University of Technology

Building and Construction Department

Second Class

Visual Basic 6.0

By

Dr. Nisreen S. Mohammed

2011-2012

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.
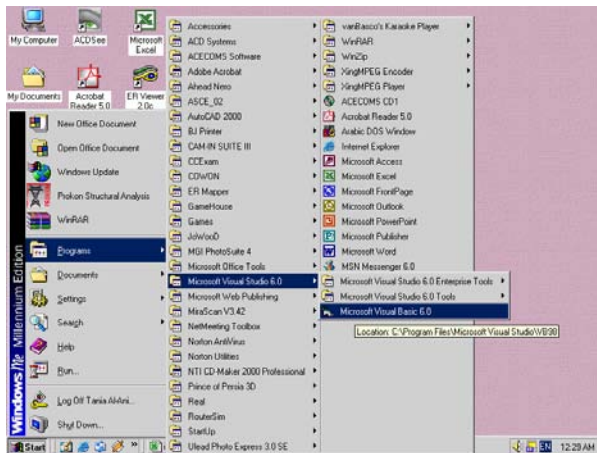

Fig.(2-1) Computer screen   **Fig.(2-2) New Project dialog.**

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):
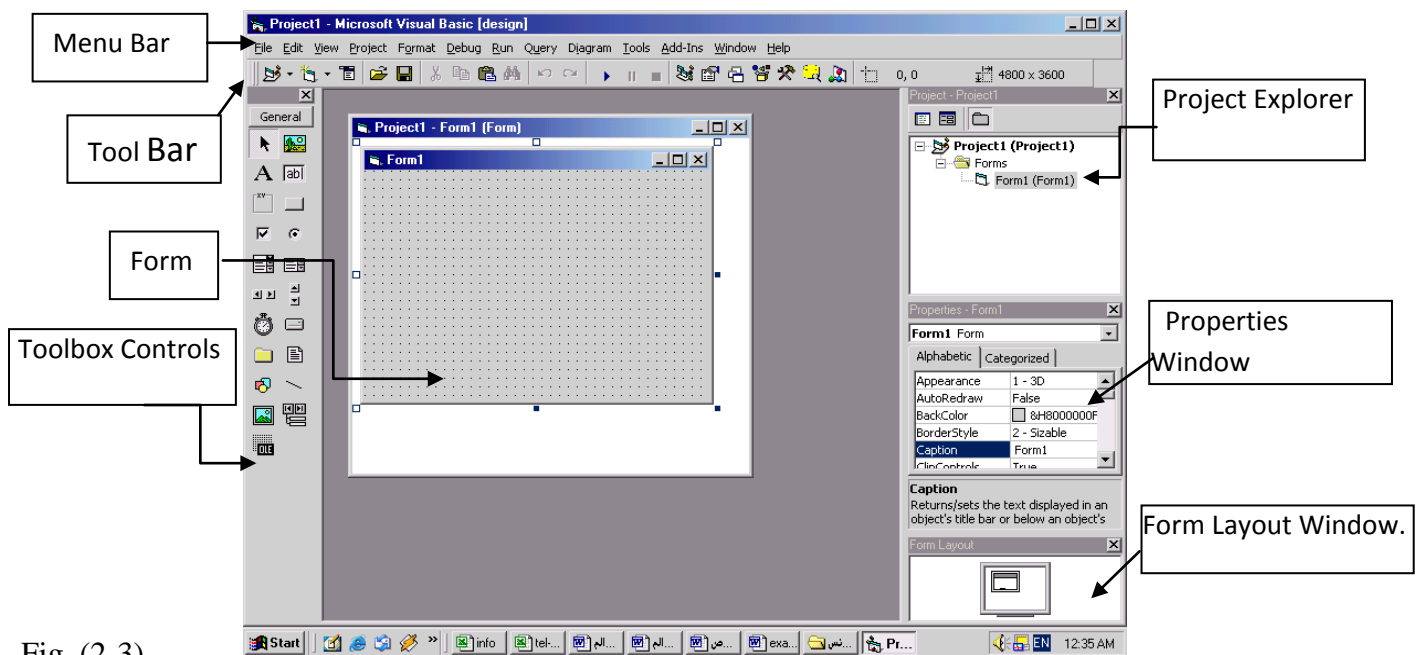
Fig. (2-3)

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 <u>Toolbox Controls:</u> 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.
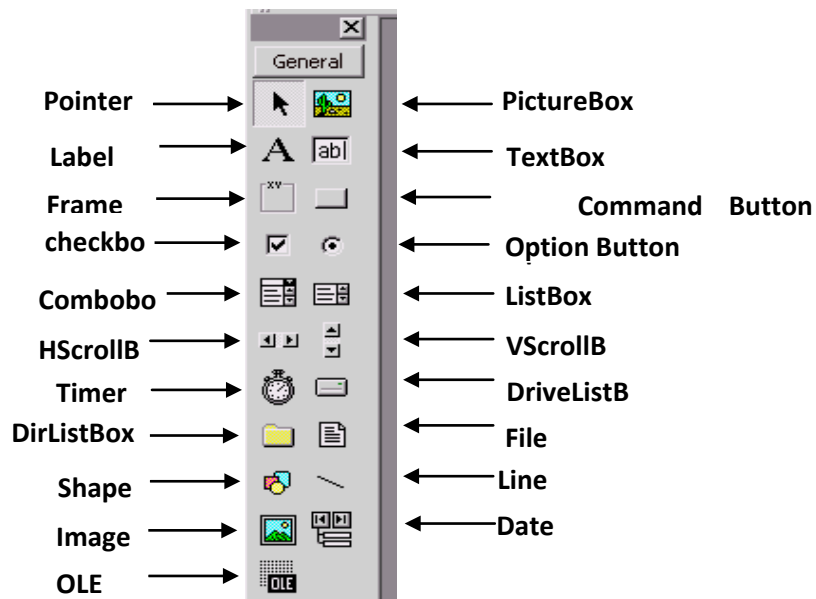

Fig.(2-4) Toolbox Window.

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** procedurename (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:

6
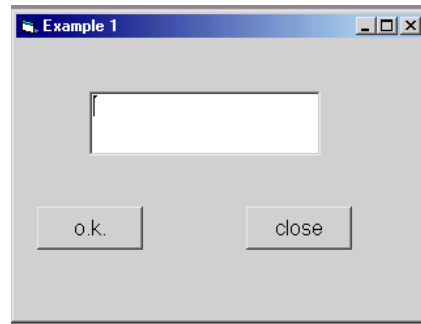
❖ **Creating the Interface**.: The first step in building a Visual Basic application is to create the forms that will be the basis for your application's interface. Then you draw the objects that make up the interface on the forms you create.

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.
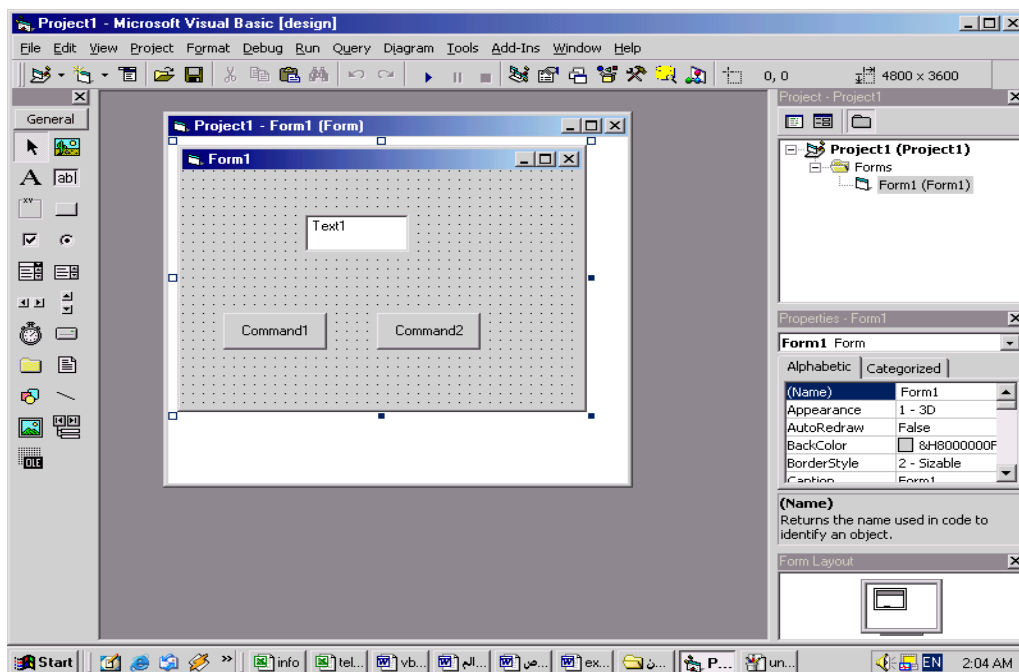
❖ **Setting Properties**

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
Form1	Name	Form1
	Caption	Example1
	Font	Bold and size 12
Command Button1	Name	Command1
	Caption	O.k.
	Font	Bold and size 12
Command Button2	Name	Command 2
	Caption	Close
	Font	Bold and size 12
TextBox	Name	Text1
	Text	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

<u>**Note:**</u> The statement **END** used to close the program runtime.
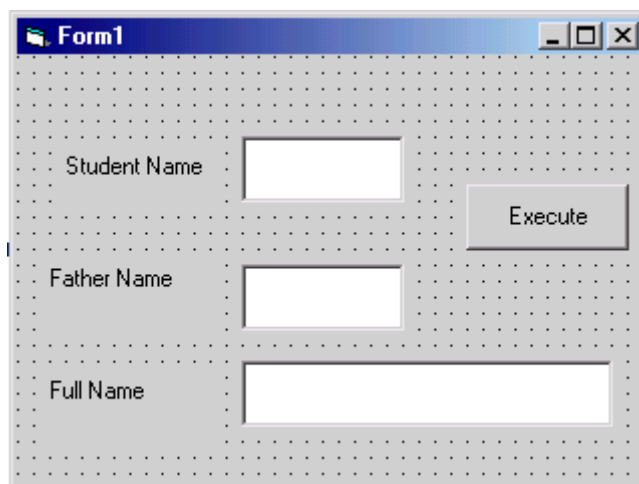
❖ **Running the Application**

   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.

❖ <u>**Saving a Project**</u>

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

<u>**Example 2-2:**</u> 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).

<u>**Solution:**</u>

❖ **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.
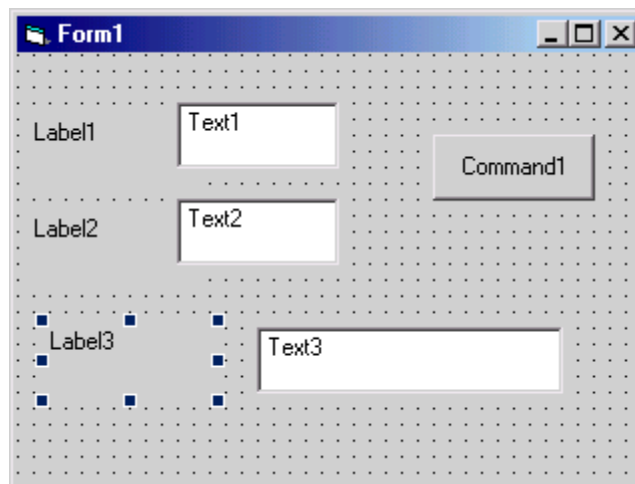
❖ **Setting Properties**

Object	Property	Setting
Form1	Name	Form1
	Caption	Example1
	Font	Bold and size 12
Command Button1	Name	Command1
	Caption	Execute
	Font	Bold and size 12
TextBox1	Name	Txt1
	Text	Empty
TextBox2	Name	Txt2
	Text	Empty
TextBox3	Name	Txt3
	Text	Empty

Labe11	Name	Label1
	Caption	Student Name
	Font	Bold and size 12
Labe12	Name	Label2
	Caption	Student Name
	Font	Bold and size 12
Labe13	Name	Label3
	Caption	Full Name
	Font	Bold and size 12

❖ **Writing Code**:

Choose the Form1 and type the following code:

Private **Sub Form1_load ( )**

Txt3.text=Form1.width-100

End Sub

Choose the command1 and type the following code:

Private **Sub Command1_click ( )**

Txt3.text=tex1.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 $-9x10^{14}$ to $9x10^{14}$.

3-Date: A variable of type Date requires 8 bytes of memory and holds numbers representing dates from January 1St 100 To December 31St 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 $1.40129x10^{-45}$ to $3.40283x10^{38}$ 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 $4.94065x10^{-324}$ to $1.7976x10^{308}$ 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 -$2x10^9$ to $2x10^9$.

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 1st 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.

- **<u>Variables used within a procedure:</u>** 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.

- **<u>Variables Used within a Module:</u>** By default, a module-level is variable 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.

- **<u>Variables used by all modules:</u>** 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

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

- **<u>Declaring All Local Variables as static:</u>** To make all local variables in a procedures 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 remains 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 be 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

X=2*X

Z= A+B

m= Sin (d)

3.8 Outputting Variable: The Print method is used to display data on the screen. The statement **Print expression** (example: **Print X**) displays the value of the expression at the current position of the cursor in the form Object. The statement **Picture.Print expression** displays the value of the expression at the current position of the cursor in the PictureBox object. Several expressions may be placed in the same Print method if separated by semicolons (to display them adjacent to one another) or by commas (to display them in successive zones).

