

Lesson1: Introduction to Visual Basic 6

Before we begin Visual Basic 6 programming, let us understand some basic concepts of programming. According to Webopedia, a computer program is an organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. Without programs, computers are useless. Therefore, programming means designing or creating a set of instructions to ask the computer to carry out certain jobs which normally are very much faster than human beings can do.

Most people think that computer CPU is a very intelligent thing, which in actual fact it is a dumb and inanimate object that can do nothing without human assistant. The microchips of a CPU can only understand two distinct electrical states, namely, the on and off states, or 0 and 1 codes in the binary system. So, the CPU only understands combinations of 0 and 1 code, a language which we called machine language. Machine language is extremely difficult to learn and it is not for us laymen to master it easily. Fortunately, we have many smart programmers who wrote interpreters and compilers that can translate human language-like programs such as BASIC into machine language so that the computer can carry out the instructions entered by the users. Machine language is known as the primitive language while Interpreters and compilers like Visual Basic are called high-level language. Some of the high level programming languages beside Visual Basic are Fortran, Cobol, Java, C, C++, Turbo Pascal, and more . Among the aforementioned programming languages, Visual Basic is the most popular. Not only it is easily to learn because of its English-like syntaxes, it can also be incorporated into all the Microsoft office applications such as Microsoft words, Microsoft Excel, Microsoft PowerPoint and more. Visual Basic for applications is known as VBA.

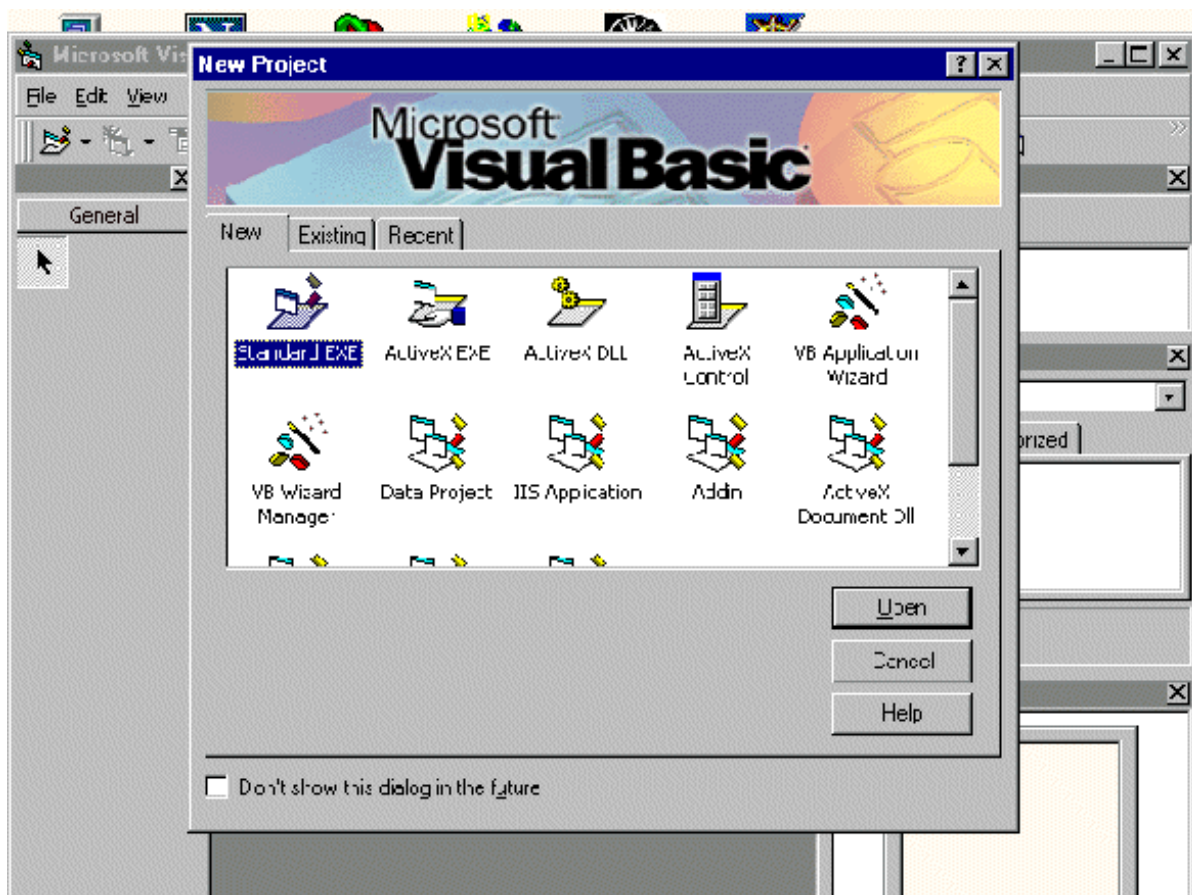
What programs can you create with Visual Basic 6?

With VB 6, you can create any program depending on your objective. For example, you can create educational programs to teach science , mathematics, language, history , geography and so on. You can also create financial and accounting programs to make you a more efficient accountant or financial controller. For those of you who like games, you can create those programs as well. Indeed, there is no limit to what program you can create!

VISUAL BASIC

The Visual Basic 6 Integrated Development Environment

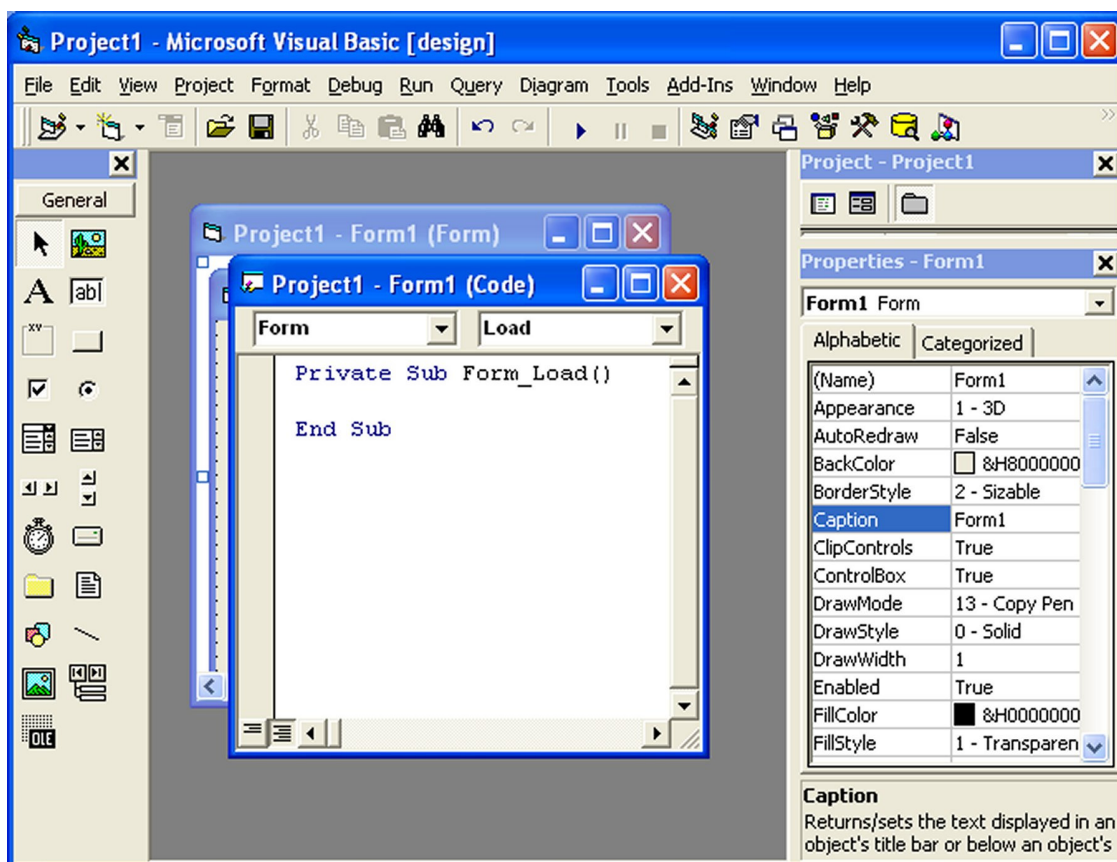
On start up, Visual Basic 6.0 will display the following dialog box as shown in **Figure 1.1**. You can choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications that we could create, however, we shall concentrate on creating Standard EXE programs (EXE means executable program). Now, click on the Standard EXE icon to go into the actual Visual Basic 6 programming environment.



VISUAL BASIC

Building Visual Basic Applications

First of all, you have to launch Microsoft Visual Basic 6. Normally, a default form with the name Form1 will be available for you to start your new project. Now, double click on Form1, the source code window for Form1 as shown in figure 2.1 will appear. The top of the source code window consists of a list of objects and their associated events or procedures. In figure 2.1, the object displayed is Form and the associated procedure is Load.



When you click on the object box, the drop-down list will display a list of objects you have inserted into your form as shown in figure 2.2. Here, you can see a form with the name Form1, a command button with the name Command1, a Label with the name Label1 and a Picture Box with the name Picture1. Similarly, when you click on the procedure box, a list of procedures associated with the object will be

VISUAL BASIC

displayed as shown in figure 2.3. Some of the procedures associated with the object Form1 are **Activate**, **Click**, **DblClick** (which means Double-Click) , **DragDrop**, **keyPress** and more. Each object has its own set of procedures. You can always select an object and write codes for any of its procedure in order to perform certain tasks.

Figure 2.2: List of Objects

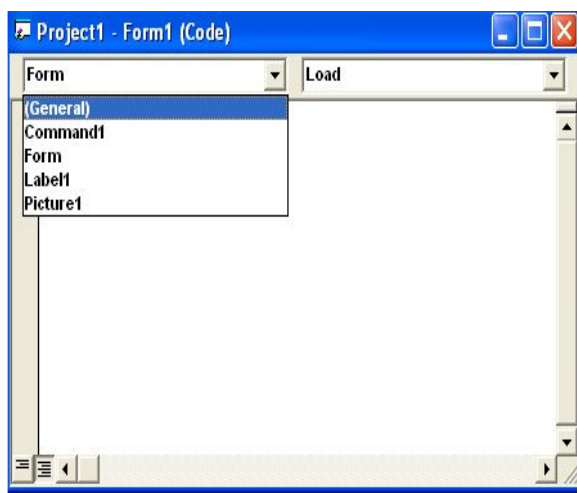
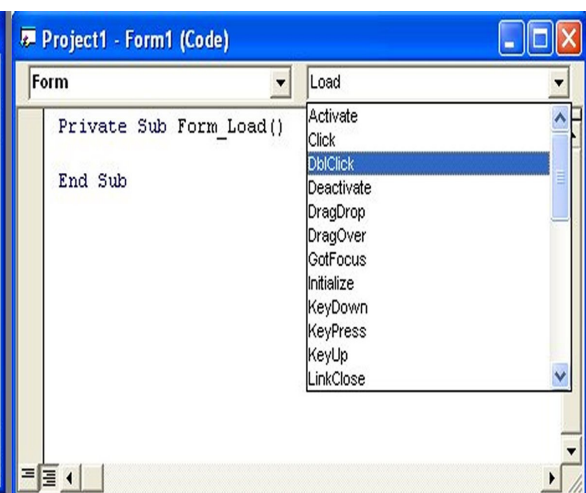


Figure 2.3: List of Procedures



You do not have to worry about the beginning and the end statements (i.e. **Private Sub Form_Load.....End Sub**.); Just key in the lines in between the above two statements exactly as are shown here. When you press **F5** to run the program, you will be surprise that nothing shown up .In order to display the output of the program, you have to add the **Form1.show** statement like in Example 2.1.1 or you can just use **Form_Activate ()** event procedure as shown in example 2.1.2. The command **Print** does not mean printing using a printer but it means displaying the output on the computer screen. Now, press F5 or click on the run button to run the program and you will get the output as shown in figure 2.4.

VISUAL BASIC

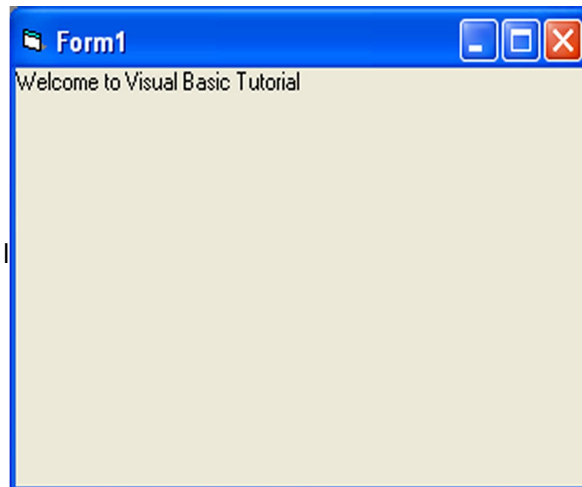
You can also perform arithmetic calculations as shown in example 2.1.2. VB uses * to denote the multiplication operator and / to denote the division operator. The output is shown in figure 2.3, where the results are arranged vertically.

