
    Text1.Text = ""
```

```
    Text2.Text = ""
```

```
    Text3.Text = ""
```

```
    Text4.Text = ""
```

```
End Sub
```

```
Private Sub Exit_Click()
```

```
End
```

```
End Sub
```

**Exercise 4.1:** Create a Visual Basic project to solve for the roots of the quadratic equation

$aX^2 + bX + c = 0$ , using quadratic formula as:  $X_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$  Design the program so that

the values of coefficient a, b, and c are entered by using input box statement. Display number of roots and value of roots in text boxes. When the value of coefficient (a) equal to zero or  $(b^2 - 4ac)$  less than zero, write a message box to enter a new value of coefficients or end the program.

**Exercise 4.2:** Create a Visual Basic project to find the value of function f(Z) from the equations are below. Write a code so that the value of variables Y and Z are entered into two boxes. Display the value of function f (Z) in separate picture box when click command button. Design form window and select all the control objects are used.

$$X = \sqrt{Y + 5}$$

$$F(Z) = \begin{cases} \frac{Z^3 + 5 \ln(Z)}{\log(Z) + 1} & Z > X^2 + 1 \\ \frac{Z^2 + 4}{2e^Z + 1} & Z < X^2 + 1 \end{cases}$$

Strictly speaking, you can avoid the test for the last OptionButton control in its group because all choices are supposed to be mutually exclusive. But the approach I just showed you increases the code's readability.

A group of OptionButton controls is often hosted in a Frame control. This is necessary when there are other groups of OptionButton controls on the form. As far as Visual Basic is concerned, all the OptionButton controls on a form's surface belong to the same group of mutually exclusive selections, even if the controls are placed at the opposite corners of the window. Actually, you can group your controls within any control that can work as a container—PictureBox, for example—but Frame controls are often the most reasonable choice.

## **5- Loops (Repetition) Structures**

Visual Basic allows a procedure to be repeated as many times as long as the processor and memory could support. This is generally called looping. Looping is required when we need to process something repetitively until a certain condition is met. In Visual Basic, we have three types of Loops, they are

- For.....Next loop,
- Do loop

### **5-1 For....Next Loop**

The format is:

**For** counter = Start **To** End **Step** [Increment]

One or more VB statements

Next [counter]

The arguments counter, start, end, and increment are all numeric. The increment argument can be either positive or negative. If increment is positive, start must be less than or equal to end or the statements in the loop will not execute. If increment is negative, start must be greater than or equal to end for the body of the loop to execute. If steps isn't set, then increment defaults to 1.

In executing the For loop, visual basic:

1. Sets counter equal to start.
2. Tests to see if counter is greater than end. If so, visual basic exits the loop (if increment is negative, visual basic tests to see if counter is less than end).
3. Executes the statements.
4. Increments counter by 1 or by increment, if it's specified.
5. Repeats steps 2 through 4.

**For Example:**

- 1- For I=0 To 10 step 5

Statements

Next I

- 2- For counter = 100 To 0 Step -5

Statements

Next counter

**Example 5-1:** Design a form and write code to find the summation of numbers (from 0 to 100).

**Solution:**

```
Private Sub form_load()  
Form1.show  
Dim I As Integer, Total As Integer  
For I = 0 To 100  
Total= Total +I  
Next I  
Print "Total=";Total  
End Sub
```

**Example 5-2:** Design a form and write code to find the summation of even numbers (from 0 to 100).

**Solution:**

```
Private Sub form_load()  
Form1.show  
Dim I As Integer, Total As Integer  
For I = 0 To 100 step 2  
Total= Total +I  
Next I  
Print "Total=";Total  
End Sub
```

**Example 5-3:** Design a form and write code to find the summation of odd numbers (from 0 to 100).

**Solution:**

```
Private Sub form_load()  
Form1.show  
Dim I As Integer, Total As Integer  
For I = 0 To 100  
If I mod 2 = 1 then Total = Total + I  
Next I  
Print "Total="; Total  
End Sub
```

### **5-2 Do –Loop:**

Use a Do loop to execute a block of statements and indefinite number of times. There are several variations of Do...Loop statement, but each evaluates a numeric condition to determine whether to continue execution. In the following Do..Loop, the statements execute as long as the condition is True.

#### **5-2-1 Do While ..Loop**

The formats are

##### **Do While *condition***

Block of one or more VB Statement

##### **Loop**

When Visual Basic executes this Do..Loop, it first tests condition. If condition is False, it skips past all the statements. If it's True, Visual Basic executes the statements and then goes back to the Do while statement and tests the condition again. Consequently, the loop can execute any number of times, as long as condition is True. The statements never execute if initially False.

**For Example:** Loop counts from 0 to 100.

```
Dim num As Integer, Total  
num = 0  
Do While num <= 100  
Total = Total + num  
num = num + 1  
Loop  
Print Total
```

#### **5-2-2 Do...Loop While:**

Another variation of the Do..Loop statement executes the statements first and then tests condition after each execution. This variation guarantees at least one execution of statements. The formats are

##### **Do**

Block of one or more VB Statement

##### **Loop While *condition***



**For Example:** Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do
Total=Total +num
num = num + 1
Loop While num <= 100
Print Total
```

### **5-2-3 Do Until ....Loop**

Unlike the **Do While...Loop** repetition structures, the **Do Until... Loop** structure tests a condition for falsity. Statements in the body of a **Do Until...Loop** are executed repeatedly as long as the loop continuation test evaluates to False.

The formats are

**Do Until *condition***

Block of one or more VB Statement

**Loop**

**For Example:** Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do until num >100
Total=Total +num
num = num + 1
Loop
Print Total
```

### **5-2-4 Do... Loop Until**

The formats are

**Do**

Block of one or more VB Statement

**Loop Until *condition***

**For Example:** Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do
Total=Total +num
num = num + 1
Loop until num>100
Print Total
```

### **5-3 Existing Loop:**

The exit statement allows you to exit directly from For Loop and Do Loop, Exit For can appear as many times as needed inside a For loop, and Exit Do can appear as many times as needed inside a Do loop (

the Exit Do statement works with all version of the Do Loop syntax). Sometimes the user might want to get out from the loop before the whole repetitive process is executed; the command to use is **Exit For** To exit a For.....Next Loop **or Exit Do** To exit a Do... Loop, and you can place the Exit For or Exit Do statement within the loop; and it is normally used together with the If...Then.....statement.

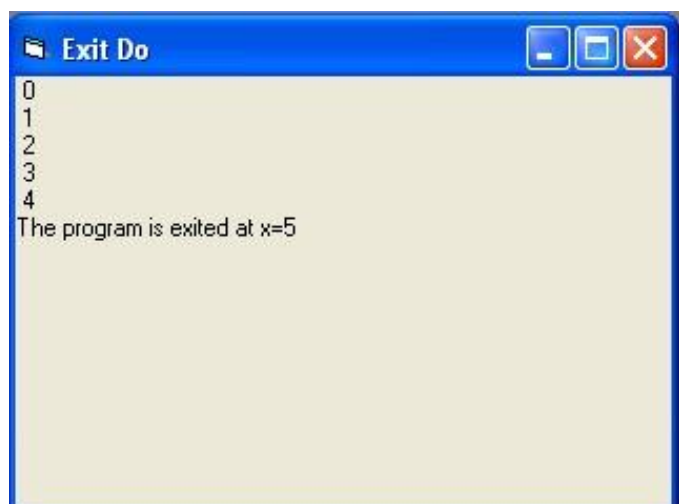
<p>• <b>Exit For</b></p> <p>The formats are:</p> <p><b>For</b> <i>counter</i> = start <b>To</b> end step (increment)</p> <p>Statements</p> <p>Exit for</p> <p>Statement</p> <p>Next <i>counter</i></p>	<p>• <b>Exit Do</b></p> <p>The formats are</p> <p><b>Do While</b> <i>condition</i></p> <p>Statements</p> <p><b>Exit Do</b></p> <p>Statements</p> <p><b>Loop</b></p>
--	---

For its application, you can refer to example:

```
1- Private sub Form_Load_( )
Form1.show
Dim n as Integer
For n=1 to 10
If n>6 then Exit For
Picture1.Print n
Next
End Sub
```



```
1- Private sub Form_Load( )
Form1.show
Dim x As Integer
x=0
Do While x < 10
Print x
x = x + 1
If x = 5 Then
Print "The program is exited at x=5"
Exit Do
End If
Loop
End Sub
```



**5-4 Nested Loop:** The nested loops are the loops that are placed inside each other. The most inner loop will be executed first, then the outer ones. These are examples of the nested loops.

