Chapter 1

Introduction

The Visual Basic programming language was developed by Alan Cooper, an American computer scientist in the late 1980's. This programming language was created with the main purpose of teaching programmers and developers how to design and develop graphical-user interface (GUI) programs easily. The reason behind the GUI name is that we work more with graphics, forms, and icons, than typing text to accomplish a particular task. In GUI approach, rather than trying to remember a hard to remember command and its corresponding parameter, we can remember easier by using pictures and images.

With Visual Basic, we can develop computer games and utilities, information systems, computer-aided instruction (CAI), multimedia powered application, and as a front-end business application system for back-end database servers such as SQL Server or ORACLE database server. Not only that, we can even develop an Internet-enabled or Web-based application system using Visual Basic 6. With these reasons, I don't think that we would have a second thought if we really wanted to learn Visual Basic programming. So let us start learning it now!

Common Elements of IDE

The Integrated Development Environment (IDE) is the workspace where we construct together all the components of our application system such as the place where we design our forms and controls as well as the place where we develop our code or programs.

Project Wizard

When we click the Visual Basic menu item under the Microsoft Visual Studio for the first time, the Project Wizard will open. We will be prompted with a New Project dialog box, where we can select several types of **projects** that will give us a head start on developing our application system. The New Project dialog box's window has three tabs: New, Existing, and Recent.

When we select the **New** tab, we let the Visual Basic to create the foundation of our application system. The New tab presents us with several project templates such as Standard.EXE, ActiveX.EXE, ActiveX.DLL and much much more. However, we will concentrate on using the Standard.EXE template for our study here in this Visual Basic programming book.

The **Existing** tab allows us to choose an existing project. Usually these existing projects are projects which we have worked on in the past, or in some cases they are sample projects which are included (bundled) in Visual Basic software package. Lastly, the **Recent** tab allows us to choose the most recently used projects. In other words, this tab presents us with a list of the existing projects we have worked on recently, instead of all of the existing projects which are displayed by selecting the Existing tab.

Menu Bar

The menu bar is actually like the menus we have seen in other Windows application software such as MS Word, MS Excel, and PowerPoint. It is the line of text that lies across the top of the Visual Basic window. This menu will give us access to many features within the integrated development environment (IDE). Some of the menu items that we can see at the Menu bar are File, Edit, View, and Project.
In the **File** menu, we will work with the actual files that make our application system. Here we can open, save or print the projects or programs that we have created.
In the **Edit** menu, we can perform the standard Clipboard options such as cut, copy, and paste. In the View menu, we can view various component and tools. We can view a form and a code module, as well as other utilities that help make our program development time more productive, easy, and fast.
In the **Project** menu, we can add or remove forms, code modules, and user controls. The menu bar also contains the Add-Ins menu where we can select an option between Visual Data Manager and Add-In Manager. The Visual Data Manager is a simple and useful tool that allows us to design and populate a database in many popular table format such as Microsoft Access, MS SQL Server, and ORACLE database server, while the Add-In Manager allows us to select other Add-In utilities to be added in the Add-Ins menu.

Toolbars

Below the menu bar are list of icons which are called toolbars. They provide us a quick access to commonly used menu bar to search for a command, we simply click an icon (which is an equivalent to that menu-bar command we need, and presto, it serves our wishes. Plus, we can dock these list of icons beneath the menu bar to drag them away from the menu bar or to make it “float” if we select the vertical bar on the left edge.

Toolbox

The toolbox is also called a control bar in other programming language such as Visual C++ or database management system (DBMS) software such as Visual FoxPro. The toolbox contains the tools or objects (controls) that we might want to place (drag and drop) on a form in our application system on which we presently design and develop. An example of these objects are Command buttons control and Text boxes control.

We can create a customized toolbox layout by selecting the Add Tab from the context menu, then add controls (objects) to the resulting tab.

Context Menus

The context menus contain shortcuts to our frequently performed commands. To open the context menu is easy, we simply click the right mouse button on the control or object we are currently using. The specific list of shortcuts available from context menu depends on the part of the environment where we click the right mouse button.

Project Explorer

The project explorer window is docked on the right side of the screen, right under the toolbar. It provides us the list of the forms and modules of our current project (program). In other words, it serves us as a quick reference to the various elements of our project. A **project** is simply a collection of files we use to create our application system. All of the objects or controls that make up our application system are packaged in a project. You will notice that the project explorer window is much like Windows Explorer, because of how it looks. Plus it allows us to expand and collapse the subfolders, the same way we do with Windows Explorer in our Windows 2000 operating system.

Properties Window

The properties window is docked right under the Project Explorer window. It provides us the lists of the property settings from the currently selected form or other control objects. A **property** in Object-Oriented Programming (OOP) term is a characteristic of an object such as its size, caption, or color. The characteristic is not limited to the appearance of an object (control) but also about the way it behaves.

Object Browser

The object browser lists the objects or controls available for manipulation in our project and gives us an easy way to navigate through our code. We can use it to examine different Visual Basic applications and see what methods, events, and properties available for those objects, then we can paste the code procedures into our application system that we presently develop.

Form Designer

The form designer is our drawing board where we design graphically the layout of the form and controls of our application system. In short, this is where we draw our graphical-user interface (GUI). Here we can add controls, pictures, or other graphics to a form to design the look we want. We have all the powers to make our GUI to look funny or elegant in the eyes of the beholder (our target users). Imagine how powerful we are? It doesn't take a professional artist to do that; but a clear mind and a heart with a good taste of beauty of an art.

Code Editor

The code editor window serves as a program editor where we can write our program (code) behind each form and control (object) we design or where we can place the code module we develop in our application system.

Form Layout Window

The form layout window gives us a thumbnail view of the current form and allows us to position the forms in our application system using a small graphical representation of the screen. The form layout window is useful for determining what screen our form will use when our application system is running.

Program Conventions Used

Here in this book, the code to be embedded in the procedure are written in bold letters. For example:

```
Private Sub Command1_Click()
 Text1.Text = "Hello World!"
End Sub
```

The above code demonstrates that the statement "**Text1.Text = Hello World!"** is the code to be embedded (typed) within the procedure Command1_Click( ). In short, the statements you are instructed to type or to embed within the procedure appears in bold letters.

In most situations, the statements which are not in bold letters are compiler system-generated code, so there is no need to type them. In the above example, they are "Private Sub Command1_Click( )" and "End Sub".

Explaining Properties, Methods, and Events briefly

Properties can be considered as an object's attributes, methods as its actions, and events as its responses. For example, a command button's attributes are its caption, color, and size. An object such as a command button can perform methods and respond to events. This event can be a click by the mouse or a keypress from the keyboard. The performed method could be to show a form object or hide it instead.

Chapter 2

Designing Basic Controls or Objects

"When people ask me what is the best way to learn programming or how to use a specific language, I tell them to experiment - write as many small programs as you can, as each one will teach you more - and each one will add another trick to your tool bag."
- Alan Cooper
(Father of Visual Basic)

Controls or objects are the primary medium of the user's interaction with the application system on which we design and develop. Mostly, we use controls to get the user input and to display the corresponding output. The other controls provide us an access to some application system and process data or information as though the remote application system is part of our program. The basic controls that we can use in our application system are command buttons, text box, label, check box, option button, list box, and combo box. By clicking or typing something on these controls, the user can accomplish their tasks.

Using Command Button and Text box

The function of a command button is to carry out a command or action (event) as the user clicks it, while the text box provides an area to input or display text. We can use the command button to extract simple responses from the user or to invoke special functions on the form. The text boxes are commonly used to display string or numeric data and to accept user input or for entering data.

Example 1:

Design and develop an application system that when the user clicks the Click Me button, the message "Hello World!" will be displayed at the text box. Follow the given figure below in designing and developing the application system.

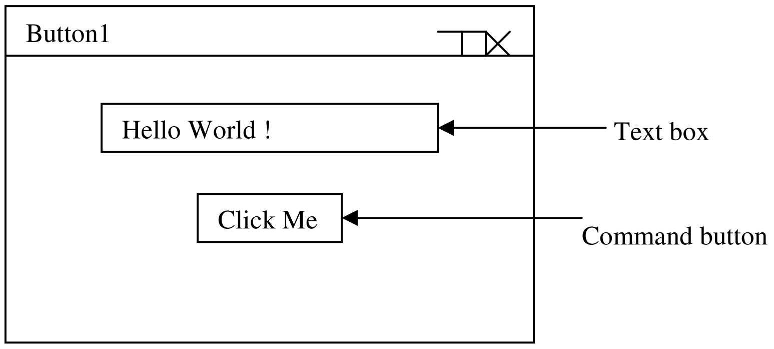


Figure 2.1 Command button and Text box design

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Button1.**
3. Double-click a text box in the Toolbox to add it to the form. Position it properly and adjust its width appropriately. Next, double-click a command button in the Toolbox to add it into the Form, then set its caption to **Click Me**.
4. Double-click the command button on the form to display the Button1 Click event procedure. Embed the following lines to the procedure:

Command1	Click

```
Private Sub Command1_Click()
 Text1.Text = "Hello World!"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Note:
You can also use the Windows Explorer to find the Visual Basic executable file, then double-click the Visual Basic icon. Or create a shortcut to Visual Basic at your desktop, and you can double-click the shortcut to activate the Visual Basic system compiler.

Explanation:

We usually use the command button to get simple responses from the user such as clicking it to do some action . We will notice here that the embedded code (within the procedure) **Text1.Text = "Hello World!"** is simply changing the `Text` property of the object named `Text1` to display "Hello World" at the Text box control. Text box is commonly used for accepting input or for entering data, but in this example it is used to display the data. In the next chapter, we will use text boxes for accepting input data. As of the moment, we will focus on how to output the data on the text box control. The syntax for our example takes the format of *object.property,* where Text1 is the object (control) and Text is the property. We can use this syntax to change property settings for any form or control in response to events that occurs while our application system is running. The specific event here is a click event (generated by clicking the command button).

Using Check Boxes, Option Buttons and Message Box

Check boxes are valid as single controls, however they are not mutually exclusive. Meaning, the user can check as many check boxes as they want, unlike in Option buttons, the user can only select one option at a time. Check boxes are applicable to the situation where you can select a lot of items because you want them all, such as you can order a pizza and choose all the ingredients you want to put on it as long as you can afford to pay. While in option buttons, you can only choose one and only one. The option buttons is applicable to this situation - marrying a girl. In the Christian world, no matter how many girlfriends you have at the same time or in the same place (shame on you brother!), you can only choose one among them to marry.
Now let us go to the function or role of a message box. A Message box (MsgBox) is simply used for displaying simple messages to the user. So that's it! Let us now have an example that deals with these controls (objects) to learn them in action.

Example 2:

Design and develop a simple Check box and Text box application that when the user clicks one of the three check boxes, it will indicate in the text box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the text box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system.

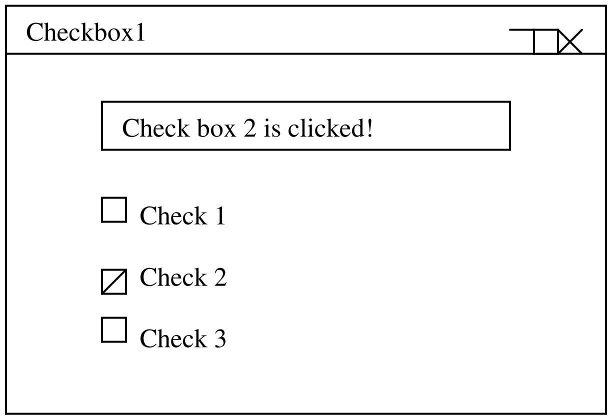


Figure 2.2 Check box and Text box design

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Checkbox1.**
3. Double-click a text box in the Toolbox to add it to the form. Position it properly and adjust its width appropriately. Next, double-click the Check box in the Toolbox to add it into the Form. Then double-click the second check box and position it properly on the Form. Do the same with the third check box.
4. Double-click the Check box 1 on the form to display the Check1 Click event procedure. Embed the following lines to the procedure:

Check1	Click

```
Private Sub Check1_Click()
 Text1.Text = "Check box 1 is clicked!"
End Sub
```

5. Now do the same with Check box 2 and Check box 3. Or you can simply select the Check2 or Check3 objects with their corresponding Click event right at the Code editor window and embed the following code:

```
Private Sub Check2_Click()
 Text1.Text = "Check box 2 is clicked!"
End Sub
```

```
Private Sub Check3_Click()
 Text1.Text = "Check box 3 is clicked!"
End Sub
```

6. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Note:
In running the application program, you can click the Start button on the toolbar or you can simply press F5.

Explanation:

Although a check box control is rather similar to an option button which we will describe in the upcoming examples, there are two fundamental differences between them. You can

select many choices in check boxes, while in option button you are only allowed to select one option at a time. You can observe that when you click the check box 1 (Check 1) then you click the check box two (Check 2), these two check boxes have the check mark on both of them. And even when you click the check box three (Check 3), it will also contain the check mark. In option button, this will not be the case, only one option button will contain the bullet or the big dot at a time, because once you click the next option button, the next option button will contain the bullet as though it transfers from the previous button.
In this example, we will notice here that the embedded code (within the procedure) **Text1.Text = "Check box 1 is clicked!"** is simply changing the Text property of the object named `Text1` to display "Check box 1 is clicked!" at the Text box control. The same thing happen to other check boxes if we click them. The syntax for our example takes the format of *object.property,* where Text1 is the object (control) and Text is the property. We can use this syntax to change property settings for any form or control in response to events that occurs while our application system is running. The specific event here is the click event which is generated by clicking the check boxes.

Example 3:

Design and develop a simple Check box and Message box application that when the user clicks one of the three check boxes, it will indicate in the Message box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the Message box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system.

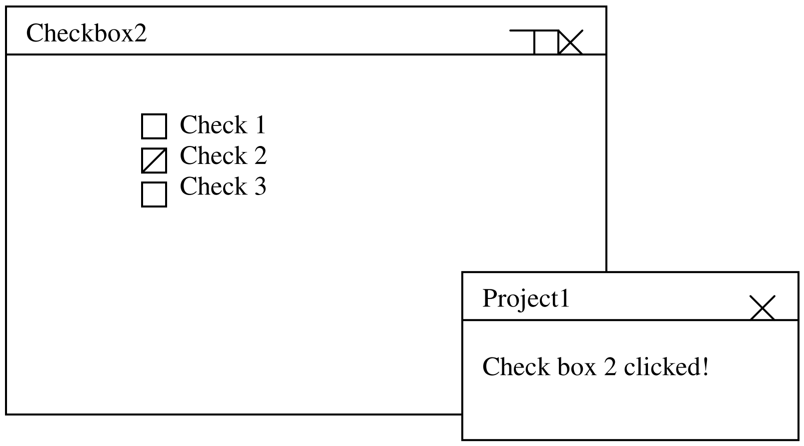


Figure 2.3 Check box and Message box design

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Checkbox2.**
3. Double-click the Check box in the Toolbox to add it into the Form. Then double-click the second check box and position it properly on the Form. Do the same with the third check box.
4. Double-click the Check box 1 on the form to display the Check1 Click event procedure. Embed the following lines to the procedure:

```
Check1          |        Click
```

```
Private Sub Check1_Click()
 MsgBox "Check box 1 is clicked!"
End Sub
```

5. Now do the same with Check box 2 and Check box 3. Or you can simply select the Check2 or Check3 objects with their corresponding Click event right at the Code editor window and embed the following code:

```
Private Sub Check2_Click()
 MsgBox "Check box 2 is clicked!"
End Sub
```

```
Private Sub Check3_Click()
 MsgBox "Check box 3 is clicked!"
End Sub
```

6. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Explanation:

A message box (MsgBox) is used for displaying simple messages to the user.
We will notice here that the embedded code (within the procedure) **MsgBox "Check box 1 is clicked!"** is simply to display the message "Check box 1 is clicked!" at the Message box control. The same thing happened to other check boxes if we click them. The Message box control is a pop-up dialog box that displays the message that we would like to convey to the user.

Example 4:

Design and develop a simple Text box and Option buttons application that when the user clicks one of the three option buttons, it will indicate in the Text box on which option button the user had clicked. For example if option button 2 was clicked by the user, it will display "Option button 2 is clicked!" at the Text box. It will do the same with Option button 1 and Option button 3. Follow the given figure below in designing and developing the application system.

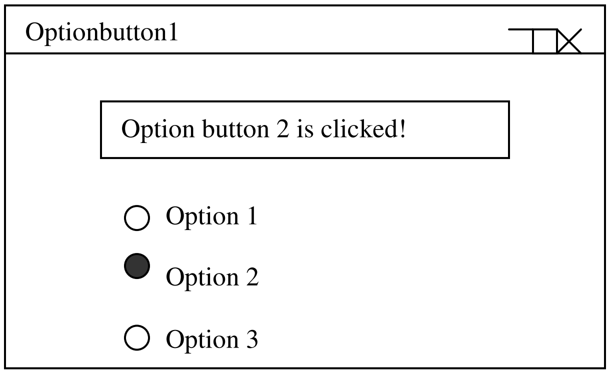


Figure 2.4 Text box and Option buttons design

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Optionbutton1.**
3. Double-click the Text box in the Toolbox to add it into the Form. Then double-click the option button to add it into the form and position it properly. Next, double-click the second option button 2 and position it properly on the Form. Do the same with the third option button 3.
4. Double-click the Option button 1 on the form to display the Option1 Click event procedure. Embed the following lines to the procedure:

```
Option1          Click

Private Sub Option1_Click()
 Text1.Text = "Option button 1 clicked!"
End Sub
```

5. Now do the same with Option button 2 and Option button 3. Or you can simply select the Option2 or Option3 objects with their corresponding Click event right at the Code window and embed the following code:

```
Private Sub Option2_Click()
 Text1.Text = "Option button 2 clicked!"
End Sub
```

```
Private Sub Option3_Click()
 Text1.Text = "Option button 3 clicked!"
End Sub
```

6. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Explanation:

Now you will see the effect of using the option button. As I have said, you can only select one option at a time. Once you click the next option button, the bullet or the big black dot will transfer from the previous button to the currently clicked option button.
We will notice here that the embedded code (within the procedure) **Text1.Text = "Option button 1 clicked!"** is simply changing the `Text` property of the object named `Text1` to display "Option button 1 clicked!" at the Text box control. The same thing happened to other option buttons if we click them. The syntax for our example takes the format of *object.property,* where Text1 is the object (control) and Text is the property. We can use this syntax to change property settings for any form or control in response to events that occur while our application system is running. The specific event here is the click event which is generated by clicking the option buttons.

Example 5:

Design and develop a simple Message box and Option buttons application that when the user clicks one of the three option buttons, it will indicate in the Message box on which option button the user had clicked. For example if option button 2 was clicked by the user, it will display "Option button 2 is clicked!" at the Message box. It will do the same with Option button 1 and Option button 3. Follow the given figure below in designing and developing the application system.

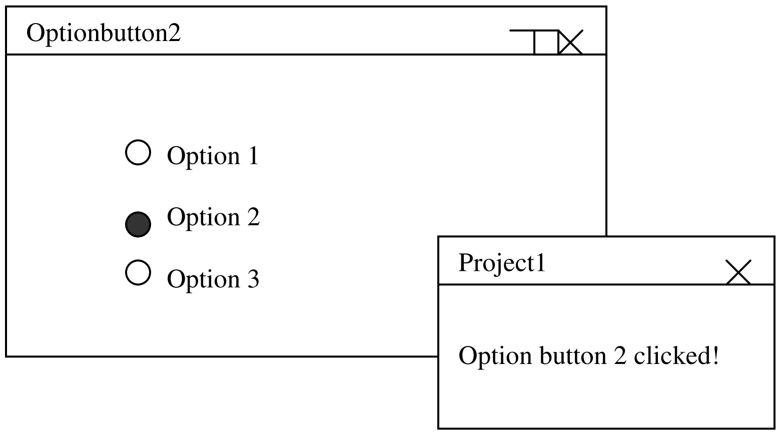


Figure 2.5 Message box and Option buttons design

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Optionbutton2.**
3. Double-click the option button to add it into the form and position it properly. Next, double-click the second option button 2 and position it properly on the Form. Do the same with the third option button 3.
4. Double-click the Option button 1 on the form to display the Option1 Click event procedure. Embed the following lines to the procedure:

```
Option1          Click

Private Sub Option1_Click()
 MsgBox "Option button 1 clicked!"
End Sub
```

5. Now do the same with Option button 2 and Option button 3. Or you can simply select the Option2 or Option3 objects with their corresponding Click event right at the Code window and embed the following code:

```
Private Sub Option2_Click()
 MsgBox "Option button 2 clicked!"
End Sub
```

```
Private Sub Option3_Click()
 MsgBox "Option button 3 clicked!"
End Sub
```

6. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Explanation:

A message box (MsgBox) is used for displaying simple messages to the user. We will notice here that the embedded code (within the procedure) **MsgBox "Option button 1 clicked!"** is simply to display the message "Option button 1 clicked!" at the Message box control. The same thing happened to other option buttons if we click them. The Message box control is a pop-up dialog box that displays the message that we would like to convey to the user.

Event-Driven Programming

In the preceding examples above, we learned how event-driven application system works. An event is an action recognized by a control or form. An event-driven application system executes Visual Basic program in response to an event. For example, if the user clicks a command button (a check box, or option button), the button's Click event procedure is executed. When we want a control such as command button or check box to respond to an event, we write a program in the event procedure for that event. As though we are writing a small program behind it, which is really the case.

Object and Classes Defined

Object is simply a combination of program and data that can be treated as a unit. Like for example, a Form or a Command button is an object. Inside (or behind) a command button is a program and data which we can use to manipulate itself. Its data such as the caption, size, or color can be modified to suit our needs. Now if we want to click it to perform a particular task, we need its Click program to accomplish the task. These program and data are encapsulated within an object, thus eliminates conflict to other code in other object or module. Classes are the blueprint or template of an object. It is just like saying, "If a floor is an object, then its floor-plan is the class. Without the floor-plan, the floor can be impossibly built. The same goes to an object, without the class, it is impossible to create it.
The object in Visual Basic (or even in Visual FoxPro or Visual C++) is defined by a class. A class defines the characteristics of an object - for instance, its size and shape. The class is used to create an object. The controls which are also called objects in the Toolbox represent classes. When we design a control (object) onto the form such as clicking, dragging, and dropping a command button or text box, we are really creating a copy or an

instance of the control class. In this case the control class is the command button class or text box class. All objects (controls) are created as identical copies of their respective classes.

To Comment or Not to Comment (That is the Question!)

Making a comment or putting one or two in your code, makes your program more readable and comprehensible to other programmers or developers. In other words, your program is a programmer-friendly if it contains comment or comments most especially to some part of your program which are hard to understand or decipher. Well, you will learn later on in your programming career that a comment can help you to remember easily what your code is doing once you have not read it for a week or month. This is but just a simple thoughtfulness that can give you and your fellow programmers an easy way to understand programs without resorting to serious and deep code analysis. You will also be surprised later on in your life as a programmer, that you too can hardly understand your own program if it contains no comments on it. Try not doing it (not putting a comment in your program), and you will be sorry after all. Take my word seriously, buddy; I really mean it.

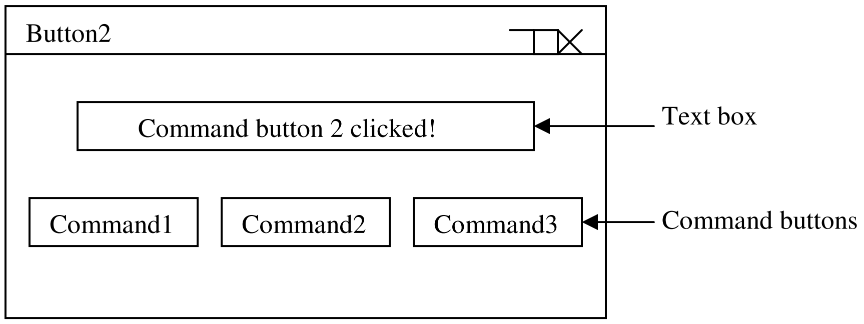
As you read through the examples in this book, you will often come across with the comment symbol (‘) of Visual Basic language (other programming languages have other comment symbols to use). This symbol tells the Visual Basic system-compiler to ignore the words that follow it. These words are remarks placed in the code for the benefit of the other programmer or developer who might examine or use your code later.

A Friendly Reminder for the Instructors

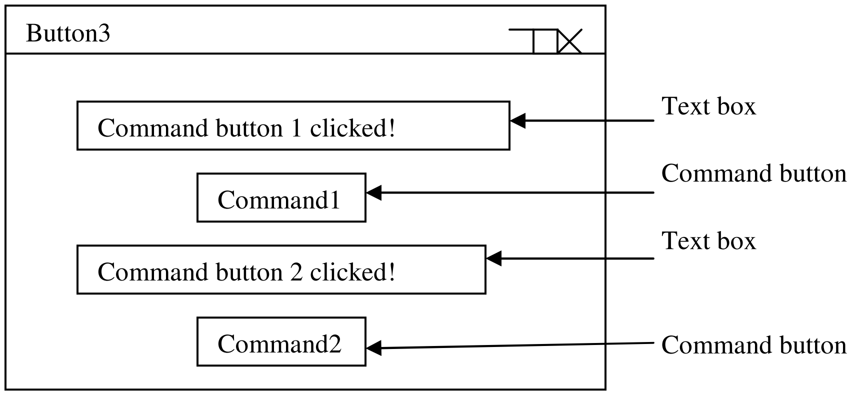
Since programming endeavor is a creative art as well as an analytical science, I would like to suggest that all the laboratory activity tests must be open notes or open books, or even an open Internet research. In this way, the students won’t spend too much time memorizing the commands and functions that they will use in solving the problems presented. The memory capacity is so precious enough to be wasted only for the things that need not to be memorized such as commands and functions. I myself cannot memorize commands and functions with accuracy most of the time. It is my belief that if the students are given enough resources such as notes, books, and Internet connection, they can spend more time in analyzing and solving the problems than cramming their minds with command syntax memorization. In the actual practice of our profession, no boss in his right mind who would instruct his or her programmers and developers to design and develop a real-world application system without the aid of notes, books, manuals or Internet connection. So why not emulate that kind of treatment to our future programmers or developers, the way the real boss do in real-world?

LAB ACTIVITY
TEST 1

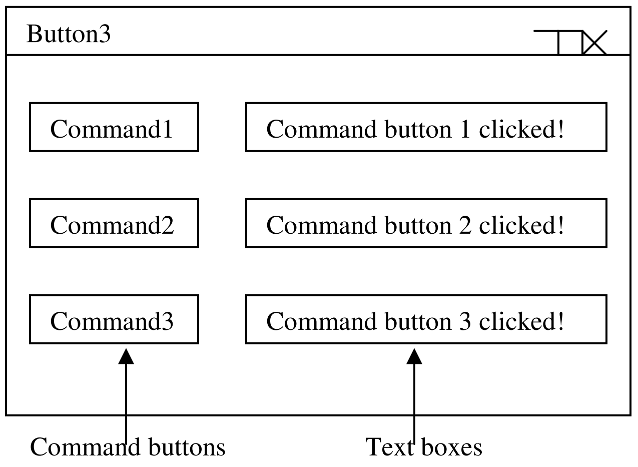
1. Design and develop an application system that when the user clicks the Command button 1, the message "Command button 1 clicked!" will be displayed at the Text box. When the user clicks the Command button 2, the message "Command button 2 clicked!" will be displayed at the Text box, and "Command button 3 clicked!" will be displayed when the user clicks the Command button 3. Follow the given figure below in designing and developing the application system.



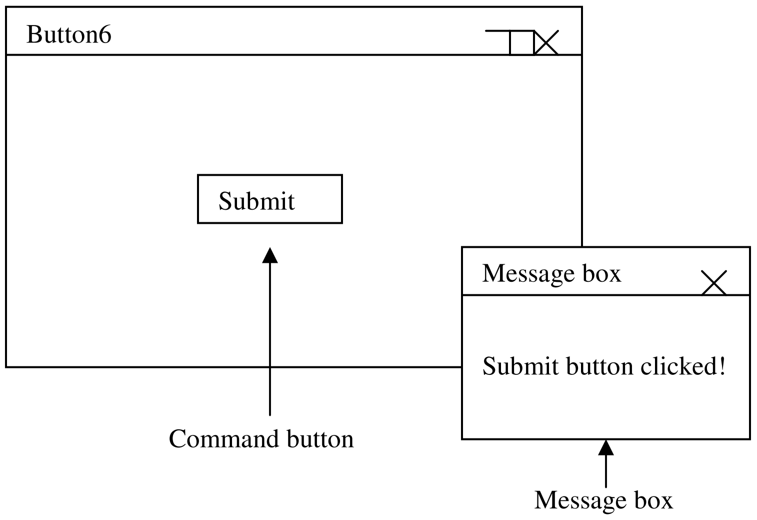
2. Design and develop an application system that when the user clicks the Command button 1, the message "Command button 1 clicked!" will be displayed at the Text box 1. When the user clicks the Command button 2, the message "Command button 2 clicked!" will be displayed at the Text box 2. Follow the given figure below in designing and developing the application system.



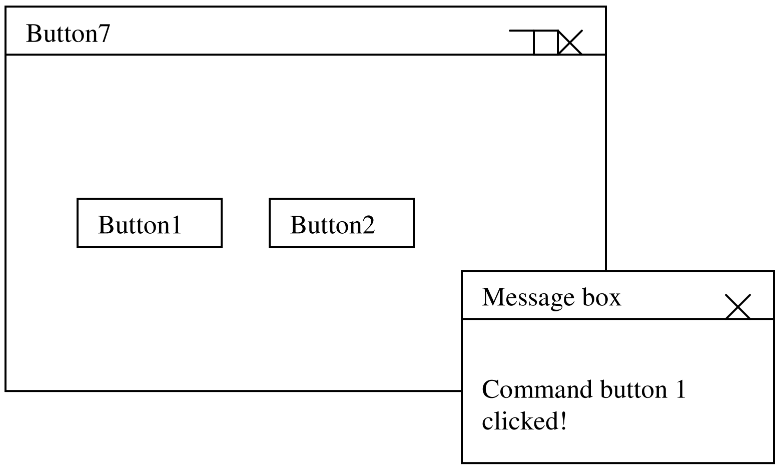
3. Design and develop an application system that when the user clicks the Command button 1, the message "Command button 1 clicked!" will be displayed at the Text box 1. When the user clicks the Command button 2, the message "Command button 2 clicked!" will be displayed at the Text box 2, and "Command button 3 clicked!" will be displayed at Text box 3 when the user clicks the Command button 3. Follow the given figure below in designing and developing the application system.


Button3
Command1
Command button 1 clicked!
Command2
Command button 2 clicked!
Command3
Command button 3 clicked!
Command buttons
Text boxes

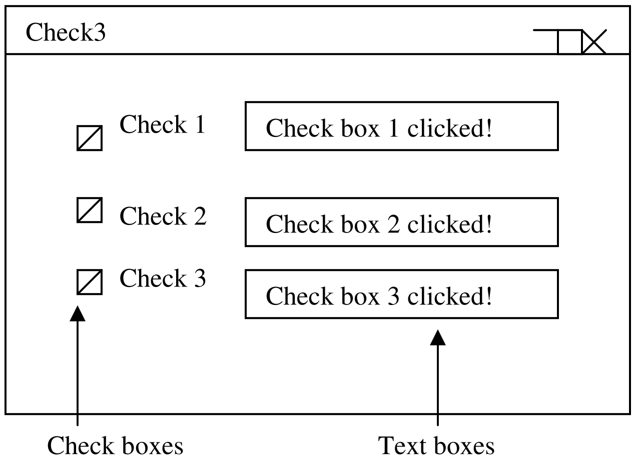
4. Design and develop an application system that when the user clicks the Command button with the caption “Submit”, the message “Submit button 1 clicked!” will be displayed at the Message box . Follow the given figure below in designing and developing the application system.



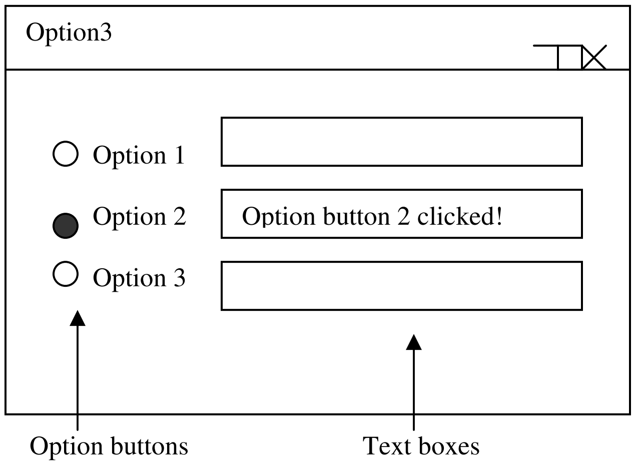
5.Design and develop an application system that when the user clicks the Command button 1, the message “Command button 1 clicked!” will be displayed at the Message box. When the user clicks the Command button 2, the message “Command button 2 clicked!” will be displayed at the Message box. Follow the given figure below in designing and developing the application system.



6. Design and develop an application system that when the user clicks the Check box 1, the message "Check box 1 clicked!" will be displayed at the Text box 1. When the user clicks the Check box 2, the message "Check box 2 clicked!" will be displayed at the Text box 2 and when the user clicks the Check box 3, the message "Check box 3 clicked!" will be displayed at the Text box 3. Follow the given figure below in designing and developing the application system.

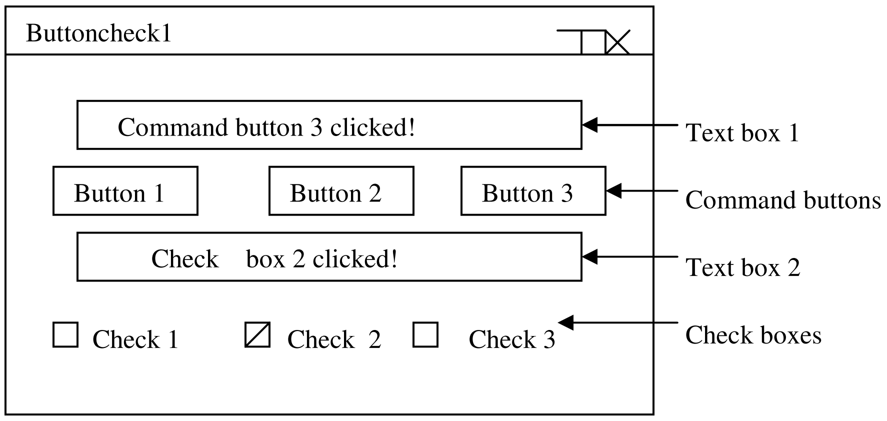


8. Design and develop an application system that when the user clicks the Option button 2, the message "Option button 2 clicked!" will be displayed at the text box 2. When the user clicks the Option button 1, the message "Option button 1 clicked!" will be displayed at the text box 1 and when the user clicks the Option button 3, the message "Option button 3 clicked!" will be displayed at the text box. Follow the given figure below in designing and developing the application system.



9. Design and develop an application system that when the user clicks the Command button 1, the message "Command button 1 clicked!" will be displayed at the text box 1. When the user clicks the Command button 2, the message "Command button 2 clicked!" will be displayed at the text box 1 and when the user clicks the Command button 3, the message "Command button 3 clicked!" will be displayed at the text box 1.

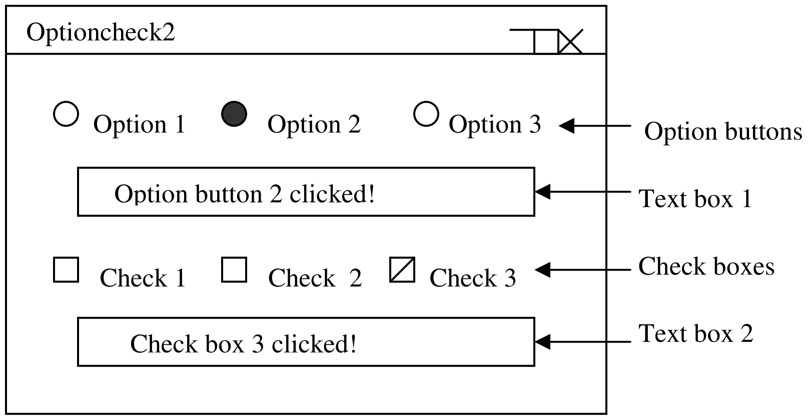
When the user clicks the Check box 1, the message "Check box 1 clicked!" will be displayed at the text box 2. When the user clicks the Check box 2, the message "Check box 2 clicked!" will be displayed at text box 2, and when the user clicks the Check box 3, the message "Check box 3 clicked!" will be displayed at the text box 2. Follow the given figure below in designing and developing the application system.



10. Design and develop an application system that when the user clicks the Option 1, the message "Option button 1 clicked!" will be displayed at the text box 1. When the user clicks the Option button 2, the message "Option button 2 clicked!" will be displayed at the text box 1 and when the user clicks the

Option button 3, the message "Option button 3 clicked!" will be displayed at the text box 1.

When the user clicks the Check box 2, the message "Check box 2 clicked!" is displayed at the text box 2. When the user clicks the Check box 1, the message "Check box 1 clicked!" will be displayed at the text box 2 and when the user clicks the Check box 3, the message "Check box 3 clicked!" will be displayed at the Text box 2. Follow the given figure below in designing and developing the application system.



Chapter 3

Using Controls with Input/Output Functions

The Legend of Val( ) and Str( ) Functions

In calculating and displaying data in our Form, we need to know how they can be manipulated using the two most important functions: Value function (Val( )) and String function (Str( )). These two functions simply convert our data from numeric to string data or vise versa. The Val( ) function converts the displayed data into numeric, while the Str( ) function converts the displayed data into string data.
The Val( ) function is applied when we want to calculate the numeric data displayed in text boxes. Now if we want to display the computed data into the text box, we need to convert it into a string data using the Str( ) function.
We need to apply the Val( ) function so that the numbers that are being entered in the text box can be converted to numeric data for computation purposes in our equation or formula, otherwise these numbers are treated as string which cannot be calculated or processed in the equation. We need also to convert the numeric data into its equivalent string data representation using the Str( ) function for the proper display of the numbers into the text box control. Without doing so, it will result to an undesired output.
Okay, let us have our example about this one. The best example of this application is the simple calculator. In our simple calculator, we will design a program where a user can input two numbers then our program will summed them up. Let us start with a program that will add two numbers.

Arithmetic Operators

+	Addition
_	Subtraction
*	Multiplication
/	Division
^	Exponentiation

The above arithmetic operators are handy for our next examples on this chapter. So remember their symbol so that we can easily understand the succeeding examples and successfully solve the upcoming Lab Activity Test. Are you ready now? Yes, you are!

Example 1:

Design and develop a simple application program that calculates the sum of two input numbers. Follow the given figure below in designing and developing the application program:

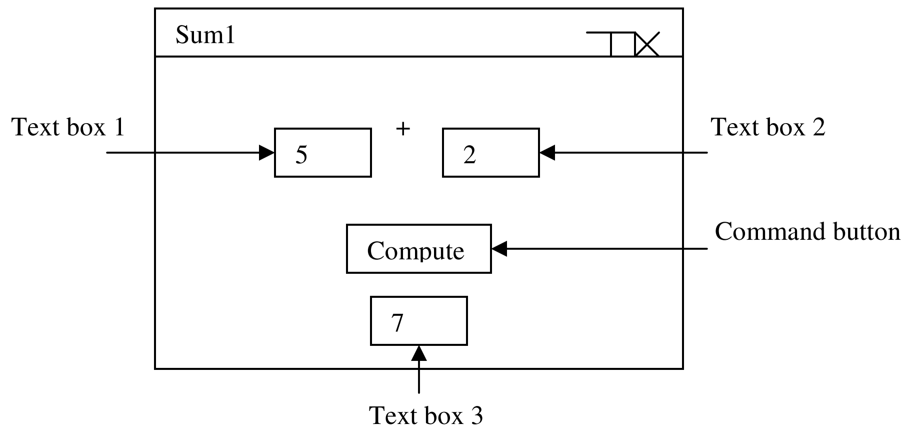


Figure 3.1 Sum of two input numbers

Note:

When the user enters two numbers in two text boxes, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Sum1.**
3. Double-click the two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click a Label in the Toolbox and put it as a plus symbol (+) between the two text boxes that you have already designed. Next, double-click a command button in the Toolbox to add it into the Form, then set its caption to **Compute**. Finally, double-click the third text box and add it into the form.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private txtNum1, txtNum2, txtSum As Integer
```

```
Private Sub Command1_Click()
 txtSum = txtNum1 + txtNum2
 Text3.Text = Str(txtSum)
```

```
End Sub
```

```
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtSum = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
 Text3.Text = nSum
End Sub
```

```
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Try to enter any numbers at the two text boxes. Then click the Compute button to see the result. Is the result okay? I hope so.

Explanation:

First, we have to embed the following code at the **(General)** and (**Declarations**) sections in our program. Select now the General item at the Object listbox and the Declaration item at the Procedure listbox.

(General)	*(Declarations)*

Private txtNum1, txtNum2, txtSum As Integer

The procedure listbox for a General section of our module contains a single selection - the Declarations section, where we place module-level variable, a constant, or even the DLL (Dynamic Link Library) declaration.
Here in our example, we declare all our variables as Private because we make it sure that only the procedures within the current module can see them. Other procedures in other modules cannot see them, therefore these private variables cannot affect nor interfere the operation of other outside modules.
In our procedure Private Sub Form_Load( ), we initialize the variables txtNum1, txtNum2, and txtSum with a value of zero(0) as what we can see in the following code:

```
     Private Sub Form_Load( )
──►  txtNum1 = 0
──►  txtNum2 = 0
──►  txtSum = 0
      Text1.Text = nNum1
      Text2.Text = nNum2
      Text3.Text = nSum
     End Sub
```

Our purpose for this initialization of values for these variables is to ensure that the moment the form is loaded, they contain a zero value so that the resulting calculation in our equation is correct. Forgetting to do so, will result to undesirable output.

You will notice also that we put the value of the variable nNum1 to Text1.Text as well as the value of nNum2 to Text2.Text and the value of nSum to Text3.Text as we can see in the following code:

```
Private Sub Form_Load( )
 txtNum1 = 0
 txtNum2 = 0
 txtSum = 0
 Text1.Text = nNum1 ◄──
 Text2.Text = nNum2 ◄──
 Text3.Text = nSum   ◄──
End Sub
```

The implicit declarations of variable nNum1, nNum2, and nSum are intentional in our part, because we want them to produce a blank text box. The "implicit" means we didn't formally declare a variable at the top of our program before using it. This is acceptable in Visual Basic, Visual FoxPro or Visual C++ but not in older programming languages. You have to take note that this one is not a recommended practice nor a good programming style. We just did this to force our text box to become blank or empty when we run our program.

In this code:

```
Private Sub Command1_Click( )
 txtSum = txtNum1 + txtNum2
 Text3.Text = Str(txtSum)
End Sub
```

we formulate the equation txtSum = txtNum1 + txtNum2 to sum up the two input numbers in text box 1 and text box 2 respectively. Then we convert the computed value that is stored in txtSum variable into a string value before we display it at the text box 3 with this code:

```
Text3.Text = Str(txtSum)
```

This is needed to display the number properly. Forgetting to do so will result to undesirable output. You will notice also that we convert the input numbers at text box 1 and text box 2 into its equivalent numeric value using the Val( ) function to properly calculate them in our equation. We can see the conversion syntax on the following code:

```
Private Sub Text1_Change( )
  txtNum1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change( )
  txtNum2 = Val(Text2.Text)
End Sub
```

When we input the numbers in our text box, they are treated as string data. Therefore, converting them to numeric data are needed, otherwise they cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.

Example 2:

Design and develop a simple application system that computes the area of a circle. Use the formula: A = πr^2 , where Pi (π) is approximately equivalent to 3.1416. Follow the given figure below in designing and developing the application system.

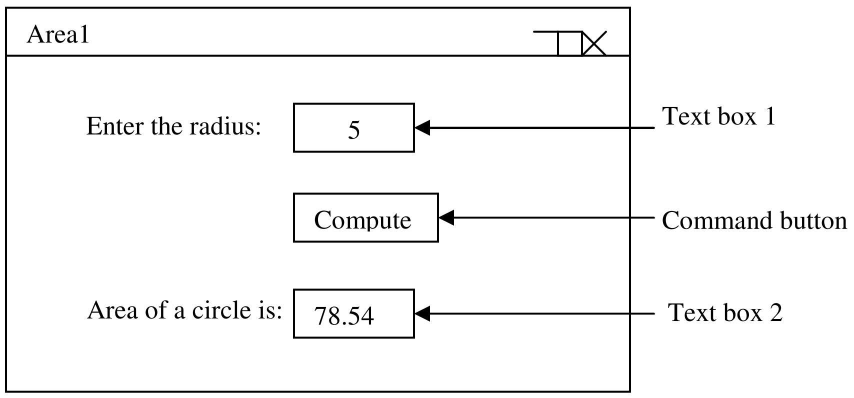


Figure 3.2 Compute the area of a circle

Note:

When the user enters the value of the radius in text box 1, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Area1.**
3. Double-click the two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click two Labels in the Toolbox and write the caption "Enter the radius:" and "Area of a circle is:" in-front of its respective text boxes. Finally, click a command button in the Toolbox to add it into the Form, then set its caption to **Compute**.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private txtRadius As Integer
Private txtArea As Double
Const conPi = 3.1416
________________________________________________
Private Sub Command1_Click()
 txtArea = conPi * txtRadius * txtRadius
 Text2.Text = Str(txtArea)
End Sub
________________________________________________
Private Sub Form_Load()
 txtRadius = 0
 txtArea = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
End Sub
________________________________________________
Private Sub Text1_Change()
 txtRadius = Val(Text1.Text)
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter any value at the text box of the radius. Then click the Compute button to see the result. Is the result okay?

Explanation:

First, we have to embed the following code at the **(General)** and (**Declarations**) sections in our program. Select now the General item at the Object listbox and the Declaration item at the Procedure listbox.

```
    (General)          |        (Declarations)

Private txtRadius As Integer
Private txtArea As Double
Const conPi = 3.1416
```

The procedure listbox for a General section of our module contains a single selection - the Declarations section, where we place module-level variable, a constant, or even the DLL (Dynamic Link Library) declaration.
Here in our example, we declare all our variables as Private because we make it sure that only the procedures within the current module can see them. Other procedures in other modules cannot see them, therefore these private variables cannot affect nor interfere the operation of other outside modules.
In this example, we declare a constant name **conPi** . This is how we declare a constant value in Visual Basic (which is similar to Pascal language, but without the semicolon (;) at the end of the statement.) We could noticed also that the variable txtArea is declared as Double data type. Variable txtArea will eventually contain a computed value with decimal point upon processing the equation. Since a variable would hold a value with a decimal point, it must be declared as Double (an equivalent to Real data type in Pascal language). This is the main reason why we declared variable txtArea as Double data type. This correct variable data type declaration also ensures the accuracy of the calculation of our equation.
In our procedure Private Sub Form_Load( ), we initialize the variables txtRadius, and txtArea with a value of zero(0) as what we can see in the following code:

```
    Private Sub Form_Load( )
→    txtRadius = 0
→    txtArea = 0
     Text1.Text = nNum1
     Text2.Text = nNum2
    End Sub
```

Our purpose for the initialization of values for these variables is to ensure that the moment the form is loaded, they contain a zero value so that the resulting calculation in our equation is correct. Forgetting to do so, will result to undesirable output.
You will notice also that we put the value of the variable nNum1 to Text1.Text as well as the value of nNum2 to Text2.Text as we can see in the following code:

```
    Private Sub Form_Load( )
```

```
 txtRadius = 0
 txtArea = 0
 Text1.Text = nNum1 ←
 Text2.Text = nNum2 ←
End Sub
```

The implicit declarations of variable nNum1 and nNum2 are intentional in our part, because we want them to produce a blank text box. The "implicit" means we didn't formally declare a variable at the top of our program before using it. This is acceptable in Visual Basic, Visual FoxPro or Visual C++ but not in older programming languages. You have to take note that this one is not a recommended practice nor a good programming style. We just did this to force our text box to become blank or empty when we run our program.

In this code:

```
Private Sub Command1_Click( )
 txtArea = conPi * txtRadius * txtRadius
 Text2.Text = Str(txtArea)
End Sub
```

we write the given equation txtArea = conPi * txtRadius * txtRadius (in programming syntax format) to calculate the area of a circle. Then we convert the computed value that is stored in txtArea variable into a string value before we display it at the text box 2 with this code:

```
Text2.Text = Str(txtArea)
```

This is needed to display the value properly. Forgetting to do so, will result to undesirable output. You will notice also that we convert the input value at text box 1 into its equivalent numeric value using the Val( ) function to properly calculate it in our equation. We can see the conversion syntax on the following code:

```
Private Sub Text1_Change( )
 txtRadius = Val(Text1.Text)
End Sub
```

When we input a number in our text box, it is treated as a string data. Therefore, converting it to numeric data is needed, otherwise it cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.

Example 3:

Design and develop a simple application system that computes the average of three input quizzes. Then display the result. Follow the given figure below in designing and developing the application system.

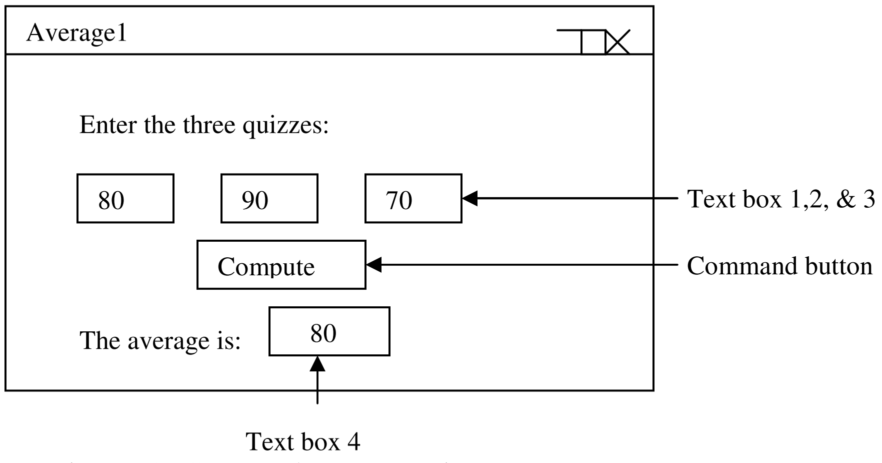


Figure 3.3. Compute the average quiz

Note:

When the user enters the score of three quizzes in text box 1, 2 and 3, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Average1.**
3. Double-click the four text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click two Labels in the Toolbox and write the caption “Enter the three quizzes:” and “The average is:” in-front of its respective text boxes. Finally, click a command button in the Toolbox to add it into the Form, then set its caption to **Compute**.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private txtQuiz1, txtQuiz2, txtQuiz3 As Integer
Private txtAve As Double
```

```
Private Sub Command1_Click()
 txtAve = (txtQuiz1 + txtQuiz2 + txtQuiz3) / 3
 Text4.Text = Str(txtAve)
```

```
End Sub
```

```
Private Sub Form_Load()
  txtQuiz1 = 0
  txtQuiz2 = 0
  txtQuiz3 = 0
  txtAve = 0
  Text1.Text = nNum1
  Text2.Text = nNum2
  Text3.Text = nNum3
  Text4.Text = nAve
End Sub
```

```
Private Sub Text1_Change()
 txtQuiz1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change()
 txtQuiz2 = Val(Text2.Text)
End Sub
```

```
Private Sub Text3_Change()
 txtQuiz3 = Val(Text3.Text)
End Sub
```

---

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter three quizzes score at the text box. Then click the Compute button to see the result. Is the result okay?

Explanation:

First, we have to embed the following code at the **(General)** and (**Declarations**) sections in our program. Select now the General item at the Object listbox and the Declaration item at the Procedure listbox.

(General)	*(Declarations)*

```
→Private txtQuiz1, txtQuiz2, txtQuiz3 As Integer
→Private txtAve As Double ←
```

The procedure listbox for a General section of our module contains a single selection - the Declarations section, where we place module-level variable, a constant, or even the DLL (Dynamic Link Library) declaration.
Here in our example, we declare all our variables as Private because we make it sure that only the procedures within the current module can see them. Other procedures in other modules cannot see them, therefore these private variables cannot affect nor interfere the operation of other outside modules.
Our variable txtAve (for average) was declared as Double data type since there is a division operation within the equation. The syntax rule in Visual Basic and in most programming languages is that any variable which hold the resulting computation of the equation which involves a division operation must be declared as Double data type. The main reason for this is that even if we divide two whole (integer) numbers, there is still a big possibility that it yields a value of type Double (a number with a decimal point). Like for example, if we divide N1/N2 (D=N1/N2) where N1=7 and N2=3, this division operation will yield a value of 2.3334 which is a Double data type. Since variable D will hold the resulting computed value with decimal point, therefore variable D must be declared as Double data type.
This correct variable data type declaration also ensures the accuracy of the calculation of our equation.
In our procedure Private Sub Form_Load( ), we initialize the variables txtQuiz1, txtQuiz2, txtQuiz3 and txtAve with a value of zero(0) as what we can see in the following code:

```
 Private Sub Form_Load( )
→ txtQuiz1 = 0
→ txtQuiz2 = 0
→ txtQuiz3 = 0
→ txtAve = 0
   Text1.Text = nNum1
   Text2.Text = nNum2
   Text3.Text = nNum3
   Text4.Text = nAve
 End Sub
```

Our purpose for this initialization of values for these variables is to ensure that the moment the form is loaded, they contain a zero value so that the resulting calculation in our equation 1 is correct. Forgetting to do so, will result to undesirable output.
You will notice also that we put the value of the variable nNum1 to Text1.Text as well as the value of nNum2 to Text2.Text, nNum3 to Text3.Text and nNum4 to Text4.Text as we can see in the following code:

```
Private Sub Form_Load()
  txtQuiz1 = 0
  txtQuiz2 = 0
  txtQuiz3 = 0
  txtAve = 0
  Text1.Text = nNum1 ←
  Text2.Text = nNum2 ←
  Text3.Text = nNum3 ←
  Text4.Text = nAve    ←
End Sub
```

The implicit declarations of variable nNum1, nNum2, nNum3 and nNum4 are intentional in our part, because we want them to produce a blank text box. The "implicit" means we didn't formally declare a variable at the top of our program before using it. This is acceptable in Visual Basic, Visual FoxPro or Visual C++ but not in older programming languages. You have to take note that this one is not a recommended practice nor a good programming style. We just did this to force our text box to become blank or empty when we run our program.

```
Private Sub Command1_Click( )
   txtAve = (txtQuiz1 + txtQuiz2 + txtQuiz3) / 3
  Text4.Text = Str(txtAve)
End Sub
```

we formulate the equation txtAve = (txtQuiz1+txtQuiz2+txtQuiz3)/3 to get the average of the three inputted quizzes in text box 1, text box 2 and text box 3 respectively. Then we convert the computed value that is stored in txtAve variable into a string value before we display it at the text box 3 with this code:

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

This is needed to display the value properly. Forgetting to do so, will result to undesirable output. You will notice also that we convert the input value at text box 1, text box 2, and text box 3 into their equivalent numeric value using the Val( ) function to properly calculate them in our equation. We can see the conversion syntax on the following code:

```
Private Sub Text1_Change( )
   txtQuiz1 = Val(Text1.Text)
```

```
End Sub
```

```
Private Sub Text2_Change( )
 txtQuiz2 = Val(Text2.Text)
End Sub
```

```
Private Sub Text3_Change( )
 txtQuiz3 = Val(Text3.Text)
End Sub
```

When we input the numbers in our text box, they are treated as string data. Therefore, converting them to numeric data are needed, otherwise they cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.

Example 4:

Design and develop a simple application system that converts the input value of Celsius into its equivalent Fahrenheit degree. Use the formula: F= (9/5) * C + 32. Follow the given figure below in designing and developing the application system.

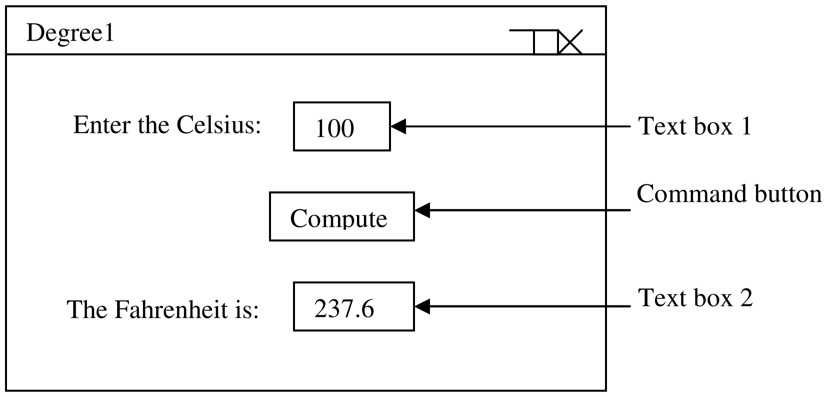


Figure 3.4 Convert Celsius to Fahrenheit degree

Note:

When the user enters the value of the Celsius in text box 1, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. **2.** Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Degree1.**
3. Double-click the two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click two Labels in the Toolbox and write the caption “Enter the Celsius:” and “The Fahrenheit is:” in-front of its respective text boxes. Finally, click a command button in the Toolbox to add it into the Form, then set its caption to **Compute**.
4. **4.** Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private txtCelsius As Integer
Private txtFahrenheit As Double
```

```
Private Sub Command1_Click()
 txtFahrenheit = (9 / 5) * (txtCelsius + 32)
 Text2.Text = Str(txtFahrenheit)
End Sub
```

```
Private Sub Form_Load()
 txtCelsius = 0
 txtFahrenheit = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
End Sub
```

```
Private Sub Text1_Change()
 txtCelsius = Val(Text1.Text)
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter any value at the text box of the Celsius. Then click the Compute button to see the result.

Explanation:

First, we have to embed the following code at the **(General)** and (**Declarations**) sections in our program. Select now the General item at the Object listbox and the Declaration item at the Procedure listbox.