Examples :
Print x , y   or   Picture1.Print x , y
Print " x & y =" ; x ; y   or   Picture1. Print " x & y =" ; x ; y

3.9 Visual Basic Operators:
1- The simplest operators carry out arithmetic operations. These operations in their order of precedence are:

Operation Code	Operation
^	Exponent
*,/	Multiplication and division
\	Integer division
Mod	Modulus – rest of division
-,+	Subtraction and addition

2- To Concatenate two strings, use the **&** symbol or the + symbol

3- There are six Comparison operators in Visual Basic.

Operation Code	Comparison
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
< > or > <	Not equal to

4-There are three logical operators:

Operation Code	Operation
Not	Logical not
And	Logical and
Or	Logical or

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)
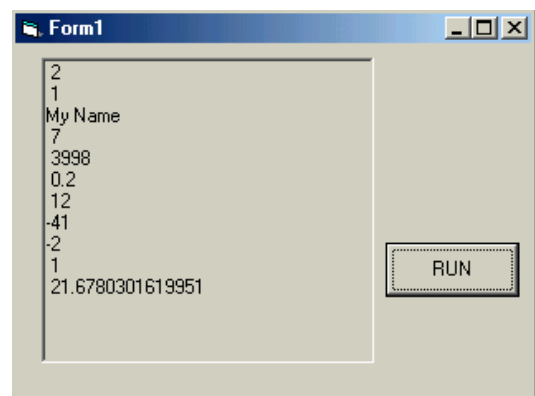
Exercise (3-1):

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

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 or 0 or 1) for (X<0 or X=0 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 <u>Converting Data Types:</u> 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
CDbl	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.
CInt	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.
CLng	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.
CSng	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 single-precision number that x determines.
CStr	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]
Str	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.
Val	The Val function is used to convert string to double-precision numbers.

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

X=	10.999	- 10.999	10.123	-10.123
FIX(X)	10	-10	10	-10
INT(X)	10	-11	10	-11
CINT(X)	11	-11	10	-10

<u>Examples:</u>

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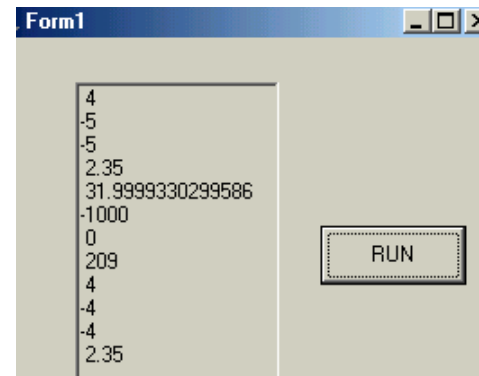
Print SGN (INT(4/3^8/4^3*3^5*2^5))

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

Form1

```
4
-5
-5
2.35
31.9999330299586
-1000
0
209
4
-4
-4
2.35
```

RUN

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

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

Example(3-2):

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

using quadratic formula as: $X_{1,2} = \dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$. 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?

Form1

الدستور

a =   Answer

b =
                       x1 =

19

c =   x2 =
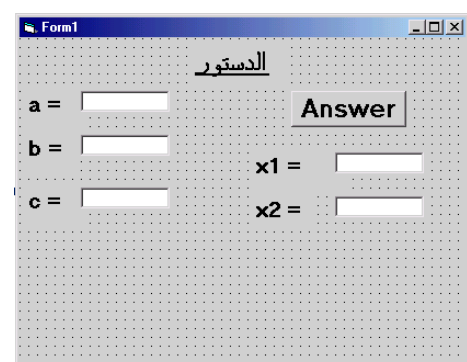
Solution:

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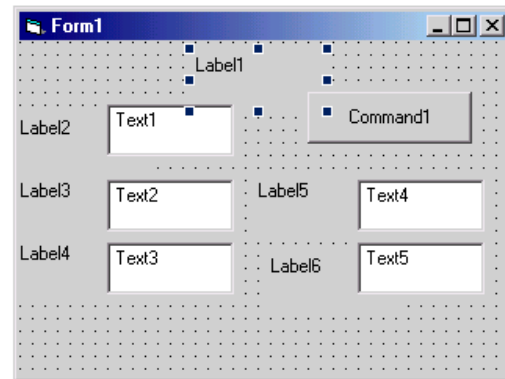
2- Set the form and object properties:

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

Box Title

Message Title

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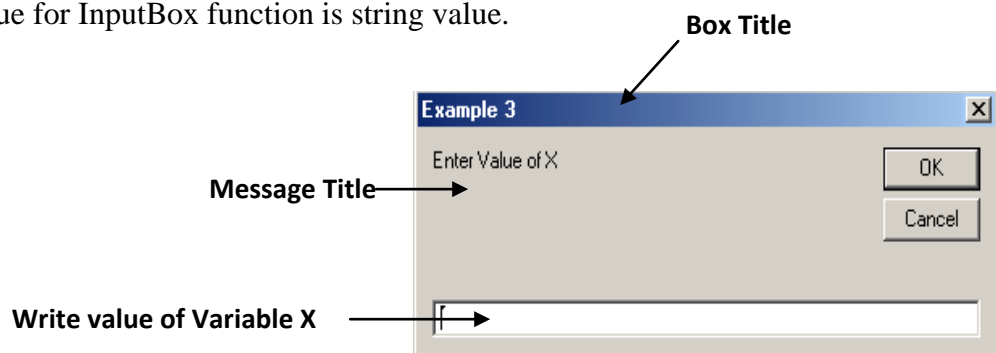
Example(3-3): Repeat Example(3-2). Using Input Box function to input value of a, b, and c.

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?

Perimeter=a+b+c   :   S= (a+b+c)/2

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

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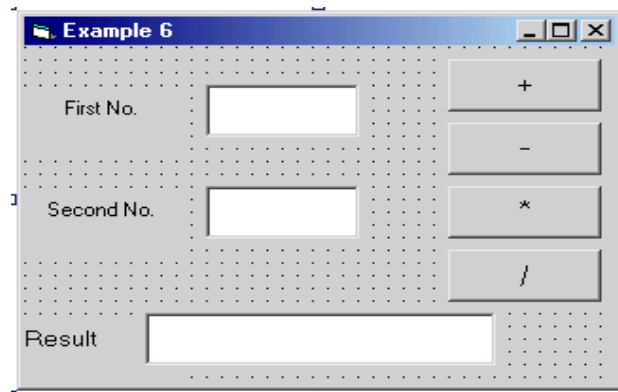
Exercise (3-3): Create a Visual Basic project to find the average value of three positive and integer variables (X1, X2, and X3). Find the deviation {(average-X1), (average-X2), and (average-X3)}. Print the value of *Average and Deviation* in PictureBox.

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.

 Angle= Degree + (Minutes/60) + (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.

3-13 Properties Method (Extension)

Before writing an event procedure for the control to response to a user's input, you have to set certain properties for the control to determine its appearance and how it will work with the event procedure. You can set the properties of the controls in the properties window at design time or at runtime. You can also change the properties of the object at runtime to give special effects such as change of color, shape, animation effect and so on. The properties that are discussed below are design-time properties.

properties	Description
Backcolor	Determines the background color of an object,
Caption	Holds the text that is to appear as the caption for a form, command button or label
Text	Holds the text that is to appear as the Text for a TextBox
Enable	Determines whether or not a form or control responds to events. The Enabled property of a control is set to (**True or False)**
FontBold	Determines whether the characters printed on an object appear in bold or normal type. The FontBold property is set to (**True or False)**
FontItalic	Determines whether the characters printed on an object appear in italic or upright type. The FontItalic property is set to (**True or False)**
FontUnderline	Determines whether the characters printed on an object appear with an underline. The FontUnderline property is set to (**True or False)**
FontSize	Determines the size, in prints, of characters printed on an object appear with an underline. Default font size is usually 8 point.
Font	Determines what type face is used when characters are printed on an object.
Vbcolor	Determines the color of object using the name of color.
Qbcolor(x)	Determines the color of object using the number (x) of color, where x between (1to15).
RGB(Red, Green, Blue)	Determines the colors, the range number for each color is between (0 To 255) .
Visible	Determines whether or not an object is displayed. The Visible property is set to (**True or False).**

 For example the following code will change the form color to yellow every time the form is loaded. Visual Basic-6 uses RGB(Red, Green, Blue) to determine the colors. The RGB code for yellow is 255,255,0. The code refer to the current form and Backcolor is the property of the form's background color. The formula to assign the Backcolor is:

Form1.Backcolor=RGB(255,255,0)   or   Form1.Backcolor=VbYellow

Here are some of the common colors and the corresponding VBcolor (Qbcolor (vbcode)) and RGB codes. You can always experiment with other combinations, but remember the maximum number for each color is 255 and the minimum number is 0.

VBCode	RGBcode	Color	Constant
0	(0,0,0)	Black	vbBlack
1	0, 0, 255	Blue	vbBlue
2	0,255,0	Green	vbGreen
3	50,100,0	Cyan	vbCyan
4	255, 165, 0	Brown	vbBrown
5	255, 0, 255	Magenta	vbMagenta
12	255,0,0	Red	vbRed
14	255, 255, 0	Yellow	VbYellow
15	255, 255, 255	White	vbWhite

Example 3-4

The following is another program that allows the user to enter the RGB codes into three different textboxes and when clicks the display color button, the background color of the form will change according to the RGB codes. So, this program allows users to change the color properties of the form at run time.

Solution:

```
Private Sub Command1_click()
Dim rgb1, rgb2, rgb3 As Integer
rgb1 = Val(Text1.Text)
rgb2 = Val(Text2.Text)
rgb3 = Val(Text13.Text)
Form1.BackColor = RGB(rgb1, rgb2, rgb3)
End Sub
```


Example 3-5: Design a form shown in figure below, with four commands button and one label. Enter "Name" in label1 from property window. When click on Command1 **(Bold)** or Command2 (**Italic)** or Command3 (**Underline),** the properties of label1 are changed. So when click on Command4 **(Normal)** replace the default properties of text are entered in Label1.

Solution:

```
Project1 - Form1 (Code)                              _ □ X
Command4            ▼    Click                        ▼

    Private Sub Command1_Click()
    Label1.FontBold = True
    End Sub

    Private Sub Command2_Click()
    Label1.FontItalic = True
    End Sub

    Private Sub Command3_Click()
    Label1.FontUnderline = True
    End Sub

    Private Sub Command4_Click()
    Label1.FontBold = False
    Label1.FontItalic = False
    Label1.FontUnderline = False
    End Sub
```
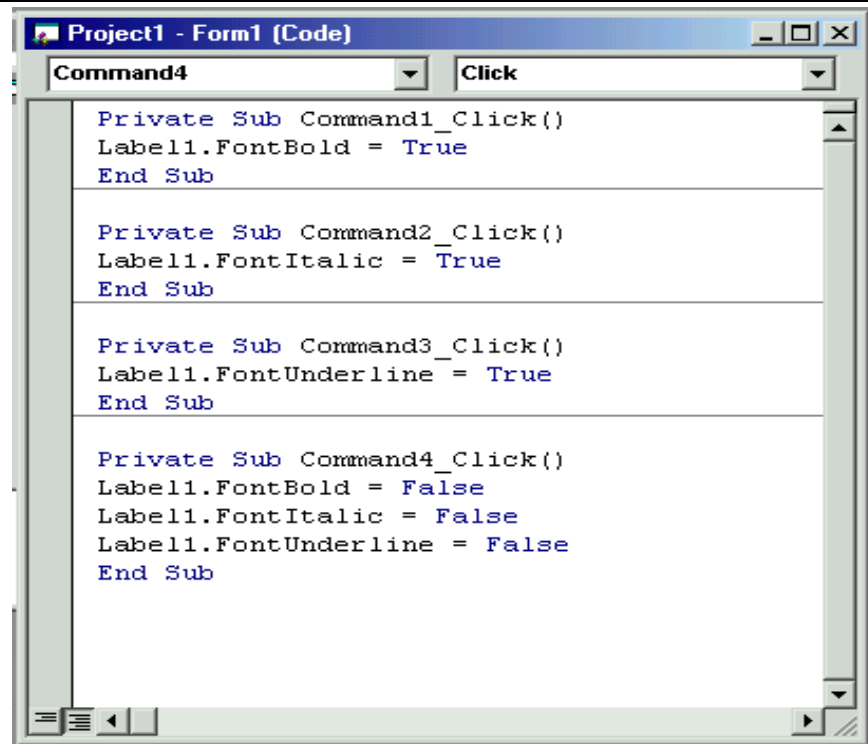
Example 3-6: Added four commands Button to the previous Example(3-5). When click on Command **(Larger)** or Command **(Smaller),** the font size of label1 (caption) is changed. Change the font color and the back ground of label1 by using commands Red and BackGreen respectively.