**Possible**

For J=1 to 5

Statement

For I=1 to 5

Statement

Next I

Statement

Next J

**Error (Not Possible)****For K=1 to 5**

Statement

For I=1 to 5

Statement

**Next K**

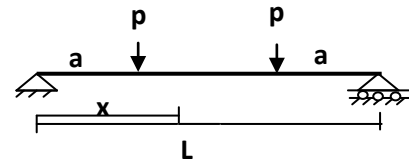
Statement

Next I

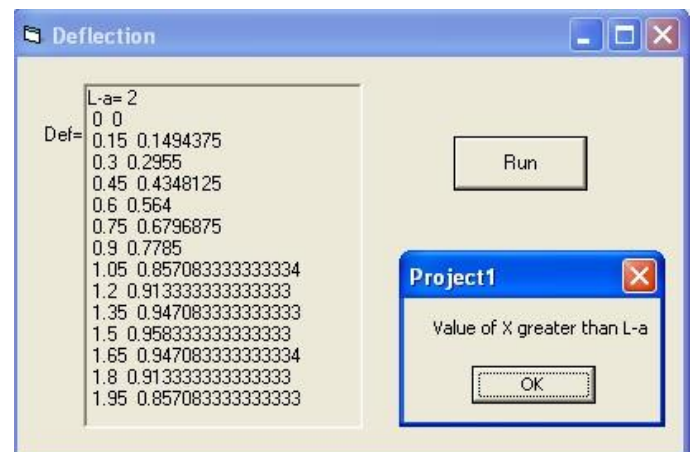
**Example5-4:** For a simply supported beam shown in Fig below. By using the input box statement, enter the value of length of the beam (L), concentrated load (P), distance (a) from support, modulus of elasticity (E) and moment of inertia (IG). Write a code program to find the value of deflection at distance (x) from support, where x increased by (0.01L) from the following equation. Print the deflection value in separate text box. Designs a form and select all control object are used.

$$\Delta_x = \frac{P x}{6EI_G} (3La - 3a^2 - x^2) \quad x < a$$

$$\Delta_x = \frac{Pa}{6EI_G} (3Lx - 3x^2 - a^2) \quad a \leq x \leq L - a$$

**Solution:**

```
Private Sub Command1_click()
Dim L, P, E, IG, a, X, Df
L=Val ( Inputbox ("L=") )
P=Val ( Inputbox ("P=") )
IG=Val ( Inputbox ("IG=") )
E= Val ( Inputbox ("E=") )
a=Val ( Inputbox ("a=") )
For x=0 To L Step 0.01 *L
If x< a Then
Df=p*X/(6*E*IG)*(3*L*a-3*a^2-X^2)
ElseIf X<= L- a Then
Df= p*a/(6*E*IG)*(3*L*X-3*X^2-a^2)
Else
Msgbox" Value of x greater than L-a" : Exit For
EndIf
Picture1.print X; Df
Next X
End Sub
```

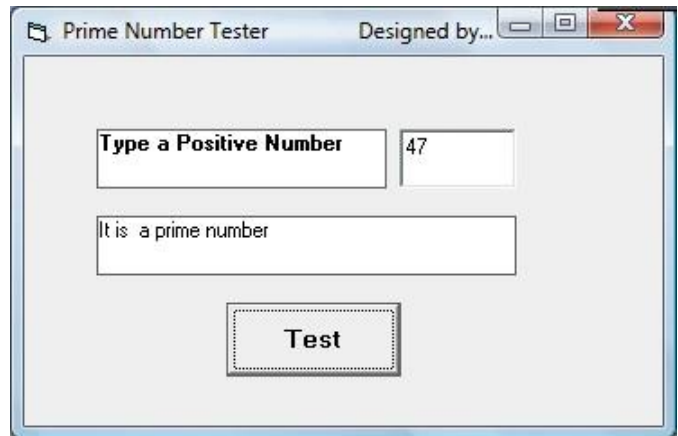


**L=3: P=1:a=1:E=1: IG=1**

**Example 5-5:** Design a form with one command and two text boxes. Enter the value of integer number (N) in separate text box. Write a code program to check if the number (N) is a prime Number or not. Display the “It is not a prime number” or “It is a prime number” in separate text box.

**Solution:**

```
Private Sub Command1_Click()
Dim N, D As Single
Dim tag As String
N = Val(Text1.Text)
Select Case N
Case Is < 2
Text2.text = "It is not a prime number"
Case 2
Text2.text = "It is a prime number"
Case Is > 2
D = 2
Do
If N / D = Int(N / D) Then
Text2.text = "It is not a prime number"
tag = "Not Prime"
Exit Do
End If
D = D + 1
Loop While D <= N - 1
If tag <> "Not Prime" Then
Text2.text = "It is a prime number"
End If
End Select
End Sub
```



**Example 5-6:** Create a Visual Basic Project to find the value of the following series.

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots \dots \dots$$

Write the code program so that the number of terms (N) is entered into text box. Display the result (Pi) in separate text box when click on command (Find Pi).

**Solution:**

```
Private Sub Command1_click()
Dim S as double, N , I , T
N=val(text1.text) : S= 0.0
For I=1 To N
T=1 / I^2
```

```

S=S+T
Next
Pi=SQR (S*6)
Text2.text=Str (Pi)
End Sub

```

**Example 5-7:** Create a Visual Basic Project to find the value of the following series.

$$\text{Sine}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Write the code program so that the value of angle (X) is entered into text box. Estimate the value of series (Sin(x)) so that the absolute value of any term is greater or equal than  $10^{-6}$ . Display the required number of terms (N) which it used in this series in a separate text box and display the result of series (Sin(x)) in another separate text box.

**Solution:**

```

Private Sub Command1_click()
Dim X, Sx, I, J, T, K, N, Fact
X = Val(Text1.Text): X = X * 3.14 / 180
N = 1: K = 1: Sx = 0
10 Fact = 1
For I = 1 To 2 * N - 1
Fact = Fact * I
Next I
T = X ^ (2 * N - 1) / Fact
If Abs(T) >= 0.000001 Then
Sx = Sx + T * K
K = -K: N = N + 1
GoTo 10
Else
Text2.Text = Str(N)
Text3.Text = (Sx)

End If: End Sub

```

**Example 5-8:** Create a Visual Basic Project to find the value of the following series.

$$\text{Sum} = \left( \sum_{i=1}^{i=N} (a * i) \right) + b$$

Write the code program so that the value of constants (a, and b) are entered into text boxes. When the users click checkbox, calculate the value of series (where the total number of terms is equal 20). When the user unchecked the checkbox, the number of terms (N) is entered into input box and calculate the value of series. Display the value of series (Sum) in a separate text box.

**Solution:**

```

Private Sub Command1_Click ( )
Dim a, b, Sum, N
a = Val (Text1.Text)
b = Val (Text2.Text)
Sum = b
If Check1.Value = 1 Then
For I = 1 To 20
Sum = Sum + a * I
Next
Else
N = Val (inputbox ("No. of terms="))
For I = 1 To N
Sum = Sum + a * I
Next
End If
Text3.Text = Str (Sum)
End Sub

```

**Exercise 5-1:** Create a Visual Basic Project to find the value of the following series.

$$\cos(x) = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Write the code program so that the value of angle (X) is entered into text box and the number of terms (N) is entered into input box. Calculate the value of series and display the result of series (Cos(x)) in another separate text box.

**Exercise 5-2:** Create a Visual Basic Project to find the value of the following series.

$$Y = 1 - \frac{X^3}{3^2} + \frac{5X^7}{7^2} - \frac{9X^{11}}{9^2} + \dots \quad X > 0$$

$$Y = \frac{X^2}{2^2} - \frac{3X^6}{6^2} + \frac{5X^{10}}{10^2} - \dots \quad X < 0$$

Write the code program so that the value of (X) is entered into text box. Estimate the value of series (Y) until the absolute value of any term is less than  $10^{-6}$ . Display the required number of terms (N) which it used in this series in a separate text box and display the result of series (Y) in another separate text box.