VISUAL BASIC

Figure 2.4 : The output of example 2.1.1

Example 2.1.1

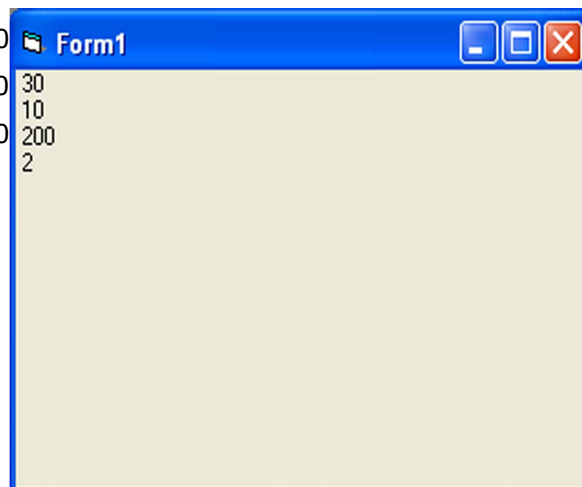
```
Private Sub Form_Load ( )  
    Form1.show  
    Print "Welcome to Visual  
    Basic tutorial"  
End Sub
```



Example 2.1.2

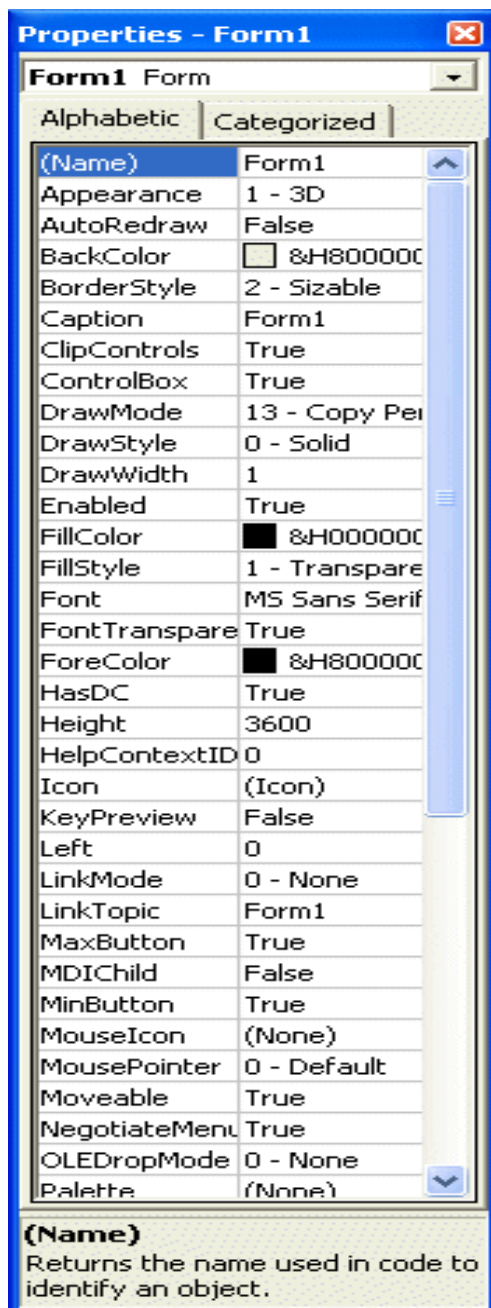
```
Private Sub Form_Activate ( )  
  
    Print 20 + 10  
    Print 20 - 10  
    Print 20 * 10  
    Print 20 / 10  
  
End Sub
```

Figure 2.5: The output of example 2.1.2



Lesson 3-Working With Controls

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 or at runtime.



VISUAL BASIC

Figure 3.1 on the right is a typical properties window for a form. You can rename the form caption to any name that you like best. In the properties window, the item appears at the top part is the object currently selected (in Figure 3.1, the object selected is Form1). At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options

VISUAL BASIC

available.

For example, in order to change the caption, just highlight Form1 under the name Caption and change it to other names. You may also try to alter the appearance of the form by setting it to 3D or flat. Other things you can do are to change its foreground and background color, change the font type and font size, enable or disable minimize and maximize buttons and etc.

You can also change the properties at runtime to give special effects such as change of color, shape, animation effect and so on. For example the following code will change the form color to red every time the form is loaded. VB uses hexadecimal system to represent the color. You can check the color codes in the properties windows which are showed up under ForeColor and BackColor .

```
Private Sub Form_Load()
```

```
Form1.Show  
Form1.BackColor = &H000000FF&
```

```
End Sub
```

Another example is to change the control Shape to a particular shape at runtime by writing the following code. This code will change the shape to a circle at runtime. Later you will learn how to change the shapes randomly by using the **RND** function.

```
Private Sub Form_Load()
```

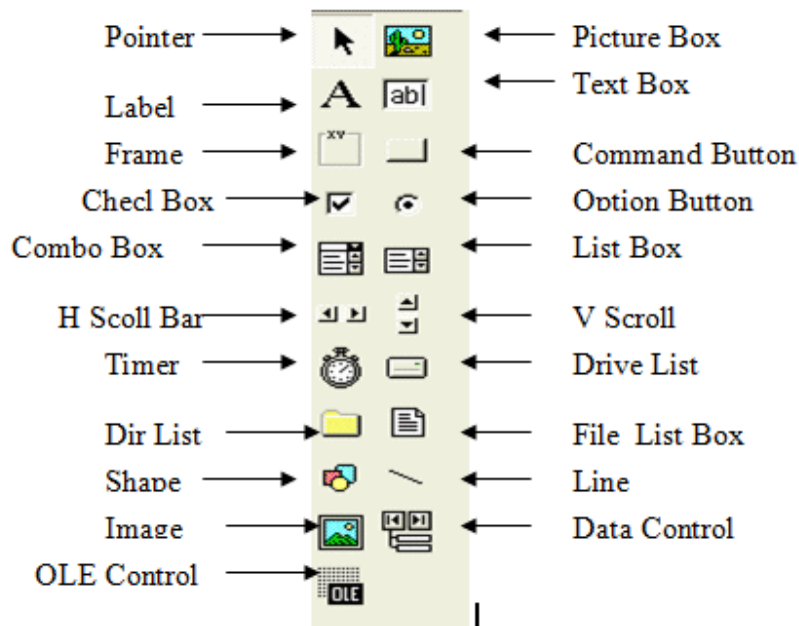
```
Shape1.Shape = 3
```

```
End Sub
```

3.2 Handling some of the common controls in VB6

When we launched Visual Basic 6 standard project, the default controls are displayed in the Toolbox as shown in Figure 3.2 below. Later , you can add more controls as you progress to more advanced programming.

VISUAL BASIC



3.2.1 The Text Box

The text box is the standard control for accepting input from the user as well as to display the output. It can handle string (text) and numeric data but not images or pictures. String in a text box can be converted to a numeric data by using the function `Val(text)`. The following example illustrates a simple program that processes the input from the user.

Example 3.1

In this program, two text boxes are inserted into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. Besides, a command button is also programmed to calculate the sum of the two numbers using the plus operator. The program use creates a variable sum to accept the summation of values from text box 1 and text box

VISUAL BASIC

2.The procedure to calculate and to display the output on the label is shown below. The output is shown in Figure 3.3

```
Private Sub Command1_Click()
```

```
'To add the values in text box 1 and text box 2
```

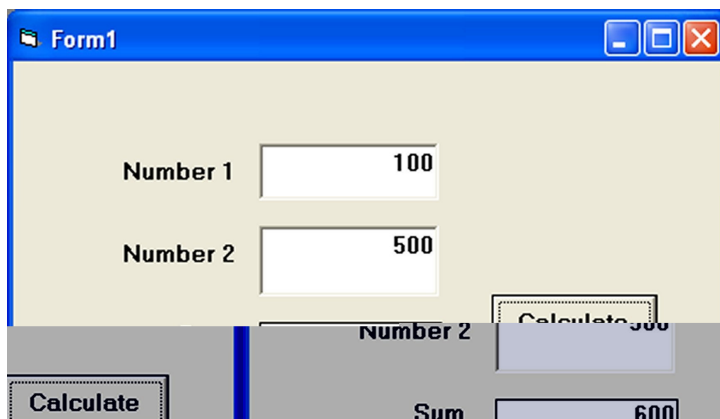
```
Sum = Val(Text1.Text) + Val(Text2.Text)
```

```
'To display the answer on label 1
```

```
Label1.Caption = Sum
```

```
End Sub
```

Figure 3.3



3.2.2 The Label

The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to

display outputs. One of its most important properties is **Caption**. Using the syntax **label.Caption**, it can display text and numeric data . You can change its caption in the properties window and also at runtime. Please refer to Example 3.1 and Figure 3.1 for the usage of label.

3.2.3 The Command Button

The command button is one of the most important controls as it is used to execute commands. It displays an illusion that the button is pressed when the user click on it. The most common event associated with the command button is the Click event, and the syntax for the procedure is

```
Private Sub Command1_Click ()
```

```
Statements
```

```
End Sub
```

3.2.4 The Picture Box

The Picture Box is one of the controls that is used to handle graphics. You can load a picture at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. You can also load the picture at runtime using the **LoadPicture** method. For example, the statement will load the picture grape.gif into the picture box.

```
Picture1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")
```

You will learn more about the picture box in future lessons. The image in the picture box is not resizable.

3.2.5 The Image Box

The Image Box is another control that handles images and pictures. It functions almost identically to the picture box. However, there is one major difference, the image in an Image Box is stretchable, which means it can be resized. This feature is not available in the Picture Box. Similar to the Picture Box, it can also use the LoadPicture method to load the picture. For example, the statement loads the picture grape.gif into the image box.

```
Image1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")
```

3.2.6 The List Box

The function of the List Box is to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the **AddItem method**. For example, if you wish to add a number of items to list box 1, you can key in the following statements

Example 3.2

```
Private Sub Form_Load ( )
```

```
List1.AddItem "Lesson1"
```

```
List1.AddItem "Lesson2"
```

```
List1.AddItem "Lesson3"
```

```
List1.AddItem "Lesson4"
```

```
End Sub
```

The items in the list box can be identified by the **ListIndex** property, the value of the ListIndex for the first item is 0, the second item has a ListIndex 1, and the second item has a ListIndex 2 and so on

3.2.7 The Combo Box

VISUAL BASIC

The function of the Combo Box is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combo box to see the items which are presented in a drop-down list. In order to add items to the list, you can also use the **AddItem method**. For example, if you wish to add a number of items to Combo box 1, you can key in the following statements

Example 3.3

```
Private Sub Form_Load ( )
```

```
    Combo1.AddItem "Item1"
```

```
    Combo1.AddItem "Item2"
```

```
    Combo1.AddItem "Item3"
```

```
    Combo1.AddItem "Item4"
```

```
End Sub
```

3.2.8 The Check Box

The Check Box control lets the user select or unselects an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements `Check1.Value=1` to mark the Check Box and `Check1.Value=0` to unmark the Check Box, as well as use them to initiate certain actions. For example, the program will change the background color of the form to red when the check box is unchecked and it will change to blue when the check box is checked. You will learn about the conditional statement `If....Then....Elesif` in later lesson. `VbRed` and `vbBlue` are color constants and `BackColor` is the background color property of the form.

Example 3.4

VISUAL BASIC

```
Private Sub Command1_Click()

If Check1.Value = 1 And Check2.Value = 0 Then
MsgBox "Apple is selected"
ElseIf Check2.Value = 1 And Check1.Value = 0 Then
MsgBox "Orange is selected"
Else
MsgBox "All are selected"
End If

End Sub
```

3.2.9 The Option Box

The Option Box control also lets the user select one of the choices. However, two or more Option Boxes must work together because as one of the Option Boxes is selected, the other Option Boxes will be unselected. In fact, only one Option Box can be selected at one time. When an option box is selected, its value is set to "True" and when it is unselected; its value is set to "False". In the following example, the shape control is placed in the form together with six Option Boxes. When the user clicks on different option boxes, different shapes will appear. The values of the shape control are 0, 1, and 2,3,4,5 which will make it appear as a rectangle, a square, an oval shape, a rounded rectangle and a rounded square respectively.

Example 3.5

```
Private Sub Option1_Click ( )

Shape1.Shape = 0

End Sub
```

```
Private Sub Option2_Click()
```

VISUAL BASIC

```
Shape1.Shape = 1
```

```
End Sub
```

```
Private Sub Option3_Click()
```

```
Shape1.Shape = 2
```

```
End Sub
```

```
Private Sub Option4_Click()
```

```
Shape1.Shape = 3
```

```
End Sub
```

```
Private Sub Option5_Click()
```

```
Shape1.Shape = 4
```

```
End Sub
```

```
Private Sub Option6_Click()
```

```
Shape1.Shape = 5
```

```
End Sub
```

3.2.10 The Drive List Box

The Drive ListBox is for displaying a list of drives available in your computer. When you place this control into the form and run the program, you will be able to select different drives from your computer as shown in Figure 3.4

3.2.11 The Directory List Box

The Directory List Box is for displaying the list of directories or folders in a selected drive. When you place this control into the form and run the program, you will be able to select different directories from a selected drive in your computer as shown in Figure 3.5

3.2.12 The File List Box

The File List Box is for displaying the list of files in a selected directory or folder. When you place this control into the form and run the program, you will be able to shown the list of files in a selected directory as shown in Figure 3.5

Lesson 4 : Writing the Code

In lesson 2, you have learned how to enter the program code and run the sample VB programs but without much understanding about the logics of VB programming. Now, let's get down to learning some basic rules about writing the VB program code.

Each control or object in VB can usually run many kinds of events or procedures; these events are listed in the dropdown list in the code window that is displayed when you double-click on an object and click on the procedures' box(refer to Figure 2.3). Among the events are loading a form, clicking of a command button, pressing a key on the keyboard or dragging an object and more. For each event, you need to write an event procedure so that it can perform an action or a series of actions

To start writing an event procedure, you need to double-click an object. For example, if you want to write an event procedure when a user clicks a command button, you double-click on the command button and an event procedure will appear as shown in Figure 2.1. It takes the following format:

Private Sub Command1_Click

(Key in your program code here)

End Sub

You then need to key-in the procedure in the space between Private Sub Command1_Click..... End Sub. Sub actually stands for sub procedure that made up a part of all the procedures in a program. The program code is made up of a number of statements that set certain

VISUAL BASIC

properties or trigger some actions. The syntax of Visual Basic's program code is almost like the normal English language though not exactly the same, so it is very easy to learn.

The syntax to set the property of an object or to pass certain value to it is :

Object.Property

where Object and Property is separated by a period (or dot). For example, the statement **Form1.Show** means to show the form with the name Form1, **Label1.Visible=true** means label1 is set to be visible, **Text1.text="VB"** is to assign the text VB to the text box with the name Text1, **Text2.text=100** is to pass a value of 100 to the text box with the name text2, **Timer1.Enabled=False** is to disable the timer with the name Timer1 and so on. Let's examine a few examples below:

Example 4.1

```
Private Sub Command1_click
Label1.Visible=false
Label2.Visible=True
Text1.Text="You are correct!"
End sub
```

Example 4.2

```
Private Sub Command1_click
Label1.Caption=" Welcome"
Image1.visible=true
End sub
```

Example 4.3

```
Private Sub Command1_click

Pictuire1.Show=true
```

VISUAL BASIC

Timer1.Enabled=True

Lable1.Caption="Start Counting

End sub

VISUAL BASIC

In Example 4.1, clicking on the command button will make label1 become invisible and label2 become visible; and the text "You are correct" will appear in TextBox1. In example 4.2, clicking on the command button will make the caption label1 change to "Welcome" and Image1 will become visible. In example 4.3, clicking on the command button will make Picture1 show up, timer starts running and the caption of label1 change to "Start Counting".