Solution:

```
Form1                    _ □ X
            AHMAD

                        Normal
   Bold
            Larger      RED
   Italic
            Smaller   Back-Green
  Underline
```

```
Project1 - Form1 (Code)                    _ □ X
Command4          ▼    Click                ▼

    Private Sub Command4_Click()
    Label1.FontBold = False
    Label1.FontItalic = False
    Label1.FontUnderline = False
    Label1.FontSize = 8
    Label1.ForeColor = vbBlack
    Label1.BackColor = vbWhite
    End Sub

    Private Sub Command5_Click()
    Label1.FontSize = Label1.FontSize + 2
    End Sub

    Private Sub Command6_Click()
    Label1.FontSize = Label1.FontSize - 2
    End Sub

    Private Sub Command7_Click()
    Label1.ForeColor = vbRed
    End Sub

    Private Sub Command8_Click()
    Label1.BackColor = vbGreen
    End Sub
```

25

3.14 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	OK
VBOKCancel	1	OK Cancel
VBAbortRetryIgnor	2	Abort Retry Ignore
VBYesNoCancel	3	Yes No Cancel
VBYesNo	4	Yes No
VBRetryCancel	5	Retry Cancel

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"

26
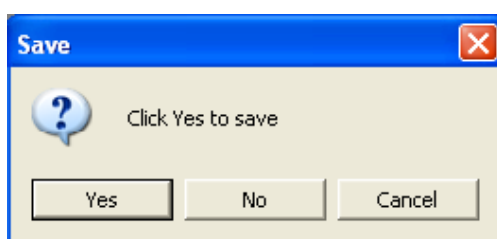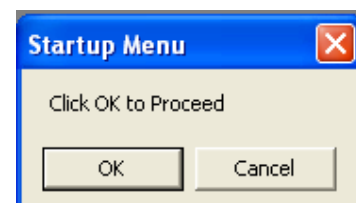
 TestMsg is a variable that holds values that are returned by the MsgBox ( ) function. The values are determined by the type of buttons being clicked by the users. It has to be declared as Integer data type in the procedure or in the general declaration section. Table 3-13.4 shows the values, the corresponding named constant and buttons.

Table 3-13.4: Return Values and Command Buttons

Value	Named Constant	Button Clicked
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

For example:
Private Sub form1_Load()
Dim testmsg As Integer
TestMsg = MsgBox("Click to test", 1, "Test message")
If TestMsg = 1 Then MsgBox "You have clicked the OK button": **End**
If TestMsg = 2 Then MsgBox "You have clicked the Cancel button": **Exit Sub**
End Sub

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.

For example:
Private Sub Form1_Load()
Dim testMsg As Integer
testMsg = MsgBox ("Click to Test", vbYesNoCancel + vbExclamation, "Test Message")
If testMsg = 6 Then MsgBox "You have clicked the yes button"
If testMsg = 7 Then MsgBox "You have clicked the NO button"
If testMsg =2 Then MsgBox "You have clicked the Cancel button")
End Sub
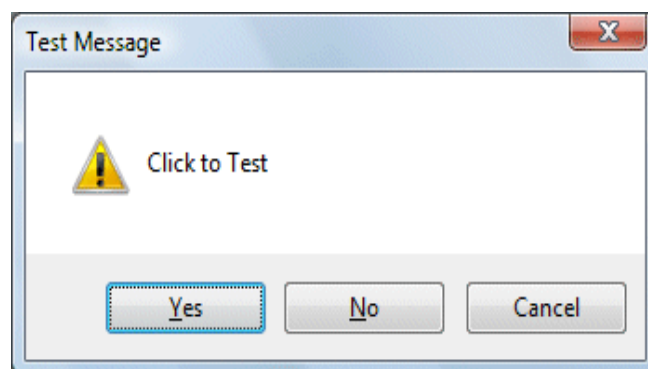
4. Control Structures

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 non-numeric 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):   or   **Solution(2):**

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

Method 1	**Method 2**
If *condition 1* Then	If *condition 1* Then
VB expression	*VB expression*
1 Else	ElseIf *condition 2* Then
If *condition 2* Then	*VB expression*
VB expression	ElseIf *condition 3* Then
2 Else	*VB expression*
If *condition 3* Then	Else 1
VB expression	*VB expression*
3 Else	End If
VB expression	
End If	
End If	
EndIf	

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:
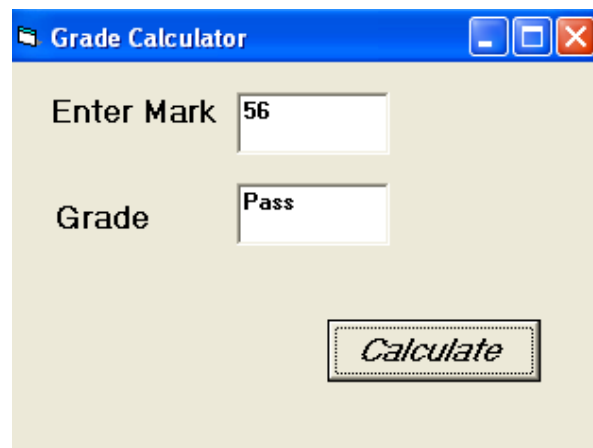
Solution1	Solution2
Private Sub Command1_click()	Dim Mark As Single , Grade as String
Dim Mark As Single , Grade as String	Mark = Val (Text1.Text)
Mark = Val (Text1.Text)	**Select Case** Mark
Select Case Mark	**Case Is** >100 , Is < 0
Case 0 To 49	Msgbox "Wrong entry, please Re-enter the mark", 16
Grade="Fail"	, " Error"
Case 50 To 59	Text1.Text=" " : Text2.Text= " " : **Exit Sub**
Grade="Pass"	**Case Is** > = 90
Case 60 To 69	Grade="Excellent"
Grade="Medium"	**Case Is** > = 80
Case 70 to 79	Grade="Very Good"
Grade="Good	**Case Is** >= 70
Case 80 To 89	Grade="Good
Grade="Very Good"	**Case Is** >= 60
Case 90 To 99	Grade="Medium"
Grade="Excellent"	**Case Is** >=50
Case Else	Grade="Pass"
Msgbox "Wrong entry, please Re-enter the mark", 16	**Case Else**
, " Error"	Grade="Fail"
Text1.Text=" " : Text2.Text= " " : **Exit Sub**	**End Select**
End Select	Text2.Text=Grade
Text2.Text=Grade	End Sub
End Sub	

Examples:

- Select Case X
  Case 3, 5, 8 : Print X   Value of X (3 or 5 or 8) only.
  End Select

32

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

- 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 $a X^2 + bX + c = 0$, using quadratic formula as: $X_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. 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.

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

$$F(Z) = \begin{cases} \dfrac{Z^3 + 5\,Ln(Z)}{Log(Z) + 1} & Z > X^2 + 1 \\[4mm] \dfrac{Z^2 + 4}{2\,e^z + 1} & Z < X^2 + 1 \end{cases}$$

Exercise 4.3: Use the Newton Raphson method to estimate the root of the function $e^{-X} = X$, employing initial value of $X_{old} = 0.0$. Write visual Basic code to compute the root X_{new} . Print the value of X_{new} when the value of $|X_{new} - X_{old}| \leq 0.0001$. Design the form with two text box and one command button.

Note: $X_{new} = X_{old} - \dfrac{f(X_{old})}{\acute{f}(X_{old})}$

4.4 OptionButton Controls:

OptionButton controls are also known as radio buttons because of their shape. You always use OptionButton controls in a group of two or more because their purpose is to offer a number of mutually exclusive choices. Anytime you click on a button in the group, it switches to a selected state and all the other controls in the group become unselected.

Preliminary operations for an OptionButton control are similar to those already described for CheckBox controls. You set an OptionButton control's Caption property to a meaningful string, and if you want you can change its Alignment property to make the control right aligned. If the control is the one in its group that's in the selected state, you also set its Value property to True. (The OptionButton's Value property is a Boolean value because only two states are possible.) Value is the default property for this control.

At run time, you typically query the control's Value property to learn which button in its group has been selected. Let's say you have three OptionButton controls, named opt10, opt100, and opt1000. You can test which one has been selected by the user as follows:

If opt10.Value=True Then
Y=X*10
ElseIf opt100.Value=True Then
Y=X*100
ElseIf opt1000.Value=True Then
Y=X*1000
End If
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.

4.5 CheckBox Control
The CheckBox control is similar to the OptionBoutton, except that a list of choices can be made using check boxes where you cannot choose more than one selection using an OptionButton. By ticking the CheckBox the value is set to True. This control can also be grayed when the state of the CheckBox is unavailable, but you must manage that state through code.

When you place a CheckBox control on a form, all you have to do, usually, is set its Caption property to a descriptive string. You might sometimes want to move the little check box to the right of its caption, which you do by setting the Alignment property to 1-Right Justify, but in most cases the default setting is OK. If you want to display the control in a checked state, you set its Value property to 1-Checked right in the Properties window, and you set a grayed state with 2-Grayed.

The only important event for CheckBox controls is the Click event, which fires when either the user or the code changes the state of the control. In many cases, you don't need to write code to handle this

event. Instead, you just query the control's Value property when your code needs to process user choices. You usually write code in a CheckBox control's Click event when it affects the state of other controls. For example, if the user clears a check box, you might need to disable one or more controls on the form and re enable them when the user clicks on the check box again.

Example 4.6: The following example illustrates the use of CheckBox control and Option Button.
* Open a new Project and save the Form as CheckBox.frm and save the Project as CheckBox.vbp
* Design the Form as shown below

Object	Property	Setting
Form	Caption Name	CheckBox Frm
CheckBox	Caption Name	Bold chkBold
CheckBox	Caption Name	Italic chkItalic
CheckBox	Caption Name	Underline chkUnderline
OptionButton	Caption Name	Red optRed
OptionButton	Caption Name	Blue optBlue
OptionButton	Caption Name	Green optGreen
TextBox	Name Text	Txt1 (empty)
CommandButton	Caption Name	Exit cmdExit
Frame1	Caption	Empty
Frame2	Caption	Empty

Solution: Following code is typed in the Click() events of the CheckBoxes

```
Private Sub chkBold_Click()
If chkBold.Value = 1 Then
  Txt1.FontBold = True
Else
  Txt1.FontBold = False
End If
End Sub
```

```
Private Sub chkItalic_Click()
If chkItalic.Value = 1 Then
  Txt1.FontItalic = True
Else
  Txt1.FontItalic = False
End If
End Sub
```

```
Private Sub chkUnderline_Click()
If chkUnderline.Value = 1 Then
  Txt1.FontUnderline = True
Else
  Txt1.FontUnderline = False
End If
End Sub
```

Following code is typed in the Click() events of the OptionButtons

```
Private Sub optRed_Click()
  Txt1.ForeColor = vbRed
End Sub
```

```
Private Sub optBlue_Click()
  Txt1.ForeColor = vbBlue
End Sub
```

```
Private Sub optGreen_Click()
  Txt1.ForeColor = vbGreen
End Sub
```

To terminate the program following code is typed in the Click() event of the Exit button

```
Private Sub cmdExit_Click()
  End
End Sub
```

Very often when you visit a website, you are presented with a list of choices for you to select which you like to receive info related to those choices. Choice selection can easily be programmed in Visual Basic, the control involved is the check box. The status of the check box is either checked or unchecked, and the syntax is Checkbox1.Value=VbChecked or Checkbox1.Value=Unchecked. In the following program, we constructed a three-choice selection list. After the user made the selection, a message box will appear to display the list of choices selected. The codes are as follow:

Example 4.7:

```
Private SubCommand1_Click()
If Check1.Value =vbChecked And Check2.Value =vbChecked And Check3.Value = vbChecked Then
MsgBox  "You like Reading, Computer and Sports"
ElseIf Check1.Value =vbChecked And Check2.Value =vbChecked And Check3.Value= vbUnchecked Then
MsgBox "You like Reading and Computer"
ElseIf Check1.Value = vbChecked And Check2.Value = vbUnchecked And Check3.Value = vbChecked Then
MsgBox "You like Reading and Sports"
ElseIf Check1.Value = vbUnchecked And Check2.Value = vbChecked And Check3.Value = vbChecked Then
MsgBox "You like Computer and Sports"
ElseIf Check1.Value = vbChecked And Check2.Value = vbUnchecked And Check3.Value = vbChecked Then
MsgBox "You like Reading and Sports"
ElseIf Check1.Value = vbChecked And Check2.Value= vbUnchecked And Check3.Value= vbUnchecked Then
MsgBox "You like Reading only "
ElseIf Check1.Value= vbUnchecked And Check2.Value= vbChecked And Check3.Value = vbUnchecked Then
MsgBox "You like computer only"
ElseIf Check1.Value=vbUnchecked And Check2.Value=vbUnchecked And Check3.Value=vbChecked Then
MsgBox "You like Sports only"
Else
MsgBox "You have no hobby"
End If
End Sub
```


38
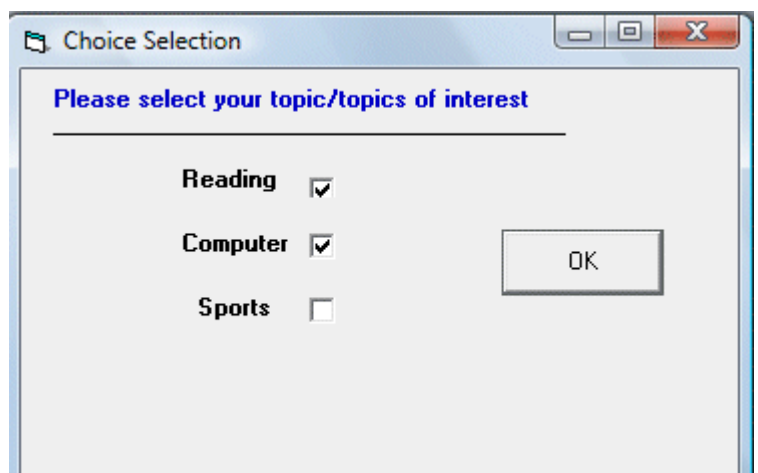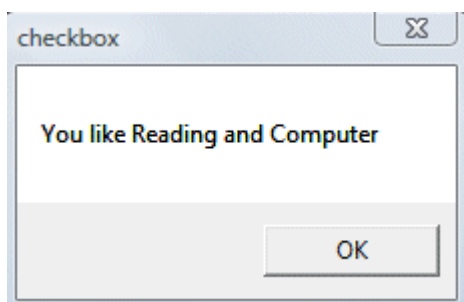
Example4.8: Write a program to enter an angle value (Degree, Minutes, and Seconds) into separate text boxes Calculate the value of angle (in degree only). Display *Angle* in separate text box. When the user click the option button, calculate the value of the function (Sin, Cos or Tan) and display in separate text box. If the value of Minutes or seconds exceeded 60, write a message box (The value of minutes or seconds exceeds 60) to stop the program. Display (division by zero) in separate text box, when calculate the tan function of angle (90, 270,..).