### **5-5 Using ListBox and ComboBox Controls In Visual Basic 6:**

The ListBox will display a single column of strings, referred to as **items**. The items to appear initially can either be specified at design time with the List property or set with code in the Form\_Load event procedure. Then code is used to access, add, or delete items from the list. If the number of items exceed the value that be displayed, scroll bars will automatically appear on the control. These scroll bars can be scrolled up and down or left to right through the list.

A ComboBox is best thought of as a text box with a help list attached. With an ordinary textbox, the user must type information into the box. With a combobox, the user has the option of either typing in information or just selecting the appropriate piece of information from a list. The two most useful types of combobox are denoted as style property combobox as shown in Figure below.



With a style 1 combo box, the list is always visible. With style 0 or 2 combobox, the list drops down when the user clicks on the arrow. In either case, when an item from the list is highlighted, the item automatically appears in the text box at the top and its value is assigned to the text property of the combo box. The items to appear initially can either be specified at design time with the combo property or set with `combo_change()` event procedure directly.

ComboBoxes have essentially the same properties, event and methods as ListBoxes.

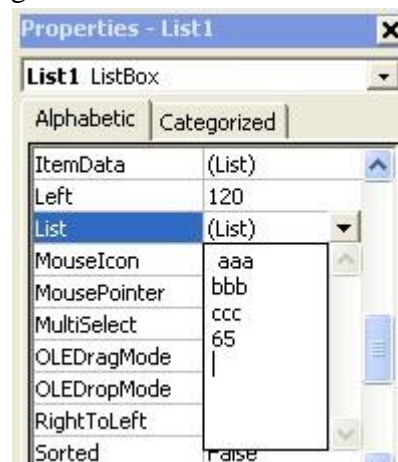
The following Figure lists some of the common **Listbox & ComboBox** properties and methods.

Property		Description
Properties		
Enabled		By setting this property to True or False user can decide whether user can interact with this control or not
List		String array. Contains the strings displayed in the drop-down list. Starting array index is 0. Use CTRL+Enter to insert values.
Sorted		Boolean. Specifies whether the ListBox &ComboBox items are sorted or not.
Style		Integer. Specifies the style of the ListBox &ComboBox appearance
Text		String. Specifies the selected item in the ComboBox.
Visible		Boolean. Specifies whether ListBox &ComboBox is visible or not at run time
Event Procedures		
Change		Called when text in ComboBox is changed
Click		Called when the ListBox &ComboBox is clicked
Methods	Description	Example
AddItem	Add an item to the ListBox &ComboBox	List1.additem str (x):Combo1.additem str (x)
ListCount	Integer. Contains the number of drop-down list items	X=List1.listcount: Y=Combo1.listcount

<b>ListIndex</b>	Integer. Contains the index of the selected ListBox & ComboBox item. If an item is not selected, ListIndex is -1	X=List1.ListIndex : Y=Combo1.ListIndex
<b>List</b>	String array. Contains the strings displayed in the drop-down list. Starting array index is 0.	X=List1.List(1): Y=Combo1.List(4) X=List1.List(List1.ListIndex)
<b>Text</b>	String. Specifies the selected item in the ListBox & ComboBox.	X=List1.Text: Y=Combo1.Text
<b>Clear</b>	Removes all items from the ListBox & ComboBox	List1.Clear: Combo1.Clear
<b>RemoveItem</b>	Removes the specified item from the ListBox & ComboBox	List1.RemoveItem 1: Combo1.RemoveItem 5 List1.RemoveItem List1.ListIndex
<b>NewIndex</b>	Integer. Index of the last item added to the ListBox & ComboBox. If the ComboBox does not contain any items, NewIndex is -1	X=list1.NewIndex: Y= Combo1.NewIndex

### **5-5.1 Adding items to a ListBox:** It is possible to populate the list at design time or run time

- **Design Time** : To add items to a list at design time, click on List property in the property box and then add the items. Press CTRL+ENTER after adding each item as shown below.



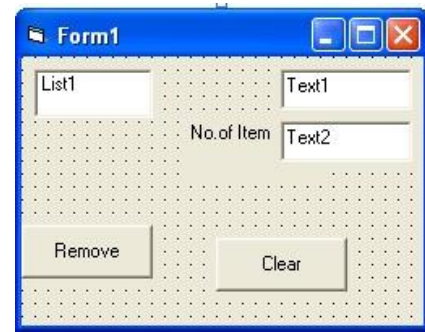
**Example 5-9:** Design a form with one list box, two textboxes and two command buttons. Write code for the following-events.

- 1- Form\_Load event, to add items.(5)
- 2- In click event of listbox, to add item to text1 from list box if item is selected
- 3- In click event of command1 (Remove), to remove item from list box if item is selected and display the number of items in the listbox into text2.
- 4- In click event of command2 (Clear), to clear items from list box.



**Solution:**

```
Private Sub Form_Load()
Dim I
For I = 0 To 4
List1.AddItem InputBox("")
Next
End Sub
```



```
Private Sub List1_Click()
Text1.Text=List1.Text           or   Text1.Text=List1.List(List1.ListIndex).
EndSub
```

```
Private Sub Command1_Click()
List1.RemoveItem List1.ListIndex
Text2.Text = List1.ListCount
EndSub
Private Sub Command2_Click()
List1.Clear
EndSub
```

**Example 5-10:** Design a form with two list boxes, one label, and one command buttons. Write code for the following-events.

- 1- Form\_Load event, to add items (n=100) to list1.
- 2- In click event of command1 (Sum), to Sum items from list1 and add to list2 at each step. Exit loop if (Sum=120).
- 3- In click event of list2, to display number of items to label1

**Solution:**

```
Private Sub Form_Load()
Dim I
For I=1 To 100
List1.AddItem Str(I)
Next I
End sub
```

```
Private Sub Command1_click()
Dim I, Sum
For I=1 to 100
Sum=Sum+I           or   Sum=Sum + list1.list (I-1)
List2.AddItem Str(Sum)
If Sum=120 Then Exit For
Next
```

---

End sub

---

```
Private Sub List2_Click()  
Label1.caption=List2.listcount  
End Sub
```

---

## **6- Arrays in Visual Basic 6**

An array is a collection of simple variables of the same type to which the computer can efficiently assign a list of values. Array variables have the same kinds of names as simple variables. An array is a consecutive group of memory locations that all have the same name and the same type. To refer to a particular location or element in the array, we specify the array name and the array element position number. The Individual elements of an array are identified using an index. Arrays have upper and lower bounds and the elements have to lie within those bounds. Each index number in an array is allocated individual memory space and therefore users must evade declaring arrays of larger size than required. We can declare an array of any of the basic data types including variant, user-defined types and object variables. The individual elements of an array are all of the same data type.

**6-1 Declaring arrays:** Arrays may be declared as Public (in a code module), module or local. Module arrays are declared in the general declarations using keyword Dim or Private. Local arrays are declared in a procedure using Dim. Array must be declared explicitly with keyword "As".

There are two types of arrays in Visual Basic namely:

- **6-1-1 Fixed-Size Array:** The size of array always remains the same-size doesn't change during the program execution. When an upper bound is specified in the declaration, a Fixed array is created. The upper limit should always be within the range of long data type.

### **One Dimension Array:**

Declaring a fixed-array, if array-Name is the name of an array variable and N is a whole number, then the statement

#### **Dim ArrayName (N) As Var Type**

Where, the Dim statement is said to dimension the array and (N) is the range of the array. The array holds either all string values or all numeric values, depending on whether **Var Type** is string or one of the numeric type names.

#### **For example:**

##### **Dim Num (5) As Integer**

In the above illustration, num is the name of the array, and the number 5 included in the parentheses is the upper limit of the array. The above declaration creates an array with 6 elements, with index numbers running from 0 to 5.