(General)	*(Declarations)*

```
→ Private txtCelsius As Integer
→ Private txtFahrenheit As Double ←
```

The procedure listbox for a General section of our module contains a single selection - the Declarations section, where we place module-level variable, a constant, or even the DLL (Dynamic Link Library) declaration.
Here in our example, we declare all our variables as Private because we make it sure that only the procedures within the current module can see them. Other procedures in other modules cannot see them, therefore these private variables cannot affect nor interfere the operation of other outside modules.
We could notice that the variable txtFahrenheit is declared as Double data type. Variable txtFahrenheit will eventually contain a computed value with decimal point upon processing the equation. Since a variable would hold a value with a decimal point, it must be declared as Double (an equivalent to Real data type in Pascal language). This is the main reason why we declared variable txtFahrenheit as Double data type. This correct variable data type declaration also ensures the accuracy of the calculation of our equation.
In our procedure Private Sub Form_Load( ), we initialize the variables txtCelsius, and txtFahrenheit with a value of zero(0) as what we can see in the following code:

```
  Private Sub Form_Load( )
→ txtCelsius = 0
→ txtFahrenheit = 0
   Text1.Text = nNum1
   Text2.Text = nNum2
  End Sub
```

Our purpose for this initialization of values for these variables is to ensure that the moment the form is loaded, they contain a zero value so that the resulting calculation in our equation is correct. Forgetting to do so, will result to undesirable output.
You will notice also that we put the value of the variable nNum1 to Text1.Text as well as the value of nNum2 to Text2.Text as we can see in the following code:

```
Private Sub Form_Load( )
   txtCelsius = 0
   txtFahrenheit = 0
   Text1.Text = nNum1 ←
  Text2.Text = nNum2 ←
End Sub
```

The implicit declaration of variables nNum1 and nNum2 are intentional in our part, because we want them to produce a blank text box. The “implicit” means we didn’t formally declare a variable at the top of our program before using it. This is acceptable in

Visual Basic, Visual FoxPro or Visual C++ but not in older programming languages. You have to take note that this one is not a recommended practice nor a good programming style. We just did this to force our text box to become blank or empty when we run our program.

In this code:

```
Private Sub Command1_Click( )
 txtFahrenheit = (9/5)*(txtCelsius+32)
 Text2.Text = Str(txtFahrenheit)
End Sub
```

we write the given equation txtFahreheit = (9/5) * (txtCelsius+32) to convert the input Celsius into its equivalent Fahrenheit degree. Then we convert the computed value that is stored in txtFahrenheit variable into a string value before we display it at the text box 2 with this code:

```
Text2.Text = Str(txtFahrenheit)
```

This is needed to display the value properly. Forgetting to do so, will result to undesirable output. You will notice also that we convert the input value at text box 1 into its equivalent numeric value using the Val( ) function to properly calculate it in our equation. We can see the conversion syntax on the following code:

```
Private Sub Text1_Change( )
 txtCelsius = Val(Text1.Text)
End Sub
```

When we input a number in our text box, it is treated as a string data. Therefore, converting it to numeric data is needed, otherwise it cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.

Variable and Constant Declarations

As what we have learned in our previous examples, variables and constants played an important role in our program. Understanding how to use them and how they work is critical to our programming endeavor and crucial to our career as systems programmer and developer. So let us start to learn about them now.
The declaration part in our program consists usually with the list of variables. In some instances, we have declared also the constant names within the program module.

Variables are memory location (address) which we can assign a specific name. For example, we can name our variables as txtSum, txtNum1, txtArea, or txtAve. We can assign or store a value or data into a variable. To declare a variable is to tell the program about it in advance. A variable name must begin with a letter which does not contain an embedded period and must be unique within the same scope (in a form or procedure).

Constants look like a variable, however unlike the variable, the value or data it holds remain fixed while the application system is running or executing. One of the purposes why we use constants in our program is to make coding and debugging easier. By putting the constant value at the declaration part in our program (usually placed at the top of the module), we can easily modify or debug it once it causes some errors. Here are some examples of the constant declarations:

```
Const conPi = 3.1416
Const conInterestRate = 0.03
Const conHoursWork=8
```

Basic Data Types of Visual Basic

Data Type	**Purpose**
Integer	An integer data type is a whole number. This number could be positive or negative and holds values in the range -32,768 to 32, 767 Examples: 9 143 32000 +123 -4567
Long	A bigger integer - means a wider range than an Integer Examples: 100,123 1,000,000 33,000
Single	A number with fractional part or decimal point and holds values up to 3.402823E+38 or 3.4 times 10 to the 38th power Examples: 3.1416 2.54 +193.20 -1,000.003
Double	A number with fractional part or decimal point and holds values up to 1.79769313486232D+308 or 1.8 times 10 to the 308th power Examples: 1143441000.5121 1000000000.151314
String	A sequence of one or more characters that are enclosed within double quotation marks Examples: “a” “Z” “*” “Ma.” “Bianca” “I love you very much, Bianca.”

Boolean A data type that holds True or False values

Example:
```
If ChkButter.Value=1 then
   FrmMain.Butter=True
Else
   FrmMain.Butter=False
EndIf
```

Byte A small integer data type with a range of 0 to 255

Example: 6 254 100 +20 -99

Date For date values data type

Examples:
```
Birthdate = #Sept-12-87#
Birthdate = #Mar 7,  1971#
Birthdate = #2-26-86  13:40#
Birthdate = #17 October 1955#
```

Object Refers to objects within Visual Basic application or other application

Example:
```
Dim objDb  As Object
Set objDb = OpenDatabase("D:\VB98\Biblio.mdb")
```

Currency For currency values data type

Example:
```
Private PerHour  As Currency
```

Variant Capable of storing all system-defined types of data

Example:
```
Dim  Increasing ‘Variant by default
     Increasing = "10"
     Increasing = Increasing + 5
     ‘Variable contains the numeric value 15
     ‘in spite of string to numeric calculation.
```

When your calculation of an equation results to a value bigger than 32,000, use the Long data type for a whole number. Forgetting to do so will result to undesirable output or wrong computation.
In the case of Variant data type, Visual Basic will try its very best to perform an automatic conversion of data (as though it has its own intelligence). But again, this is not a good programming practice nor a good programming style. Specifying a data type for a declared variable is still the best way to accomplish our programming task. Just to keep away from a “hard to find bug” situation caused by a “no data type variable”.

Dim Variable Declaration

Before closing this chapter with the challenging Lab Activity Test, I would like to introduce the most significant variable declaration in Visual Basic programming. This is about the Dim variable. To declare a variable with the Dim statement, we need the following syntax:

Dim *variablename* [As Type]

The variables declared with the Dim statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the value of the variable disappears. Furthermore, the value of a variable in a procedure is *local* to that procedure - meaning we cannot access a variable in one procedure from another procedure. With these characteristic of a local variable, we can use the same variable names in other procedures without causing a conflict or accidental changes to any procedure. To simplify our application to this Dim variable, let us revisit our previous examples in Chapter 2 by converting them with Dim declaration. Let us start it now.

Sample 1:

Design and develop a simple Check box and Text box application that when the user clicks one of the three check boxes, it will indicate in the text box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the text box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system. Apply the Dim variable declaration.

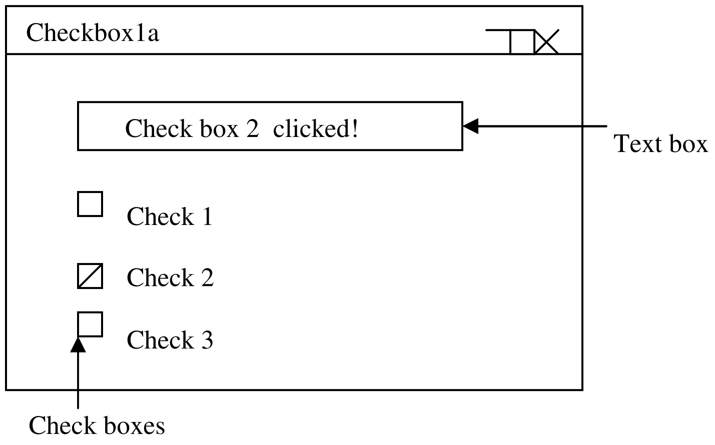


Figure 3.5 Check boxes and Dim Declaration

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Checkbox1a.**
3. Double-click a text box in the Toolbox to add it to the form. Position it properly and adjust its width appropriately. Next, double-click the Check box in the Toolbox to add it into the Form. Then double-click the second check box and position it properly on the Form. Do the same with the third check box.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Check1_Click()
Dim sMessage As String
 sMessage = "Check box 1 is clicked!"
 Text1.Text = sMessage
End Sub
```

```
Private Sub Check2_Click()
Dim sMessage As String
 sMessage = "Check box 2 is clicked!"
 Text1.Text = sMessage
End Sub
```

```
Private Sub Check3_Click()
Dim sMessage As String
 sMessage = "Check box 3 is clicked!"
 Text1.Text = sMessage
End Sub
```

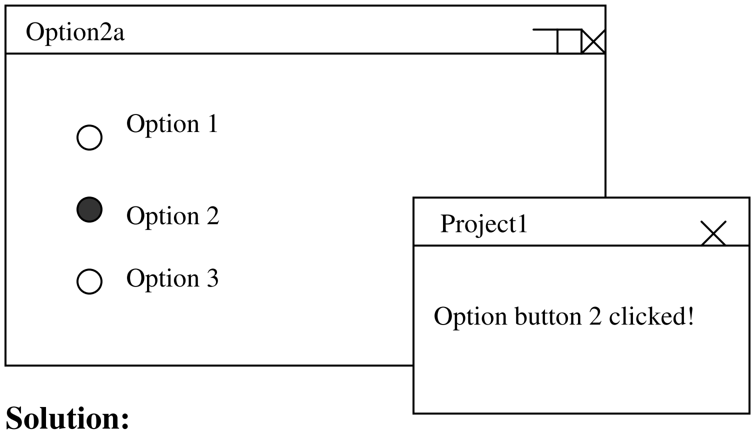
5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Explanation:

You will notice that we locally declared here the sMessage variable as String data type with the Dim statement on each procedure. This is syntactically correct though the sMessage variable was declared similarly in three procedures, because its scope is local only to the procedure where it was declared. So there is no conflict nor accidental changes that happens to the code. This is the beauty of a locally declared variable using the Dim statement.
To learn more, consider the succeeding sample on how to apply the Dim statement.

Sample 2:

Design and develop a simple Message box and Option buttons application that when the user clicks one of the three option buttons, it will indicate in the Message box on which option button the user had clicked. For example if option button 2 was clicked by the user, it will display "Option button 2 is clicked!" at the Message box. It will do the same with Option button 1 and Option button 3. Follow the given figure below in designing and developing the application system. Apply the Dim variable declaration.



Solution:

1.Select and click the Visual Basic 6 under the Microsoft Visual Studio 6 at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the OK (or Open) command button. Set the caption of the Form to **Option2a.**
3. Double-click the option button to add it into the form and position it properly. Next, double-click the second option button 2 and position it properly on the Form. Do the same with the third option button 3.

4.Double-click now the form to display the Code editor. Embed the following code to each respective procedures:

```
Private Sub Option1_Click()
Dim sMessage As String
 sMessage = "Option button 1 clicked!"
 MsgBox sMessage
End Sub
```

```
Private Sub Option2_Click()
Dim sMessage As String
 sMessage = "Option button 2 clicked!"
 MsgBox sMessage
End Sub
```