Syntaxes that do not involve setting of properties are also English-like, some of the commands are **Print, If...Then....Else....End If, For...Next, Select Case.....End Select, End** and **Exit Sub**. For example, **Print " Visual Basic"** is to display the text Visual Basic on screen and **End** is to end the program. Other commands will be explained in details in the coming lessons.

Program code that involve calculations is very easy to write, you need to write them almost like you do in mathematics. However, in order to write an event procedure that involves calculations, you need to know the basic arithmetic operators in VB as they are not exactly the same as the normal operators we use, except for **+** and **-**. For multiplication, we use *****, for division we use **/**, for raising a number x to the power of n, we use **x ^n** and for square root, we use **Sqr(x)**. VB offers many more advanced mathematical functions such as **Sin, Cos, Tan** and **Log**, they will be discussed in lesson 10. There are also two important functions that are related to arithmetic operations, i.e. the functions **Val** and **Str\$** where Val is to convert text entered into a textbox to numerical value and Str\$ is to display a numerical value in a textbox as a string (text). While the function Str\$ is as important as VB can display a numeric values as string implicitly, failure to use Val

VISUAL BASIC

will results in wrong calculation. Let's examine example 4.4 and example 4.5.

Example 4.4

```
Private Sub Form_Activate()
```

```
Text3.text=text1.text+text2.text
```

```
End Sub
```

Example 4.5

```
Private Sub Form_Activate()
```

```
Text3.text=val(text1.text)+val(text2.text)
```

```
End Sub
```

When you run the program in example 4.4 and enter 12 in textbox1 and 3 in textbox2 will give you a result of 123, which is wrong. It is because VB treat the numbers as string and so it just joins up the two strings. On the other hand, running exampled 4.5 will give you the correct result, i.e., 15.

Lesson 5: Managing Visual Basic Data

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and more everyday. Similarly in Visual Basic, we have to deal with all sorts of of data, some can be mathematically calculated while some are in the form of text or other forms. VB divides data into different types so that it is easier to manage when we need to write the code involving those data.

5.1 Visual Basic Data Types

Visual Basic classifies the information mentioned above into two major data types; they are the numeric data types and the non-numeric data types.

Table 5.1: Numeric Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767

VISUAL BASIC

Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

5.1.2 Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .They are summarized in Table 5.2

Table 5.2: Nonnumeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double

VISUAL BASIC

Variant(text)	Length+22 bytes	Same as variable-length string
---------------	-----------------	--------------------------------

.1.3 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use `num=1.3089#` for a Double type data. Some of the suffixes are displayed in Table 5.3.

Table 5.3

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

```
memberName="Turban," John."  
TelNumber="1800-900-888-777"  
LastDay=#31-Dec-00#  
ExpTime=#12:00 am#
```

Lesson 5: Managing Visual Basic Data

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and more everyday. Similarly in Visual Basic, we have to deal with all sorts of data, some can be mathematically calculated while some are in the form of text or other forms. VB divides data into different types so that it is easier to manage when we need to write the code involving those data.

5.1 Visual Basic Data Types

Visual Basic classifies the information mentioned above into two major data types, they are the numeric data types and the non-numeric data types.

5.1.1 Numeric Data Types

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and more.

Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others. In Visual Basic, numeric data are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures or data that does not need precision can use Integer or Long integer in the computation. Programs that

VISUAL BASIC

require high precision calculation need to use Single and Double decision data types, they are also called floating point numbers. For currency calculation, you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve a many decimal points, we can use the decimal data types. These data types summarized in Table 5.1

Table 5.1: Numeric Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/-

79,228,162,514,264,337,593,543,950,335
if no decimal is use
+/- 7.9228162514264337593543950335
(28 decimal places).

5.1.2 Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type. They are summarized in Table 5.2

Table 5.2: Nonnumeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

5.1.3 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need

to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data. Some of the suffixes are displayed in Table 5.3.

Table 5.3

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

```
memberName="Turban, John."  
TelNumber="1800-900-888-777"  
LastDay=#31-Dec-00#  
ExpTime=#12:00 am#
```

5.2 Managing Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In term of VB, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Visual Basic, you have to follow a set of rules.

5.2.1 Variable Names

The following are the rules when naming the variables in Visual Basic

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

VISUAL BASIC

Examples of valid and invalid variable names are displayed in Table 5.4

Table 5.4

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy
Long_Name_Can_beUSE	He&HisFather acceptable
	*& is not

5.2.2 Declaring Variables

In Visual Basic, one needs to declare the variables before using them by assigning names and data types. They are normally declared in the general section of the codes' windows using the **Dim** statement. The format is as follows:

Dim Variable Name As Data Type

Example 5.1

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim doDate As Date
```

You may also combine them in one line , separating each variable with a comma, as follows:

```
Dim password As String, yourName As String, firstnum As Integer,.....
```

If data type is not specified, VB will automatically declare the variable as a Variant. For string declaration, there are two possible formats, one for the

variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as example 5.1 above. However, for the fixed-length string, you have to use the format as shown below:

*Dim VariableName as String * n*, where n defines the number of characters the string can hold.

Example 5.2:

```
Dim yourName as String * 10
```

yourName can holds no more than 10 Characters.

5.3 Constants

Constants are different from variables in the sense that their values do not change during the running of the program.

5.3.1 Declaring a Constant

The format to declare a constant is

Const Constant Name As Data Type = Value

Example 5.3

```
Const Pi As Single=3.142
```

```
Const Temp As Single=37
```

```
Const Score As Single=100
```

Lesson 6: Working with Variables

6.1 Assigning Values to Variables

VISUAL BASIC

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is

Variable=Expression

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and more. The following are some examples:

```
firstNumber=100
secondNumber=firstNumber-
99
userName="John      Lyan"
userpass.Text  = password
Label1.Visible = True
Command1.Visible = false
Label4.Caption =
textbox1.Text
ThirdNumber    =
Val(usernum1.Text)
total = firstNumber +
secondNumber+ThirdNumber
```

6.2 Operators in Visual Basic

To compute inputs from users and to generate results, we need to use various mathematical operators. In Visual Basic, except for + and -, the symbols for the operators are different from normal mathematical operators, as shown in Table 6.1.

Table 6.1: Arithmetic Operators

Operator	Mathematical function	Example
^	Exponential	2^4=16
*	Multiplication	4*3=12, (5*6))2=60
/	Division	12/4=3
Mod	Modulus(return the remainder from an integer division)	15 Mod 4=3 255 mod 10=5
\	Integer Division(discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

Example 6.1

```
Dim firstName
```

Example 6.2

```
Dim number1, number2, number3 as Integer
```


VISUAL BASIC

String	Dim total, average as variant
Dim secondName As	Private sub Form_Click
String	number1=val(Text1.Text)
Dim yourName As	number2=val(Text2.Text)
String	number3= val(Text3.Text)
	Total=number1+number2+number3
Private	Average=Total/5
Sub	Label1.Caption=Total
Command1_Click()	= Label2.Caption=Average
firstName	End Sub
Text1.Text	
secondName	
Text2.Text	In the example above, three variables are declared as
yourName	= integer and two variables are declared as variant. Variant
secondName + " " +	means the variable can hold any data type. The program
firstName	computes the total and average of the three numbers
Label1.Caption	that are entered into three text boxes.
= yourName	
End Sub	
In this example,	
three variables are	
declared as string.	
For variables	
firstName and	
secondName will	
receive their data	
from the user's input	
into textbox1 and	
textbox2, and the	
variable yourName	
will be assigned the	
data by combining	
the first two	
variables. Finally,	
yourName is	
displayed on Label1.	

Lesson 7 : Controlling Program Flow

7.1 Conditional Operators

To control the VB program flow, we can use various conditional operators. Basically, they resemble mathematical operators. Conditional operators are very powerful tools, they let the VB program compare data values and then decide what action to take, whether to execute a program or terminate the program and more. These operators are shown in Table 7.1.

7.2 Logical Operators

In addition to conditional operators, there are a few logical operators which offer added power to the VB programs. There are shown in Table 7.2.

Table 7.1: Conditional Operators		Table 7.2: Logical Operators	
Operator	Meaning	Operator	Meaning
=	Equal to	And	Both sides must be true
>	More than	or	One side or other must be true
<	Less Than		
>=	More than and equal	Xor	One side or other must be true but not both
<=	Less than and equal		
<>	Not Equal to	Not	Negates truth

* You can also compare strings with the above operators. However, there are certain rules to follow: Upper case letters are less than lowercase letters, "A"<"B"<"C"<"D".....<"Z" and number are less than letters.

<div><div><div>7.3</div><div>If.....Then.....Else Statements Operators</div></div><div><div>Using with</div><div><p>To effectively control the VB program flow, we shall use If...Then...Else statement together with the conditional operators and logical operators. The general format for the if...then...else statement is</p><p>If conditions Then VB expressions Else VB expressions End If</p><p>* any If..Then..Else statement must end with End If. Sometime it is not necessary to use Else.</p></div></div></div>	<div><div>Example:</div><div><div>Private Sub</div><div>OK_Click()</div><div>firstnum = Val(usernum1.Text)</div><div>secondnum = Val(usernum2.Text)</div><div>total = Val(sum.Text)</div><div>If total = firstnum + secondnum And Val(sum.Text) <> 0</div><div>Then</div><div>correct.Visible = True</div><div>wrong.Visible = False</div><div>Else</div><div>correct.Visible = False</div><div>wrong.Visible = True</div><div>End If</div><div>End Sub</div></div></div>
---	---

Lesson 8 : Select Case....End select Control Structure

In the previous lesson, we have learned how to control the program flow using the **If...ElseIf** control structure. In this chapter, you will learn another way to control the program flow, that is, the **Select Case** control structure. However, 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 make decision on one expression or dimension (for example the examination grade) 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 preferred when there exist many different conditions because using If...Then..ElseIf statements might become too messy.

The format of the Select Case control structure is show below:

Select Case expression

```
Case value1
    Block of one or more VB statements
Case value2
    Block of one or more VB Statements
Case value3
    .
    .
Case Else
    Block of one or more VB Statements
```

Example 8.1

```
Dim grade As String

Private Sub Compute_Click( )

grade=txtgrade.Text

Select Case grade

Case "A"
    result.Caption="High Distinction"

Case "A-"
    result.Caption="Distinction"
```

End Sub

Visual Basic allows a procedure to be repeated many times as long as the processor until a condition or a set of conditions is fulfilled. This is generally called looping . Looping is a very useful feature of Visual Basic because it makes repetitive works easier. There are two kinds of loops in Visual Basic, the Do...Loop and the For.....Next loop

The formats are

d) Do Block of one or more VB statements Loop Until condition

Sometime we need exit to exit a loop prematurely because of a certain condition is fulfilled. The syntax to use is known as Exit Do. You can examine Example 9.2 for its usage.

9.3 For....Next Loop

The `For....Next` format is:

```
For counter=startNumber to endNumber (Step increment)
    One or more VB statements
Next
```

Please refer to example 9.3a,9.3b and 9.3 c for its usage.

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, you can place the **Exit For** statement within the loop; and it is normally used together with the If.....Then... statement. Let's examine example 9.3 d.

Example 9.2

```
Dim sum, n As Integer
Private Sub Form_Activate()
List1.AddItem "n" & vbTab & "sum"
Do
    n = n + 1
    Sum = Sum + n
    List1.AddItem n & vbTab & Sum
    If n = 100 Then
        Exit Do
    End If
Loop
End Sub
```

Explanation

In the above example, we compute the summation of 1+2+3+4+.....+100. In the design stage, you need to insert a ListBox into the form for displaying the output, named List1. The program uses the AddItem method to populate the ListBox. The statement List1.AddItem "n" & vbTab & "sum" will display the headings in the ListBox, where it uses the vbTab function to create a space between the headings n and sum.

VISUAL BASIC

Example 9.3 a For counter=1 to 10 display.Text=counter Next	Example 9.3 b For counter=1 to 1000 step 10 counter=counter+1 Next
Example 9.3 c For counter=1000 to 5 step -5 counter=counter-10 Next *Notice that increment can be negative	Example 9.3 d Private Sub Form_Activate() For n=1 to 10 If n>6 then Exit For End If Else Print n End If End Sub

Lesson 10: Introduction to VB Built-in Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers.

The general format of a function is **FunctionName (arguments)**

The arguments are values that are passed on to the function.

In this lesson, we are going to learn two very basic but useful internal functions of Visual basic , i.e. the **MsgBox()** and **InputBox ()** functions.

10.1 MsgBox () Function

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he /she can continues. This format is as follows:

yourMsg=MsgBox(Prompt, Style Value, Title)

VISUAL BASIC

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, please refer Table 10.1 for types of command button displayed. The Title argument will display the title of the message board.

Table 10.1: Style Values		
Style Value	Named Constant	Buttons Displayed
0	vbOkOnly	Ok button
1	vbOkCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons.
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons

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

Example: `yourMsg=MsgBox("Click OK to Proceed", 1, "Startup Menu")`

and `yourMsg=Msg("Click OK to Proceed".vbOkCancel,"Startup Menu")`

are the same.

`yourMsg` 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

VISUAL BASIC

in the procedure or in the general declaration section. **Table 10.2** shows the values, the corresponding named constant and buttons.

Table 10.2 : 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

Example 10.1

i. The Interface:

You draw three command buttons and a label as shown in Figure 10.1

Figure 10.1

ii. The procedure for the test button:

```

Private Sub Test_Click()
Dim testmsg As Integer
testmsg = MsgBox("Click to test", 1, "Test message")
If testmsg = 1 Then
Display.Caption = "Testing Successful"
Else
Display.Caption = "Testing fail"
End If
End Sub

```

When a user click on the test button, the image like the one shown in Figure 10.2 will appear. As the user click on the OK button, the message "Testing successful" will be displayed and when he/she clicks on the Cancel button, the message "Testing fail" will be displayed.

Figure 10.2

VISUAL BASIC

To make the message box looks more sophisticated, you can add an icon besides the message. There are four types of icons available in VB as shown in Table 10.3		Example 10.2 You draw the same Interface as in example 10.1 but modify the codes as follows: Private Sub test2_Click() Dim testMsg2 As Integer testMsg2 = MsgBox("Click to Test", vbYesNoCancel + vbExclamation, "Test Message") If testMsg2 = 6 Then display2.Caption = "Testing successful" ElseIf testMsg2 = 7 Then display2.Caption = "Are you sure?" Else display2.Caption = "Testing fail" End If End Sub In this example, the following message box will be displayed: Figure 10.3

Table 10.3

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

10.2 The InputBox() Function

An InputBox() function will

display a message box where the user can enter a value or a message in the form of text. The format is

myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)

myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

- Prompt - The message displayed normally as a question asked.
- Title - The title of the Input Box.
- default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.
- x-position and y-position - the position or the coordinate of the input box.

Lesson 11: Mathematical Functions

The mathematical functions are very useful and important in programming because very often we need to deal with mathematical concepts in programming such as chance and probability, variables, mathematical logics, calculations, coordinates, time intervals and etc. The common mathematical functions in Visual Basic are **Rnd, Sqr, Int, Abs, Exp, Log, Sin, Cos, Tan , Atn, Fix** and **Round**.

(i) Rnd is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. In Example 11.1. When you run the program, you will get an output of 10 random numbers between 0 and 1. Randomize Timer is a vital statement here as it will randomize the process.

Example 11.1:

VISUAL BASIC

```
Private Sub Form_Activate
    Randomize Timer
    For x=1 to 10
        Print Rnd
    Next x
```

Random numbers in its original form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 random integers ranging from 1 to 6, which make the program behave as a virtual die, we need to convert the random numbers using the format **Int(Rnd*6)+1**. Let's study the following example:

In this example, `Int(Rnd*6)` will generate a random integer between 0 and 5 because the function **Int** truncates the decimal part of the random number and returns an integer. After adding 1, you will get a random number between 1 and 6 every time you click the command button. For example, let say the random number generated is 0.98, after multiplying it by 6, it becomes 5.88, and using the integer function `Int(5.88)` will convert the number to 5; and after adding 1 you will get 6.

In this example, you place a command button and change its caption to 'roll die'. You also need to insert a label into the form and clear its caption at the designing phase and make its font bigger and bold. Then set the border value to 1 so that it displays a border; and after that set the alignment to center. The statement `Label1.Caption=Num` means the integer generated will be displayed as the caption of the label.

Example 11.2

```
Dim num as integer
Private Sub Command1_Click ( )
    Randomize Timer
    Num=Int(Rnd*6)+1
    Label1.Caption=Num
End Sub
```

The Numeric Functions

The numeric functions are **Int**, **Sqr**, **Abs**, **Exp**, **Fix**, **Round** and **Log**.

VISUAL BASIC

a) **Int** is the function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than the number. For example, $\text{Int}(2.4)=2$, $\text{Int}(4.8)=4$, $\text{Int}(-4.6)= -5$, $\text{Int}(0.032)=0$ and so on.

b) **Sqr** is the function that computes the square root of a number. For example, $\text{Sqr}(4)=2$, $\text{Sqr}(9)=2$ and etc.

c) **Abs** is the function that returns the absolute value of a number. So $\text{Abs}(-8) = 8$ and $\text{Abs}(8)= 8$.

d) **Exp** of a number x is the value of e^x . For example, $\text{Exp}(1)=e^1 = 2.7182818284590$

e) **Fix** and **Int** are the same if the number is a positive number as both truncate the decimal part of the number and return an integer. However, when the number is negative, it will return the smallest integer that is larger than the number. For example, $\text{Fix}(-6.34)= -6$ while $\text{Int}(-6.34)=-7$.

f) **Round** is the function that rounds up a number to a certain number of decimal places. The Format is $\text{Round}(n, m)$ which means to round a number n to m decimal places. For example, $\text{Round}(7.2567, 2) = 7.26$

g) **Log** is the function that returns the natural Logarithm of a number. For example,

$\text{Log } 10= 2.302585$

Example 11.3

This example computes the values of $\text{Int}(x)$, $\text{Fix}(x)$ and $\text{Round}(x,n)$ in a table form. It uses the Do Loop statement and the Rnd function to generate 10 numbers. The statement $x = \text{Round}(\text{Rnd} * 7, 7)$ rounds a random number between 0 and 7 to 7 decimal places. Using commas in between items will create spaces between them and hence a table of values can be created. The program and output are shown below

```
Private Sub Form_Activate ()  
    n = 1  
    Print " n", "      x", "Int(x)", "Fix(x)", "Round(x, 4)"  
    Do While n < 11  
        Randomize Timer
```

```
x = Round (Rnd * 7, 7)
Print n, x, Int(x), Fix(x), Round(x, 4)
n = n + 1
Loop
End Sub
```

The Trigonometric Functions

The common trigonometric functions are **Sin, Cos, Tan and Atn**.

- a) Sin is the function that computes the value of sine of an angle in radian.
- b) Cos is the function that computes the value of cosine of an angle in radian.
- c) Tan is the function that computes the value of tangent of an angle in radian.
- d) Atn is the function that computes the value of arc tangent of an angle in radian.

An angle in degree has to be converted to radian before it can be calculated by the above trigonometric functions. From high school mathematics, we know that π radian is equivalent to 180° ; which means 1 radian is equivalent to π divided by 180. Therefore, in order to convert an angle x from degree to radian, we have to multiply x by $(\pi/180)$. However, there is a small problem because it is rather difficult to obtain the precise value of π , fortunately, there is a way to do it in VB. First of all, we know that an arc tangent of 1 will return the value of 45° which is $\pi/4$ radian. So, to obtain the value of π , just multiply the arc tangent of 1 with 4. Let's examine how all the above calculations can be done in the following examples:

Example 11.4

In this example, the program will display the values of sine, cosine and tangent for various angles in degree between 0° and 360° in a table form. The value of π is obtained using the equation $\pi=4*\text{Atn}(1)$. The angle in degree is converted to radian by multiplying the angle by

VISUAL BASIC

($\pi/180$). Different angles are obtained through the use of For...Next Loop. The program is shown below and the output is shown in Figure 12.4.

```
Private Sub Form_Activate ()
pi = 4 * Atn(1)
Print "angle", "Sin x", "Cos x", "Tan x"
For degree = 0 To 360 Step 30
angle = degree * (pi / 180)
Print degree, Round(Sin(angle), 4), Round(Cos(angle), 4),
Round(Tan(angle), 4)
Next degree
End Sub
```

The output diagram of Example 4

Lesson 12: Formatting Functions

VISUAL BASIC

Formatting output is a very important part of programming so that the data can be presented systematically and clearly to the users. Data in the previous lesson were presented fairly systematically through the use of commas and some of the functions like Int, Fix and Round. However, to have better control of the output format, we can use a number of formatting functions in Visual basic.

The three most common formatting functions in VB are **Tab**, **Space**, and **Format**

(i) The Tab function

Tab (n); x

The item x will be displayed at a position that is n spaces from the left border of the output form. There must be a semicolon in between Tab and the items you intend to display (VB will actually do it for you automatically).

Example1

```
.Private Sub Form_Activate  
    Print "I"; Tab(5); "like"; Tab(10); "to"; Tab(15); "learn";  
    Tab(20); "VB"  
  
    Print  
  
    Print Tab(10); "I"; Tab(15); "like"; Tab(20); "to"; Tab(25);  
    "learn"; Tab(20); "VB"  
  
    Print  
  
    Print Tab(15); "I"; Tab(20); ; "like"
```

(ii) The Space function

The **Space** function is very closely linked to the Tab function. However, there is a minor difference. While Tab (n) means the item is placed n spaces from the left border of the screen, the Space function specifies the number of spaces between two consecutive items. For example, the procedure

Example 2

```
Private Sub Form_Activate()  
    Print "Visual"; Space(10); "Basic"
```


VISUAL BASIC

End Sub

Means that the words Visual and Basic will be separated by 10 spaces

(iii) The Format function

The **Format** function is a very powerful formatting function which can display the numeric values in various forms. There are two types of Format function, one of them is the built-in or predefined format while another one can be defined by the users.

(i) The format of the predefined Format function is

Format (n, "style argument")

where n is a number and the list of style arguments is given in the table

Style argument	Explanation	Example
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal places.	Format(6648972.265, "Standard")=6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")=\$6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

Example 3

VISUAL BASIC

```
Private Sub Form_Activate()  
Print Format (8972.234, "General Number")  
Print Format (8972.2, "Fixed")  
Print Format (6648972.265, "Standard")  
Print Format (6648972.265, "Currency")  
Print Format (0.56324, "Percent")  
End Sub
```

Now, run the program and you will get an output like the figure below:

VISUAL BASIC

Lesson 13: String Manipulation Functions

In this lesson, we will learn how to use some of the string manipulation function such as Len, Right, Left, Mid, Trim, Ltrim, Rtrim, Ucase, Lcase, Instr, Val, Str, Chr and Asc.(i)The Len Function

The length function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The format is

**Len
("Phrase")**

For example,

```
Len  
(VisualBasic)  
= 11 and Len  
(welcome to  
VB tutorial) =  
22
```

The Len function can also return the number of digits or memory locations of a number that is stored in the computer. For example,

```
Private sub  
Form_Activate  
( )  
  
X=sqr (16)  
  
Y=1234  
  
Z#=10#  
  
Print Len(x),  
Len(y), and  
Len (z)  
  
End Sub
```

VISUAL BASIC

will produce the output 1, 4, 8. The reason why the last value is 8 is because z# is a double precision number and so it is allocated more memory spaces.

(ii) The Right Function

The Right function extracts the right portion of a phrase. The format is

Right ("Phrase", n)

Where n is the starting position from the right of the phrase where the portion of the phrase is going to be extracted. For example,

Right("Visual Basic", 4) = asic

(iii) The Left Function

The Left\$ function extract the left portion of a phrase. The format is

Left("Phrase", n)

Where n is the starting position from the left of the phrase where the portion of the phrase is going to be extracted. For example,

Left ("Visual Basic", 4) = Visu

(iv) The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the phrase. The format is

Ltrim("Phrase")

.For example,

Ltrim (" Visual Basic", 4)= Visual basic

(v) The Rtrim Function

The Rtrim function trims the empty spaces of the right portion of the phrase. The format is

Rtrim("Phrase")

.For example,

Rtrim ("Visual Basic ", 4) = Visual basic

(vi) The Trim function

The Ttrim function trims the empty spaces on both side of the phrase. The format is

VISUAL BASIC

Trim("Phrase")

.For example,

Trim (" Visual Basic ") = Visual basic

(viii) The Mid Function

The **Mid** function extracts a substring from the original phrase or string. It takes the following format:

Mid(phrase, position, n)

Where position is the starting position of the phrase from which the extraction process will start and n is the number of characters to be extracted. For example,

Mid("Visual Basic", 3, 6) = ual Bas

(ix) The InStr function

The **InStr** function looks for a phrase that is embedded within the original phrase and returns the starting position of the embedded phrase. The format is

Instr (n, original phase, embedded phrase)

Where n is the position where the Instr function will begin to look for the embedded phrase. For example

Instr(1, "Visual Basic", " Basic")=8

(x) The Ucase and the Lcase functions

The **Ucase** function converts all the characters of a string to capital letters. On the other hand, the **Lcase** function converts all the characters of a string to small letters. For example,

Ucase("Visual Basic") =VISUAL BASiC

Lcase("Visual Basic") =visual basic

(xi) The Str and Val functions

The **Str** is the function that converts a number to a string while the **Val** function converts a string to a number. The two functions are important when we need to perform mathematical operations.

(xii) The Chr and the Asc functions

The **Chr** function returns the string that corresponds to an ASCII code while the **Asc** function converts an ASCII character or symbol to the corresponding ASCII code. ASCII stands for "American Standard Code for Information Interchange". Altogether there are 255 ASCII codes and as many ASCII characters. Some of the characters may not be displayed as they may represent some actions such as the pressing of a key or produce a beep sound. The format of the Chr function is

Chr(charcode)

and the format of the Asc function is

Asc(Character)

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% , Asc("B")=66, Asc("&")=38

Lesson 14: Creating User-Defined Functions

14.1 Creating Your Own Function

The general format of a function is as follows:

Public Function functionName (Arg As dataType,.....) As dataType

or

Private Function functionName (Arg As dataType,.....) As dataType

* Public indicates that the function is applicable to the whole project and
Private indicates that the function is only applicable to a certain module or procedure.

Example 14.1

VISUAL BASIC

In this example, a user can calculate the future value of a certain amount of money he has today based on the interest rate and the number of years from now, supposing he will invest this amount of money somewhere. The calculation is based on the compound interest rate.

VISUAL BASIC

The code

```
Public Function FV(PV As Variant, i As Variant, n As Variant) As Variant
```

```
'Formula to calculate Future Value(FV)
```

```
'PV denotes Present Value
```

```
FV = PV * (1 + i / 100) ^ n
```

```
End Function
```

```
Private Sub compute_Click()
```

```
'This procedure will calculate Future Value
```

```
Dim FutureVal As Variant
```

```
Dim PresentVal As Variant
```

```
Dim interest As Variant
```

```
Dim period As Variant
```

```
PresentVal = PV.Text
```

```
interest = rate.Text
```

```
period = years.Text
```

```
'calling the function
```

```
FutureVal = FV(PresentVal, interest, period)
```

```
MsgBox ("The Future Value is " & FutureVal)
```

```
End Sub
```

Example 14.2

The following program will automatically compute examination grades based on the marks that a student obtained. The code is shown on the right.

The Code

```
Public Function  
grade(mark As  
Variant) As  
String  
Select Case  
mark  
Case Is >= 80  
grade = "A"  
Case Is >= 70  
grade = "B"  
Case Is >= 60  
grade = "C"  
Case Is >= 50  
grade = "D"  
Case Is >= 40  
grade = "E"  
Case Else  
grade = "F"  
End Select  
End Function  
  