The numbers inside the parentheses of the individual variables are called **subscripts**, and each individual variable is called a **subscripted variable** or **element**. The elements of an array are assigned successive memory locations. The following figure shows the memory location for the array Num(5)

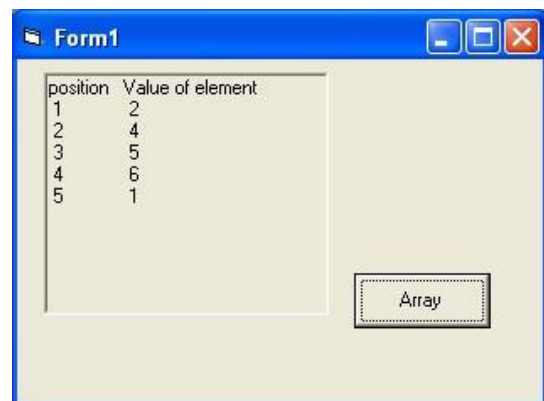
Num (5)	Num (0)	Num (1)	Num (2)	Num (3)	Num (4)	Num (5)
	1	3	-10	5	3	2

**Example 6-1:** Write a code program to read of one dimensional array A(5). Print the value and position of each element.

$$A = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 6 \\ 1 \end{bmatrix}$$

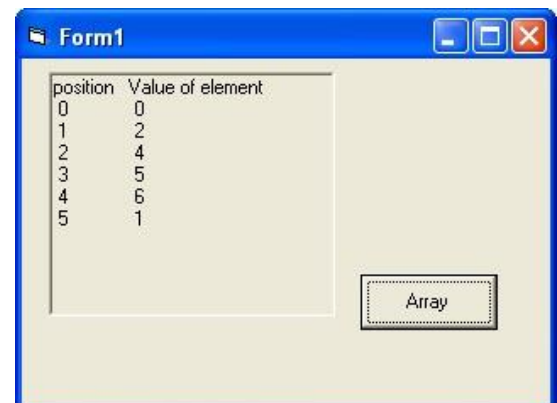
**Solution 1:**

```
Dim A(5) as single
Picture1.cls
Picture1.Print "position"; Space (3); "Value of element"
For I=1 To 5
A( I)= Val(InputBox(" "))
Next I
For I= 1 to 5
Picture1.Print I; Space (11); A(I)
Next
```



**Solution 2:**

```
Dim A(5) as single
Picture1.cls
Picture1.Print "position"; Space (3); "Value of element"
For I= 1 To 5
A( I)= Val(InputBox(" "))
Next I
For I= 0 to 5
Picture1.Print I; Space (11); A(I)
Next
```



**Note:** In solution 2, The type value of array (A) as single, then the default value of A(0)=0, If Type value of array(A) as Variant, then empty value in position A(0).

position	Value of element
0	2
1	4
2	5
3	6
4	1

**Example 6-2:** Suppose A is a one dimension array with 10 elements is entered into listbox. Write a program segment to find the location J such that A (J) contains the largest value in A. Display the Largest value and the location into textboxes.

**Solution:**

```

Dim A(10) as single
For I=1 To 10
A(I)=Val(list1.list(I-1))
Next
Max=A(1) : P=1
For J=1 to 10
If A(J)> Max Then
Max=A(J) : P= J
EndIf
Next
Text1.text= Str(Max)
Text2.text= Str(P)
End Sub

```

**Example 6-3:** Suppose A is a one dimension array with 10 elements is entered into **listbox**. Write a program segment to create the one dimension array (B) contains the even value in array (A). Display the new array (B) into list box2.

**Solution:**

```

List2.Clear
Dim A(10) As Single, B(10) As Single
For I = 1 To 10
A(I) = Val(List1.List(I - 1))
Next
For I = 1 To 10
If A(I) Mod 2 = 0 Then
k = k + 1
B(k) = A(I)

```

```

End If
Next
For I = 1 To k
List2.AddItem Str(B(I))
Next
End Sub

```

**Example 6-4:** Suppose X, Y is linear arrays each with 7 elements into inputbox which element from X and Y in one row respectively. Write a program segment to compute the value of (S) from the following formula. Display the value of (S) into textbox.

$$S = \frac{\sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}}{\sqrt{\sum_{i=1}^n X_i y_i}}$$

**Solution:**

```

Dim X(7) As Single, Y(7) As Single
For I=1 To 7
X(I)= Val (InputBox("X(i) "))
Y(I)= Val (InputBox("Y(i)"))
Next I
For I=1 To 7
S1=S1+X(I)^2 : S2=S2+Y(I)^2 : S3= S3 +X(I)*Y(I)
Next I
S= Sqr(S1) * Sqr(S2) / Sqr(S3)
Text1.text=Str(S)
End Sub

```

**Example 6-5:** Suppose A is a one dimension array with (10) elements. Write a code program which sorts A so that its elements are increasing and sorters into a new array B. Display the origin array (A) and create array (B) into picturebox which element from A and B in one row respectively.

**Solution:**

```

Dim A (10 ), B(10)
For I=1 To 10
A ( I )=Val ( InputBox ("A=")) : B ( I ) = A ( I )
Next
For I = 1 To 9
For J = I+1 To 10
If B( J ) < B( I ) Then
D=B(I)
B( I )=B( J )
B( J )=D
EndIf
Next J , I

```

```

For I=1 To 10
Picture1.Print A ( I ); space (4); B ( I )
Next

```

### Two Dimensional Arrays:

Arrays can have multiple dimensions. A common use of multidimensional arrays is to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we must specify two indexes: The first (by convention) identifies the element's row and the second (by convention) identifies the element's column.

Tables or arrays that require two indexes to identify a particular element are called two dimensional arrays. The following statement declares a two-dimensional array (3 by 3) within a procedure.

**Dim Avg ( 3, 3) as Single**

Avg ( Row, Col.)	Avg (0,0)	Avg (0,1)	Avg (0,2)	Avg (0,3)
	Avg (1,0)	Avg (1,1)	Avg (1,2)	Avg (1,3)
	Avg (2,0)	Avg (2,1)	Avg (2,2)	Avg (2,3)
	Avg (3,0)	Avg (3,1)	Avg (3,2)	Avg (3,3)

Avg ( 3, 3)	2	6	1	0
	3	1	6	-3
	7	3	1	5
	5	4	-2.5	9

It is also possible to define the lower limits for one or both the dimensions as for fixed size arrays.

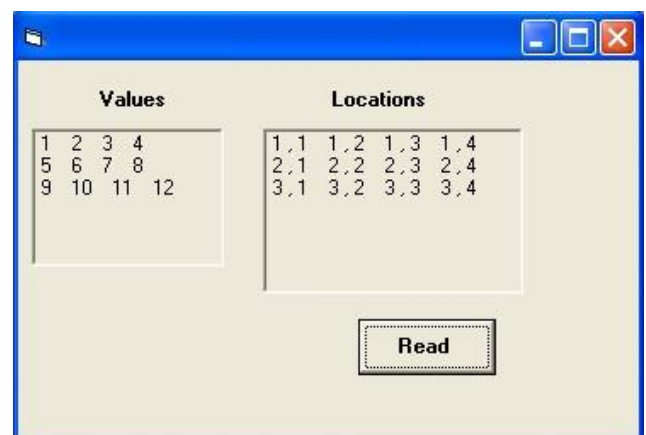
**Example 6-6:** Write a code program to read of two dimensional array A(3,4) on a row by row. Print the value and position of each element.

### Solution:

```

Dim A(3,4) As Single
For I=1 To 3                                (Rows)
For J= 1 To 4                                (Columns)
A(I,J) =Val(InputBox(""))
Next J
Next I
For I=1 To 3
For J= 1 To 4
Picture1.Print A(I, J) ; Space(2) ;
Picture2.Print I ; " , " ; J ; Space(2) ;

```



Next J

Picture1.Print : Picture2.Print

Next I

**Example 6-7:** Write a code program to read of two dimensional array A(3,4) on a column by column. Print the value and position of each element.