Solution:
```
Private Sub Command1_Click()
Dim deg As Double, minut As Double, second As Double
Dim total As Double, p As Double
deg = Val(Text1.Text)
minut = Val(Text2.Text)
second = Val(Text3.Text)
If minut > 60 Or second > 60 Then
res = MsgBox("The value of minutes or seconds exceeds 60", 0)
If res = 1 Then  Text1.Text = "": Text2.Text = "": Text3.Text = "": Exit Sub
End If
total = deg + (minut / 60) + (second / 3600)
Text4.Text = Str(total)
p = 3.141596 / 180
If Option1.Value = True Then
Text5.Text = Str(Sin(p * total))
End If
If Option2.Value = True Then
Text5.Text = Str(Cos(p * total))
End If
If Option3.Value = True Then
```
If Cos(p * total) <= Abs(0.00001) Then Text5.Text = "division by zero": Exit Sub
```
Text5.Text = Str(Tan(p * total))
End If
End Sub
```

Example 4.9: For a simply supported beam subjected to a uniform load (W) on the length of span (L) and a concentrated load (P) on a mid span (L/2). When the user click checkbox1, enter the value of (P) and display the value of bending moment (Mom) in a separate text box, when the user click on the checkbox2, enter the value of (W) and display the value of bending moment (Mom) in a separate textbox. Write a program in a separate command button (Calculate) to find the value of (Mom) at mid span of beam subjected to (W) or (P) or both of them.

Solution:

```
Private Sub Command1_Click()
Private Sub Command1_Click()
Dim l, p, w, mom1, mom2, mom
l = Val(Text1.Text)
If Check1.Value = 1 Then
p = Val(Text2.Text)
mom = p * l / 4:mom=mom1
Text4.Text = Str(mom)
Else
Text4.Text = ""
End If
If Check2.Value = 1 Then
w = Val(Text3.Text)
mom = w * l ^ 2 / 8: mom2=mom
Text4.Text = Str(mom2)
Else
End If
If Check1.Value = 1 And Check2.Value = 1 Then
mom = mom1 + mom2
Text4.Text = Str(mom)
End If
End Sub
```

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 *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.

• **Exit For**	• **Exit Do**
The formats are:	The formats are
For *counter*= start **To** end step (increment)	**Do While** *condition*
Statements	Statements
Exit for	**Exit Do**
Statement	Statements
Next *counter*	**Loop**

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
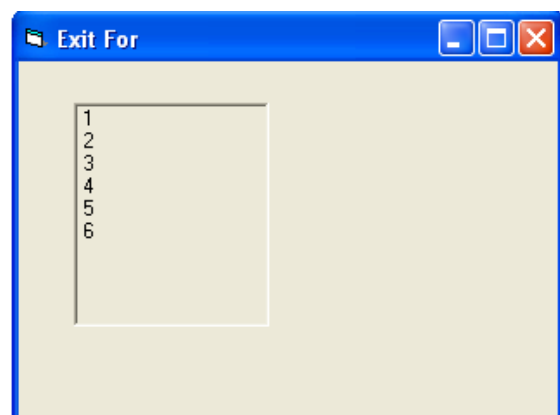
1- Private sub Form Load_( )
   Form1.show
   Dim x As Integer
   X=0
   Do While x < 10
   Print x
   x = x + 1
   If x = 5 Then
   Print "The program is exited at x=5"
   Exit Do
   End If
   Loop
   End Sub
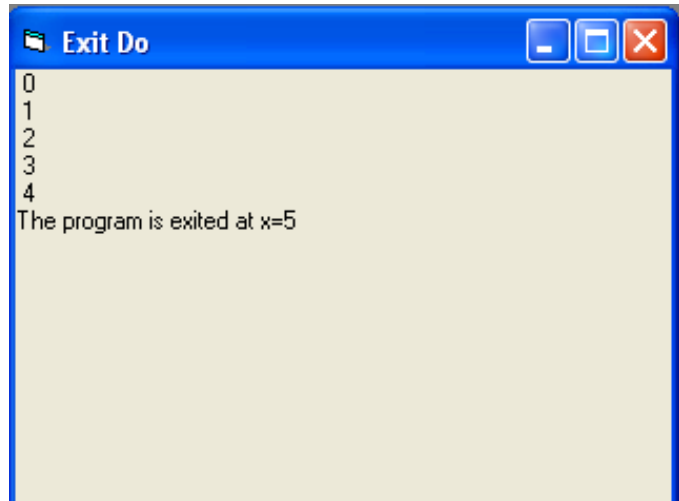
5-4 Nested Loop: The nested loops are the loops that are placed inside each other. The most inner loop will be executed first, then the outer ones. These are examples of the nested loops.

Possible	**Error (Not Possible)**
For J=1 to 5	**For K=1 to 5**
Statement	Statement
For I=1 to 5	For I=1 to 5
Statement	Statement
Next I	**Next K**
Statement	Statement
Next J	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.

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

$$\Delta_x = \frac{P\,a}{6EI_G}\left(3L\,x - 3\,x^2 - a^2\right) \quad a \le x \le L - a$$
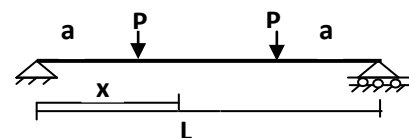
Solution:

```
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-a= 2
0 0
Def= 0.15 0.1494375
0.3 0.2955
0.45 0.4348125
0.6 0.564
0.75 0.6796875
0.9 0.7785
1.05 0.857083333333334
1.2 0.913333333333333
1.35 0.947083333333333
1.5 0.958333333333333
1.65 0.947083333333334
1.8 0.913333333333333
1.95 0.857083333333333

Value of X greater than L-a
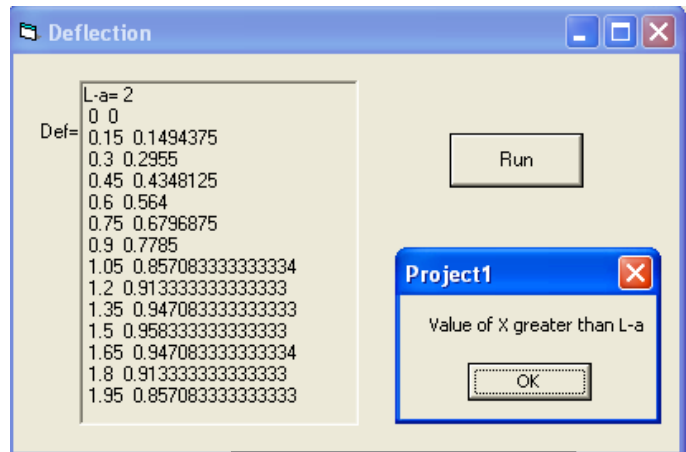
L=3: P=1:a=1:E=1: IG=1

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"
```

46
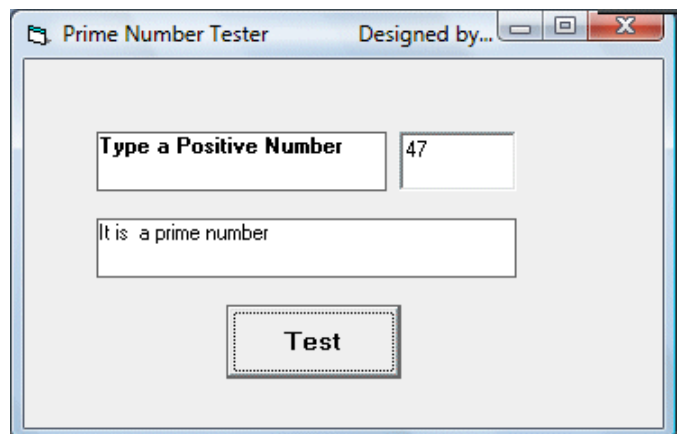
```
End If
End Select
End Sub
```

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

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

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.

$$Sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

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#
  10Fact = 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.

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

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.

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

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.

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

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

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 through 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.
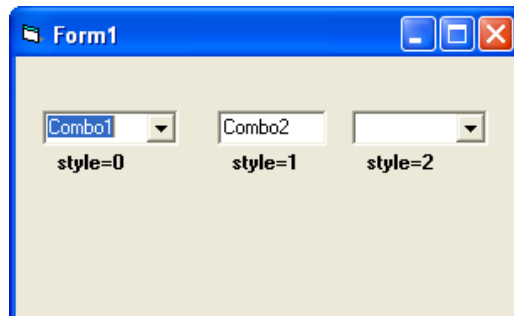
With a style 1 combo box, the list is always visible. With style 0 or 2 combobox, the list drops down when the user clicks on the arrow. In either case, when an item from the list is highlighted, the item automatically appears in the text box at the top and its value is assigned to the text property 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
ListIndex	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
List	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)
Text	String. Specifies the selected item in the ListBox &ComboBox.	X=List1.Text: Y=Combo1.Text
Clear	Removes all items from the ListBox &ComboBox	List1.Clear: Combo1.Clear
RemoveItem	Removes the specified item from the ListBox &ComboBox	List1.RemoveItem 1: Combo1. RemoveItem 5 List1.RemoveItem List1.ListIndex
NewIndex	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 &ComboBox : 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.
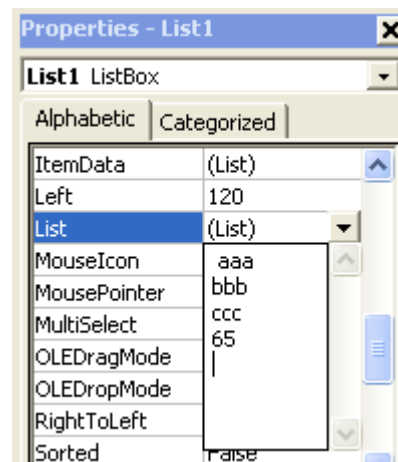
Example 5-9: Design a form with one list box, two textboxes and two command buttons. Write code for the following-events.
1- Form_Load event, to add items.(5)
2- In click event of listbox,to add item to text1 from list box if item is selected
3- In click event of command1 (Remove), to remove item from 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
Text2.Text=List1.ListCount
EndSub
```
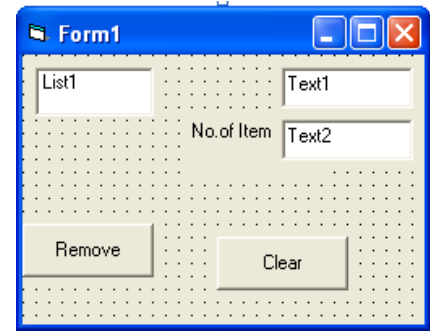
```
Private Sub List1_Click()
Text1.text=List1.Text          or      Text1.Text=List1List(List1.ListIndex).
EndSub
```

```
Private Sub Command1_Click()
List1.RemoveItem List1.ListIndex
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
```

```
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
```

Example 5-11: Write a code program to design a simple calculator. This program uses a combo box which includes four operators, addition, subtraction, multiplication and division and two list boxes included the integer numbers. It can perform the above four basic calculations by changing the operators.

Solution:

```
Private Sub Form_Load()
Dim I
For I=1 To 10
List1.AddItem Str(I)
List1.AddItem Str(I)
Next I
Combo1.AddItem "+"
Combo1.AddItem "-"
Combo1.AddItem "*"
Combo1.AddItem "/"
```