Private Sub  
compute_Click()  
grading.Caption  
= grade(mark)  
  
End Sub
```

Lesson 16: Arrays

16.1 Introduction to Arrays

By definition, an array is a list of variables, all with the same data type and name. When we work with a single item, we only need to use one variable. However, if we have a list of items which are of similar type to deal with, we need to declare an array of variables instead of using a variable for each item. For example, if we need to enter one hundred names, we might have difficulty in declaring 100 different names, this is a waste of time and efforts. So, instead of declaring one hundred different variables, we need to declare only one array. We differentiate each item in the array by using subscript, the index value of each item, for example name(1), name(2), name(3)etc. , which will make declaring variables streamline and much systematic.

16.2 Dimension of an Array

An array can be one dimensional or multidimensional. One dimensional array is like a list of items or a table that consists of one row of items or one column of items. A two dimensional array will be a table of items that make up of rows and columns. While the format for a one dimensional array is ArrayName(x), the format for a two dimensional array is ArrayName(x,y) while a three dimensional array is ArrayName(x,y,z) . Normally it is sufficient to use one dimensional and two dimensional array ,you only need to use higher dimensional arrays if you need with engineering problems or even some accounting problems. Let me illustrate the arrays with tables.

Table 16.1. One dimensional Array

Student Name	Name(1)	Name(2)	Name(3)	Name(4)	Name(5)	Name(6)
--------------	---------	---------	---------	---------	---------	---------

Table 16.2 Two Dimensional Array

Name(1,1)	Name(1,2)	Name(1,3)	Name(1,4)
Name(2,1)	Name(2,2)	Name(2,3)	Name(2,4)

16.2 Declaring Arrays

We could use Public or Dim statement to declare an array just as the way we declare a single variable. The Public statement declares an array that can be used throughout an application while the Dim statement declare an array that could be used only in a local procedure.

The general format to declare a one dimensional array is as follow:

Dim arrayName(subs) as dataType

VISUAL BASIC

where subs indicates the last subscript in the array.

Example 16.1

```
Dim CusName(10) as String
```

will declare an array that consists of 10 elements if the statement [Option Base 1](#) appear in the declaration area, starting from CusName(1) to CusName(10). Otherwise, there will be 11 elements in the array starting from CusName(0) through to CusName(10)

CusName(1)	CusName(2)	CusName(3)	CusName(4)	CusName(5)	CusName(6)	CusName(7)	CusName(8)
------------	------------	------------	------------	------------	------------	------------	------------

Example 16.2

```
Dim Count(100 to 500) as Integer
```

declares an array that consists of the first element starting from Count(100) and ends at Count(500)

The general format to declare a two dimensional array is as follow:

```
Dim ArrayName(Sub1,Sub2) as dataType
```

Example 16.3

Dim StudentName(10,10) will declare a 10x10 table make up of 100 students' Names, starting with StudentName(1,1) and end with StudentName(10,10).

16.3 Sample Programs(ii)

(i) The code

The Code

Dim studentName(10) As String	Dim studentName(10) As String
Dim num As Integer	Dim num As Integer
Private Sub addName()	Private Sub addName()
For num = 1 To 10	For num = 1 To 10
studentName(num) =	studentName(num) =
InputBox("Enter the student name", "Enter Name", "", 1500, 4500)	InputBox("Enter the student name")
If studentName(num) <> studentName(num)	List1.AddItem studentName(num)
"" Then	Next
Form1.Print studentName(num)	End Sub
Else	Private Sub Start_Click()
End	addName

VISUAL BASIC

End If

End Sub

Next

End Sub

The above program accepts data entry through an input box and displays the entries in the form itself. As you can see, this program will only allow a user to enter 10 names each time he clicks on the start button.

The above program accepts data entries through an InputBox and displays the items in a list box.

Lesson 17: Working with Files

17.1 Introduction

Up until lesson 13 we are only creating programs that could accept data at runtime, when the program is terminated, the data also disappear. Is it possible to save data accepted by a VB program into a storage device, such as a hard disk or diskette, or even CDRW? The answer is possible. In this chapter, we will learn how to create files by writing them into a storage device and then retrieve the data by reading the contents of the files using a customized VB program

17.2 Creating files

To create a file , we use the following command

Open "fileName" For Output As #fileNumber

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

Examples:

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

will create a text file by the name of sample.txt in My Document folder. The accompany file number is 1. If you wish to create and save the file in A drive, simply change the path, as follows"

Open "A:\sample.txt" For Output As #1

If you wish to create a HTML file , simply change the extension to .html

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

17.2.1 Sample Program : Creating a text file

```
Private Sub create_Click()
```

```
Dim intMsg As String
```

```
Dim StudentName As String
```

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

```
intMsg = MsgBox("File sample.txt opened")
```

```
StudentName = InputBox("Enter the student Name")
```

```
Print #1, StudentName
```

```
intMsg = MsgBox("Writing a" & StudentName & " to sample.txt ")
```

```
Close #1
```

```
intMsg = MsgBox("File sample.txt closed")
```

```
End Sub
```

VISUAL BASIC

* The above program will create a file sample.txt in the My Documents' folder and ready to receive input from users. Any data input by users will be saved in this text file.

17.3 Reading a file

To read a file created in section 17.2, you can use the input # statement. However, we can only read the file according to the format when it was written. You have to open the file according to its file number and the variable that hold the data. We also need to declare the variable using the DIM command.

17.3.1 Sample Program: Reading file

```
Private Sub Reading_Click()  
Dim variable1 As String  
Open "c:\My Documents\sample.txt" For Input As #1  
Input #1, variable1  
Text1.Text = variable1  
Close #1  
  
End Sub
```

* This program will open the sample.txt file and display its contents in the Text1 textbox.

Example 17.3.2 Creating and Reading files using Common Dialog Box

This example uses the common dialog box to create and read the text file, which is much easier than the previous examples as many operations are handled by the common dialog box. The following is the program:

```
Dim linetext As String  
Private Sub open_Click()  
CommonDialog1.Filter =  
"Text files{*.txt}|*.txt"  
CommonDialog1.ShowOpen  
  
If  
CommonDialog1.FileName  
<> "" Then
```

The syntax **CommonDialog1.Filter = "Text files{*.txt}|*.txt"** ensures that only the textfile is read or saved. The statement **CommonDialog1.ShowOpen** is to display the open file dialog box and the statement **CommonDialog1.ShowSave** is to display the save file dialog box. **Text1.Text = Text1.Text & linetext** is to read the data and display them in the Text1 textbox

The Output window is shown below:

VISUAL BASIC

```
Open
CommonDialog1.FileName
For Input As #1
Do
Input #1, linetext
Text1.Text = Text1.Text &
linetext
Loop Until EOF(1)
End If
Close #1
End Sub
Private Sub save_Click()
CommonDialog1.Filter =
"Text files{*.txt}|*.txt"
CommonDialog1.ShowSave
If
CommonDialog1.FileName
<> "" Then
Open
CommonDialog1.FileName
For Output As #1
Print #1, Text1.Text
Close #1
End If
End Sub
```


Lesson 19: Creating Multimedia Applications-Part I

You can create various multimedia applications in VB that could play audio CD, audiofiles, VCD , video files and more.

To be able to play multimedia files or multimedia devices, you have to insert **Microsoft Multimedia Control** into your VB applications that you are going to create. However, Microsoft Multimedia Control is not normally included in the startup toolbox, therefore you need to add the **MM control** by pressing Ctrl+T and select it from the components dialog box that is displayed.

19.1 Creating a CD player

In this program, you can create a CD player that resembles an actual CD player. It allows the user select a track to play, to fast forward, to rewind and also to eject the CD. It can also display the track being played. The interface and code are shown below.

a) The Interface.

The Code

```
Private Sub Form_Load()  
'To position the page at the center  
Left = (Screen.Width - Width) \ 2  
Top = (Screen.Height - Height) \ 2  
'Initialize the CD  
myCD.Command = "Open"  
End Sub
```

```
Private Sub myCD_StatusUpdate()
```

```
'Update the track number
```

```
trackNum.Caption = myCD.Track  
End Sub
```

```
Private Sub Next_Click()  
myCD.Command = "Next"  
End Sub
```

```
Private Sub Play_Click()  
myCD.Command = "Play"  
End Sub  
Private Sub Previous_Click()  
myCD.Command = "Prev"  
End Sub
```

```
Private Sub Stop_Click()  
myCD.Command = "Stop"  
End Sub
```

Lesson 20: Creating Multimedia Applications-Part II

In previous lesson, we have programmed a CD player. Now, by making some modifications, you can transform the CD player into an audio player. This player will be created in such a way that it could search for wave and midi files in your drives and play them.

In this project, you need to insert a **ComboBox**, a **DriveListBox**, a **DirListBox**, a **TextBox** and a **FileListBox** into your form. I shall briefly discuss the function of each of the above controls. Besides, you must also insert **Microsoft Multimedia Control(MMControl)** into your form, you may make it visible or invisible. In my program, I choose to make it invisible so that I can use the command buttons created to control the player.

- ComboBox- to display and enable selection of different type of files.
- DriveListBox- to allow selection of different drives available on your PC.
- DirListBox - To display directories
- TextBox - To display selected files
- FileListBox- To display files that are available

Relevant code must be written to coordinate all the above controls so that the application can work properly. The program should follow in the following logical way:

Step 1: User chooses the type of files he wants to play.

Step2:User selects the drive that might contains the relevant audio files.

Step 3:User looks into directories and subdirectories for the files specified in step1. The files should be displayed in the **FileListBox**.

VISUAL BASIC

Step 4: User selects the files from the **FileListBox** and click the **Play** button.

Step 5: User clicks on the **Stop** button to stop playing and **Exit** button to end the application.

The Interface

The Code	
<pre>Private Sub Combo1_Change() ' to determine file type If ListIndex = 0 Then File1.Pattern = (*.wav) ElseIf ListIndex = 1 Then File1.Pattern = (*.mid) Else File1.Pattern = (*.*) End If End Sub ii Private Sub Dir1_Change() 'To change directories and subdirectories(or folders and subfolders) File1.Path = Dir1.Path If Combo1.ListIndex = 0 Then File1.Pattern = (*.wav) ElseIf Combo1.ListIndex = 1 Then</pre>	<pre>ii Private Sub play_Click() 'To play WaveAudio file or Midi File Command2_Click If Combo1.ListIndex = 0 Then AudioPlayer.DeviceType = "WaveAudio" ElseIf Combo1.ListIndex = 1 Then AudioPlayer.DeviceType = "Sequencer" End If AudioPlayer.FileName = Text1.Text AudioPlayer.Command = "Open" AudioPlayer.Command = "Play" ii</pre>

VISUAL BASIC

<pre> File1.Pattern = ("*.mid") Else File1.Pattern = ("*.wav") End If End Sub </pre>	<pre> End Sub Private Sub stop_Click() If AudioPlayer.Mode = 524 Then Exit Sub If AudioPlayer.Mode <> 525 Then AudioPlayer.Wait = True AudioPlayer.Command = "Stop" End If AudioPlayer.Wait = True AudioPlayer.Command = "Close" End Sub </pre>
<pre> Private Sub Drive1_Change() 'To change drives Dir1.Path = Drive1.Drive End Sub Private Sub File1_Click() If Combo1.ListIndex = 0 Then File1.Pattern = ("*.wav") ElseIf Combo1.ListIndex = 1 Then File1.Pattern = ("*.mid") Else File1.Pattern = ("*.wav") End If If Right(File1.Path, 1) <> "\" Then filenam = File1.Path + "\" + File1.FileName Else filenam = File1.Path + File1.FileName End If Text1.Text = filenam End Sub Private Sub Form_Load() 'To center the Audioplayer startup page Left = (Screen.Width - Width) \ 2 Top = (Screen.Height - Height) \ 2 Combo1.Text = "*.wav" Combo1.AddItem "*.wav" Combo1.AddItem "*.mid" </pre>	

Combo1.AddItem "All files" End Sub	
---------------------------------------	--

Lesson 21: Creating Multimedia Applications-PartIII

In lesson 20, we have created an audio player. Now, by making further modifications, you can transform the audio player into a picture viewer. This viewer will be created in such a way that it could search for all types of graphics files in your drives and displays them in a picture frame.

Similar to the previous project, in this project, you need to insert a **ComboBox**, a **DriveListBox**, a **DirListBox**, a **TextBox** and a **FileListBox** into your form. I shall briefly explain again the function of each of the above controls.

- ComboBox- to display and enable selection of different type of files.
- DriveListBox- to allow selection of different drives available on your PC.
- DirListBox - To display directories
- TextBox - To display selected files
- FileListBox- To display files that are available

Relevant codes must be written to coordinate all the above controls so that the application can work properly. The program should flow in the following logical way:

Step 1: User chooses the type of files he wants to play.

Step2:User selects the drive that might contains the relevant graphic files.

Step 3:User looks into directories and subdirectories for the files specified in step1. The files should be displayed in the FileListBox.

Step 4: User selects the files from the FileListBox and click the Show button.

Step 5: User clicks on Exit button to end the application.

VISUAL BASIC

The Interface

The Code

```
Private Sub Form_Load()
```

```
'To center the player
```

```
Left = (Screen.Width - Width) \ 2
```

```
Top = (Screen.Height - Height)\2
```

```
Combo1.Text = "All graphic files"
```

```
Combo1.AddItem "All graphic files"
```

```
Combo1.AddItem "All files"
```

```
End Sub
```

```
ii
```

```
Private Sub Combo1_Change()
```

```
If ListIndex = 0 Then
```

```
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")
```

```
Else
```

```
File1.Pattern = ("*.*)"
```

```
End If
```

```
End Sub
```

```
'Specific the types of files to load
```

```
Private Sub Dir1_Change()
```

```
File1.Path = Dir1.Path
```

```
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")
```

```
End Sub
```

```
'Changing Drives
```

```
Private Sub Drive1_Change()
```

```
Dir1.Path = Drive1.Drive
```

```
End Sub
```

```
Private Sub File1_Click()
```

```
If Combo1.ListIndex = 0 Then
```

```
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")
```

```
Else
```

```
File1.Pattern = ("*.*)"
```

```
End If
```

VISUAL BASIC

```
If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If
Text1.Text = filenam