**Solution:**

Dim A(3,4) As Single

For J=1 To 4 (Columns)

For I= 1 To 3 (Rows)

A(I,J)=Val(InputBox(""))

Next I

Next J

For J=1 To 4

For I= 1 To 3

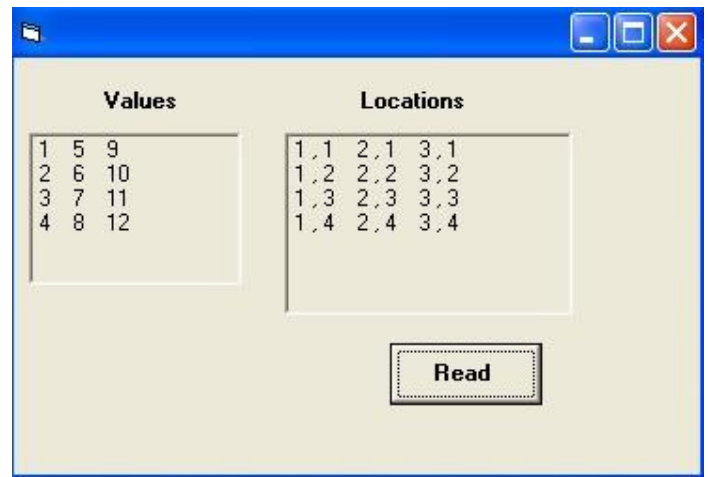
Picture1.Print A(I, J) ; Space(2) ;

Picture2.Print I ; " , " ; J ; Space(2) ;

Next I

Picture1.Print : Picture2.Print

Next J



**Example 6-8:** Write a code program to create a two dimensional array N (5X2) into List Box on row by row. Print the values of array N.

**Solution:**

Dim N(5,2) As Single

K=0

For I = 1 To 5

For J=1 To 2

N(I,J)= Val (List1.List (K))

K=K+1

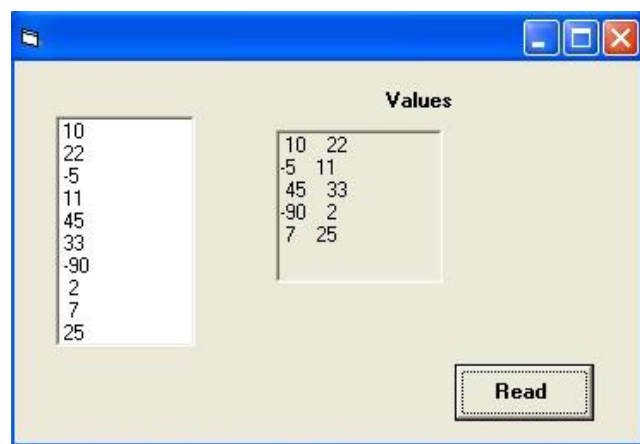
Next J, I

For I=1 To 5

For J= 1 To 2

Picture1.Print N(I, J) ; Space(2) ;

Next J : Picture1.Print : Next I



**Example 6-9:** Suppose N is a (5x2) matrix array is entered into ListBox on row by row. Write a program segment to find the location I and J such that N (I,J) contains the largest value in N. Print the values of array N. Display the Largest value and the location into textboxes.

**Solution:**

Dim N(5,2) As Single

K=0

```

For I = 1 To 5
For J=1 To 2
N(I,J)= Val (List1.List (K))
K=K+1
Next J, I
Max = N(1, 1): R = 1: C = 1
For I = 1 To 5
For J = 1 To 2
If N(I, J) > Max Then
Max = N(I, J)
R = I: C = J
End If
Next J, I
For I = 1 To 5
For J = 1 To 2
Picture1.Print N(I, J); Space(2);
Next J: Picture1.Print: Next I
Text1.Text = Str(Max)
Text2.Text = Str(R)
Text3.Text = Str(C)

```

The screenshot shows a Visual Basic application window with a blue title bar and standard Windows controls. The main area is light beige and contains a 5x2 grid of numbers. To the right of the grid are three text boxes labeled 'Largest Value=', 'Location(Row)=', and 'Location(Col)='. At the bottom right is a button labeled 'Max.'.

Values	
10	22
-5	11
45	33
-90	2
7	25

Largest Value= 45  
Location(Row)= 3  
Location(Col)= 1

Max.

**Example 6-10:** Write a code program to defined the array H (5,5) Calculate the elements of the numeric array (H). Each element of H is determined by the formula (  $h_{ij} = i + j - 1$  ). Create the one dimensional array X contains the elements of array H(5,5) on row by row. Print the array X into List Box.

**Solution:**

```

Dim H(5,5) As Single , X(25) As Single
For I=1 To 5
For J=1 To 5
H(I,J)=(I+J-1)
Next J, I
For I=1 To 5
For J=1 To 5
K=K+1
X(K) =H (I , J)
Next J, I
For I=1 To K
List1.AddItem str(X(I))
Next I

```



**Example 6-11:** Write a code program to read the elements of the array T(5,3) on a row by row. Calculate the SUM of elements in each row and stored in column 4. Print a new array T(5,4) and the sum of all individual row sums, the cumulative sum for all rows.

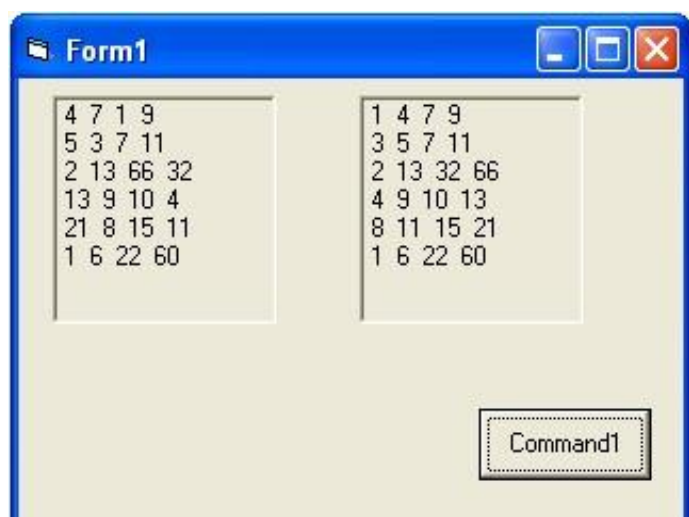
**Solution:**

```
Dim T (5, 4) As Single
For I =1 To 5
For J=1 To 3
T(I,J)=Val (InputBox(""))
Next J, I
For I=1 To 5
Sum=0
For J=1 To 3
Sum=Sum + T(I ,J)
Next J
T (I,4) = Sum
Total=Total +T(I,4)
Next I
For I=1 To 5
For J=1 To 4
Picture1.print T (I , J);
Next J: Picture1.Print : Next I
Text1.text=Str(Total)
End Sub
```

**Example 6-12:** Suppose W is a two dimension array with (6,4). Write a code program which sorts W on row by row so that its elements are increasing (Ascending) and sorters into a same array. Display the new array (W) into picturebox which element.

**Solution:**

```
Dim W(6,4) as Single
For I=1 To 6
For J=1 To 4
W ( I, J)= Val (InputBox(""))
Next J , I
For I=1 To 6
For J= 1 To 3
For K= J+1 To 4
If W(I,K) < W(I,J) Then
C=W(I,J)
W(I,J)=W(I,K) : W(I,K)=C
End If
Next K , J, I
For I=1 To 6
```



```

For J =1 To 4
Picture1.print W(I,J);
Next J : Picture1.Print : Next I

```

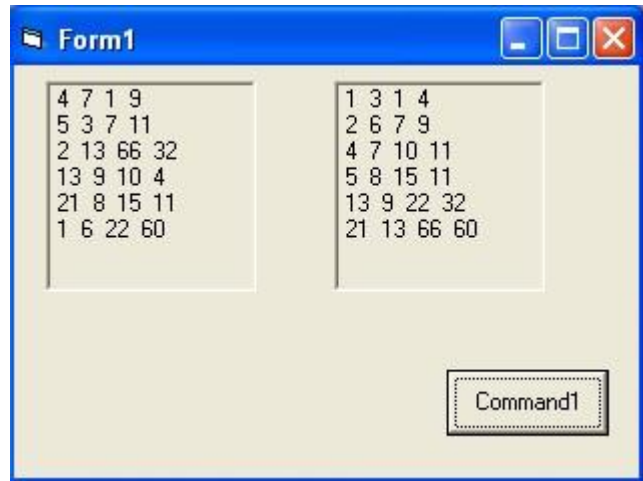
**Example 6-13: :** Suppose W is a two dimension array with (6,4). Write a code program which sorts W on column by column so that its elements are increasing (Ascending) and stores into a same array. Display the new array (W) into picturebox which element.