```
Private Sub Option3_Click()
Dim sMessage As String
 sMessage = "Option button 3 clicked!"
 MsgBox sMessage
End Sub
```

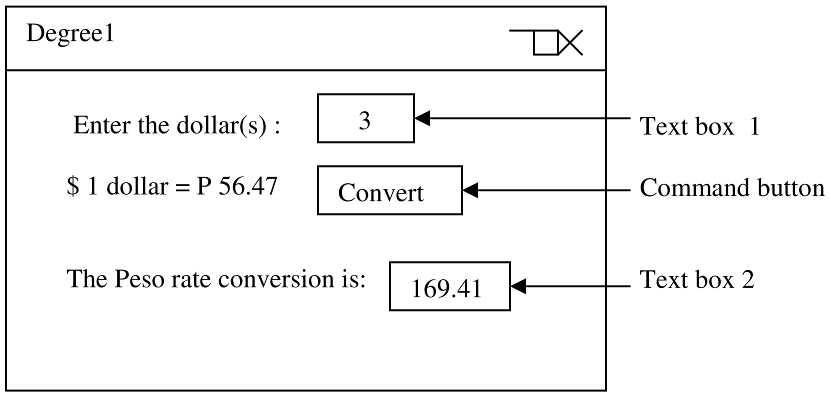
5.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Explanation:

Like in our previous sample application, you will again notice that we locally declared here the sMessage variable as String data type with the Dim statement on each procedure. This is syntactically correct though the sMessage variable was declared similarly in three procedures, because its scope is local only to the procedure where it was declared. So there is no conflict nor accidental changes that happens to the code. This is the beauty of a locally declared variable using the Dim statement.
So this time, you probably understand how this Dim declaration is being used in our application system. What you have learned from these samples is applicable to the succeeding chapters. So keep that knowledge and skill for you will use them later on.

LAB ACTIVITY TEST 3

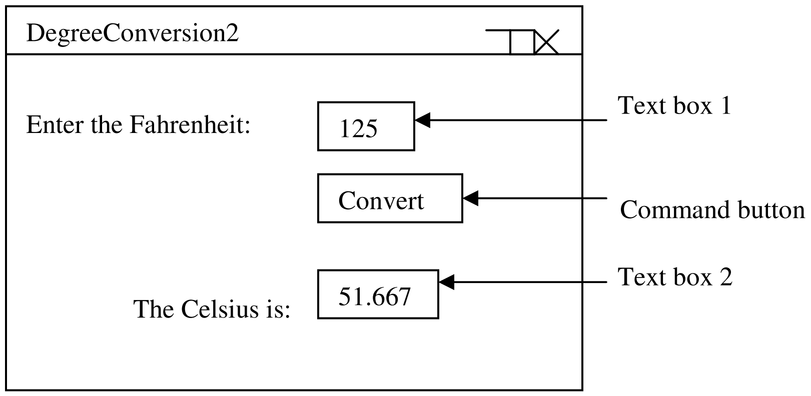
1. Design and develop a simple application system that converts the input dollar(s) into its equivalent Peso rate. Follow the given figure below in designing and developing the application system.



Note:

When the user enters the value of the dollar(s) in text box 1, the user should click the Command button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

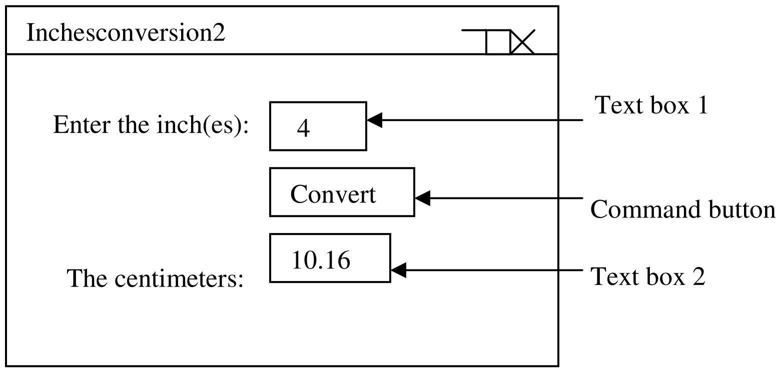
2. Design and develop a simple application system that converts the input Fahrenheit into its equivalent Celsius degree. Use the formula: C = (5/9) * F - 32. Follow the given figure below in designing and developing the application system.



Note:

When the user enters the value of the Fahrenheit in text box 1, the user should click the Command button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

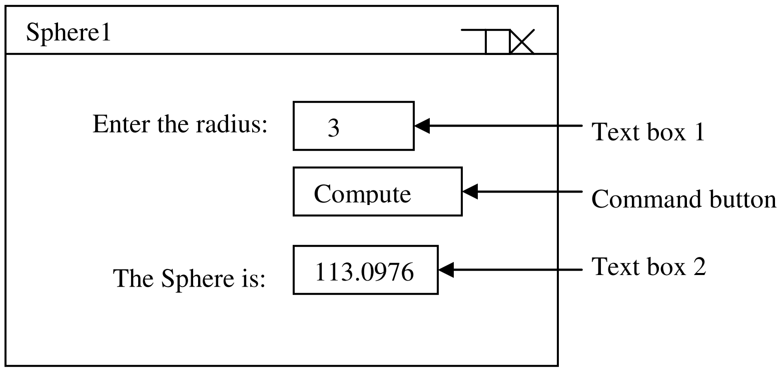
3. Design and develop a simple application system that converts the input inch(es) into its equivalent centimeters. One inch is equivalent to 2.54 cms. Follow the given figure below in designing and developing the application system.



Note:

When the user enters the value of the inch(es) in text box 1, the user should click the Command button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

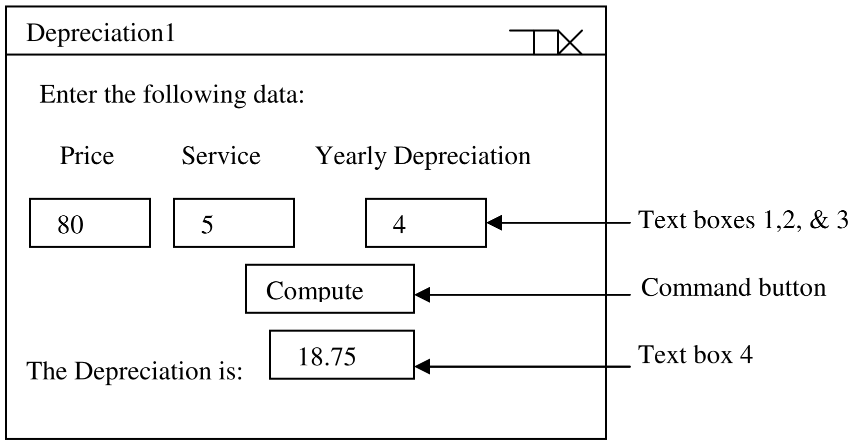
4. Design and develop a simple application system that computes the volume of a sphere. Use the formula: $V = 4/3\ \pi r^3$, where Pi (π) is approximately equivalent to 3.1416. Follow the given figure below in designing and developing the application system.



Note:

When the user enters the value of the radius in text box 1, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

5.Design and develop a simple application system that computes the Depreciation cost of the item (D). Takes as input the purchase Price of an item (P), its expected number of years of Service (S), and Yearly depreciation for the item (Y).
Use the formula: D=P-S /Y. Follow the given figure below in designing and developing the application system.



Note:

After the user enters the purchase Price of an item (P), its expected number of years of Service (S), and Yearly depreciation for the item (Y) at text boxes 1,2 & 3, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

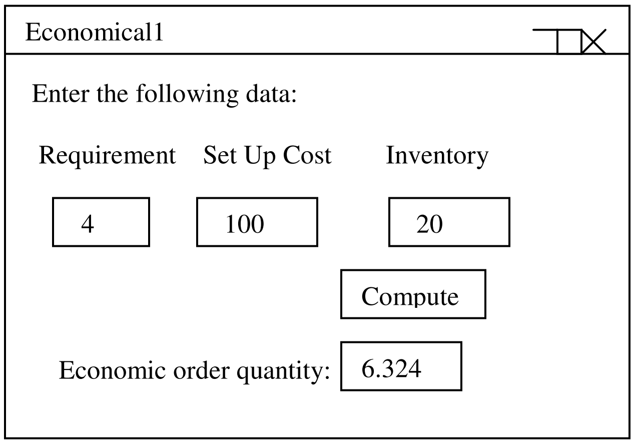
6.Design and develop a simple application system that determines the most economical quantity to be stocked for each product that a manufacturing company has in its inventory. This quantity called *economic order quantity* (EOQ) is calculated as follows:

EOQ = sqrt(2RS/I)

where:

R = total yearly production Requirement
S = Set Up Cost per order
I = Inventory Carrying Cost per unit

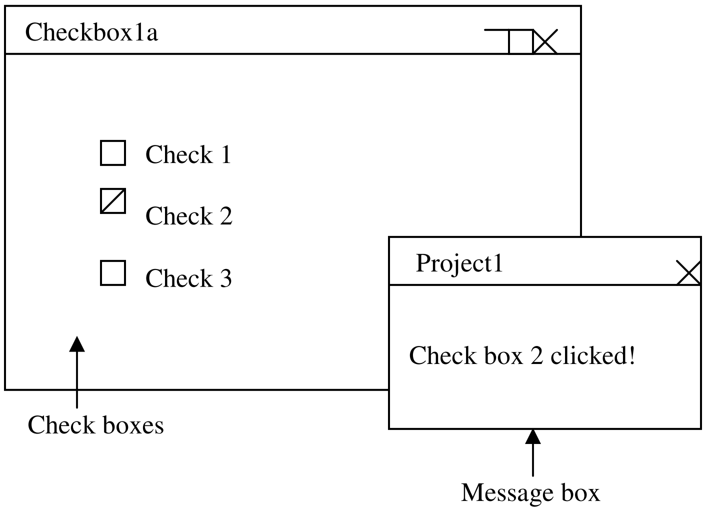
Follow the given figure below in designing and developing the application system.



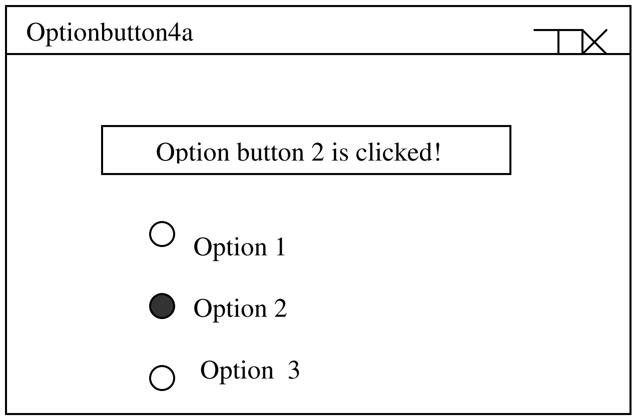
Note:

After the user enters the total year production Requirement (R), its Set up cost per order (S), and Inventory carrying cost per unit (I) at text boxes 1,2 & 3, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

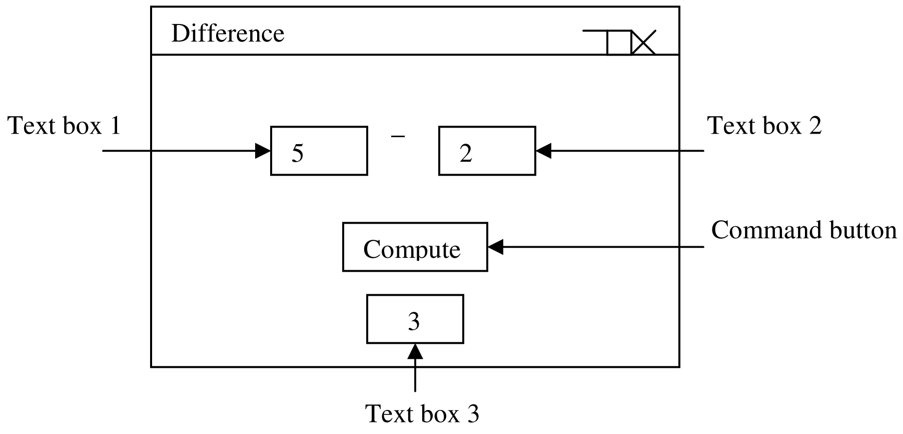
7.Design and develop a simple Check box and Message box application system that when the user clicks one of the three check boxes, it will indicate in the Message box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the Message box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system. Apply the Dim variable declaration.



8. Design and develop a simple Text box and Option buttons application system that when the user clicks one of the three option buttons, it will indicate in the Text box on which option button the user had clicked. For example if option button 2 was clicked by the user, it will display "Option button 2 is clicked!" at the Message box. It will do the same with Option button 1 and Option button 3. Follow the given figure below in designing and developing the application system. Apply the Dim variable declaration.



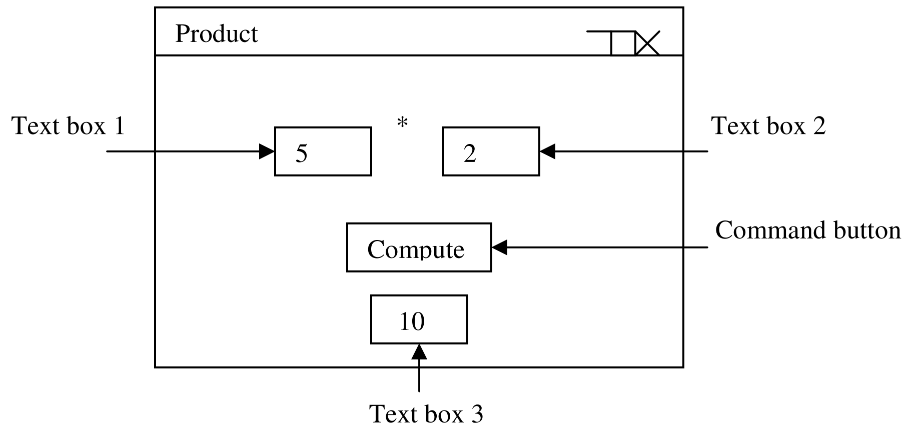
9.Design and develop a simple application system that calculates the difference of two input numbers. Follow the given figure below in designing and developing the application system.



Note:

When the user enters two numbers in two text boxes, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

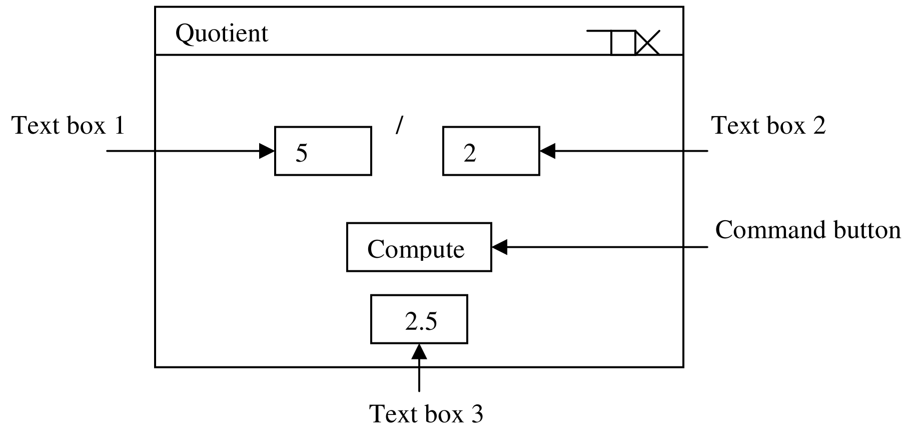
10.Design and develop a simple application system that calculates the product of two input numbers. Follow the given figure below in designing and developing the application system.



Note:

When the user enters two numbers in two text boxes, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

11.Design and develop a simple application system that calculates the quotient of two input numbers. Follow the given figure below in designing and developing the application system.



Note:

When the user enters two numbers in two text boxes, the user should click the Command button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

Chapter 4

Using Controls with Conditional Statements

Computers can make decisions based on a given condition. This condition can be evaluated or tested whether true or false. The computer will act appropriately based on the result of the evaluation. Usually, the conditions are conditional expressions with the use of relational operators and logical operators.

Relational Operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

Logical Operators

Operators	**Rules**
`And`	If both or all expressions are evaluated to True, then the result is True.
`Or`	When either or any (or just one) of the expression(s) is True, the result is True.
`Not`	If the expression is True, then the result is the opposite of the expression.

Logical AND Truth Table

In logical operators, 0 is equivalent to False and 1 is equivalent to True. So the first table below is equivalent to the second table that follows:

X	Y	Z=X * Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	Z = X*Y
False	False	False
False	True	False
True	False	False
True	True	True

Logical OR Truth Table

X	Y	Z=X +Y
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	Z = X + Y
False	False	False
False	True	True
True	False	True
True	True	True

Logical NOT Truth Table

X	Z
0	1
1	0

X	Z
False	True
True	False

In logical Not operator, the value of variable X is simply inverted, so the value of variable Z is the inverted value of variable X.

The Visual Basic language supports the following three conditional statements (decision structures):

- **`If-Then-Else`**
- **`If-Then-ElseIf`**
- **`Select Case - Else`**

If-Then-Else Syntax

The simple if-then-else conditional statement allows the computer to choose one and only one of the given two alternatives. Its general syntax is:

If *condition* **Then**
Statement1
Else
Statement2
End If

If the *condition* is evaluated to True, then *statement1* is executed, otherwise *statemenet2* is executed instead. The condition is usually a comparison using the relational operators and in some cases it can be any expression that evaluates to a numeric value. The Visual Basic system compiler interprets this value as True or False, where a zero (0) numeric value is False, and most of the time a numeric value of one (1) is considered True (or any non-zero value).

If-Then-ElseIf Syntax

The If-Then-ElseIf control construct allows the computer to evaluate three or more conditional statements, but to execute only one associated statement which is the statement of the first conditional expression that is evaluated to True. In other words, use the If-Then-ElseIf control structure to define several blocks of statements, one of which will execute. Its general syntax is:

If *condition1* **Then**
Statement1
ElseIf *condition2* **Then**
Statement2
ElseIf *condition3* **Then**
Statement3
.

```
.
.
Else
   Statementn
End If
```

Visual Basic will evaluate first the condition1. If it is False, the Visual Basic's invisible program pointer will proceed to evaluate condition2, and so on, until it finds a True condition. When True condition is found, only then that its associated statement will be executed. We can also add the Else conditional statement as the option to choose in the If-Then-ElseIf conditional statement. This will provide a way to catch some impossible to meet conditions or the "none of the above" situation. We may use the Else statement for error-checking, sometimes.

Select Case Syntax

The Select Case conditional statement is an alternative to If-Then-ElseIf conditional statement. With the use of Select Case statement, we can simplify some of the code that are tedious, long, and cumbersome when we use the If-Then-ElseIf statement. In other words, the Select Case can be the best substitute conditional statement to write a clearer and concise multi-way decision program compared to the messy If-Then-ElseIf statement. Here is the Select Case general syntax:

```
Select Case var_expression
        Case expression1
                Statement1
        Case expression2
                Statement2
        Case expression3
                Statement3
        Case Else
                Statementn
End Select
```

Each expression is a list of one or more values to be tested or evaluated. If there are more than one value in a list, it must be separated by commas, otherwise a syntax error will occur. Like in the rules of If-Then-ElseIf statement, if more than one Case matches the var_expression, only the statement associated with the first matching Case Select will be executed. Again, like the rules of If-Then-Else If, the Else statement will be executed if none of the values in the above expression list matches the var_expression.
The Else statement is optional to the three conditional statements (*if/else*, *if/elseif/else*, and *select case*). In other words, you can use it or not.
We have to take note also that although the Select Case statement is the best substitute statement for the If/ElseIf statement, it has an inherent limitations. For example, we cannot apply most of the relational operators that are mixed with logical operators with it

such as less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), and not equal (<>) and the logical Or and logical And operators. We cannot say:

Case Is>0 Or Case Is<100

Or

Case Is>0 And Case Is<100

This is a syntax error. Instead, we have to rewrite the code in Select Case format which in many cases look so very different from the If/ElseIf syntax and yet produce the same result. To rewrite the above code, we have the following Select Case syntax:

Case Is >0, Is<100 ‘For the logical Or

Or

Case 0 To 100 ‘For the logical And

See? They are really a complete rewrite. It takes you to learn deeply the Select Case syntax and its limitations and capabilities. Otherwise, you would have a hard time converting an If/ElseIf code into Select Case code. Let us try converting the example below from If/ElseIf conditional statement to Select Case statement code. Are you ready?

Example 1:

Design and develop a simple application program that determines if the input number at Text box 1 is equivalent to the magic words: “I love you” or not. Display the result “I love you!” at the Text box 2 if the input number is 143, and “Wrong guess!” if it is not. Follow the given figure below in designing and developing the application system.

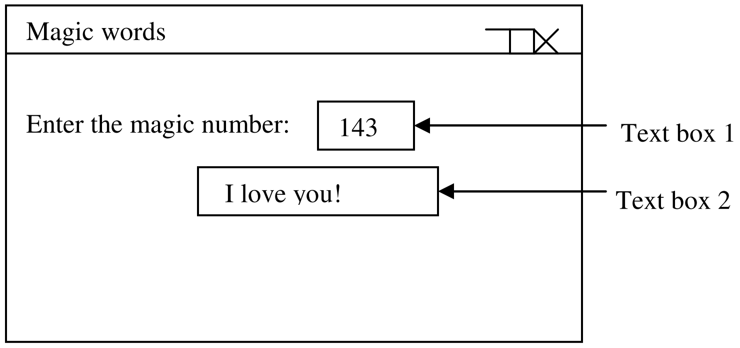


Figure 4.1 Determining a magic number

Solution: (Using the If/ElseIf Conditional Statement)

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.

2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Magic words.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click one Label in the Toolbox and write the caption "Enter the magic number" in front of text box 1.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = sText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change()
If Text1.Text = "143" Then
   Text2.Text = "I love you!"
Else
   Text2.Text = "Wrong guess!"
End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter a number "143" at text box 1 and see what happens. Then try another number. You will see that the output message at text box 2 changes as you enter another number on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

```
 Private Sub Form_Load( )
    Text1.Text = sText1 ←
    Text2.Text = sText2 ←
End Sub
```

Without doing so, our text boxes will contain the default display message "Text1" and "Text2" at text box 1 and text box 2 respectively. You can try it if you want, by not including the above code in your program. Dare?

Our variant variables here are sText1 and sText2. The variant variable is a kind of variable which has no declared data type, and usually they are not formally listed at the top of our program or module (as what happened here in our program).
In our conditional statement:

If Text1.Text = “143” Then

We are evaluating if the conditional expression is true. Meaning, we are trying to know if the user is entering a number 143 at text box 1 (Text1). If it is, then we will display the message “I love you!” at text box 2. Displaying the message can be accomplished through this code:

Text2.Text = “I love you!”

Otherwise if the other numbers are entered, then the message “Wrong guess!” will be displayed at text box 2 (Text2) instead.
One noticeable here in our program syntax is the way the number 143 is formatted. It is formatted as a string data (enclosed in double quotation mark). We did this so that the “Runtime error: Type mismatch” can be prevented. Remember that we are testing the conditional statement with this syntax:

$$\textit{if }\underset{\substack{\uparrow\\\text{object}}}{\textit{Text1}}.\overset{\substack{\text{property}\\\downarrow}}{\textit{Text}} = \textit{“143” Then}$$

therefore the right side of the assignment operator (=) must be a string (text) data, otherwise it mismatches with the Text property of object Text1.
You can try experiencing this “Type mismatch” runtime error by omitting the double quotation mark in the number 143. Do you want to try?

Example 2:

Design and develop a simple application program that determines if the input number is Positive or Negative. Consider 0 as positive number. Follow the given figure below in designing and developing the application system.

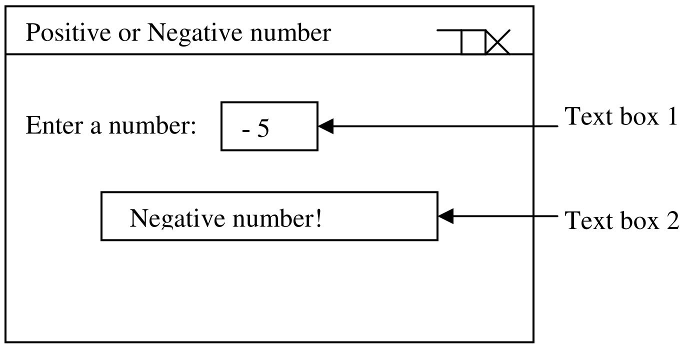


Figure 4.2 Determine if Negative or Positive number

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Positive/Negative number.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click one Label in the Toolbox and write the caption "Enter a number:" in front of text box 1. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = nText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change()
 If Val(Text1.Text)>=0 Then
     Text2.Text = "Positive number!"
 Else
     Text2.Text = "Negative number!"
 End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter a -5 number at text box 1 and see what happens. Then try another number. You will see that the output message at text box 2 changes as you enter another number on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

Private Sub Form_Load( )
Text1.Text = nText1 ←

```
Text2.Text = sText2 ←
End Sub
```

Without doing so, our text boxes will contain the default display message “Text1” and “Text2” at text box 1 and text box 2 respectively. Our variant variables here are nText1 and nText2. The variant variable is a kind of variable which has no declared data type, and usually they are not formally listed at the top of our program or module.
We need to apply the Val( ) function here in our (Text1.Text) expression so that the number entered at text box 1 (Text1) which is a string (text) by default will be converted into its numeric equivalent. In this way, we can evaluate if the number entered is greater than or equal to the number at the right side of the conditional expression such as in the following code:

$$\textit{If } \underbrace{\textit{Val(Text1.Text)>=0}}_{\text{Conditional expression}} \textit{ Then}$$

By the way, how would we know if a number is Positive? Well, a number is positive if it is greater than or equal to zero. Considering 0 as positive since it is an unsigned number. How about a Negative number? Well, it is a number that is not greater than or equal to zero. With this logic in mind, we can simply construct the following code to accomplish the given task:

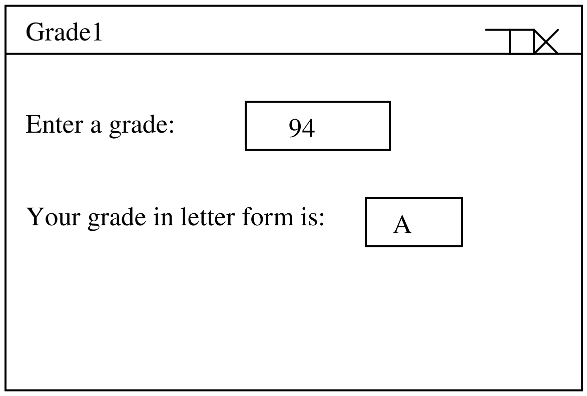
```
If Val(Text1.Text)>=0 Then
    Text2.Text = "Positive number!"
Else
    Text2.Text = "Negative number!"
End If
```

Example 3:

Design and develop a simple application program that will assist a teacher in converting a range of numerical grade into its equivalent letter form grade, based on the given scale:

Range	Grade
90 and above	A
80 - 89	B
70 - 79	C
60 - 69	D
below 60	F

Follow the given figure below in designing and developing the application system.


Grade1

Enter a grade: 94

Your grade in letter form is: A

Figure 4.3 Numeric to Letter form grade conversion

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Grade1.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click two Labels in the Toolbox and write the caption “Enter a grade:” in front of text box 1 and “Your grade in letter form is:” in front of text box 2.

4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = nText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change()
If Val(Text1.Text)>=90 Then
    Text2.Text = "A"
ElseIf Val(Text1.Text)>=80 And Val(Text1.Text)<=89 Then
    Text2.Text = "B"
```

```
ElseIf Val(Text1.Text)>=70 And Val(Text1.Text)<=79 Then
    Text2.Text = "C"
ElseIf Val(Text1.Text)>=60 And Val(Text1.Text)<=69 Then
    Text2.Text = "D"
ElseIf Val(Text1.Text)<60 Then
    Text2.Text = "F"
End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

   Now try to enter a number "94" at text box 1 and see what happens. Then try another number. You will see that the output message at text box 2 changes as you enter another number on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