End Sub
```

```
Private Sub show_Click()

If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If

'To load the picture into the picture box
picture1.Picture = LoadPicture(filenam)

End Sub
```

Lesson 23: Creating database applications in VB-Part I

Visual basic allows us to manage databases created with different database programs such as MS Access, Dbase, Paradox and etc. In this lesson, we are not dealing with how to create database files but we will see how we can access database files in the VB environment. In the following example, we will create a simple database application which enable one to browse customers' names. To create this application, insert the data control into the new form. Place the data control somewhere at the bottom of the form. Name the data control as data_navigator. To be able to use the data control, we need to connect it to any database. We can create a database file using any database application but I suggest we use the database files that come with VB6. Let select NWIND.MDB as our database file.

VISUAL BASIC

To connect the data control to this database, double-click the [DatabaseName](#) property in the properties window and select the above file, i.e NWIND.MDB. Next, double-click on the [RecordSource](#) property to select the customers table from the database. You can also change the caption of the data control to anything but I use "Click to browse Customers" here. After that, we will place a label and change its caption to Customer Name. Last but not least, insert another label and name it as cus_name and leave the label empty as customers' names will appear here when we click the arrows on the data control. We need to bind this label to the data control for the application to work. To do this, open the label's [DataSource](#) and select data_navigator that will appear automatically. One more thing that we need to do is to bind the label to the correct field so that data in this field will appear on this label. To do this, open the [DataField](#) property and select ContactName. Now, press F5 and run the program. You should be able to browse all the customers' names by clicking the arrows on the data control.

The Design Interface.

The Runtime Interface

VISUAL BASIC

You can also add other fields using exactly the same method. For example, you can add address, City and telephone number to the database browser.

VISUAL BASIC

End Sub

```
Private Sub Command4_Click()  
dtaBooks.Recordset.MoveLast  
End Sub
```

Run the application and you shall obtain the interface below and you will be able to browse the database using the four command buttons.

Lesson 25: Creating VB database applications using ADO control

In Lesson 22 and Lesson 23, we have learned how to build VB database applications using data control. However, data control is not a very flexible tool as it could only work with limited kinds of data and must work strictly in the Visual Basic environment. To overcome these limitations, we can use a much more powerful data control in Visual Basic, known as ADO control. ADO stands for ActiveX data objects. As ADO is ActiveX-based, it can work in different platforms (different computer systems) and different programming languages. Besides, it can access many different kinds of data such as data displayed in the Internet browsers, email text and even graphics other than the usual relational and non relational database information.

To be able to use ADO data control, you need to insert it into the toolbox. To do this, simply press Ctrl+T to open the components dialog box and select Microsoft ActiveX Data Control 6. After this, you can proceed to build your ADO-based VB database applications.

The following example will illustrate how to build a relatively powerful database application using ADO data control. First of all, name the new form as frmBookTitle and change its caption to Book Titles- ADO Application. Secondly, insert the ADO data control and name it as adoBooks and change its caption to book. Next, insert the necessary labels, text boxes and command buttons. The runtime interface of this program is shown in the diagram below, it allows adding and deletion as well as updating and browsing of data.

VISUAL BASIC

VISUAL BASIC

The properties of all the controls are listed as follow:

Form Name	frmBookTitle
Form Caption	Book Titles - ADOApplication
ADO Name	adoBooks
Label1 Name	lblApp
Label1 Caption	Book Titles
Label 2 Name	lblTitle
Label2 Caption	Title :
Label3 Name	lblYear
Label3 Caption	Year Published:
Label4 Name	lblISBN
Label4 Caption	ISBN:
Label5 Name	lblPubID
Label5 Caption	Publisher's ID:
Label6 Name	lblSubject
Label6 Caption	Subject :
TextBox1 Name	txttitle
TextBox1 DataField	Title
TextBox1 DataSource	adoBooks
TextBox2 Name	txtPub
TextBox2 DataField	Year Published
TextBox2 DataSource	adoBooks
TextBox3 Name	txtISBN
TextBox3 DataField	ISBN
TextBox3 DataSource	adoBooks
TextBox4 Name	txtPubID
TextBox4 DataField	PubID
TextBox4 DataSource	adoBooks
TextBox5 Name	txtSubject
TextBox5 DataField	Subject
TextBox5 DataSource	adoBooks
Command Button1 Name	cmdSave
Command Button1 Caption	&Save
Command Button2 Name	cmdAdd
Command Button2 Caption	&Add
Command Button3 Name	cmdDelete
Command Button3 Caption	&Delete
Command Button4 Name	cmdCancel
Command Button4 Caption	&Cancel
Command Button5	cmdPrev

VISUAL BASIC

Name	
Command Button5 Caption	&<
Command Button6 Name	cmdNext
Command Button6 Caption	&>
Command Button7 Name	cmdExit
Command Button7 Caption	E&xit

To be able to access and manage a database, you need to connect the ADO data control to a database file. We are going to use **BIBLIO.MDB** that comes with VB6. To connect ADO to this database file , follow the steps below:

- a) Click on the ADO control on the form and open up the properties window.
- b) Click on the ConnectionString property, the following dialog box will appear.

when the dialog box appear, select the Use **Connection String's** Option. Next, click build and at the **Data Link dialog box**, double-Click the option labeled **Microsoft Jet 3.51 OLE DB provider**.

VISUAL BASIC

After that, click the Next button to select the file **BIBLO.MDB**. You can click on Text Connection to ensure proper connection of the database file. Click OK to finish the connection.

Finally, click on the RecordSource property and set the **command type** to **adCmdTable** and **Table name** to **Titles**. Now you are ready to use the database file.

ii

Now, you need to write code for all the command buttons. After which, you can make the ADO control invisible.

ii

ii

For the **Save** button, the program codes are as follow:

```
Private Sub cmdSave_Click()
```

```
adoBooks.Recordset.Fields("Title") = txtTitle.Text  
adoBooks.Recordset.Fields("Year Published") = txtPub.Text  
adoBooks.Recordset.Fields("ISBN") = txtISBN.Text  
adoBooks.Recordset.Fields("PubID") = txtPubID.Text  
adoBooks.Recordset.Fields("Subject") = txtSubject.Text  
adoBooks.Recordset.Update
```

```
End Sub
```

For the **Add** button, the program codes are as follow:

```
Private Sub cmdAdd_Click()
```

VISUAL BASIC

```
adoBooks.Recordset.AddNew
```

```
End Sub
```

For the **Delete** button, the program codes are as follow:

```
Private Sub cmdDelete_Click()
```

```
    Confirm = MsgBox("Are you sure you want to delete this record?", vbYesNo,  
    "Deletion Confirmation")
```

```
    If Confirm = vbYes Then
```

```
        adoBooks.Recordset.Delete
```

```
        MsgBox "Record Deleted!", , "Message"
```

```
    Else
```

```
        MsgBox "Record Not Deleted!", , "Message"
```

```
    End If
```

```
End Sub
```

```
ii
```

For the **Cancel** button, the program codes are as follow:

```
Private Sub cmdCancel_Click()
```

```
    txtTitle.Text = ""
```

```
    txtPub.Text = ""
```

```
    txtPubID.Text = ""
```

```
    txtISBN.Text = ""
```

```
    txtSubject.Text = ""
```

```
End Sub
```

For the Previous (<) button, the program codes are

```
Private Sub cmdPrev_Click()
```

```
    If Not adoBooks.Recordset.BOF Then
```

```
        adoBooks.Recordset.MovePrevious
```

```
    If adoBooks.Recordset.BOF Then
```

```
        adoBooks.Recordset.MoveNext
```

VISUAL BASIC

End If

End If

End Sub

For the Next(>) button, the program codes are

```
Private Sub cmdNext_Click()
```

```
    If Not adoBooks.Recordset.EOF Then
```

```
        adoBooks.Recordset.MoveNext
```

```
    If adoBooks.Recordset.EOF Then
```

```
        adoBooks.Recordset.MovePrevious
```

```
    End If
```

```
End Sub
```

End Sub

Lesson 26: Using Microsoft DataGrid Control 6.0

In the previous chapter, we use textboxes to display data by connecting them to a database via Microsoft ADO data Control 6.0. The textbox is not the only control that can display data from a database, many other controls in Visual Basic can display data. One of the them is the Data Grid control. Data Grid control can be used to display the entire table of a record set of a database. It allows users to view and edit the data.

Data Grid control is the not the default item in the Visual Basic control toolbox, you have add it from the VB6 components. To add the Data Grid control, click on the project in the menu bar and select components where a dialog box that displays all the available VB6 components. Select Microsoft Data Grid Control 6.0 by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the Microsoft ADO data control so that you are able to access the database. Lastly, click on the OK button to exit the dialog box. Now you should be able to see that the Data Grid control and the ADO data control are added to the toolbox. The next step is to drag the DataGrid control and the ADO data control into the form.

The components dialog box is shown below:

VISUAL BASIC

Before you proceed , you need to create a database file using Microsoft Access. Here I created a file to store my the information of my books and I name the table book. After you have created the table, enter a few records such as mine. The table is shown below:

VISUAL BASIC

Now you need to connect the database to the ADO data control. To do that, right click on the ADO data control and select the ADODC properties, the following dialog box will appear.

Next click on the Build button and the Data Link Properties dialog box will appear (as shown below). In this dialog box, select the database file you have created, in my case, the file name is books.mdb. Press test connection to see whether the connection is successful. If the connection is successful, click OK to return to the ADODC property pages dialog box. At the ADODC property pages dialog box, click on the Recordsource tab and select 2-adCmdTable under command type and select book as the table name, then click OK.

VISUAL BASIC

VISUAL BASIC

Finally you need to display the data in the DataGrid control. To accomplish this, go to the properties window and set the DataSource property of the DataGrid to Adodc1. You can also permit the user to add and edit your records by setting the AllowUpdate property to True. If you set this property to false, the user cannot edit the records. Now run the program and the output window is shown below:

Lesson 27: Using SQL queries in Visual Basic 6

In the previous chapter, we have learned to use the DataGrid Control to display data from a database in Visual Basic 6 environment. However, it does not allow users to search for and select the information they want to see. In order to search for a certain information, we need to use SQL query. SQL stands for **Structures Query Language**. Using SQL keywords, we are able to select specific information to be displayed based on certain criteria. The most basic SQL keyword is **SELECT**, it is used together with the keyword **FROM** to select information from one or more tables from a database. The syntax is:

SELECT fieldname1,fieldname2,.....,fieldnameN
FROM TableName

fieldname1, fieldname2,.....fieldnameN are headings of the columns from a table of a database. You can select any number of fieldname in the query. If you wish to select all the information, you can use the following syntax:

SELECT * FROM TableName

In order to illustrate the usage of SQL queries, lets create a new database in Microsoft Access with the following filenames **ID, Title, Author, Year, ISBN, Publisher, Price** and save the table as **book** and the database as **books.mdb** in a designated folder.

Next, we will start Visual Basic and insert an ADO control, a DataGrid and three command buttons. Name the three command buttons as **cmdAuthor, cmdTitle** and **cmdAll**. Change their captions to **Display Author ,Display Book Title** and **Display All** respectively. You can also change the caption of the form to **My Books**. The design interface is shown below:

VISUAL BASIC

Now you need to connect the database to the ADO data control. Please refer to [lesson 25](#) for the details. However, you need to make one change. At the ADODC property pages dialog box, click on the Recordsource tab and select **1-adCmdText** under command type and under Command Text(SQL) key in **SELECT * FROM book.**

VISUAL BASIC

ii

ii

Next, click on the command button cmdAuthor and key in the following statements:

```
Private Sub cmdAuthor_Click()  
  
Adodc1.RecordSource = "SELECT Author FROM book"  
Adodc1.Refresh  
Adodc1.Caption = Adodc1.RecordSource  
  
End Sub
```

and for the command button cmdTitle, key in

```
Private Sub cmdTitle_Click()  
  
Adodc1.RecordSource = "SELECT Title FROM book"  
Adodc1.Refresh  
Adodc1.Caption = Adodc1.RecordSource  
  
End Sub
```

Finally for the command button cmdAll, key in

```
Private Sub cmdAll_Click()  
  
Adodc1.RecordSource = "SELECT * FROM book"  
Adodc1.Refresh  
Adodc1.Caption = Adodc1.RecordSource  
  
End Sub
```

Now, run the program and when you click on the Display Author button, only the names of authors will be displayed, as shown below:

! ; !
!
!

8 ! ; ! !

(; G -R S

,)Z8 !)080\$
*L3# , !
)Z8 ! 3)Z8 !
90L0 !
)! I

- \$) @) (@""""@) 8
!7*G & 8 9 7 \$

E W X XE WE XW 8

& !
* " ; ^
;3 ;3;\$!
L 6 \$ 4 !
\$ 5)Z86 ! - \$ 2 !7*G & %" /
; ^
)Z8 M ! , Z !
I

VISUAL BASIC

Example 21d1: Query based on Author

Run the program and key in the following SQL query statement