```
Private Sub Command1_Click()
Dim z
Select Case Combo1.Text
Case "+"
z = Val(List1.Text) + Val(List2.Text)
Case "-"
z = Val(List1.Text) - Val(List2.Text)
Case "*"
z = Val(List1.Text) * Val(List2.Text)
Case "/"
z = Val(List1.Text) / Val(List2.Text)
Case Else
Exit Sub
End Select
Text1.Text = Str(z)
End Sub
```
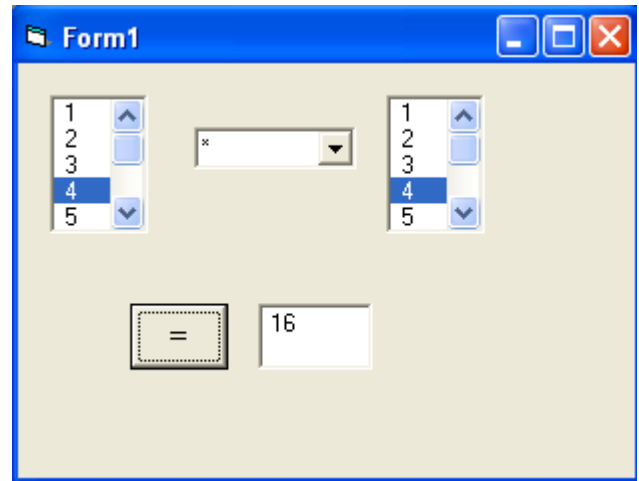
6- Arrays in Visual Basic 6

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 6 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

If we want to specify the lower limit, then the parentheses should include both the lower and upper limit along with the To keyword. An example for this is given below.

Dim Num (6) As Integer

Num (6)	Num (0)	Num (1)	Num (2)	Num (3)	Num (4)	Num (5)	Num(6)
	-	1	3	-10	5	3	2

In the above statement an array of 10 elements (Num(10)) is declared but with indexes running from 1 to 6.

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).

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
End Sub

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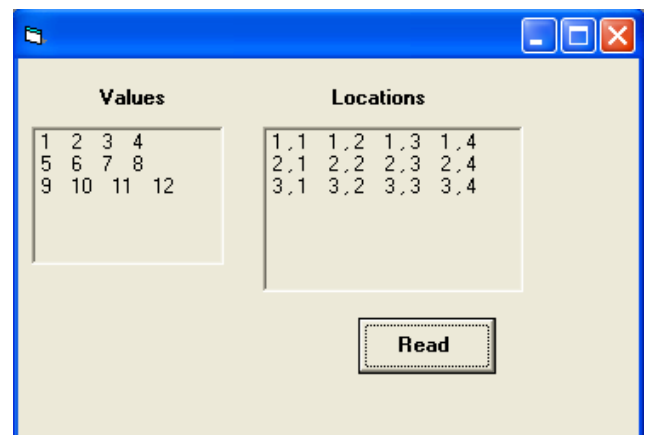
Example 6-7: Write a code program to read of two dimensional array A(3,4) on a column by column. Print the value and position of each element.

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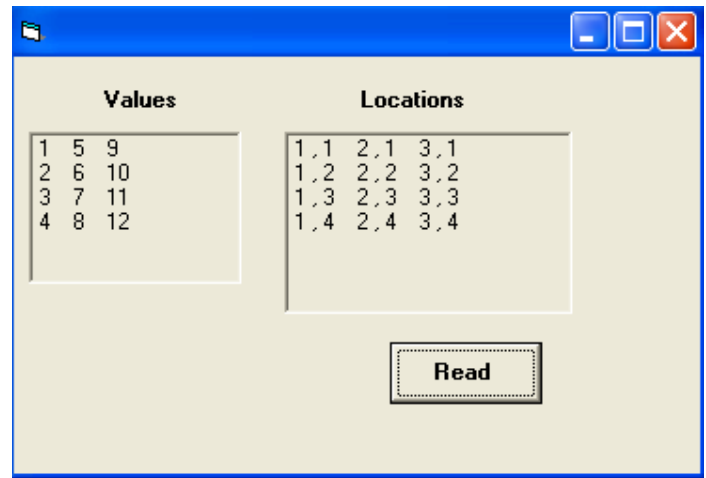
Example 6-8: Write a code program to create a two dimensional array N (5X2) into List Box on row by row. Print the values of array N.

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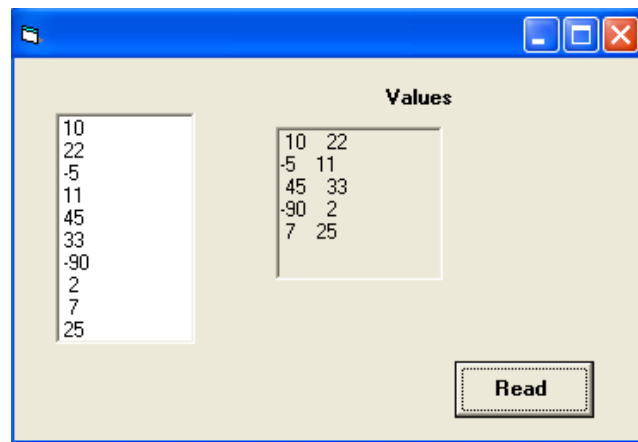
Example 6-9: Suppose N is a (5x2) matrix array is entered into ListBox on row by row. Write a program segment to find the location I and J such that N (I,J) contains the largest value in N. Print the values of array N. Display the Largest value and the location into textboxes.

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)

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 3
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
```
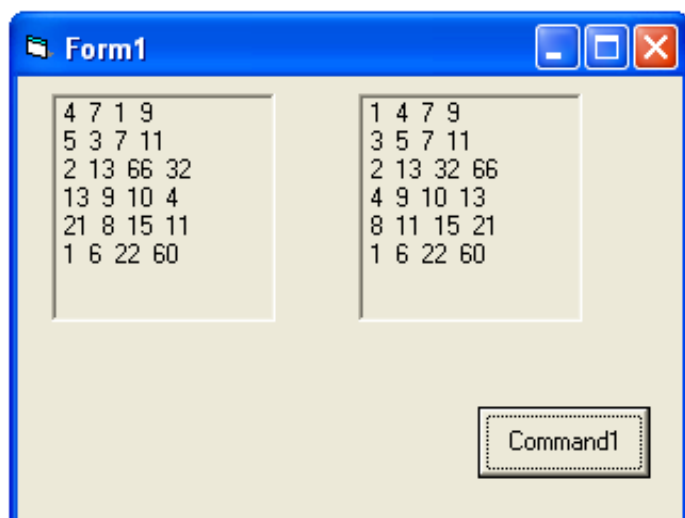
For I=1 To 6
For J =1 To 4
Picture1.print W(I,J);
Next J : Picture1.Print : Next I

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
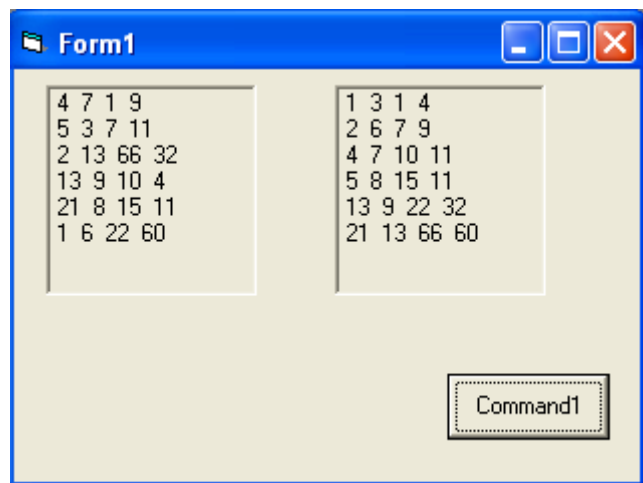
- **6-1-2 Dynamic Array:** The size of the array can be changed at the run time- size changes during the program execution.

In actual practice, the amount of data that a program will be processing is not known in advance. Programs should be flexible and incorporate a method for handling varying amounts of data. Visual basic makes this possible with the statement

ReDim *ArrayName* (N) As *Var Type*

This can use variables or expression when indicating the subscript range. However, ReDim statements can only be used inside procedures.

.

Example 6-14: Suppose a one dimension array A with N elements is entered into inputbox. Write a visual basic program segment which is calculated the sum of the squares: Sum=$\sum_{i=1}^{N}(A_i)^2$

Solution:
```
Dim  N
N= Val (InputBox("N="))
ReDim A(N)
For I=1 To N
A(I) =Val(InputBox(""))
Sum=Sum+A(I)^2
Next
Text1.Text=Str (Sum)
```

Example 6-15: An MxN matrix array A is entered into input box. Write a visual basic program segment which is calculated the SUM of elements in each Column and Stored in a one dimension Z. Print the arrays A and Z and the sum of all elements of array Z.

Solution:
```
Dim M, N
M=Val (Text1.Text)  :  N= Val (Text2.Text)
ReDim A (M, N), Z(N)
For I=1 To M
For J=1 To N
A(I,J) = Val (InputBox(""))
Next J , I
For J=1 To N
Sum=0
For I=1 To M
Sum=Sum+ A(I,J)
Next I
Z(J)=Sum
Total=Total +Z(J)
Next J
For I =1 To M
For J=1 To N
Picture1.Print A(I,J);
Next J: picture1.print: Next I
For I=1 To N
Picture2.print Z(I)
Next
Text3.text=Str(Total)
```

Example 6-16: Suppose a string array Name(N) and a numeric array Grade(N,8). Write A code program to read the student name and student grade for 8 subjects. Extend the program to calculate student grades average, stored the average grades into (column number 9). Sort the average grades so that its elements are increasing. Display the student name and the average grades for all students.

Solution:

```
Dim N As Single
N= Val (InputBox("Number of Student"))
ReDim Name (N) As String,Grade(N,9) As Single
For I=1 To N
Name(I)=Val(InputBox("Name"))
For J=1 To 8
Grade(I,J) =Val(InputBox("grade" & Str(J) ) )
Next J, I
For I=1 To N
Sum=0
For J=1 To 8
Sum=Sum+ Grade (I, J)
Next J
Grade (I,9)=Sum / 8
Next I
For I=1 To N-1
For J=I+1 To N
If Grade (J, 9) < Grade(I, 9) Then
For K=1 To 9
C=Grade (I,K)
Grade (I,K)=Grade(J,K)
Grade (J,K)=C
D=Name (I)
Name (I)= Name(J)
Name(J)=D
Next K
End If
Next J, I
For I=1 To N
Picture1.Print Name(I)
For J= 1 To 9
Picture2.Print  Grade(I,J);
Next J: Picture2.print
Next I
```

Example 6-17: Suppose a numeric array A having M row and N columns, and a numeric array X having N elements. Write a code Program to calculate a numeric array Y by carrying out the following operations. Print a new array Y.

$$[Y]_m = [A]_{m,n} * [X]_n$$

Solution:

Dim M, N

M=Val (Text1.Text) : N= Val (Text2.Text)

ReDim A (M, N), X(N), Y(M)

For I=1 To M

For J=1 To N

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

Picture1.Print A(I, J);

Next J: Picture1.Print: Next I

For I=1 To N

X(I)=Val (Inputbox(""))

Picture2.Print X(I)

Next I

For I=1 To M

For J=1 To N

Y(I)=Y(I)+A(I,J)*X(J)

Next J, I

For I=1 To M

Picture3.Print Y(I)

Next I

Note: Each element of Y is determined from the following formula:

Y (**1**) = A(**1, 1**)*X(**1**)+A(**1, 2**)*X(**2**)

Y (**2**) = A(**2, 1**)*X(**1**)+ A(**2, 2**)*X(**2**)

Y (**3**) = A(**3, 1**)*X(**1**)+A(**3, 2**)*X(**2**)

Example 6-18: Suppose a numeric array (A) having (5 row and 3 columns), and that (B) is a numeric array having (3 rows and 5 columns). Write a code Program to calculate the elements of the numeric array C, where a numeric array C is determined from the following formula:

$[C] = [A] * [B]$. Print a numeric array C.

Solution: Each element of array C is determined by $[C]_{m,n} = [A]_{m,k} * [B]_{k,n}$

Dim A(5,3) , B(3,4), C(5,4)

For I=1 To 5

For J=1 To 3

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

Next J , I

For I=1 To 3

```
For J=1 To 4
B(I,J)=Val(InputBox("B"))
Next J , I
For I=1 To 5
For J=1 To 4
For K=1 To 3
C(I,J)=C(I,J) + A( I,K ) * B( K,J )
Next K, J, I
For I=1 To 5
For J=1 To 4
Picture1.Print C(I,J);
Next J: Picture1.Print: Next I
```

Note: Each element of C is determined from the following formula:
C(1,1)= A(1,1)*B(1,1)+ A(1,2)*B(2,1) +A(1,3)*B(3,1)
C(2,2)=A(2,1)*B(1,2)+ A(2,2)*B(2,2)+ A(2,3)*B(3,2)

Example 6-19: Suppose a numeric array (A) having M elements. Write a code Program to calculate the elements of the numeric array C, where a numeric array C is determined from the following formula: $[C] = [A] * [A]^T$. Print a numeric array C.