```
Private Sub Form_Load( )
  Text1.Text = nText1 ←
  Text2.Text = sText2 ←
End Sub
```

Without doing so, our text boxes will contain the default display message "Text1" and "Text2" at text box 1 and text box 2 respectively. Our variant variables here are nText1 and nText2. The variant variable is a kind of variable which has no declared data type, and usually they are not formally listed at the top of our program or module.
We need to apply the Val( ) function here in our (Text1.Text) expression so that the number entered at text box 1 (Text1) which is a string (text) by default will be converted into its numeric equivalent. In this way, we can evaluate if the number entered is greater than, less than, or equal to the number at the right side of the conditional expression such as in the following code:

```
If Val(Text1.Text)>=90 Then
.
.
.
ElseIf Val(Text1.Text)<=60
```

We apply here also the logical And operator so that we can trap the numeric range of the grade entered. In doing so, we can ensure that the number entered falls within a specific range, and our program will respond appropriately based on it. Remember that in logical And, both or all conditional expressions must be evaluated to True so that its associated statement(s) will be executed. Like for example, if we input a numeric grade of 75, the conditional expressions

```
If Val(Text1.Text)>=90 Then
   Text2.Text = "A"
ElseIf Val(Text1.Text)>=80 And Val(Text1.Text)<=89 Then
   Text2.Text = "B"
ElseIf Val(Text1.Text)>=70 And Val(Text1.Text)<=79 Then  ←
   Text2.Text = "C"  ←
ElseIf Val(Text1.Text)>=60 And Val(Text1.Text)<=69 Then
   Text2.Text = "D"
ElseIf Val(Text1.Text)<60 Then
   Text2.Text = "F"
End If
```

were evaluated to True since the number 75 is greater than 70 and less than 79 (so both conditional expressions are tested true). Therefore, the associated statement: Text2.Text = "C" was executed that results to the output on text box 2 (Text2) with a letter "C".
In the If-ElseIf syntax, after finding a True evaluation for a particular conditional statement, the Visual Basic system compiler will jump right to the end of the If statement, thus ending the evaluation process. In other words, the remaining conditions below it will no longer evaluated or tested (or simply ignored). Technically this will save much microprocessor's processing power compared to using all Ifs conditional statements in your program. Meaning, using the If statement all throughout instead of ElseIf statement. With the If statement, all conditional statements will be evaluated, regardless of how many True evaluations found at the upper part of the conditional block. This makes very taxing to the microprocessor's processing energy. In other words, the following code is not advisable to use (since there is only one option to be selected by our Visual Basic system compiler to comply with the given program requirement):

```
  → If Val(Text1.Text)>=90 Then
        Text2.Text = "A"
     End If
 → If Val(Text1.Text)>=80 And Val(Text1.Text)<=89 Then


     Text2.Text = "B"
  End If
→  If Val(Text1.Text)>=70 And Val(Text1.Text)<=79 Then
     Text2.Text = "C"
   End If
→ If Val(Text1.Text)>=60 And Val(Text1.Text)<=69 Then
    Text2.Text = "D"
  End If
→ If Val(Text1.Text)<60 Then
   Text2.Text = "F"
```

End If

Notice all the conditional statements pointed by the arrows. It contained all Ifs. It's bad in programming. There are cases or situations in real-world application that we have no other choice but to use all the Ifs throughout in our sub procedure, however such cases or situations rarely occur. So use all Ifs with utmost care, okay?

Here is the equivalent code in Select Case syntax:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Magic words.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click one Label in the Toolbox and write the caption "Enter the magic number" in front of text box 1.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = nText1
Text2.Text = nText2
End Sub
```

```
Private Sub Text1_Change()
Dim nGrade As Integer
nGrade = Val(Text1.Text)
Select Case nGrade
  Case Is >= 90
     Text2.Text = "A"
  Case 80 To 89
     Text2.Text = "B"
  Case 70 To 79
     Text2.Text = "C"
  Case 60 To 69
     Text2.Text = "D"
  Case Is < 60
     Text2.Text = "F"
End Select
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter a number "143" at text box 1 and see what happens. Then try another number. You will see that the output message at text box 2 changes as you enter another number on text box 1.

Explanation:

You can observe that the Select Case statement equivalent code of our If/ElseIf statement example above are so very different in syntax format. You really need to have a good background of the Select Case statement's limitations and capabilities to be able to convert the If/ElseIf code into Select Case statement's equivalent code successfully. Let us dissect some of the code fragments.

```
Dim nGrade As Integer
nGrade = Val(Text1.Text)
Select Case nGrade
  Case Is >= 90
```

First, we declare a local variable nGrade as integer data type using the Dim statement. Then we apply the Val( ) function to convert the input data from a string (in Text box 1 - Text1) into its numerical value equivalent so that we can evaluate it against the numeric data which is located at the left side of the conditional expression: Is >=90. In this example, it is the numeric 90 value.
The Case Is statement here means if nGrade is greater than or equal to 90.
Now to trap the value between 80 to 89 we have used the following Select Case syntax:

```
Case 80 To 89
```

which is equivalent to If/ElseIf statements :

```
ElseIf Val(Text1.Text)>=80 And Val(Text1.Text)<=89 Then
```

If you try to compare the two syntax implementation, you can really say that they are far different from each other. But what is noticeable is that in Select Case statement, the code is more simple and clearer than the long and cumbersome If/ElseIf statement code. Do you agree?

Example 4:

Design and develop a simple application program that displays an equivalent color once an input letter matches its first character. For example **b** for **Blue**, **r** for **Red**, and so on. Here is the given criteria:

Letters	Colors

‘B’ or ‘b’	Blue
‘R’ or ‘r’	Red
‘G’ or ‘g’	Green
‘Y’ or ‘y’	Yellow
other letters	‘Unknown Color’

Follow the given figure below in designing and developing the application system.

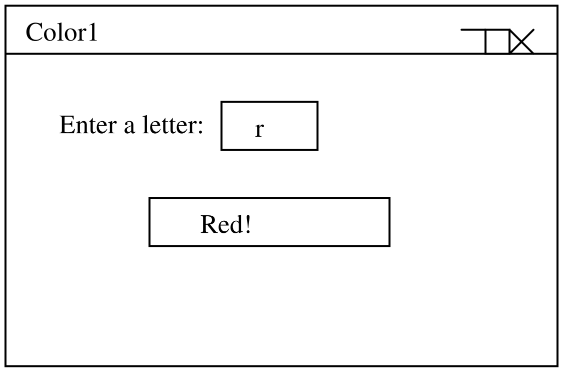


Figure 4.4 Determine a color

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Color1.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click a Label in the Toolbox and write the caption “Enter a letter:” in front of text box 1.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = sText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change()
If Text1.Text = "B" Or Text1.Text = "b" Then
   Text2.Text = "Blue!"
ElseIf Text1.Text = "R" Or Text1.Text = "r" Then
   Text2.Text = "Red!"
ElseIf Text1.Text = "G" Or Text1.Text = "g" Then
   Text2.Text = "Green!"
ElseIf Text1.Text = "Y" Or Text1.Text = "y" Then
```

```
   Text2.Text = "Yellow!"
Else
    Text2.Text = "Unknown Color"
End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Now try to enter a letter "r" at text box 1 and see what happen. Then try another letter. You will see that the output message at text box 2 changes as you enter another letter on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

```
Private Sub Form_Load( )
  Text1.Text = sText1  ←
  Text2.Text = sText2  ←
End Sub
```

Without doing so, our text boxes will contain the default display message "Text1" and "Text2" at text box 1 and text box 2 respectively.
Remember that in logical Or, at least one conditional expression which is evaluated True will trigger the computer to execute the associated statement(s). The logical Or is applied to this kind of program requirement, because the user's input whether uppercase(capital) letter or lowercase(small) letter should be accepted as valid. In programming, the data or value small 'r' is not equivalent to capital 'R'. Now if we input the small letter **r** then the message "Red" will be displayed at text box 2 (Text2), and if we change the small letter **r** to capital letter **R**, the output is still "Red". The same goes to small letter **b** or capital letter **B**. You will notice also that as we change the input data, the default output message at text box 2 is always "Unknown Color". This is because the Visual Basic system compiler will consider a space as a blank data in which results to the evaluation of False to all conditional expressions. That's the reason why the associated statement of the Else conditional statement was always executed every time we change the input data or the letter that we entered at text box 1 (Text1). That's the end of our discussion. Now let us go to the application of Case Select conditional statement. Are you excited for the next topic? I hope so.
In our next example, let us try to use the Select Case conditional statement to the above example instead of the If/ElseIf construct. And here it is!

Example 5:

Design and develop a simple application program that displays an equivalent color once an input letter matches its first character. For example **b** for **Blue**, **r** for **Red**, and so on. This time, use the Select Case conditional statement instead of If/Else conditional statement. Here is the given criteria:

Letters	Colors
‘B’ or ‘b’	Blue
‘R’ or ‘r’	Red
‘G’ or ‘g’	Green
‘Y’ or ‘y’	Yellow
other letters	‘Unknown Color’

Follow the given figure below in designing and developing the application system.

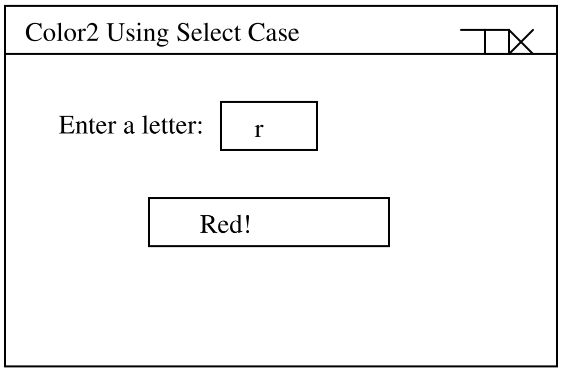


Figure 4.5 Determine a color

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Color2 Using Select Case.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click a Label in the Toolbox and write the caption “Enter a letter:” in front of text box 1.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = sText1
Text2.Text = sText2
```

```
End Sub

Private Sub Text1_Change()
Dim sColor As String
sColor = Text1.Text
Select Case sColor
  Case "B", "b"
    Text2.Text = "Blue!"
  Case "R", "r"
    Text2.Text = "Red!"
  Case "G", "g"
    Text2.Text = "Green!"
  Case "Y", "y"
    Text2.Text = "Yellow!"
  Case Else
     Text2.Text = "Unknown Color!"
End Select
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Now try to enter a letter "r" at text box 1 and see what happens. Then try another letter. You will see that the output message at text box 2 changes as you enter another letter on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

Private Sub Form_Load( )
 Text1.Text = sText1$\leftarrow$
 Text2.Text = sText2$\leftarrow$
End Sub

Without doing so, our text boxes will contain the default display message "Text1" and "Text2" at text box 1 and text box 2 respectively. We declared a local variable sColor as string by using the Dim statement. We need this variable to hold the Text1.Text statement. This will help to simplify our Select Case statement code.
You will notice that instead of using the logical Or operator (to evaluate the expression), we use the comma (,) to separate the expressions in the list. To understand this explanation easily, let us have this comparative presentation:
In the If/ElseIf, we have these statements:

If Text1.Text = "B" Or Text1.Text = "b" Then

```
Text2.Text = "Blue!"
```

while in Select Case, its equivalent statements are:

```
Select Case sColor  ←
   Case "B", "b"   ←
       Text2.Text = "Blue!" ←
```

(where sColor = Text1.Text)

The system compiler will test first the first expression in the list of the Select Case block, then if it is evaluated to False, it will proceed testing the next expression and so on until it will find a True evaluated expression. However, if all listed expressions in the Select Case block are evaluated to False then the associated statement of the optional Else will be executed. In the absence of this optional Else statement, our application system will simply display no output (or it seems our application program is hanging). This is the beauty of putting an Else for this kind of situation to prompt the user about what happened. Informing the user on what happened is always the best defense. Without doing so, the user might misunderstand how the application system reacts or behaves in this type of circumstances. As a result, the user might reboot the computer, which should not be the case, since there is no need to reboot it.

Example 6:

Design and develop a simple application program to display the high school level of a student, based on its year-entry number. For example, the year-entry 1 means the student is a freshman, 2 for sophomore, and so on. Here is the given criteria:

Year-entry number	High-School Level
1	Freshman
2	Sophomore
3	Junior
4	Senior
Other entry no.	Out-Of-School

Follow the given figure below in designing and developing the application system.

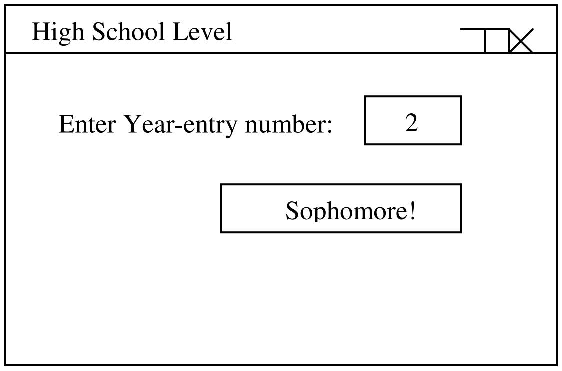


Figure 4.6 Determine the high school level

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **High School Level.**
3. Double-click two text boxes in the Toolbox to add them into the form. Position them properly and adjust their width appropriately. Now double-click a Label in the Toolbox and write the caption "Enter a letter:" in front of text box 1.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
Text1.Text = sText1
Text2.Text = sText2
End Sub

Private Sub Text1_Change()
Dim nYearEntry As Integer
nYearEntry = Val(Text1.Text)
Select Case nYearEntry
  Case 1
    Text2.Text = "Freshman!"
  Case 2
    Text2.Text = "Sophomore!"
  Case 3
    Text2.Text = "Junior!"
  Case 4
    Text2.Text = "Senior!"
  Case Else
```

```
    Text2.Text = "Out-Of-School!"
End Select
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Now try to enter number 2 at text box 1 and see what happens. Then try another number. You will see that the output message at text box 2 changes as you enter another number on text box 1.

Explanation:

To display an empty box, we need to initialize the text box with a variant type of variable. In this example, we can accomplish this through the following code:

```
Private Sub Form_Load( )
  Text1.Text = sText1 ←
  Text2.Text = sText2 ←
End Sub
```

Without doing so, our text boxes will contain the default display message “Text1” and “Text2” at text box 1 and text box 2 respectively. We declared a local variable nYearEntry as integer by using the Dim statement. We need this variable to hold the Text1.Text statement. This will help to simplify our Select Case statement code.
The moment we enter a number at any text boxes, its default data type is a string. To convert the input data to numeric value, we need to use the Val ( ) function. This is the reason why we have the following statement:

NYearEntry = Val(Text1.Text)

With the above statement, the nYearEntry variable will contain the converted data into numeric value. If Case 1 is evaluated to True then its associated statement:

Text2.Text = “Freshman!”

is executed. After that, the system compiler’s invisible program pointer will jump to the End Select statement right away (ignoring all the remaining list of expressions below). If Case 1 is evaluated to False, then the next Case statement is evaluated. The system compiler will test the remaining Case statements until it finds a True evaluated expression. If none is found, then the associated statement of Else conditional statement will be executed. In the absence of the optional Else in our code, the result will simply a “no output” in our program’s text box 2 (Text2). It is recommended that we will always put an Else statement in our code that uses conditional statements in order to catch an impossible input data or to the “none of the above” selection situation.

Remember that we can use the input string data on the text box directly in our code by not using the Val( ) function. However we will declare the sYearEntry variable as String and enclose the numeric expression in the Case statement with double quotes so that it can be read by the compiler system as a string expression. Examine the changes in our code below :

```
Private Sub Form_Load( )
Text1.Text = sText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change( )
Dim sYearEntry As String ←
sYearEntry = Text1.Text
Select Case sYearEntry
  Case "1"   ←
    Text2.Text = "Freshman!"
  Case "2"
    Text2.Text = "Sophomore!"
  Case "3"
    Text2.Text = "Junior!"
  Case "4"
    Text2.Text = "Senior!"
  Case Else
    Text2.Text = "Out-Of-School!"
End Select
End Sub
```

We can also use the *Case Is* statement to accomplish the given task. Like for example, instead of using

```
Case 2
```

we will use

```
Case  Is = 2
```

Let us examine its code below:

```
Private Sub Form_Load( )
Text1.Text = sText1
Text2.Text = sText2
End Sub
```

```
Private Sub Text1_Change()
Dim sYearEntry As Integer
```

```
sYearEntry = Val(Text1.Text)
Select Case sYearEntry
→ Case Is = 1
     Text2.Text = "Freshman!"
→ Case Is = 2
     Text2.Text = "Sophomore!"
→ Case Is = 3
    Text2.Text = "Junior!"
→ Case Is = 4
     Text2.Text = "Senior!"
   Case Else
    Text2.Text = "Out-Of-School!"
 End Select
End Sub
```

We can also rewrite the code using the string declaration equivalent with the Case Is statement. Examine the equivalent code below:

```
Private Sub Form_Load( )
Text1.Text = sText1
Text2.Text = sText2
End Sub

Private Sub Text1_Change( )
Dim sYearEntry As String
sYearEntry = Text1.Text
Select Case sYearEntry
  Case Is = "1"
    Text2.Text = "Freshman!"
  Case Is = "2"
    Text2.Text = "Sophomore!"
  Case Is = "3"
    Text2.Text = "Junior!"
  Case Is = "4"
    Text2.Text = "Senior!"
  Case Else
    Text2.Text = "Out-Of-School!"
End Select
End Sub
```

That’s the end of our discussion, and let us go to other application that will make us more excited!

Using Option Buttons and Check Boxes Together
(It's All Together Now!)

Here in our next example, we will use the Option Buttons together with Check Boxes in a hypothetical application system. Let us learn how to manipulate them together. Well, we will apply other popular controls too, such as the Text box and Command button. Okay, let us see it in action!
By the way, the Label control displays read-only text, and its main function is to provide information to the user. This can be in the form of a message shown on the form or a prompt. A message such as labeling an output data of what it is or telling the user what kind of data to input. This Label application is usually with a conjunction of a text box that holds the input or output data. Like for example that the numeric output in the text box is the Area of a circle or a Fahrenheit degree. Or simply telling the user to enter a numeric data for calculation or a character to process. The label as a prompt is usually combined with a text box, list box, combo box or other control. It provides the user an idea of the nature of the referenced control. For instance, if we had a list box that contains the abbreviated different States in America. We can put a label at the top of list box that tells about the abbreviations.

Example 7:

Design and develop a simple application system that when the user chooses the Deluxe pizza option, the ingredients to be checked are: Cheese, Bacon & Ham, and Onion & Chili; its corresponding price is 185 pesos (to be displayed at text box). When the user chooses the Special pizza option, the ingredients to be checked are: Cheese, Pepper, Bacon & Ham, Mushroom, and Onions & Chili; its price is 250 pesos. Now if the user chooses the Primo pizza option, all ingredients should be marked with a check, and its price is 290 pesos.

Follow the given figure below in designing and developing the application system.

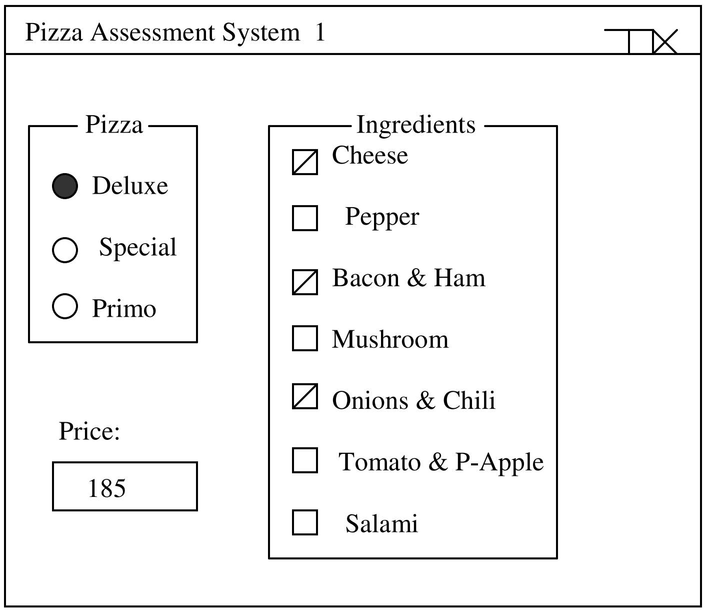


Figure 4.7 Pizza Assessment System Part 1

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Pizza Assessment System 1.**
3. Drag and drop a Frame from a Toolbox and change its caption to "Pizza", then put three Option buttons inside it and name the Option buttons with the captions : Deluxe, Special, and Primo. Drag and drop another Frame with a caption "Ingredients", then put seven check boxes on it and name the Check boxes captions with these: Cheese, Pepper, Bacon & Ham, Mushroom, Onions & Chili, Tomato & P-Apple (for Pineapple), and Salami. Finally, drag and drop a Text box from a Toolbox and place it below the Pizza Frame. Then put a label at the top of it with the word "Price:".
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Option1_Click()
Check1.Value = 1
Check2.Value = 0
Check3.Value = 1
Check4.Value = 0
Check5.Value = 1
Check6.Value = 0
```

```
Check7.Value = 0
Text1.Text = "185"
End Sub
```

```
Private Sub Option2_Click()
Check1.Value = 1
Check2.Value = 1
Check3.Value = 1
Check4.Value = 1
Check5.Value = 1
Check6.Value = 0
Check7.Value = 0
Text1.Text = "250"
End Sub
```

```
Private Sub Option3_Click()
Check1.Value = 1
Check2.Value = 1
Check3.Value = 1
Check4.Value = 1
Check5.Value = 1
Check6.Value = 1
Check7.Value = 1
Text1.Text = "290"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Try to click the Deluxe Pizza option, then the Special Pizza option, and finally the Primo Pizza option, to see how our program works. Is it working? I hope so.

Explanation:

Since we want to put a check mark for the ingredients - Cheese (positioned at Check 1 object), Bacon & Ham (positioned at Check 3 object), Onions & Chili (positioned at Check 5 object) when the user clicks the Deluxe pizza option (positioned at Option 1 object), therefore we have to assign them with a Boolean value of 1 (one). The 0 (zero) Boolean value means the check box will be displayed without a check mark or with unselected status. We can accomplish this task with the following code:

Private Sub Option1_Click( )
Check1.Value = 1
Check2.Value = 0
Check3.Value = 1
Check4.Value = 0