```
SELECT Title, Author FROM book WHERE Author='Liew Voon Kiong'
```

Where you click on the query button, the DataGrid will display the author name Liew Voon Kiong. as shown below:

ii

You can also try following queries:

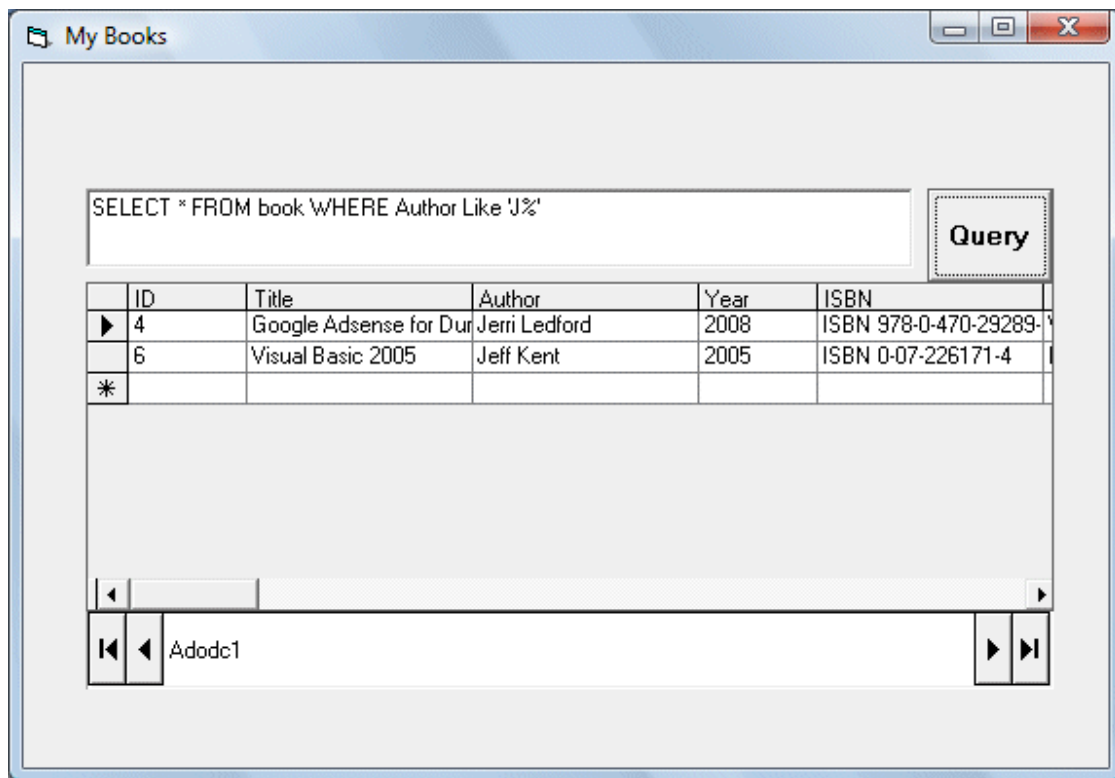
- SELECT * FROM book WHERE Price<=80
- SELECT * FROM book WHERE Year=2008
- SELECT * FROM book WHERE Author<>'Liew Voon Kiong'

You may also search for data that contain certain characters by pattern matching. It involves using the **Like** operator and the **%** symbol. For example, if you want to search for a author name that begins with alphabet J, you can use the following query statement

```
SELECT * FROM book WHERE Author Like 'J%'
```


VISUAL BASIC

Where you click on the query command button, the records where authors' name start with the alphabet J will be displayed, as shown below:



Next, if you wish to rank order the data, either in ascending or descending order, you can use the **ORDER By , ASC (for ascending) and DESC(Descending)** SQL keywords.

The general formats are

SELECT fieldname1, fieldname2.....FROM table ORDER BY fieldname ASC

SELECT fieldname1, fieldname2.....FROM table ORDER BY fieldname DESC

Example 21d3:

The following query statement will rank the records according to Author in ascending order.

SELECT Title, Author FROM book ORDER BY Author ASC

VISUAL BASIC

My Books

SELECT Title, Author FROM book ORDER BY Author ASC

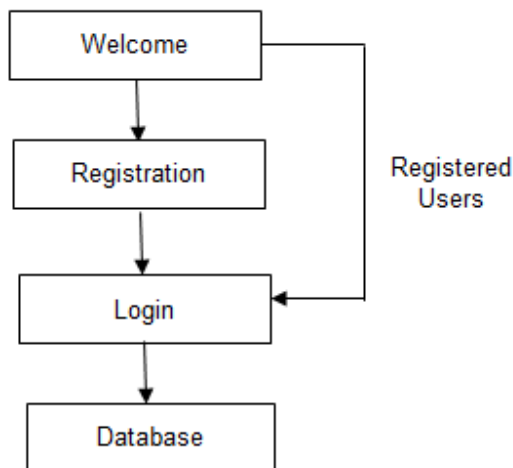
Query

	Title	Author
▶	Visual Basic 6_How To	Deitel & Deitel T.R. Niet
	Visual Basic 2005	Jeff Kent
	Google Adsense for Dur	Jerri Ledford
	Visual Basic 6 Made Ea	Liew Voon Kiong
	Visual Basic 6 & Internet	P.Sellappan
	Secrets of Internet Millio	Stuart Tan
*		

Adodc1

Lesson 29: Creating Advanced VB database application using ADO control

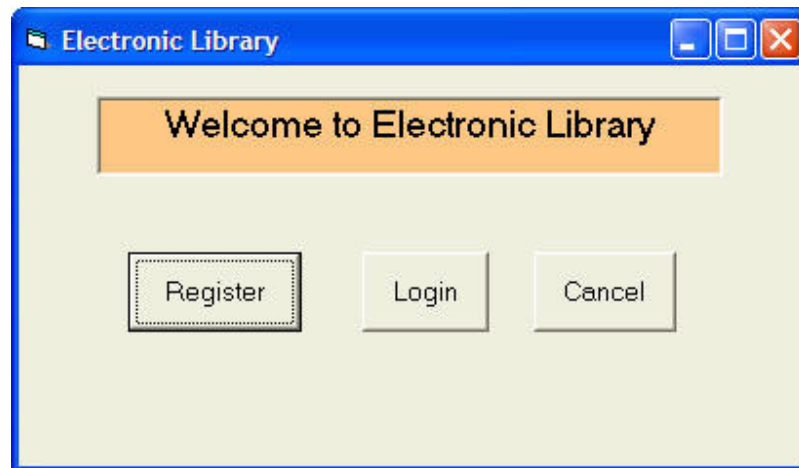
In previous lessons, you have learned how to design database applications using data control and ADO control. However, those applications are very simple and plain. In this lesson, you will learn how to create a more advanced database application using ADO control. The application you are going to create is known as an electronic library. This electronic library will be able to accept the user registration as well as handling login command that require the user's password, thus enhancing the security aspect of the database. Basically, the application will constitute a welcome menu, a registration menu, a Login menu and the main database menu. The sequence of the menus are illustrated as follow:



2.1 The Welcome Menu

First of all, you need to design the Welcome menu. You can follow the example as follow:

VISUAL BASIC



In this form, you need to insert three command buttons and set their properties as follow:

ii

Form name	main_menu
command button 1 Name	cmdRegister
command button 1 Caption	Register
command button 2 Name	cmdLogin
command button 2 Caption	Login
command button 3 Name	cmdCancel
command button 3 Caption	Cancel

The code is as follows:

```
Private Sub cmdCancel_Click()  
End  
End Sub
```

```
Private Sub cmdLogin_Click()  
main_menu.Hide  
Login_form.Show  
End Sub
```

```
Private Sub cmdRegister_Click()
```

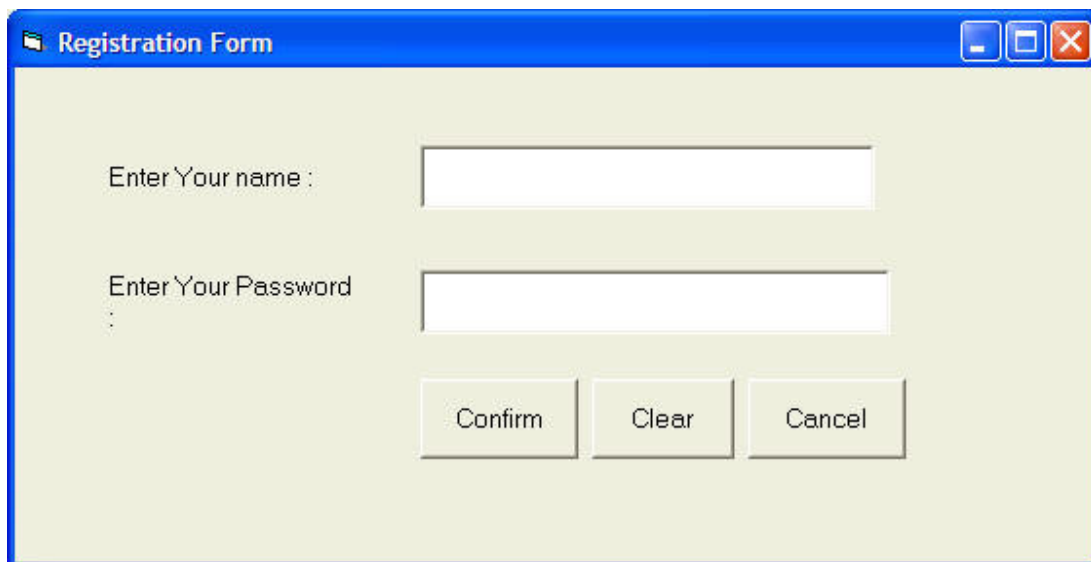
VISUAL BASIC

```
main_menu.Hide  
Register.Show  
End Sub
```

ii

29.2 The Registration Form

If a new user click the Register button, the registration form will appear. An example is illustrated as follow:



This registration forms consist of two text boxes , three command buttons and an ADO control. Their properties are set as follow:

Form name	Register
textbox 1 name	txtName
textbox 2 name	txtpassword
textbox 2 PasswordChar	*
command button 1 name	cmdConfirm
command button 1 Caption	Confirm
command button 2 name	cmdClear
command button 2 Caption	Clear
command button 3 name	cmdCancel

VISUAL BASIC

command button 3 Caption	Cancel
ADO control name	UserInfo

note that the PasswordChar of textbox 2 is set as * which means users will not be able to see the actual characters they enter, they will only see the * symbol.

The codes are as follow:

```
Private Sub cancel_Click( )
```

```
End
```

```
End Sub
```

```
Private Sub cmdClear_Click( )
```

```
txtName.Text = ""
```

```
txtpassword.Text = ""
```

```
End Sub
```

```
Private Sub cmdConfirm_Click()
```

```
UserInfo.Recordset.Fields("username") = txtName.Text
```

```
UserInfo.Recordset.Fields("password") = txtpassword.Text
```

```
UserInfo.Recordset.Update
```

```
Register.Hide
```

```
Login_form.Show
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
UserInfo.Recordset.AddNew
```

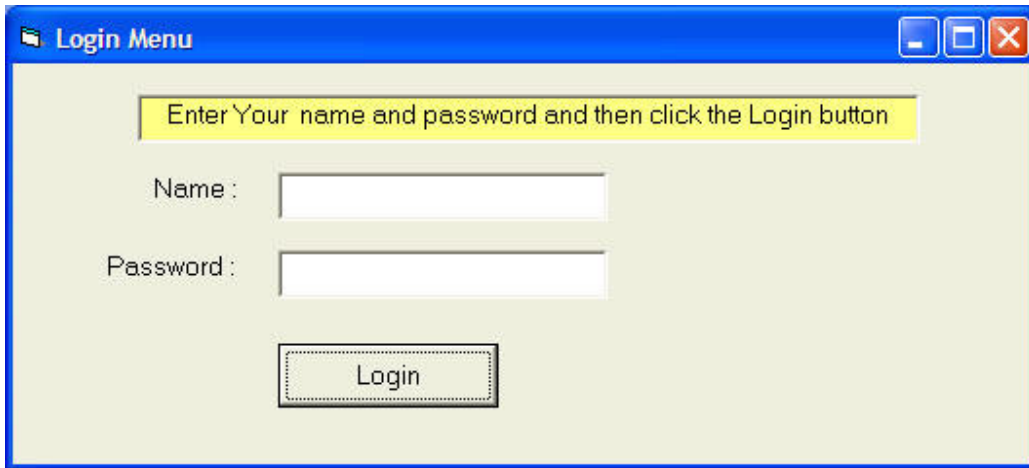
```
End Sub
```

```
ii
```

29.3 The Login Menu

The Login menu is illustrated as follow:

VISUAL BASIC



There are two text boxes and a command button, their properties are set as follow:

Textbox 1 name	txtName
Textbox 2 name	txtpassword
Command button 1 name	cmdLogin
Command button 1 Caption	Login
Form name	Login_form

The codes are as follow:

```
Private Sub cmdLogin_Click()
```

```
Dim username As String  
Dim psword As String  
Dim usernam As String  
Dim pssword As String  
Dim Msg As String
```

```
Register.UserInfo.Refresh  
username = txtName.Text  
psword = txtpassword.Text
```

```
Do Until Register.UserInfo.Recordset.EOF  
If Register.UserInfo.Recordset.Fields("username").Value = username And  
Register.UserInfo.Recordset.Fields("password").Value = psword Then  
Login_form.Hide
```

VISUAL BASIC

```
frmLibrary.Show
```

```
Exit Sub
```

```
Else
```

```
Register.UserInfo.Recordset.MoveNext
```

```
End If
```

```
Loop
```

```
Msg = MsgBox("Invalid password, try again!", vbOKCancel)
```

```
If (Msg = 1) Then
```

```
Login_form.Show
```

```
txtName.Text = ""
```

```
txtpassword = ""
```

```
Else
```

```
End
```

```
End If
```

```
End Sub
```

```
ii
```

29.4 The Main Database Manager

The main database manager is illustrated as follow:

VISUAL BASIC

The screenshot shows a standard Windows-style application window. The title bar is blue and contains the text 'Home Library - Created by LiewVK on 2003-5-26'. The main area has a light beige background. The form consists of five text boxes arranged vertically, each with a label to its left. The labels are 'Title', 'Author', 'Publisher', 'Year', and 'Category'. The text boxes contain the following values: 'Visual Basic 6 & Internet', 'P.Sellappan', 'Sejana Publishing', '2001', and 'Programming'. Below the text boxes is a horizontal line, and underneath that line are seven command buttons. The buttons are labeled 'Save', 'New', 'Delete', 'Cancel', 'Next', 'Previous', and 'Exit'.

The properties of all controls are listed in the table below:

ii

Form name	frmLibrary
ADO control name	adoLibrary
ADO visible	False
TextBox 1 name	txtTitleA
TextBox 2 name	txtAuthor
TextBox 3name	txtPublisher
TextBox 4 name	txtYear
TextBox 5 name	txtCategory
Command button 1 name	cmdSave
Command button 1 caption	&Save
Command button 2 name	cmdNew
Command button 2 caption	&New
Command button 3 name	cmdDelete
Command button 3 caption	&Delete
Command button 4 name	cmdCancel
Command button 4	&Cancel

VISUAL BASIC

caption	
Command button 5 name	cmdNext
Command button 5 caption	N&ext
Command button 6 name	cmdPrevious
Command button 6 caption	&Previous
Command button 7 name	cmdExit
Command button 7 caption	E&xit

ii

The codes are as follow:

ii

Private Sub cmdCancel_Click()

txtTitle.Text = ""

txtAuthor.Text = ""

txtPublisher.Text = ""

txtYear.Text = ""

txtCategory.Text = ""

End Sub

Private Sub cmdDelete_Click()

Confirm = MsgBox("Are you sure you want to delete this record?", vbYesNo, "Deletion Confirmation")

If Confirm = vbYes Then

adoLibrary.Recordset.Delete

MsgBox "Record Deleted!", , "Message"

Else

MsgBox "Record Not Deleted!", , "Message"

End If

End Sub

Private Sub cmdExit_Click()

End

End Sub

VISUAL BASIC