Solution: Where B=A^T

```
Dim M As Single
M=Val(Text1.text)
ReDim A(M),B(1,M), C(M,M)
For I=1 To M
A(I) =Val (InputBox(""))
B(1,I)=A(I)
Picture1.Print A(I)
Picture2.Print A(I);
Next I
For I= 1 To M
For J=1 To M
C(I,J)= C(I,J) +A(I)*B(1,J)
Next J, I
For I=1 To M
For J=1 To M
Picture3.print C(I,J);
Next J: Picture3.Print: Next I
```

Exercises:

6-1: Suppose a one dimension array A with N elements is entered into inputbox. Write a visual basic program segment which is calculated the product: $Prod = \prod_{i=1}^{N}(1 - A_i)$

6-2: Consider a sequence of real numbers, X_i, i= 1,2,...., M. the mean is defined as:

$\overline{X} = \frac{\sum_{i=1}^{m} X_i}{m}$, the standard deviation is $\sigma = \sqrt{\frac{\sum(X_i - \overline{X})^2}{m}}$

Write a code program to read in the first M elements of a one dimension array. Calculate the mean and the standard deviation.

6-3: Write a code program that will read in a set of temperatures T(M), determine an average, and then calculate the deviation of each temperature about the average D(M). The deviation is defined as D(I)=T (I)- A

Where, A represents the average temperature. Print out the average temperature, followed by three columns containing the values for I, T (I) and D(I), respectively.

6-4: To compute the area under the curve for the function $f(x) = 3x^3 - 4x^2 + 6x + 5$ in an interval $(a \leq x \leq b)$ is equal to $\int_{a}^{b} f(x)dx$. Write a code program to reads a, b and N (where N is a number of segments) respectively. Calculate the numerical array F(N) and the approximate area by the Trapezoid rule from the following formula. Display the approximate area into separate text box.

$$area \approx \frac{1}{2}h\left[f(a) + f(b) + 2\sum_{i=1}^{N-1} f(x_i)\right]$$

Where: $h = \frac{b-a}{N}$: $x_i = a + h.i$

And F(0)=F(a): F(N)=F(b)

6-5: Suppose a numeric array S(10,3). Write a code program to read the elements of the array S (three positive integers a, b, c) for each Pythagorean triplet. Calculate the area for all Pythagorean triplets and stored into a numeric one dimension array (Area), where a form a Pythagorean triplet if $a^2 + b^2 = c^2$. Print the arrays S and Area.

7- Graphics in Visual Basic

7.1 Introduction: Graphics are the elements of a picture. Colors, lines, rectangles, patterns, text, etc. are all graphics. Graphics are visual. Visual Basic provides graphics capabilities for drawing shapes in different colors and patterns. Visual basic is also capable of displaying many popular image formats. Although the graphics capabilities may not be as feature rich as graphics software programs, visual basic's graphic capabilities are integral to creating polished windows applications.

7.2 Coordinate Systems: To draw in visual basic, we must understand Visual Basic's coordinate system as shown in figure below, that identifies points on the screen (such as forms or pictureboxes). By default, the upper-left point on the screen has coordinate (0,0), which is commonly called the origin. A coordinate pair is composed of an *x coordinate* (the horizontal coordinate) and *y coordinate* (the vertical coordinate). The x coordinate is the horizontal distance on the x axis from the origin. The y coordinate is the vertical distance on the y axis from the origin. The unit that a coordinate system is measured in is *called a scale*. Visual basic provides eight coordinate system scales. Most controls as well as the form use twips by default. Property *ScaleMode* specifies the scale.
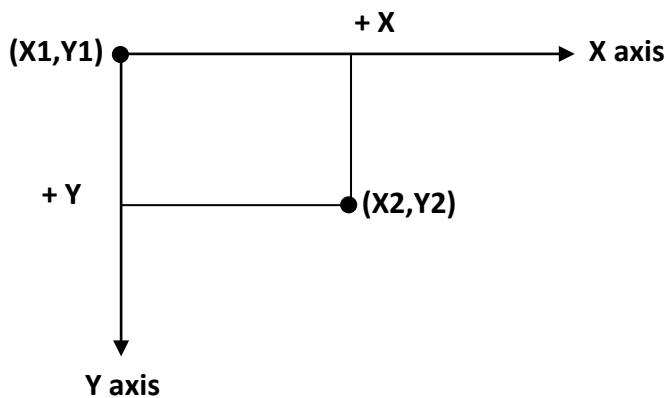
User-defined coordinates are fdefined using method scale. Two set of coordinates define the scale. The first coordinate set defines the upper-left corner and the second coordinate set defines the lower-right corner. The statement,

Scale (xx1 , yy1) - (xx2 , yy2)

For example:

- **Scale(0,0)-(100,100)**
- **Scale (100,100)-(0,0)**
- **Scale(-100,100)-(100,-100)**

7.2 Graphics Method:
Visual basic provides several methods for creating graphics. The graphics methods, summarized in the following table, apply to forms.

Method	Description
Line	**Draws lines on a form. Can also be used to draw rectangles.**
Circle	**Draws circles on a form. Can also be used to draw ellipses**
Pset	**Sets a point's color**
Print	**Draw text on a form.**

❖ **Method Line:** draws lines and rectangles between two sets of coordinates. The first set of coordinates is the starting point and the second is the ending point.

Line (x1,y1)-(x2,y2),color

For example:
- **Line (0,0)-(100,100),VbBlue**
- **Line(100,50)-(50,50),QbColor(5)**
- **Line(50,50)-(50,100),RGB(45,100,10)**

For rectangles (also called boxes) the first coordinate set specifies the upper-left corner and the second specifies the lower-right corner.

Line (x1,y1)-(x2,y2),color, B   [or BF]
The visual basic constant (Vb) which represents the color name, the third argument (**B**), indicates that the method should draw a rectangle. A third argument of (**BF**) would indicate that the rectangleshould be filled (solid). For example

- **Line (0,0)-(55,21), , B**
- **Line(25,50)-(75,100), , Bf**
  -

➢ **Note:** There are three ways to specify a color value at run time.
  1- Use RGB(1To 255, 1To 255,1 To 255) function
  2- Use the QBColor(1 to 15) function to choose one of 15 Microsoft QuickBasic color as shown in table below
  3- Enter a color value directly (VbColor) as shown in table below.

Vb Code	Color	Constant	Vb Code	Color	Constant
0	Black	vbBlack	8	Grey	vbGrey
1	Blue	vbBlue	9	Light Blue	vbLightBlue
2	Green	vbGreen	10	Light Green	vbLightGreen
3	Cyan	vbCyan	11	Light Cyan	vbLightCyan
4	Red	vbRed	12	Light Red	vbLightRed
5	Magenta	vbMagenta	13	Light Magenta	vbLightMagenta
6	Brown	vbBrown	14	Yellow	vbYellow
7	White	vbWhite	15	Bright White	vbBrightWhite

❖ **Method Circle:** draws circles, ellipses, arcs, and sectors. A circle's radius is the distance from the circle's center to any circle point. An ellipse differs from a circle in that its aspect ratio (the ratio of height to width) is not 1.Arcs is the curved portion of sectors. Sectors are wedge shaped pieces of a circle. Radians (from **0 to 2 π**) must be used for sector and arc angles.

Circle (x1,y1), radius, color, start angle, end angle, proportion

> **For Example**
> **Scale (0, 0)-(100, 100)**
> **pi = 3.14156**

- **Circle (50, 25), 5**   **circle**

Circle (15, 15), 5, , 0, pi / 2   **Arc (C.C.W angle (+))**

- **Circle (15, 30), 5, , 0, pi**   **Arc (C.C.W angle (+))**

- **Circle (15, 45), 5, , pi / 2, 3 * pi / 2**   **Arc (C.C.W angle (+))**

- **Circle (15, 60), 5, , pi, 0**   **Arc (C.C.W angle(+))**
-
- **Circle (15, 80), 5, , -pi / 2, -3 * pi / 2**   **Sector (C.W angle (-))**

- **Circle (45, 80), 5, , -pi, -pi / 2**   **Sector (C.W angle (-))**

- **Circle (65, 80), 5, , -pi / 2 , -pi**   **Sector (C.W angle (-))**

- **Circle (85, 80), 5, , pi / 2 , - pi**   **Sector (C.W angle (-))**
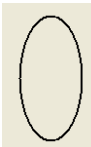
- **Circle (50, 45), 5 , , , , , 2**   **Ellipse $\frac{Y}{X} > 1$**
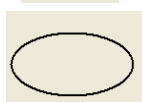
- **Circle (50, 65), 5 , , , , , 0.5**   **Ellipse $\frac{Y}{X} < 1$**

❖ **Method Pset:** turns on a point by changing the color at the point for example, the statement
   Pset (x,y),color
   Pset(40,40),VbRed

❖ **Method Print:** To draw text on the form. The default X coordinate is 0 and visual Basic automatically increments the y coordinate to draw on the next line. The current drawing coordinates are stored in properties *currentX* and *currentY.* For Example:

CurrentX=1
CurrentY=3
Print "Visual Basic"

Note: you can use the (*Form1.ForeColor*) property to specify any color to draw text.

7.3 Graphics Properties: Several drawing properties can be used with drawing methods. In this section, we introduce properties:

1- *DrawWidth*: The draw width property specifies the width of line for output from the graphics methods. For Example :

Private Sub Form Activate ()
Scale(0,0)-(100,200)
Drawwidth=1
Line(20,20)-(50,20)
Drawwidth=5
Line(20,40)-(50,40)
Drawwidth=8
Line(20,60)-(50,60)
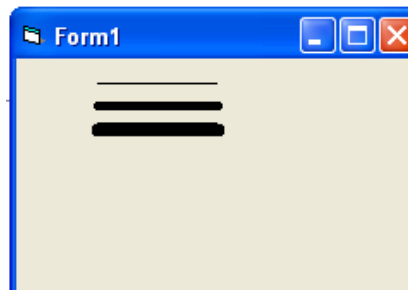
2- *DrawStyle*: the draw style property specifies whether the lines created with graphics methods are solid or have a broken pattern control. There are seven different draw style values (from 0 to 6). For Example:

Private Sub Form_Activate()
Scale (0, 0)-(100, 100)
DrawWidth = 1
Y = 10
For I = 0 To 6
DrawStyle = I
Y = Y + 10
Line (20, Y) - (70, Y)
Next I
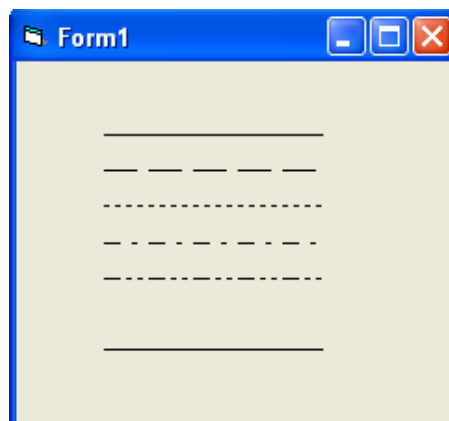
3- *FillStyle:* As long as you don't change the setting of the fill style Property, the box appears empty.(The box does get filled with default *FillStyle* and Settings, But *FillStyle* default to 1-Tranpartenet). You can change the *FillStyle* property to any the settings listed in the following table:

Setting	Description
0	**Solid. Fills in box with the color set for the *FillColor* Property**
1	**Transparent (the default). Graphical object appears empty, no matter what color is used**
2	**Horizontal lines**
3	**Vertical lines**
4	**Upward diagonal lines**
5	**Downward diagonal lines**
6	**Crosshatch**
7	**Diagonal Crosshatch**

For Example:

```
Private Sub Form_Activate()
Scale (0, 0)-(100, 100)
For i = 0 To 7
Y = Y + 10
FillStyle = i
FillColor = QBColor(i)
Line (Y, Y)-(Y + 10, Y + 10),  , B
Circle (Y + 15, 10), 5
Next i
```
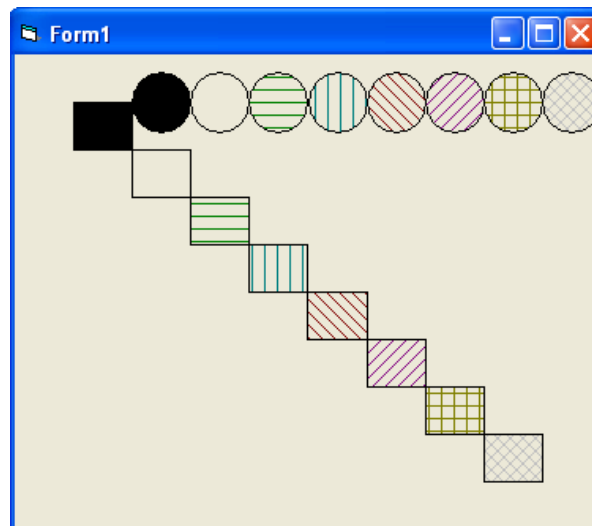

Example 7-1: Write a code program to draw the figures below.