**Solution:**

```

Dim W(6,4) as Single
For I=1 To 6
For J=1 To 4
W ( I, J)= Val (InputBox(""))
Next J , I
For J =1 To 4
For I= 1 To 5
For K= I+1 To 6
If W(K,J) < W(I,J) Then
C=W(I,J)
W(I,J)=W(K,J) : W(K,J)=C
End If
Next K , I, J
For I=1 To 6
For J =1 To 4
Picture1.print W(I,J);
Next J : Picture1.Print : Next I

```



## 9- Sub Procedure and Function Procedure

Most computer programs that solve real-world problems are much larger than those presented in the first few chapters. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces each of which is more manageable than the original program. This technique is called divide and conquer. This chapter describes many key features that facilitate the design, implementation, operation and maintenance of large programs.

**Functions and Subroutines** are programs designed for specific task, and could be called from the main program or from sub-procedures without pre definition or declaration. Users are allowed to call in any number of times which save the main program space, since it avoids repetition of code these subroutines could be designed by user or could be previously built. The concepts and descriptions are summarized in the following table.

Item	Subroutine	Function
<b>Code</b>	Sub Name (arguments) Statements End Sub	Function Name (arguments) Statements End Function

<b>Remark</b>	<ul style="list-style-type: none"> <li>• Need call statement</li> <li>• Return values by arguments</li> <li>• Return many values (arguments)</li> <li>• Used for Input/output, condition treatment</li> <li>• Could be used without arguments.</li> </ul>	<ul style="list-style-type: none"> <li>• Used in arithmetic statement</li> <li>• Return value by its name</li> <li>• Return one value</li> <li>• Used for arithmetic's or conversion of variable type.</li> </ul>
<b>Call Statement</b>	<b>Call Name(value1,value2,,,) )</b>	<b>Z=name(value1)</b>
<b>Exit statement</b>	<b>Exit Sub</b>	<b>Exit Function</b>

### 9.1 Sub Procedures

Sub procedure are created with the add procedure dialog (displayed when add procedure is selected from the tools menu). The add procedure menu item is grayed unless the code window is visible. Figure (9-1) displays the add procedure dialog. The procedure name is entered in TextBox Name and can be any valid identifier. Frame Type contains option buttons for selecting the procedure type (Sub or Function). Frame scope contains option buttons for selecting keyword public or keyword private that will procedure, we will use keyword private, which also preceded our event procedures.



Figure (9-1): add procedure dialog

Once a valid name has been type into textbox name (add) has been passed, the procedure appears in the code window. Figure (9-2) shows procedure (add) which we created with the add procedure dialog. The code representing (add) in figure (9-2) is called the sub procedure definition.

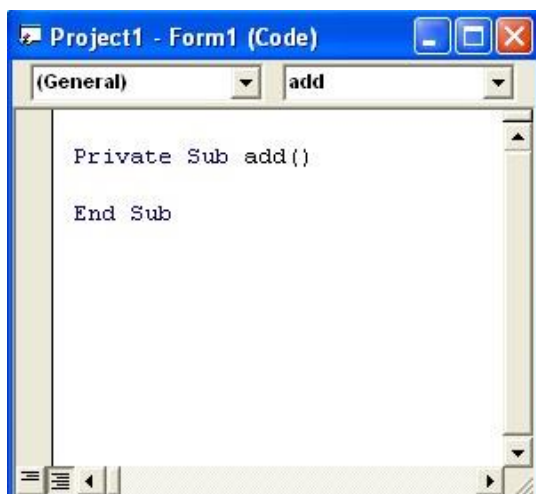


Figure (9-2): A sub procedure created with the add Procedure dialog.

Sub Procedures can also be created by typing the sub procedure directly into the code window. Once a line such as

Private Sub add2 ( )

Is typed and the enter key pressed, visual basic automatically creates the end sub line. Figure (9-3) shows the results when (add2) is typed directly into the code window.

The line

Private Sub add2 ( )

is the sub procedure header. The header contains keyword private, keyword sub, the procedure name, and parentheses. Any declarations and statements the programmer places between the header and end sub form the sub procedure body. Every time the sub procedure is called (or invoked) the body is immediately executed.

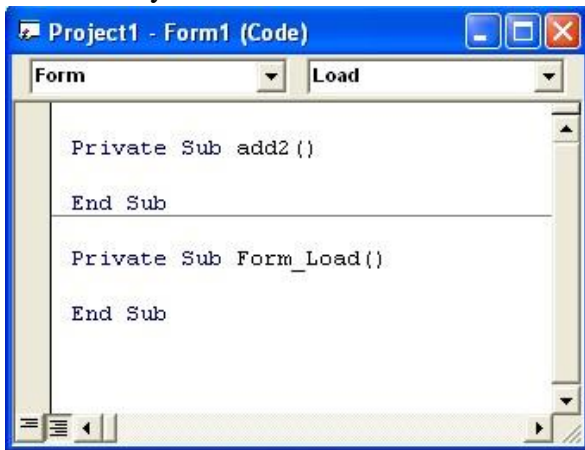


Figure (9-3): A Sub procedure created by typing directly into the code window.

Execution of the sub procedure terminates when end sub is reached. Program execution then continues with the statement immediately following the call to (add2).

All **Sub** procedure definitions contain parentheses which may be empty (e.g., add2). Consider the following sub procedure:

**Private Sub Calculate** (a as single, b as double)

Picture1.print a\*b

**End Sub**

Which declares two parameter variables, (a, and b), in the parameter list. Parameter variables are declared using the **As** keyword. Parameter variables are not explicitly given a type default to **Variant**. Parameter variables receive their values from the procedure call and are used in the procedure body. The call to **Calculate** could also have been written as

**Call** Calculate (30,10.0)

Which uses keyword **Call** and encloses the arguments passed in a set of parentheses. The arguments passed can be variable names as well, for example, the call

Call Calculate (a, b)

Would pass a, and b to Calculate.

**Example 9-1:** Write a code program to read three integer numbers. Using a define sub procedure (Minimum) to determine the smallest of three integers. Display the smallest value in textbox.

**Solution:**

Private Sub Command1\_Click()

Dim Num1 As Single, Num2 As Single, Num3 As Single

Num1 = Fix(Text1.Text)

Num2 = Fix(Text2.Text)

```

Num3 = Fix(Text3.Text)
Call Minimum(Num1, Num2, Num3, min)
Text4.Text = Str(min)
End Sub

```

---

```

Private Sub Minimum(Num1, Num2, Num3, min)
min = Num1
If Num2 < min Then min = Num2
If Num3 < min Then min = Num3
End Sub

```

---

```

(General) Minimum
Private Sub Minimum(Num1, Num2, Num3, min)
min = Num1
If Num2 < min Then min = Num2
If Num3 < min Then min = Num3
End Sub

Private Sub Command1_Click()
Dim Num1 As Single, Num2 As Single, Num3 As Single
Num1 = Fix(Text1.Text)
Num2 = Fix(Text2.Text)
Num3 = Fix(Text3.Text)
Call Minimum(Num1, Num2, Num3, min)
Text4.Text = Str(min)
End Sub

```

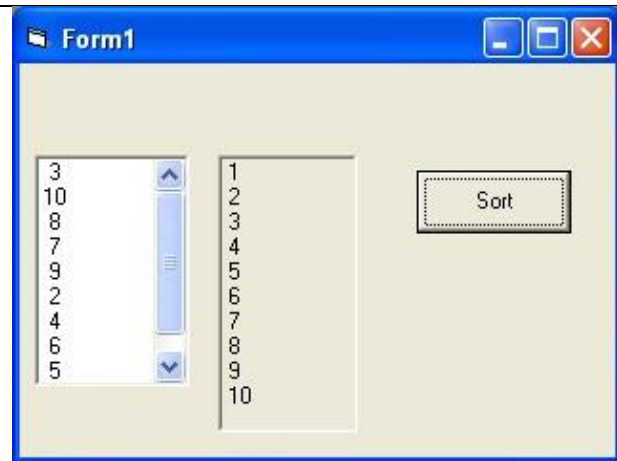
**Example 9-2:** Write a code program to read a one dimension array A (10). Using a define sub procedure (Sort) to Sort (increasing) the array A. Display the new array into picturebox.