```
Check5.Value = 1
Check6.Value = 0
Check7.Value = 0
Text1.Text = "185"
End Sub
```

Once the Deluxe pizza option is clicked by the user, we will also display the 185 pesos price of Deluxe pizza at the text box 1. We can accomplish the task with this code:

```
Text1.Text = "185"
```

The above explanation applies also (with a slight modifications to fit with the given situation or requirement) to the Special and Primo pizza options. So there is no need for me to explain it in details.

What we have done previously is a simple Assessment system. How about an Assessment system where we can input a quantity of a particular item, then it will output its computation such as its amount. Is it not more flexible? Okay, we will do it now in our next example.

Example 8:

Design and develop a simple application system that when the user chooses the Deluxe pizza option, the ingredients to be marked are Cheese, Bacon & Ham, and Onions & Chili. Its price is P 185 pesos. If the user inputs 2 in the Qty (Quantity) box, the Amt (Amount) to be displayed should be 270, because 185 * 2 is equal to P 270 pesos. If the input quantity is 3 then the amount to be displayed is 455 pesos, and so on.
When the user chooses the Special pizza option, the ingredients to be checked are Cheese, Pepper, Bacon & Ham, Mushroom, and Onions & Chili. Its price is P250 pesos. If the user inputs 2 at the Qty box, the Amt to be displayed is P500 pesos. If the input quantity is 3 then the amount to be displayed is P750 pesos, and so on.
When the user chooses the Primo pizza option, all ingredients should be marked with a check. Its price is P290 pesos. If the user inputs 2 at the quantity box, the amount to be displayed is P380 pesos. If the input quantity is 3 then the amount to be displayed is P870, and so on.
Follow the given figure below in designing and developing the application system.

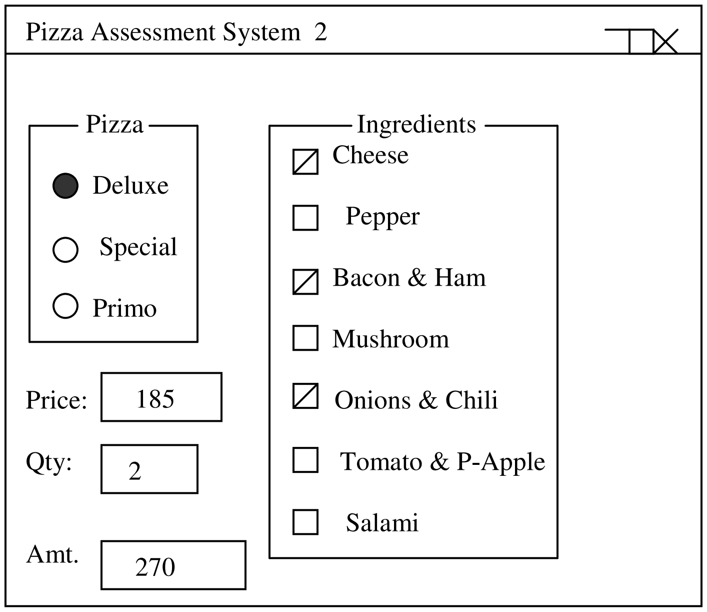


Figure 4.8 Pizza Assessment System Part 2

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Pizza Assessment System 2.**
3. Drag and drop a Frame from a Toolbox and change its caption to "Pizza", then put three Option buttons inside it and name the Option buttons with the captions : Deluxe, Special, and Primo. Drag and drop another Frame with a caption "Ingredients", then put seven check boxes on it and name the Check boxes captions with these: Cheese, Pepper, Bacon & Ham, Mushroom, Onions & Chili, Tomato & P-Apple (for Pineapple), and Salami. Finally, drag and drop three Text boxes from a Toolbox and place them below the Pizza Frame. Then put its corresponding labels in front of them, such as the Price, Qty., and Amt.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private InitialValue As Integer
Const conDeluxePrice = 185, conSpecialPrice = 250,
conPrimoPrice = 290
```

```
Private Sub Form_Load()
Text2.Text = InitialValue
Text3.Text = InitialValue
End Sub
```

```
Private Sub optDeluxe_Click()
Check1.Value = 1
Check2.Value = 0
Check3.Value = 1
Check4.Value = 0
Check5.Value = 1
Check6.Value = 0
Check7.Value = 0
Text1.Text = Str(conDeluxePrice)
Text2.Text = InitialValue
Text3.Text = InitialValue
optDeluxe.Value = True
optSpecial.Value = False
optPrimo.Value = False
End Sub
```

```
Private Sub optSpecial_Click()
Check1.Value = 1
Check2.Value = 1
Check3.Value = 1
Check4.Value = 1
Check5.Value = 1
Check6.Value = 0
Check7.Value = 0
Text1.Text = Str(conSpecialPrice)
Text2.Text = InitialValue
Text3.Text = InitialValue
optDeluxe.Value = False
optSpecial.Value = True
optPrimo.Value = False
End Sub
```

```
Private Sub optPrimo_Click()
Check1.Value = 1
Check2.Value = 1
Check3.Value = 1
Check4.Value = 1
Check5.Value = 1
Check6.Value = 1
Check7.Value = 1
Text1.Text = Str(conPrimoPrice)
Text2.Text = InitialValue
Text3.Text = InitialValue
optDeluxe.Value = False
optSpecial.Value = False
optPrimo.Value = True
```

```
End Sub

Private Sub Text2_KeyPress(KeyAscii As Integer)
Dim txtAmount As Single
txtAmount = 0
If optDeluxe Then
  txtAmount = conDeluxePrice * Val(Text2.Text)
  Text3.Text = Str(txtAmount)
End If

If optSpecial Then
  txtAmount = conSpecialPrice * Val(Text2.Text)
  Text3.Text = Str(txtAmount)
End If

If optPrimo Then
   txtAmount = conPrimoPrice * Val(Text2.Text)
   Text3.Text = Str(txtAmount)
End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click its equivalent icon at the tool bar).

Try to click the Deluxe pizza option, to see how our program works. Now input a number in the Qty (Quantity) text box, then press the Enter key. Does the program display the amount correctly? Now try to click the Special pizza option and change the input number at the Qty (Quantity) text box, then press the Enter key, and see what happens. Is the amount correct? I hope so.

Explanation:

First, we declare all the variables and constants we use in our program. We type them under the **(General)** object and **(Declarations)** procedure in our module. Here is our code for it:

(General)	*(Declarations)*

```
Private InitialValue As Integer
Const conDeluxePrice=185, conSpecialPrice=250, conPrimoPrice=290
```

After the declaration of variable and constants, we will initialize the Text box 2 (Text2) and Text box 3 (Text3) with a value of 0 (zero) with the following code:

```
Private Sub Form_Load( )
  Text2.Text = InitialValue
  Text3.Text = InitialValue
End Sub
```

Since we want to put a check mark for the ingredients - Cheese (positioned at Check 1 object), Bacon & Ham (positioned at Check 3 object), Onions & Chili (positioned at Check 5 object) when the user clicks the Deluxe pizza option (positioned at Option 1 object), therefore we have to assign them with a Boolean value of 1 (one). The 0 (zero) Boolean value means the check box will be displayed without a check mark or with unselected status. We can accomplish this task with the following code:

```
Private Sub optDeluxe_Click( )
Check1.Value = 1
Check2.Value = 0
Check3.Value = 1
Check4.Value = 0
Check5.Value = 1
Check6.Value = 0
Check7.Value = 0
Text1.Text = Str(conDeluxePrice)
Text2.Text = InitialValue
Text3.Text = InitialValue
optDeluxe.Value = True
optSpecial.Value = False
optPrimo.Value = False
End Sub
```

The above explanation applies also (with a slight modification to fit with the given situation or requirement) to the Special and Primo pizza options. So there's no need to repeat the discussion for each of them.
We apply here the Str( ) function to convert the numeric constant value of pizza price. This numeric to string conversion is needed to properly display the numeric data in Text box 1 (Text1). We would like to ensure also that if the Deluxe pizza is currently selected, then the other options must be explicitly deselected or unselected by assigning a False value on its corresponding option button. That is the reason why we have the following code:

```
optDeluxe.Value = True
optSpecial.Value = False
optPrimo.Value = False
```

Our explanation here also applies to the other modules, specifically the modules for Special pizza option and Primo pizza option.
At the object Text box 2 (Text2) and KeyPress procedure, we declare a local variable txtAmt (using the Dim statement) as a Single (a number with a decimal point) data type and initialize it with a value of 0 (zero) before using it in our equation's calculation. This

is to ensure that the calculation of the equation is accurate since the txtAmt variable is initialized with a value of 0 (zero).
We are using here the *if* conditional statement to evaluate if the conditional expressions optDeluxe, optSpecial, and optPrimo are tested True. If the optDeluxe is evaluated to True, then its associated statements (which is an equation and a display at Text box 3 (Text3) will be executed or performed. The same thing applies to optSpecial and optPrimo conditional expressions too. Remember that the conditional expressions such as optDeluxe, optSpecial, and optPrimo can be evaluated to True only if they are selected by the user through clicking its corresponding option button. Try to examine its code below:

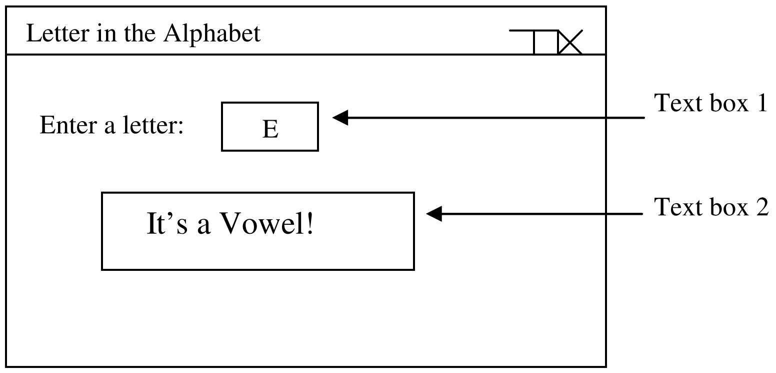
```
  Private Sub Text2_KeyPress(KeyAscii As Integer)
  Dim txtAmount As Single
  txtAmount = 0
→ If optDeluxe Then
    txtAmount = conDeluxePrice * Val(Text2.Text)
    Text3.Text = Str(txtAmount)
  End If

→ If optSpecial Then
    txtAmount = conSpecialPrice * Val(Text2.Text)
    Text3.Text = Str(txtAmount)
  End If