         1- Solution:


```
Private Sub Command1_Click()
' Assign a scale
Scale (0, 0)-(100, 100)
DrawWidth = 3

'Draw 2 rectangles
Line (20, 20)-(60, 50),  , B
Line (50, 40)-(90, 70),  , B

' Draw 4 connecting lines (the last is dotted)
Line (20, 20)-(50, 40)
Line (60, 20)-(90, 40)
Line (20, 50)-(50, 70)
DrawStyle = 2
Line (60, 50)-(90, 70)
' DrawStyle =1 means solid line
' DrawStyle =2 means dotted line
End Sub
```

2-
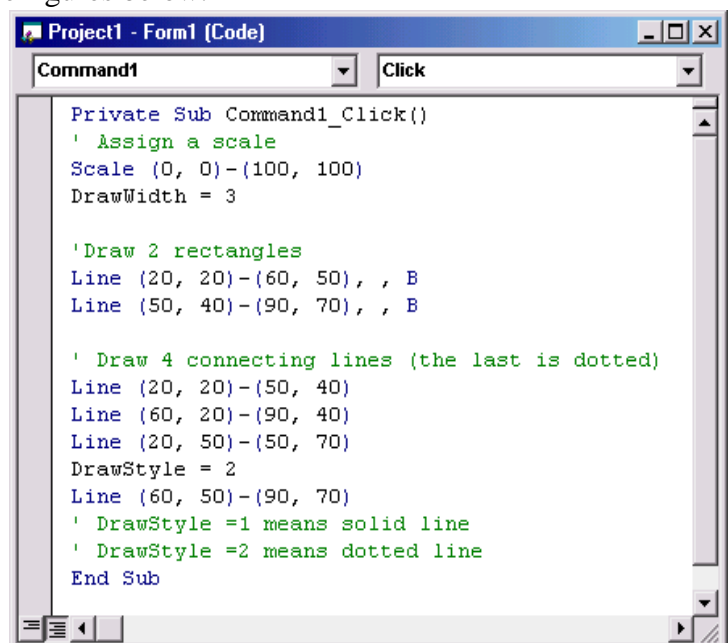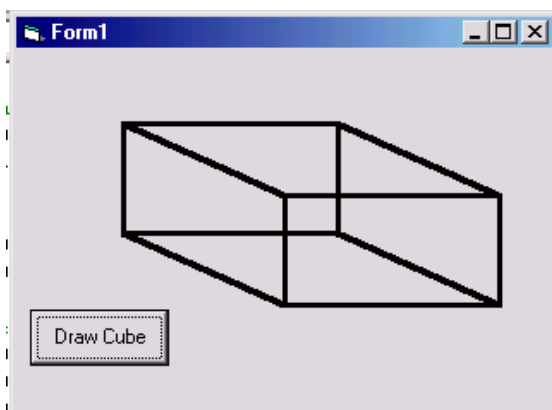
```vb
Private Sub Command1_Click()

'Assign scale
Scale (0, 0)-(100, 100)
DrawWidth = 3

' Draw 9 squares each 10 unit smaller than the other
' note that the sum of the coordinates of the start
' of the each square and the coordinates of the end
' of the same square is equal to 100

For i = 10 To 50 Step 5
Line (i, i)-(100 - i, 100 - i), , B
Next i

' note that the last square is a point
' You can also draw the squares using 9 different commands

End Sub
```

3-
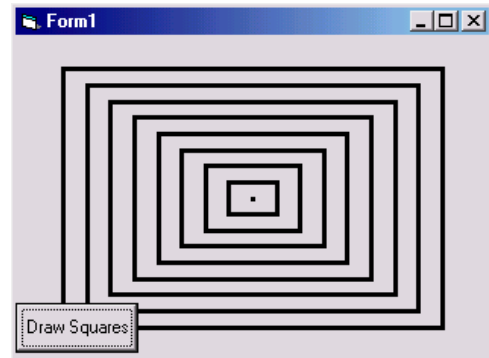
```vb
Private Sub Command1_Click()
' Assign a scale
Scale (0, 0)-(10, 10)

' to draw the face (circle) filled with yellow
FillColor = vbYellow
FillStyle = 0
Circle (5, 5), 3, vbBlack

'To draw 2 eyes filled with black color
FillColor = vbBlack
Circle (4, 4), 0.25, vbBlack
Circle (6, 4), 0.25, vbBlack

' To draw a smile mouth
DrawWidth = 3
pi = 3.14159
Circle (5, 5), 1.5, vbRed, 225 * pi / 180, 315 * pi / 180

End Sub
```

3-
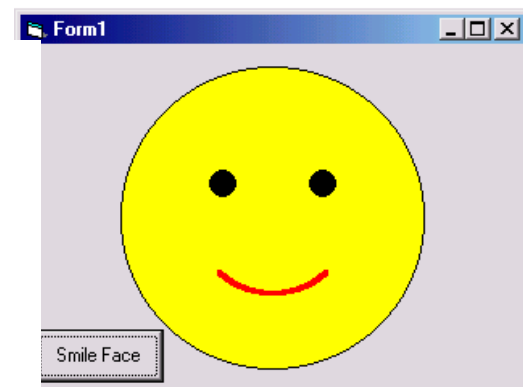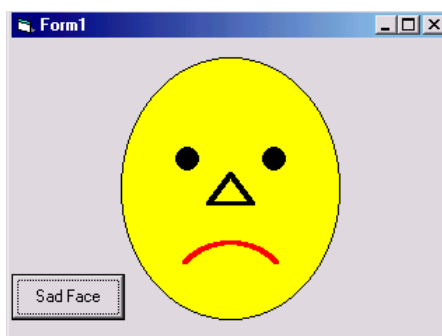
```vb
Private Sub Command1_Click()

' Assign a scale
Scale (0, 0)-(10, 10)

' To draw big face (circle)filled with the color yellow
FillStyle = 0
FillColor = vbYellow
Circle (5, 5), 3, vbBlack, , , 1.2

' To draw two eyes filled with black
FillColor = vbBlack
Circle (4, 4), 0.25, vbBlack
Circle (6, 4), 0.25, vbBlack

' To draw smile mouth
Form1.DrawWidth = 3
pi = 3.14159
Circle (5, 9), 1.5, vbRed, 45 * pi / 180, 135 * pi / 180

' To draw a nose
Line (5, 4.5)-(5.5, 5.5)
Line (5, 4.5)-(4.5, 5.5)
Line (4.5, 5.5)-(5.5, 5.5)
End Sub
```

72

Example7-2: The following statements represent of visual Basic program that are used to generate the graph. Draw the figure and write all the necessary coordinates position into the graph.

1-Private Sub Command1_Click()
Scale (0, 0)-(100, 100)
For i = 0 To 2 Step 2
Line (15 * i, 5)-(15 * (i + 1), 20), , B
Line (5 + 15 * i, 10)-(10 + 15 * i, 15), , BF
Next
Line (15, 10)-(30, 15), , B
End Sub
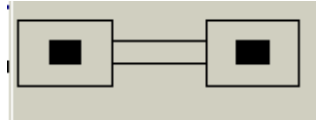
2-
Scale (0, 0)-(100, 100)
Line (20, 45)-(80, 95), , B
Line (20, 45)-(50, 5)
Line (50, 5)-(80, 45)
Circle (50, 70), 10, , , , 1.5
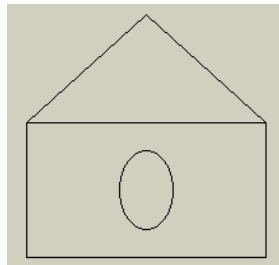
3-Private Sub Command1_Click()
Scale (-10, 10)-(10, -10)
Circle (0, 0), 8, , , , 0.5
Circle (0, 0), 2
Circle (-6, 0), 1
Circle (6, 0), 1
Line (10, 0)-(-10, 0)
Line (0, 10)-(0, -10)
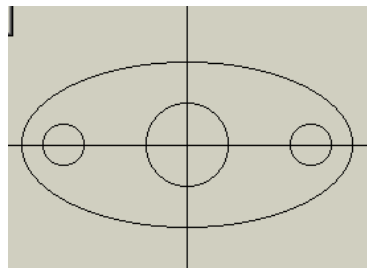
4-
Scale (0, 0)-(40, 40)
Line (20, 15)-(30, 35), , B
Circle (25, 15), 5, , 0, 3.14
FillStyle = 0
Circle (25, 12), 1, , , , 1.5
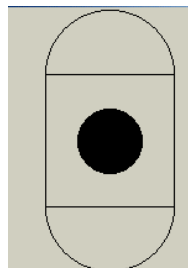
5-
Scale (0, 0)-(120, 120)
Line (30, 30)-(70, 90), , B
FillStyle = 0
Circle (50, 30), 20, , 0, 3.14
Circle (50, 90), 20, , 3.14, 2 * 3.14
Circle (50, 60), 10

73
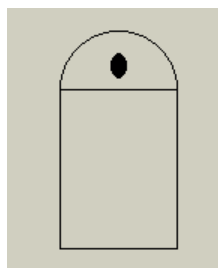
6-
Scale (-10, 10)-(10, -10)
For i = 0 To 6 Step 2
DrawStyle = 0
Line (0, 0)-(i, i), , B
Line (0, 0)-(-i, -i), , B
Next
For i = 0 To 6 Step 2
DrawStyle = 2
Line (0, 0)-(i, -i), , B
Line (0, 0)-(-i, i), , B
Next
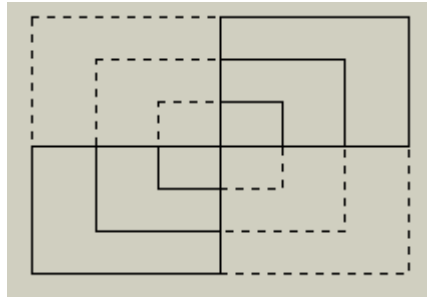
7.4 Graphics Controls: The Visual Basic provides three controls designed to create graphical effects in an application:
- The line Control
- The Shape Control
- The image control

These controls don't have event procedure. We first discuss drawing lines with the **Line** control. Unlike method line, which must be used at run-time, the line control can be used at design time. Lines can also be drawn at run-time with the **Line** control.

A line's color is specified using **Line** control's *BorderColor* property and a line's style is specified by setting the **Line** control's *BorderStyle* property. **Line** control line width(or thickness) is specified by setting the *BorderWidth.* Line length and position are specified using properties *X1,Y1, X2,* and *Y2*. *X1* and *Y1* specify the starting coordinate. *X2* and *Y2* specify the ending coordinate.

For Example
- Line1.BorderStyle=1
- Line1.BorderColor=Vbwhite
- Line1.X1=10
- Line1.Y1=15

The **Shape** control can be used to draw **rectangles, ellipses, rounded rectangles, squares, circles and rounded squares.** The **Shape** property specifies which **Shape** is drawn.

Value	Description
0	Rectangle
1	Square
2	Oval (i.e., an ellipse)
3	Circle
4	Rounded Rectangle
5	Rounded square

Shape control property *FillStyle* specifies how the shape is to be filled. The *BorederStyle* property specifies the style using the values from 0 to 6. *BackColor* , *FillColor* and *BorderColor* specify coloring. Note that *FillColor* and *BackColor* are ignored when either *FillStyle* or *BackStyle* is (Tranparent). Property *BorderWidth* changes the width of lines.

74

8- Menus

Windows applications provide groups of related commands in menus. These commands depend on the application, but some- such as Open and Save – are frequently found in applications. Visual Basic provides an easy way to create menus with the modal Menu Editor dialog. The dialog is displayed when Menu Editor is selected from the tools menu.
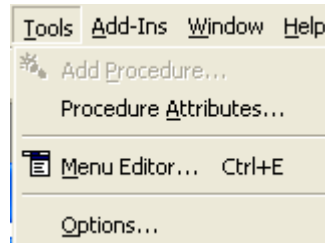
The menu editor command is grayed unless the form is visible. The menu dialog can also be displayed by right-clicking the form and selecting menu editor.
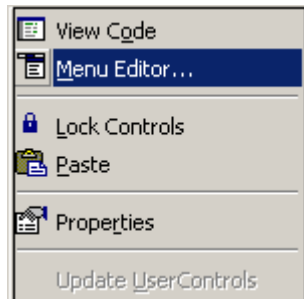
The menu editor dialog, shown in figure below, contains the textboxes **Caption** and **Name**. The value entered in the **Caption Textbox** is the menu name the user sees. The value entered in the **Name Textbox** is the variable name the programmer uses. Every menu must have a **Caption** and a **Name**.
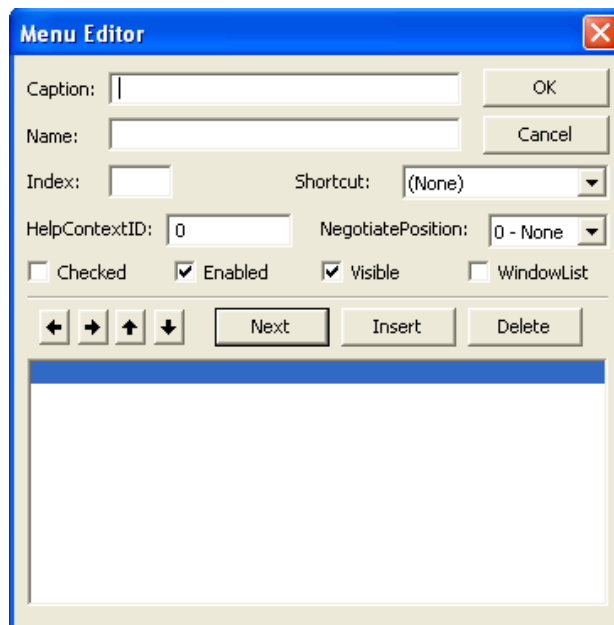
Menus are like other controls in that they have properties and events. The menu editor is a way of setting select properties for a menu. Once a menu is created, its properties can be viewed in the properties window and its events in the code window. The programmer can create menu control

arrays. The **Index Textbox** specifies the menu's index in the control array. Menus that are not top-level menus can have shortcut keys (combinations of Ctrl, Shift and letter keys). Shortcut keys are specified using the **Shortcut ComboBox**. All shortcut keys listed in the shortcut ComboBox are predefined by visual basic. Programmers may not define their own.