```
Private Sub cmdNew_Click()  
adoLibrary.Recordset.AddNew
```

```
End Sub
```

```
Private Sub cmdNext_Click()  
If Not adoLibrary.Recordset.EOF Then  
adoLibrary.Recordset.MoveNext  
If adoLibrary.Recordset.EOF Then  
adoLibrary.Recordset.MovePrevious  
End If  
End If
```

```
End Sub
```

```
Private Sub cmdPrevious_Click()  
If Not adoLibrary.Recordset.BOF Then  
adoLibrary.Recordset.MovePrevious  
If adoLibrary.Recordset.BOF Then  
adoLibrary.Recordset.MoveNext  
End If  
End If
```

```
End Sub
```

```
Private Sub cmdSave_Click()
```

```
adoLibrary.Recordset.Fields("Title").Value = txtTitle.Text  
adoLibrary.Recordset.Fields("Author").Value = txtAuthor.Text  
adoLibrary.Recordset.Update
```

```
End Sub
```

Lesson 30: Animation-Part I

Animation is always an interesting and exciting part of programming. Although visual basic is not designed to handle advance animations, you can still create some interesting animated effects if you put in some hard thinking. There are many ways to create animated effects in VB6, but for a start we will focus on some easy methods.

The simplest way to create animation is to set the **VISIBLE** property of a group of images or pictures or texts and labels to true or false by triggering a set of events such as clicking a button. Let's examine the following example:

This is a program that create the illusion of moving the jet plane in four directions, North, South ,East, West. In order to do this, insert five images of the same picture into the form. Set the visible property of the image in the center to be true while the rest set to false. On start-up, a user will only be able to see the image in the center. Next, insert four command buttons into the form and change the labels to Move North, Move East, Move West and Move South respectively. Double click on the move north button and key in the following procedure:

```
Sub Command1_click( )
```

```
Image1.Visible = False  
Image3.Visible = True  
Image2.Visible = False  
Image4.Visible = False  
Image5.Visible = False
```

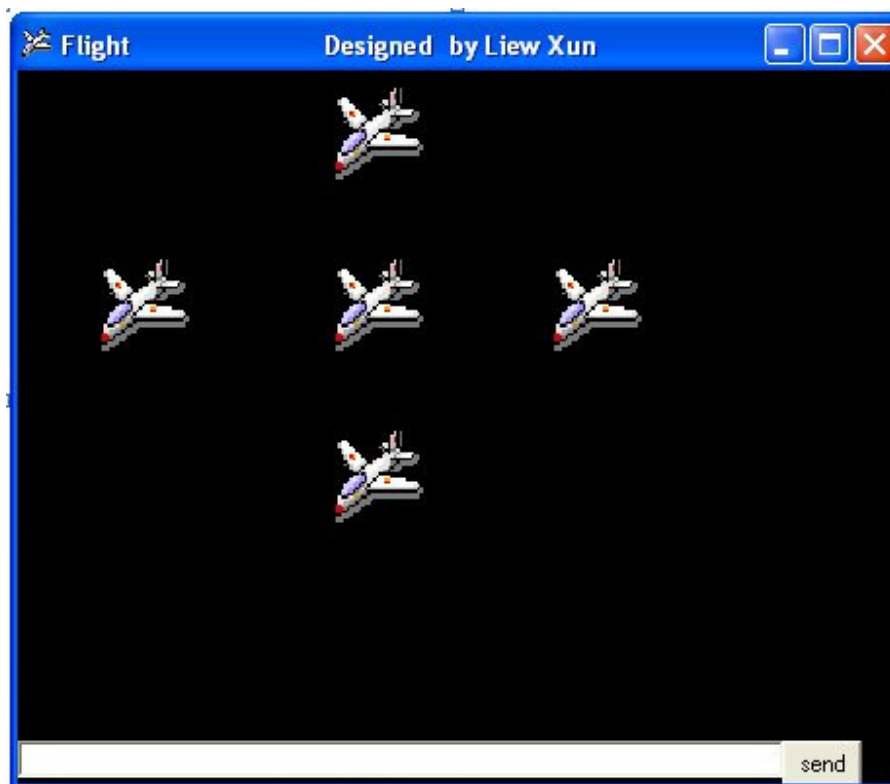
```
End Sub
```

By clicking on the move north button, only image 3 is displayed. This will give an illusion that the jet plane has moved north. Key in similar procedures by double clicking other command buttons. You can also insert an addition command button and label it as Reset and key in the following codes:

```
Image1.Visible = True  
Image3.Visible = False  
Image2.Visible = False  
Image4.Visible = False  
Image5.Visible = False
```

Clicking on the reset button will make the image in the center visible again while other images become invisible, this will give the false impression that the jet plane has move back to the original position.

VISUAL BASIC



VISUAL BASIC

You can also issue the commands using a textbox, this idea actually came from my son Liew Xun (10 years old). His program is shown below:

```
Private Sub Command1_Click()
```

```
    If Text1.Text = "n" Then
```

```
        Image1.Visible = False
```

```
        Image3.Visible = True
```

```
        Image2.Visible = False
```

```
        Image4.Visible = False
```

```
        Image5.Visible = False
```

```
    ElseIf Text1.Text = "e" Then
```

```
        Image1.Visible = False
```

```
        Image4.Visible = True
```

```
        Image2.Visible = False
```

```
        Image3.Visible = False
```

```
        Image5.Visible = False
```

```
    ElseIf Text1.Text = "w" Then
```

```
        Image1.Visible = False
```

```
        Image3.Visible = False
```

```
        Image2.Visible = False
```

```
        Image4.Visible = False
```

```
        Image5.Visible = True
```

```
    ElseIf Text1.Text = "s" Then
```

```
        Image1.Visible = False
```

```
        Image3.Visible = False
```

```
        Image2.Visible = True
```

```
        Image4.Visible = False
```

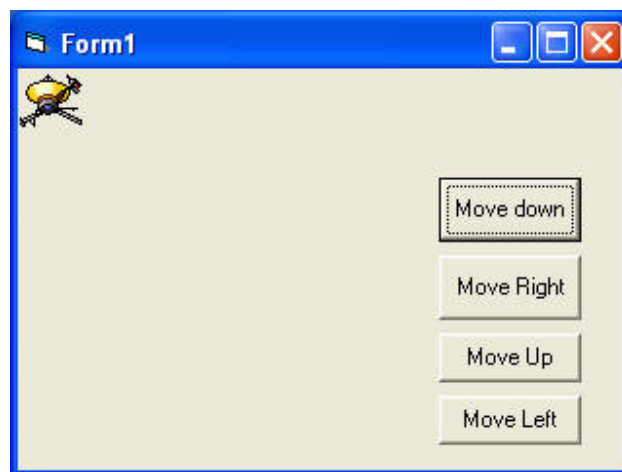
```
        Image5.Visible = False
```

```
    End If
```

```
End Sub
```

VISUAL BASIC

Another simple way to simulate animation in VB6 is by using the Left and Top properties of an object. Image.Left give the distance of the image in twips from the left border of the screen, and Image.Top give the distance of the image in twips from the top border of the screen, where 1 twip is equivalent to 1/1440 inch. Using a statement such as Image.Left-100 will move the image 100 twips to the left, Image.Left+100 will move the image 100 twip away from the left(or 100 twips to the right), Image.Top-100 will move the image 100 twips to the top and Image.Top+100 will move the image 100 twips away from the top border (or 100 twips down).Below is a program that can move an object up, down. left, and right every time you click on a relevant command button.



The Code

```
Private Sub Command1_Click()  
Image1.Top = Image1.Top + 100  
End Sub
```

```
Private Sub Command2_Click()  
Image1.Top = Image1.Top - 100  
End Sub
```

```
Private Sub Command3_Click()  
Image1.Left = Image1.Left + 100  
End Sub
```

```
Private Sub Command4_Click()  
Image1.Left = Image1.Left - 100  
End Sub
```

VISUAL BASIC

The fourth example let user magnify and diminish an object by changing the height and width properties of an object. It is quite similar to the previous example. The statements `Image1.Height = Image1.Height + 100` and `Image1.Width = Image1.Width + 100` will increase the height and the width of an object by 100 twips each time a user click on the relevant command button. On the other hand, The statements `Image1.Height = Image1.Height - 100` and `Image1.Width = Image1.Width - 100` will decrease the height and the width of an object by 100 twips each time a user click on the relevant command button



The Code

```
Private Sub Command1_Click()  
Image1.Height = Image1.Height + 100  
Image1.Width = Image1.Width + 100  
End Sub
```

```
Private Sub Command2_Click()  
  
Image1.Height = Image1.Height - 100  
Image1.Width = Image1.Width - 100  
  
End Sub
```

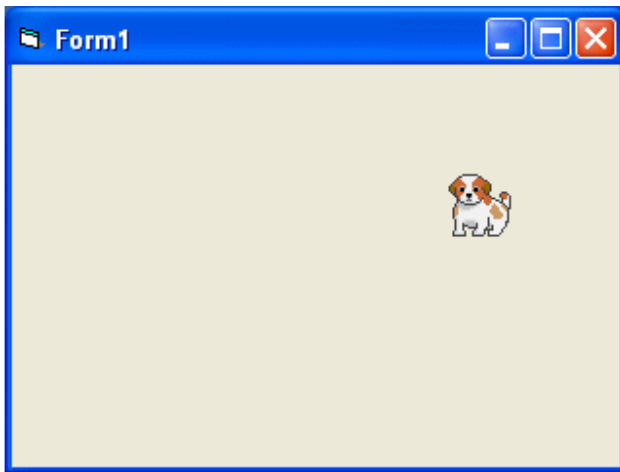
You can try to combine both programs above and make an object move and increases or decreases in size each time a user click a command button.

Lesson 32: Animation - Part III

32.1 Animation using Timer

All preceding examples of animation that you have learn in lesson 23 and lesson 24 only involve manual animation, which means you need to keep on clicking a certain command button or pressing a key to make an object animate. In order to make it move automatically, you need to use a timer. The first step in creating automatic animation is to drag the timer from the toolbox into the form and set its interval to a certain value other than 0. A value of 1 is 1 milliseconds which means a value of 1000 represents 1 second. The value of the timer interval will determine the speed on an animation.

In the following example, I use a very simple technique to show animation by using the properties `Visible=False` and `Visible=true` to show and hide two images alternately. When you click on the program, you should see the following animation.



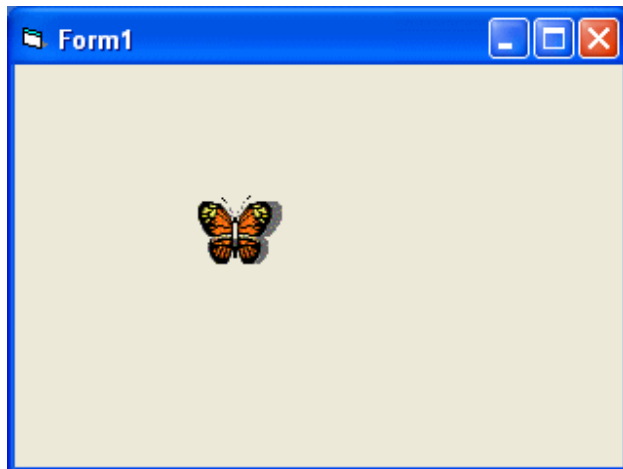
The Code

```
Private Sub  
Timer1_Timer()  
  
If Image1.Visible  
= True Then  
Image1.Visible =  
False  
Image2.Visible =  
True  
ElseIf  
Image2.Visible =  
True Then  
Image2.Visible =  
False  
Image1.Visible =  
True  
End If  
  
End Sub
```

Next example shows a complete cycle of a motion such as the butterfly flapping its wing. Previous examples show only manual animation while this example will display an automatic animation once you start the program or by clicking a command button. Similar to the example under lesson 24.2, you need to insert a group of eight images of a butterfly flapping its wings at different stages. Next, insert a timer into the form and set the interval to 10 or any value you like. Remember to make image1 visible while other images invisible at start-up. Finally, insert a command button, rename its caption as `Animate` and key in the following statements by double clicking on this button. Bear in mind that you should enter the statements for hiding

VISUAL BASIC

and showing the images under the timer1_timer subroutine otherwise the animation would work. Clicking on the animate button make timer start ticking and the event will run after every interval of 10 milliseconds or whatever interval you have set at design time. In future lesson, I will show you how to adjust the interval at runtime by using a slider bar or a scroll bar. When you run the program, you should see the following animation:



```
Private Sub Form_Load()  
Image1.Visible = True  
x = 0  
End Sub  
  
Private Sub Command1_Click()  
Timer1.Enabled = True  
End Sub  
  
Private Sub Timer1_Timer()  
If Image1.Visible = True Then  
Image1.Visible = False  
Image2.Visible = True  
  
ElseIf Image2.Visible = True Then  
Image2.Visible = False  
Image3.Visible = True  
  
ElseIf Image3.Visible = True Then  
Image3.Visible = False  
Image4.Visible = True  
ElseIf Image4.Visible = True Then  
Image4.Visible = False  
Image5.Visible = True  
ElseIf Image5.Visible = True Then  
Image5.Visible = False  
Image6.Visible = True  
ElseIf Image6.Visible = True Then
```

VISUAL BASIC

```
Image6.Visible = False
Image7.Visible = True
ElseIf Image7.Visible = True Then
Image7.Visible = False
Image8.Visible = True
ElseIf Image8.Visible = True Then
Image8.Visible = False
Image1.Visible = True
End If
End Sub
```