**Solution:**

```

Private Sub Command1_Click()
Dim A(10) As Single
For I = 1 To 10
A(I) = Val(List1.List(I - 1))
Next I
Call Sort(A, 10)
For I = 1 To 10
Picture1.Print A(I)
Next I
End Sub

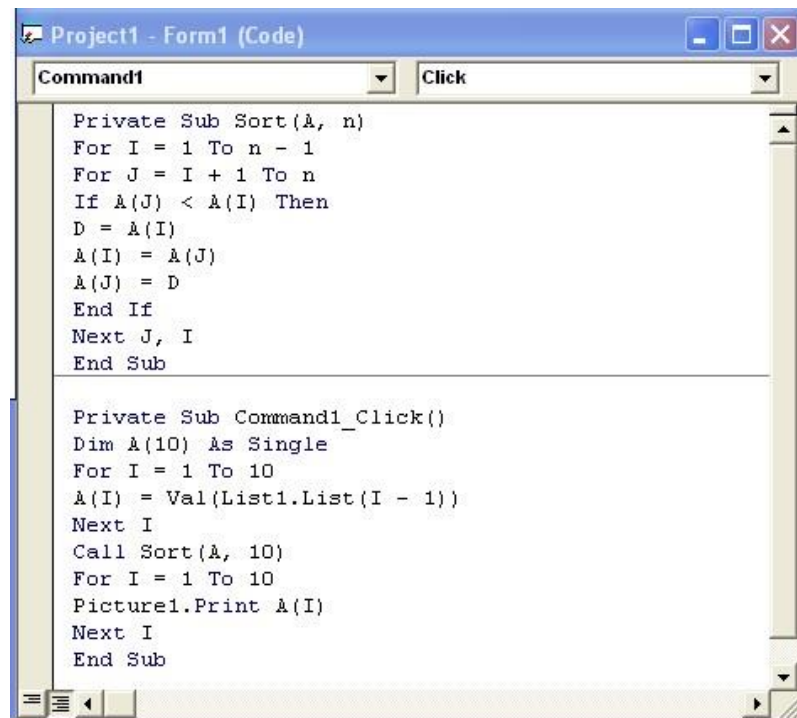
```



```

Private Sub Sort(A, n)
For I = 1 To n - 1
For J = I + 1 To n
If A(J) < A(I) Then
D = A(I)
A(I) = A(J)
A(J) = D
End If
Next J, I
End Sub

```



**9.2 Function Procedures:** function procedures and sub procedures share the same characteristics, with one important difference- function procedures return a value (i.g., give a value back) to the caller, whereas sub procedures do not.

Function Procedures can be created with the add procedure dialog shown in figure (9-1) by selecting function. Figure (9-4) shows a function procedure. **Fact**, created with the add procedure dialog. Fact implicitly returns variant.

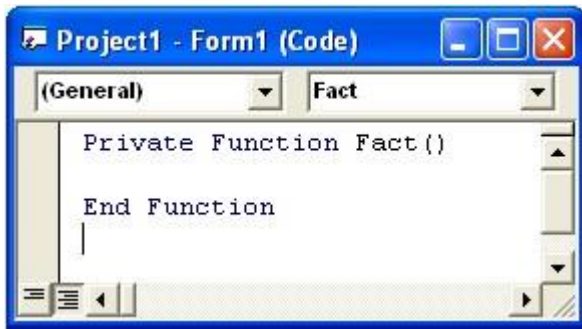


Figure (9-4): Function procedure created with add procedure dialog

**Fact** could also have been created by typing the function procedure directly into the code window. The line

**Private Function Fact()**

is the function procedure header. The header contains the keyword function, the function name and parentheses. The declarations and statements that the programmer will insert between the header and **End Function** form the function procedure body, Fact is invoked with the line.

**Result= Fact( )**

When a function procedure name (such as **Fact**) is encountered at run time, the function procedure is called, causing its body statements to execute. Consider the complete definition for **Fact**

**Private Function Fact( N )**

Fact=N^2

**End Function**

A function procedure return value is specified in the body by assigning a value to the function procedure name, as in

Fact=N^2

Then returns (along with the value returned) to the calling statement

**Result=Fact (N)**

And the return value is assigned to variable result. Program execution then continues with the next statement after the call to Fact.

All function procedure definitions contain parentheses, the parentheses may be empty (e.g. Fact) or may contain one parameter variable declarations. Consider the following function procedure:

**Private Function Area (s1 as single,s2 as single)**

Area=s1\*s2

**End Function**

Which declare two parameter variables s1, and s2. Area's return type is variant. Area is called with the statement

Square=area(8.5, 7.34)

The value 8.5 is stored in s1 and the value 7.34 is stored in s2.

**Example 9-3:** Write a code program to read three integer numbers. Using a define sub Function (Min) to determine the smallest of three integers. Display the smallest value in textbox.

**Solution:**



```

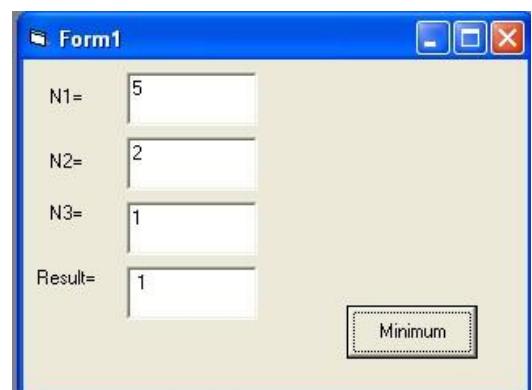
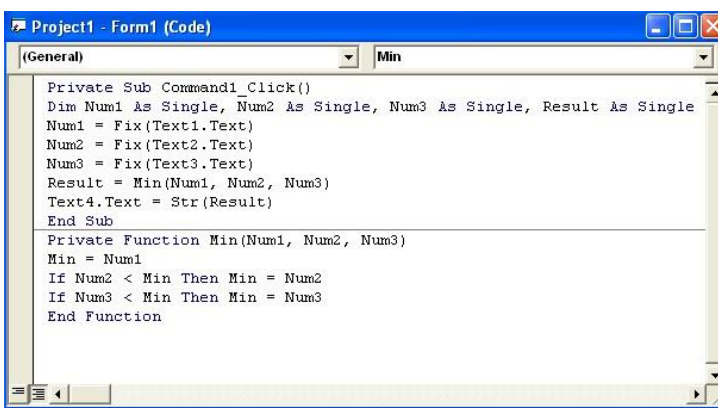
Private Sub Command1_Click()
Dim Num1 As Single, Num2 As Single, Num3 As Single, Result As Single
Num1 = Fix(Text1.Text)
Num2 = Fix(Text2.Text)
Num3 = Fix(Text3.Text)
Result = Min(Num1, Num2, Num3)
Text4.Text = Str(Result)
End Sub

```

```

Private Function Min(Num1, Num2, Num3)
Min = Num1
If Num2 < Min Then Min = Num2
If Num3 < Min Then Min = Num3
End Function

```



**Example 9-4:** Write a code program to input the value of N. Using a define sub function fact to determine(N!). Display the result into text box.