The menu editor dialog also several provides several **CheckBoxes** for controlling the appearance of menus. The checked checkbox is unchecked (False) by default and allows the programmer the option of creating a checked menu item (a menu item that acts as a toggle and displays a check mark when selected). We discuss the menu editor buttons momentarily.

In our next example, we create two top-level menus, **Draw** and **Exit**, using the menu editor. Building the menus does not require us to write any code. Creating and designing menus is an excellent example of visual programming (the ability to create windows GUIs writing only minimal code).

The following figure displays a GUI containing two menus, **Draw** and **Exit** – each of which is created using menu editor. Note that the separator bars are created by placing a hyphen, -, in the caption textbox.
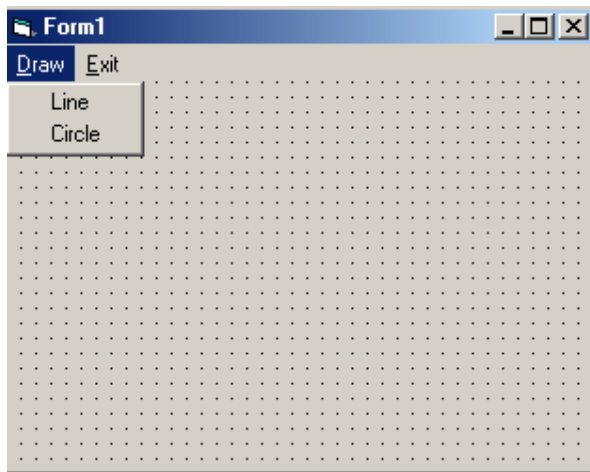
The menu editor dialog shown in figure below, corresponding to the program of example. The next button allows the next menu or menu item to be edited. Insert allows a menu or menu item to be inserted between two existing menus or menu items. Delete remove the selected menu or menu item. The up and down arrow buttons perform insertion in the same manner as insert.

Notice that the top-level menus appear in the left column. Any submenus or menu items are indented (using the second arrow button). Submenu menu items are indented beneath their respective menus (using the second arrow button).

```
Private Sub DrawLine _click ()
Line (25,25)-(75,50),vbYellow
End Sub
```

```
Private Sub Drawcircle _click ()
Circle (25,15), 5,vbGreen
End Sub
```

```
Private Sub MenuClose _click ()
END
End Sub
```


76
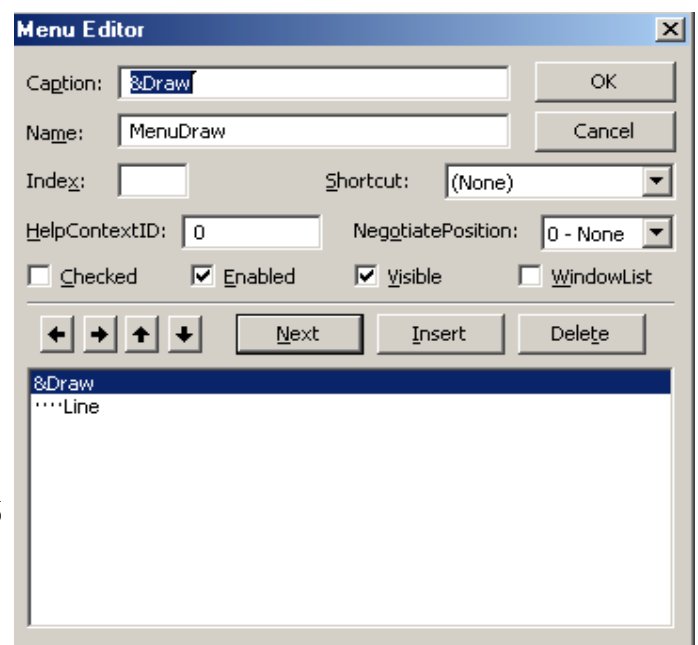
9- <u>Sub Procedure and Function Procedure</u>

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 reputation 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
Code	Sub Name (arguments) Statements End Sub	Function Name (arguments) Statements End Function
Remark	• Need call statement • Return values by arguments • Return many values (arguments) • Used for Input/output, condition treatment • Could be used with out arguments.	• Used in arithmetic statement • Return value by its name • Return one value • Used for arithmetic's or conversion of variable type.
Call Statement	**Call Name(value1,value2,,,,)**	**Z=name(value1)**
Exit statement	**Exit Sub**	**Exit Function**

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.
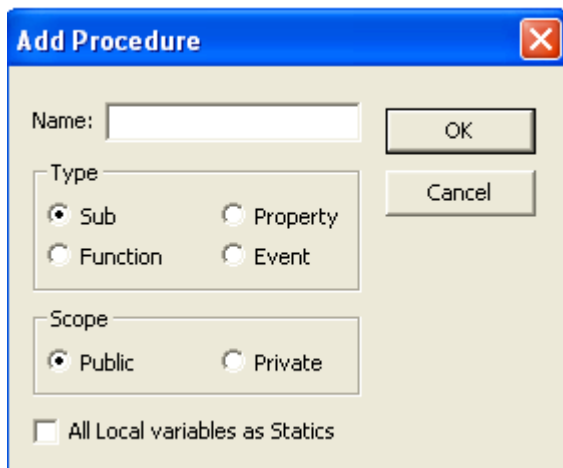

Figure (9-1): add procedure dialog

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.
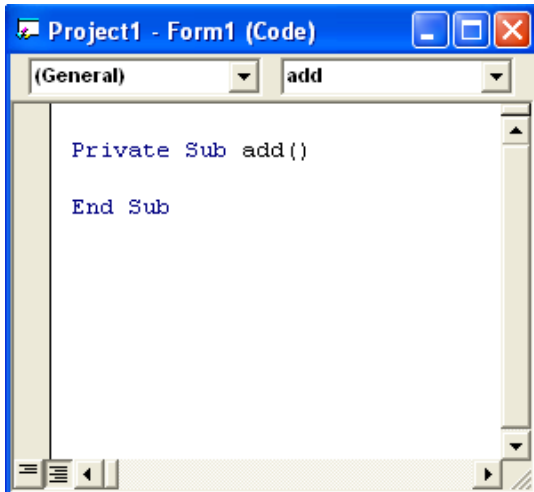

Figure (9-2): A sub procedure created with the add Procedure dialog.

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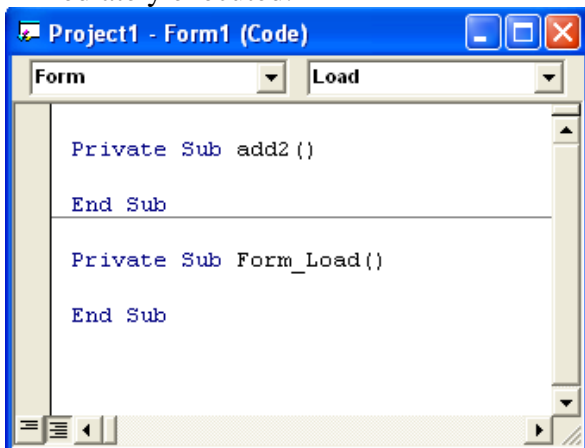
Figure (9-3): A Sub procedure created by typing directly into the code window.

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 be 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
```

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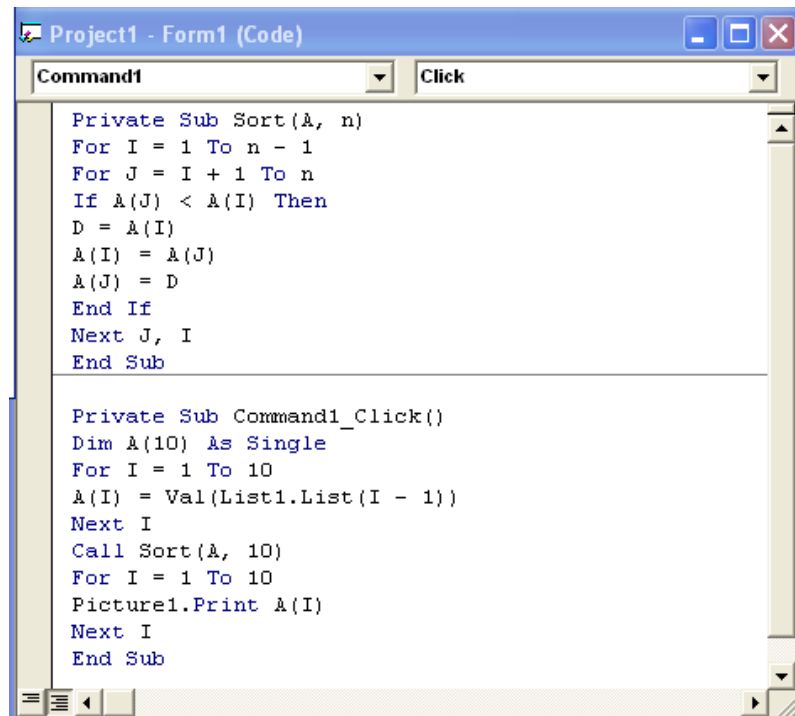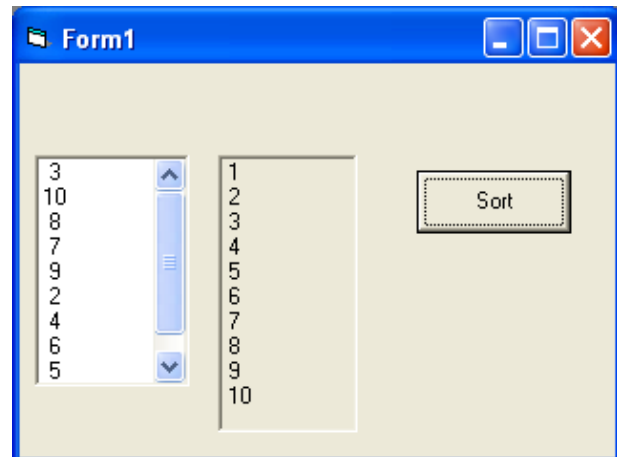
9.2 Function Procedures: function procedures and sub procedures share the same characteristics, with one important difference- function procedures return a value (i.g., give a value back) to the caller, whereas sub procedures do not.

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.
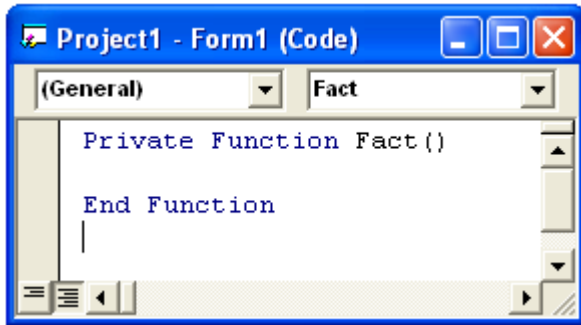

Figure (9-4): Function procedure created with add procedure dialog

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 =MinNum1, Num2, Num3)

Text4.Text = Str(Result)

End Sub
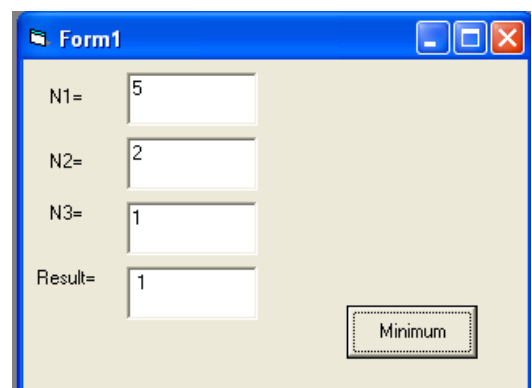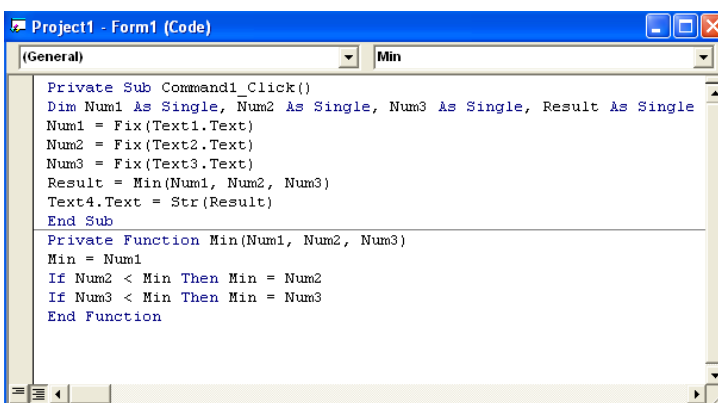
Private Function Min(Num1, Num2, Num3)

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
```