**Solution:**

```

Private Sub Command1_Click()
Dim N As Single, Result As Double
N = Val(Text1.Text)
Result = Fact(N)
Text2.Text = Str(Result)
End Sub

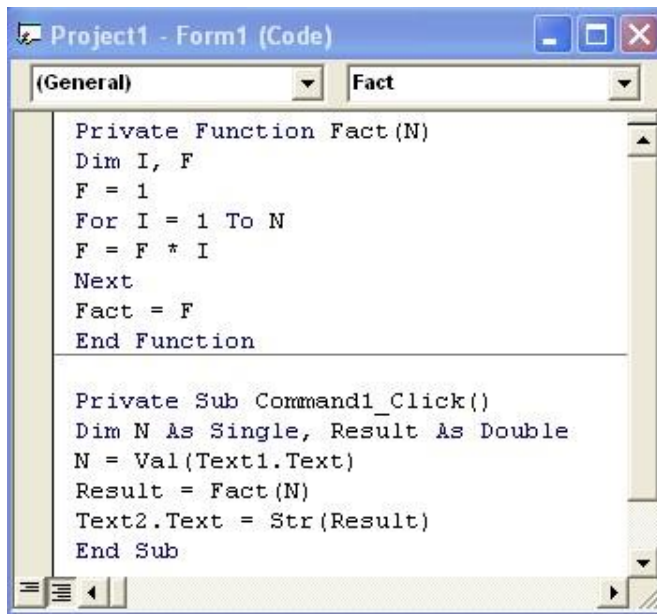
```

```

Private Function Fact(N)
Dim I, F
F = 1
For I = 1 To N
F = F * I
Next
Fact = F
End Function

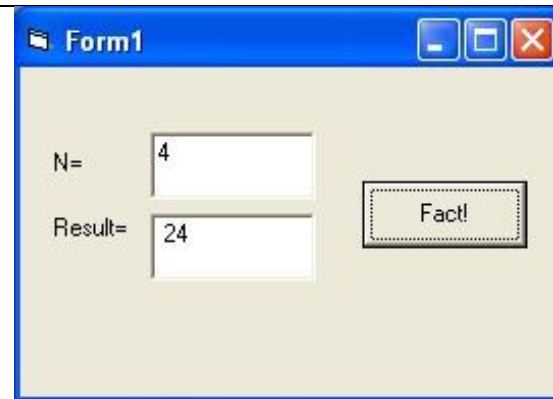
```





```
Project1 - Form1 (Code)
(General) Fact
Private Function Fact(N)
    Dim I, F
    F = 1
    For I = 1 To N
        F = F * I
    Next
    Fact = F
End Function

Private Sub Command1_Click()
    Dim N As Single, Result As Double
    N = Val(Text1.Text)
    Result = Fact(N)
    Text2.Text = Str(Result)
End Sub
```



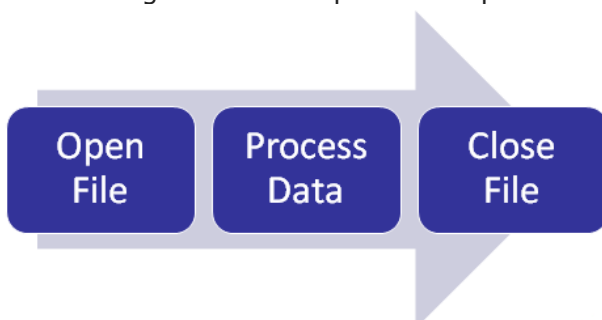
## File Handling

A file is a collection of bytes stored on the disk with a given name (called as filename). Every development tool provides access to these files on the disk. In this chapter we will understand how to access and manipulate files using Visual Basic.

There are three special controls, called as File controls, which deal with files and directories. We will also understand how to use these controls in this chapter.

### File handling

The following are three important steps in handling a file.



- Opening the file
- Processing the file, i.e. either reading the content of the file or writing the required data into file or both.
- Closing the file

## Defining new terms

- **Record:** one logical section of a file that holds a related set of data. If the file contains Student information, a record would hold the information on one student: name, address, studentID, etc. If there are 5,000 students registered, the file contains 5,000 records.
- **Field:** part of a record that defines a specific information. In the Student record, FirstName, LastName, StudentID, are fields. The field is the lowest element in the file. Even if the information consists of one character, Sex is M or F, it is still considered a separate field. The field is the equivalent of the variable - we call it a variable when it is used to store data in memory and call it a field when it stores in a file.
- **I/O:** stands for Input/Output. Whenever you work with a file you have to have ways of reading data from the file (that's Input) and ways of writing data to the file (that's Output). I/O operations consist of all those commands that let you read and write files.

## Types of files

There are basically three types of files you can work with:

- **Sequential file:** this is a file where all the information is written in order from the beginning to the end. To access a given record you have to read all the records stored before it. It is in fact like listening to a tape - you can go forward or back but you can't jump directly to a specific song on the tape. In fact, in the old days, magnetic tape was the most commonly used medium to store data and all files were organized this way. Now, it is still useful when there is a small amount of data to store, a file of application settings, for example. It can even be of use when there is a large amount of data to be stored, provided it all has to be processed at one time, eg: a file of invoices to produce a statement at month-end.
- **Random file:** a file where all records are accessible individually. It is like a CD where you can jump to any track. This is useful when there is a large quantity of data to store and it has to be available quickly: you have to know if a part is in stock for a customer who is on the phone; the program doesn't have time to search through 10,000 records individually to locate the correct one. This method of storage became popular when hard-disk drives were developed.
- **Binary file:** this is a special, compacted form of the random file. Data is stored at the byte level and you can read and write individual bytes to the file. This makes the file access very fast and efficient.

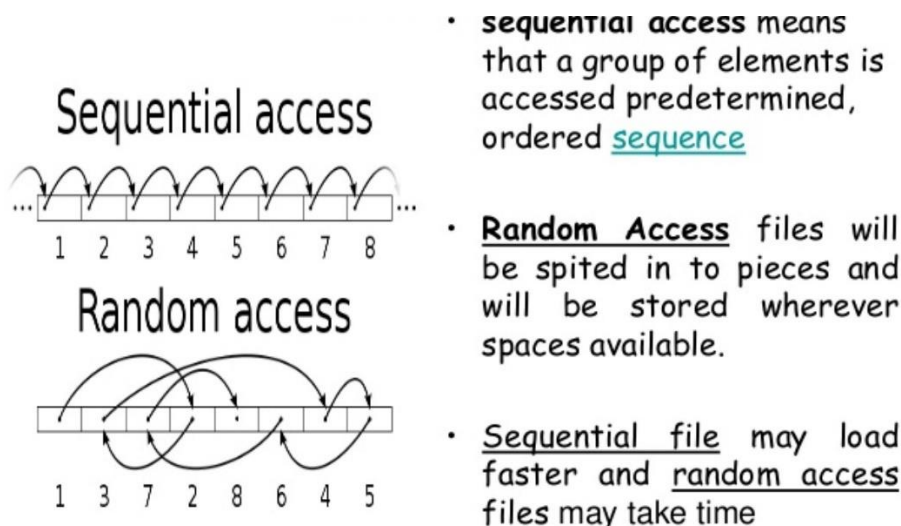
- Access Mode

For Mode in the Open statement indicates how the file will be used. There are five access modes:

- Input: open for sequential input; the file will be read sequentially starting at the beginning.
- Output: open for sequential output; records will be written sequentially starting at the beginning; if the file does not exist, it is created; if it does exist, it is overwritten.
- Random: open for random read and write; any specific record can be accessed.
- Append: sequential output to the end of an existing file; if the file does not exist it is created; it does not overwrite the file.
- Binary: open for binary read and write; access is at byte level.

If access mode is not specified in the Open statement, For Random is used by default.

### Sequential File



### Creating a File

To create a file , we use the following command

Open "fileName" For Output As #fileNumber

Each file created must have a file name and a file number for identification. As for the file name, you must also specify the path where the file will reside. For example:

Open "c:\My Documents\sample.txt" For Output As #1

will create a text file by the name of sample.txt in My Document folder in C drive. The accompanied file number is 1. If you wish to create a HTML file , simply change the extension to .html

Open "c:\My Documents\sample.html" For Output As # 2

Sample Program : Creating a text file

```
Private Sub create_Click()  
Dim intMsg As String  
Dim StudentName As String  
Open "c:\My Documents\sample.txt" For Output As #1  
intMsg = MsgBox("File sample.txt opened")  
StudentName = InputBox("Enter the student Name")  
Print #1, StudentName  
intMsg = MsgBox("Writing a" & StudentName & " to sample.txt ")  
Close #1  
intMsg = MsgBox("File sample.txt closed")  
End Sub
```