→ If optPrimo Then
     txtAmount = conPrimoPrice * Val(Text2.Text)
     Text3.Text = Str(txtAmount)
  End If
  End Sub
```

LAB ACTIVITY
TEST 3

1. Design and develop a simple application program that determines if the input letter at text box 1 (Text1) is a Vowel or Consonant. The output message should be displayed at text box 2 (Text2). The Vowels are: A E I O U; while the remaining letters are Consonants. Your program must be able to handle an uppercase or lowercase input letter. Follow the given figure below in designing and developing the application program:

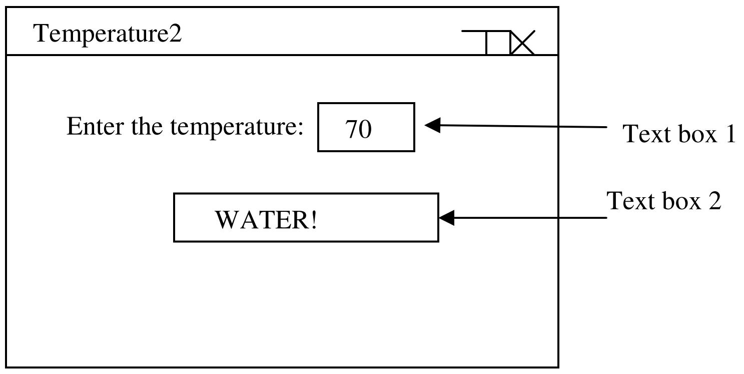


1st Solution: Use the If/ElseIf conditional statement
2nd Solution: Use the Select Case conditional statement

2. Design and develop a simple program that examines the input value of a variable nTemp (Temperature) at text box 1 (Text1). Then display the following messages at text box 2 (Text2) depending on the value assigned to nTemp variable.

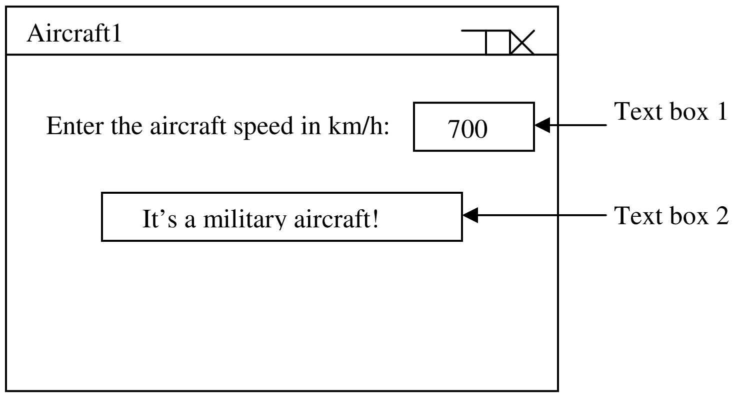
Temperature	Message
Less than 0	ICE
Between 0 and 100	WATER
Exceeds 100	STEAM

Follow the given figure below in designing and developing the application program:



1st Solution: Use the If/ElseIf conditional statement
2nd Solution: Use the Select Case conditional statement

3. Design and develop a simple program for the Air Force to label an aircraft as military or civilian. The program is to be given the plane's observed speed in km/h (kilometer per hour).The speed will serve as its input at text box 1 (Text1). For planes traveling in excess of 1100 km/h, you should label them as "civilian" aircraft; between 500 km/h to 1100 km/h, label them as "military" aircraft and for planes traveling at more slower speed - less than 500 km/h, you should label them as an "It's a bird!" message. The labeled message should be displayed at text box 2 (Text2). Follow the given figure below in designing and developing the application program:

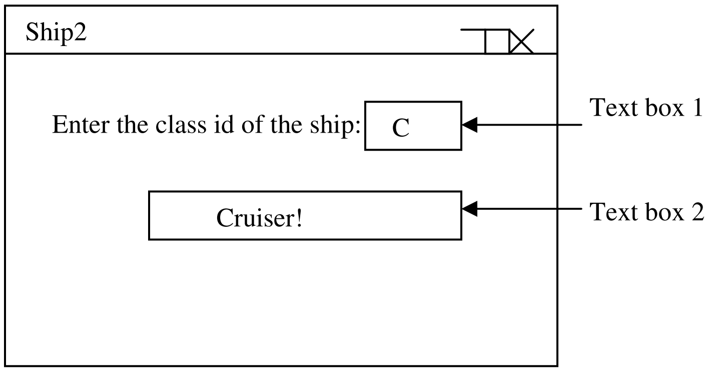


1st Solution: Use the If/ElseIf conditional statement
2nd Solution: Use the Select Case conditional statement

4. Design and develop a simple program that determines the class of the Ship depending on its class ID (identifier). Here is the criteria. The class ID serves as the input data at text box 1 (Text1) and the class of the ship serves as the output information at text box 2 (Tex2). Your program should be able to handle both capital or lower case letter as an input data.

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

Follow the given figure below in designing and developing the application program:

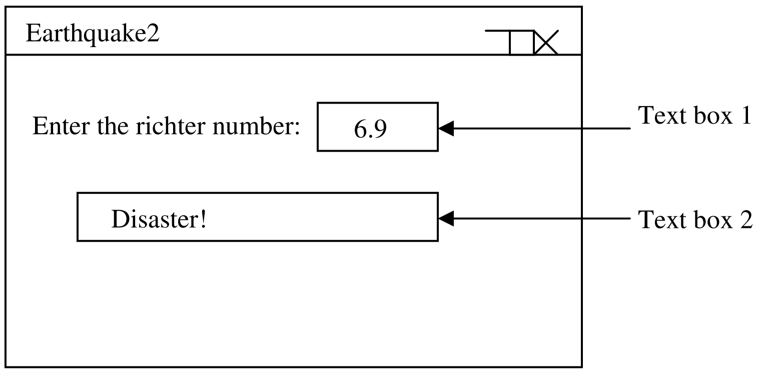


1st Solution: Use the If/ElseIf conditional Statement
2nd Solution: Use the Select Case conditional Statement

5. The National Earthquake Information Center has the following criteria to determine the earthquake's damage. Here is the given richter scale criteria and its corresponding characterization. The richter scale serves as an input data at text box 1 (Text1) and the characterization as an output information at text box 2 (Text2).

Richter Number	Characterization
Less than 5	Little or No Damage
From 5 up to 5.5	Some Damage
From 5.6 up to 6.5	Serious Damage
From 6.6 up to 7.5	Disaster
Higher than 7.5	Catastrophe

Follow the given figure below in designing and developing the application program:

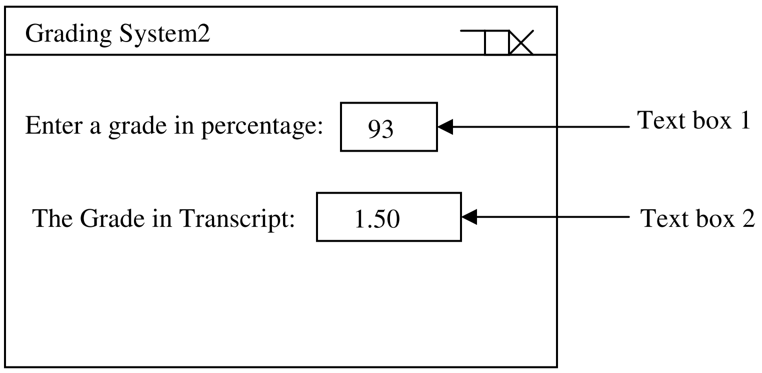


1st Solution: Use the If/ElseIf conditional statement
2nd Solution: Use the Select Case conditional statement

6. Design and develop a simple application program that accepts an input grade in percentile form at text box 1 (Text1) and output its grade equivalent at text box 2 (Text2) based on the given range of percentile and grade equivalent table below:

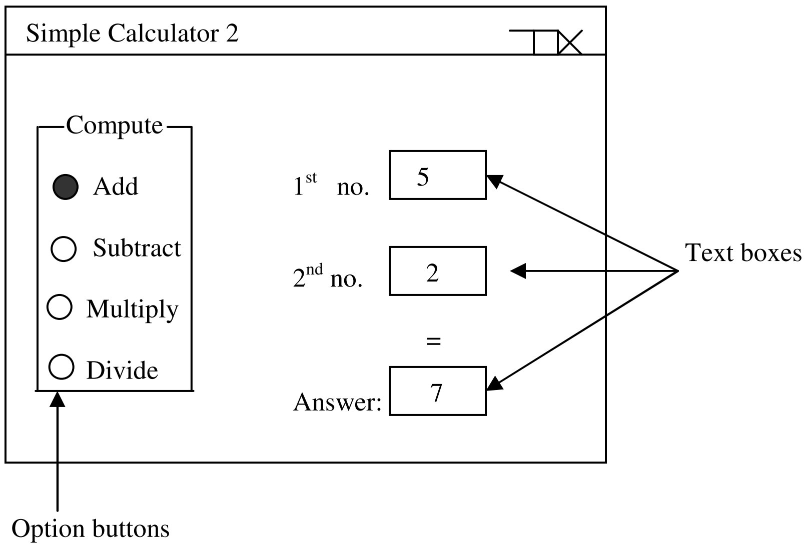
Range in Percentile Form	Grade in Transcript
98 - 100	1.00
95 - 97	1.25
92 - 94	1.50
89 - 91	1.75
85 - 88	2.00
82 - 84	2.25
80 - 81	2.50
77 - 79	2.75
75 - 76	3.00
other grades	“Out - Of-Range”

Follow the given figure below in designing and developing the application program:

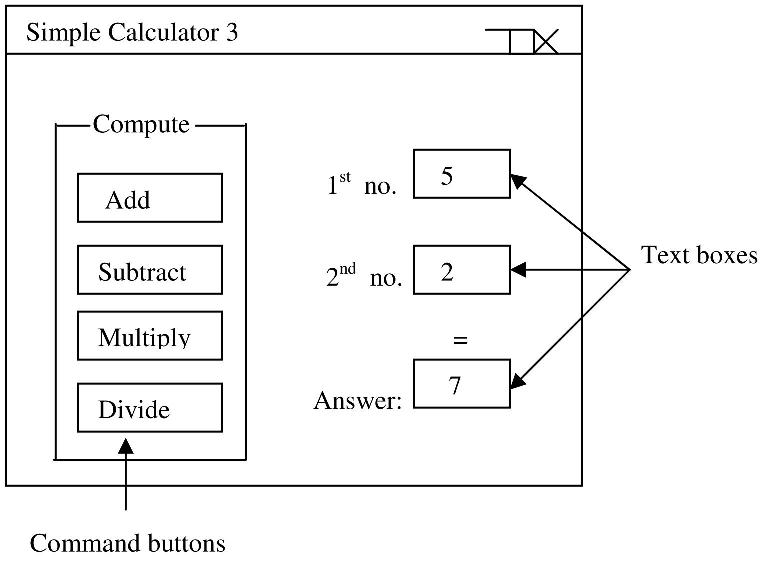


1st Solution: Use the If/ElseIf conditional statement
2nd Solution: Use the Select Case conditional statement

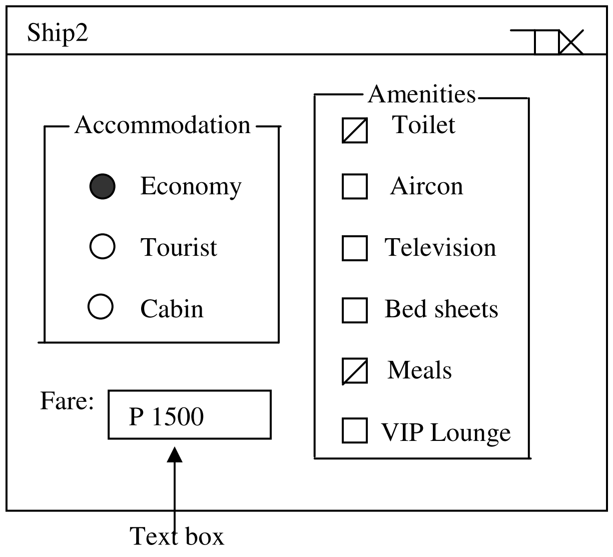
7. Design and develop a simple application program that adds, subtracts, multiplies, or divides two input numbers using the option buttons. Follow the given figure below in designing and developing the application program:



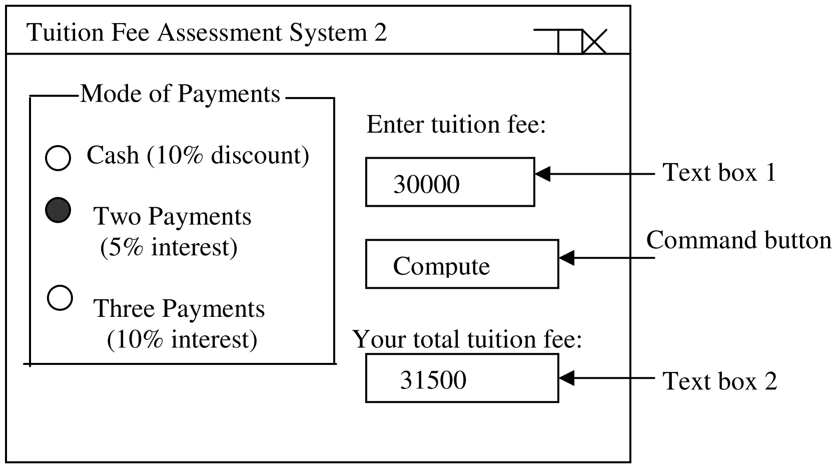
8. Design and develop a simple application program that adds, subtracts, multiplies, or divides two input numbers using a group of command buttons. Follow the given figure below in designing and developing the application program:



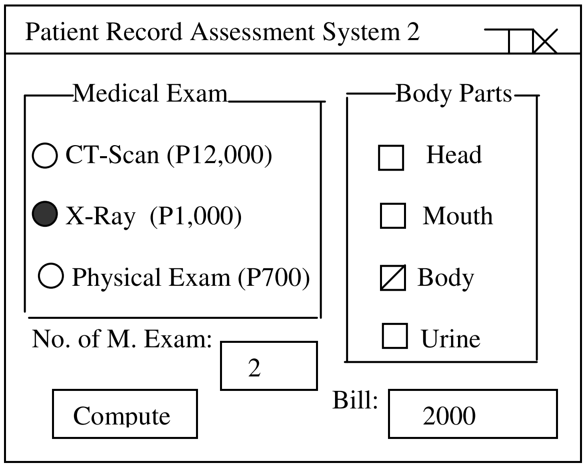
9. Design and develop a simple application system so that when the user chooses the Economy accommodation, the Toilet and Meals amenities should be marked with a check and the fare is P1,500. When the user chooses the Tourist accommodation, the Toilet, Aircon, Bed sheets, and Meals amenities should be marked with a check and the fare is P1,700. When the user chooses the Cabin accommodation, all amenities should be marked with a check and the fare is P2,000 pesos. . Design and develop a simple application program that adds, subtracts, multiplies, or divides two input numbers using the option buttons. Follow the given figure below in designing and developing the application program:



10.Design and develop this simple application system. Here in our program, we will input the tuition fee. Now if the user clicks the Compute command button, the total tuition fee will be displayed at Text box 2 (Text2). In our sample input data, we input the 30,000 tuition fee, now its 5 % (percent) interest is 1,500. Thus, output at text box 2 (Text2) is 31,500 (30,000 + 1,500).
Now for example we enter a tuition fee of 20,000 and then we select the Cash as the mode of payment, the output at text box 2 (Text2) should be 18,000 since there is a 10 % (percent) discount for paying a Cash tuition fee.
Follow the given figure below in designing and developing the application system:



11. Design and develop a simple application system that if the user selects CT-Scan, only the Head body parts is marked with a check. If X-Ray is chosen, only the Body will be marked with a check. And if the Physical exam is chosen, then all the body parts are marked with a check. In our example, we chose the X-ray as the Medical exam chosen, and since the number of Medical exam is 2, therefore, the Bill is 2000 (1000 * 2).



Chapter 5

Using Controls With Looping Statements

Looping Statements

The looping statement is a program instruction that repeats some statement or sequence of statements in a specified number of times. Looping statement allows a set of instructions to be performed all over and over again until a certain condition is reached, met, proven or tested as false or true. In other words, looping statement allows us to execute one or more lines of code repetitively. The looping statement that Visual Basic supports are:

- Do Loop
- For Next
- For Each Next

Different Do Loop Syntax

We use the Do Loop statement to execute a block of statements an indefinite number of times. The Do loops work well when we don't know how many times we need to execute the statements in the loop.
Here are the syntax of the different variations of the Do Loop statements :

Do While *condition*
  Statements 'Statements that do something repetitively
Loop

When Visual Basic executes a Do- While- Loop, it first evaluates the *condition*. If the *condition* is False (0), it will not execute all the statements within the body of the loop. If it's True (1), Visual Basic executes the statements and then goes back to the Do While statement and evaluates the *condition* again.

Do
  Statements 'Statements that do something repetitively ⟵ *body of the loop*
Loop While *condition*

When Visual Basic executes a Do-Loop-While, it executes the statements first and then evaluates the *condition* after each execution. This Do-Loop-While statement guarantees at least one execution of statements.

Do Until *condition*
  Statements 'Statements that do something repetitively ⟵ *body of the loop*
Loop

In a Do-Until-Loop statement, it loops zero or more times.

Do
  Statements *'Statements that do something repetitively* ← *body of the loop*
Loop Until *condition*

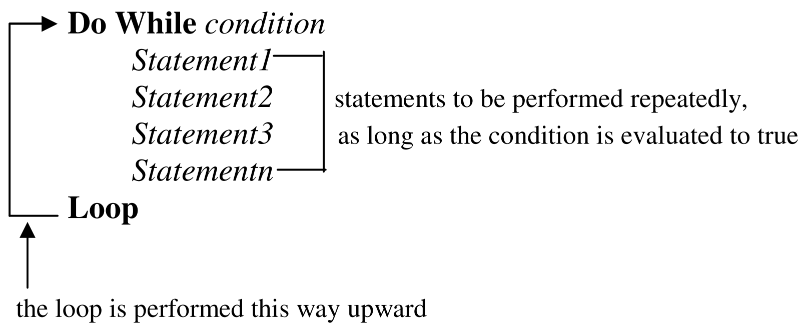
In a Do-Loop-Until statement, it loops at least once.

While *condition*
  Statements *'Statements that do something repetitively* ← *body of the loop*
Wend

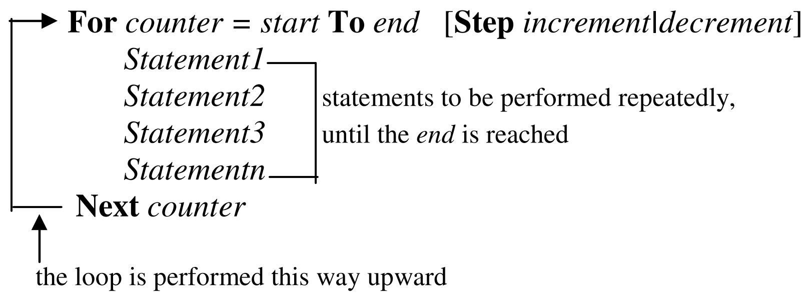
The While-Wend loop statement works like the Do-While-Loop statement. You can use the While-Wend statement as a substitute to the Do-While-Loop.

How The Do-While-Loop Iterates3

Here are the graphical representation on how the computer performs the iteration process:


Do While condition
Statement1
Statement2
Statement3
Statementn
statements to be performed repeatedly, as long as the condition is evaluated to true
Loop
the loop is performed this way upward

How The For-Next Loop Iterates


For counter = start To end [Step increment|decrement]
Statement1
Statement2
Statement3
Statementn
statements to be performed repeatedly, until the end is reached
Next counter
the loop is performed this way upward

To understand completely how this For-Next loop statement, we will have the complete syntax of it in our next discussion. Here it is!

Different For Loop Syntax

The For Next loop statement is a better choice when we know in advance how many times a loop executes. A For loop uses a variable called a *counter* that increases or decreases in value during each repetition of the loop. Here are the syntax of the different variations of For loop statements:

For *counter* = *start* **To** *end* [**Step** *increment/decrement*]
    Statements *'Statements that do something repetitively* ⟵ *body of the loop*
Next [*counter*]

The *counter*, *start*, *end*, and *increment*(positive) or *decrement*(negative) arguments are all numeric data.
If the *increment* argument is used after the Step syntax, the *start* argument must be less than or equal to *end* argument, otherwise, the statements in the loop will not execute. If the decrement argument is used after the Step syntax, the *start* argument must be greater than or equal to the *end* syntax in order for the statements within the body of the loop to execute. If the Step is not set, the default value of the increment is positive 1.
The Visual Basic executes the For loop in the following steps:

1. First, it sets the *counter* equal to the *start* argument.
2. Second, it evaluates to see if *counter* is greater than the *end* argument. If it is, Visual Basic exits the loop. Now if the decrement is used, Visual Basic evaluates to see if the *counter* is less than the *end* argument.
3. At this point, the For loop executes the statements within the body of the loop.
4. Finally, the counter argument is incremented by 1, if it is specified.
5. Then, the For loop will perform the steps 2 through 4 over and over again until the condition is finally satisfied or met such as the *end* argument value is reached.

For Each element In group
  Statements *'Statements that do something repetitively* ⟵ *body of the loop*
Next

The For-Each-Next loop repeats a group of statements for each element in a collection of objects or in an array instead of repeating the statements in a specified number of times. This is especially helpful if we don't know how many elements are in a collection.

Example 1:

Design and develop a simple looping statement program that generates the given sequence numbers using the For Next syntax.
Follow the given figure below in designing and developing the application system.

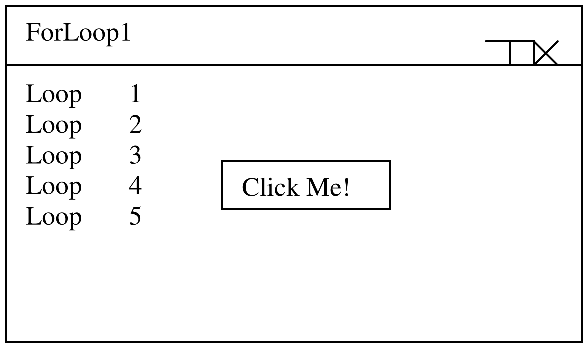


Figure 5.1 Generating sequence number using For Next syntax

Note:

When the user clicks the Click Me! command button, the Loop 1 up to Loop 5 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **ForLoop1.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 For nCounter = 1 To 5
    Print "Loop", nCounter
 Next nCounter
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

The *start* argument value of this example is 1, while the *end* argument value is 5.

$$\begin{tikzcd}[row sep=small] \textit{For nCounter} = & \textit{1} \arrow[d, leftarrow] & \textit{To} & \textit{5} & \textit{end} \arrow[l] \\ & \textit{start} & & & \end{tikzcd}$$

The loop starts counting from 1 (start) and stops at 5 (end). The nCounter increases by 1 every time it loops, until the counter reaches the *end* value of 5. This further means that that incrementation operation happened within the line of *for* loop statement. This is the main reason why the incrementation formula is not included within the body of the loop. We will notice that since the loop is performed 5 times, the following statement within the body of the loop will be executed 5 times:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 For nCounter = 1 To 5
   Print "Loop", nCounter  ←—— statement within the body of the loop
 Next nCounter
End Sub
```

The incrementation and decrementation formula are designed like the following:

$I = I + 1 \longleftarrow$ incrementation formula

$D = D - 1 \longleftarrow$ decrementation formula

In other looping statements such as any variations of the Do-While loop, these incrementation and decrementation formulas are commonly used.
We can also explicitly specify the Step 1 syntax here in our For loop statement, though this is no longer needed since this is already a default syntax for the For loop statement. However, for the sake of clarity, we can do so. Here is its complete code:

```
Private Sub Command1_Click( )
Dim nCounter As Integer
For nCounter = 1 To 5 Step 1 ←—
  Print "Loop", nCounter
Next nCounter
End Sub
```

Explicit declaration such as this Step 1 syntax is a good programming practice. What it requires is an extra effort from us to make our program more readable and programmer-friendly.

Example 2:

Design and develop a simple looping statement program that generates the given sequence numbers using the Do-While-Loop syntax.
Follow the given figure below in designing and developing the application system.

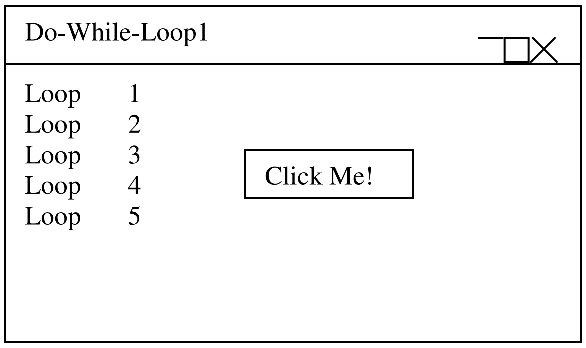


Figure 5.2 Generating sequence numbers using Do-While-Loop syntax.

Note:

When the user clicks the Click Me! command button, the Loop 1 up to Loop 5 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Do-While-Loop1.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 nCounter = 1
 Do While nCounter <= 5
    Print "Loop", nCounter
    nCounter = nCounter + 1
 Loop
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

The initial value of nCounter variable is 1. So when the program pointer evaluates or tests the *condition* :

nCounter<=5

it is evaluated to True, since the value of variable nCounter is less than 5. Because the *condition* is evaluated to true, all statements within the body of the loop:

```
                condition
           ┌──────────────────┐
           ▼                  ▼
  Do While   nCounter <= 5
    Print "Loop", nCounter     ◄──┐
    nCounter = nCounter + 1    ◄──┘ statements within the body of the loop
Loop
```

are executed. The execution of all the statements within the body of the loop are repeated until the evaluation of the *condition* becomes False.

We can also use the other variations of the Do-While loop statements. Let us examine some of the code of these Do - While loop variations below:

The Do-Loop -While Solution:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 nCounter = 1
 Do
   Print "Loop", nCounter
   nCounter = nCounter + 1
 Loop While nCounter <= 5
End Sub
```

You will notice that in the Do-Loop-While solution, the condition is tested at the last part of our loop. This ensures that even if the condition is evaluated or tested as False, the statements within the body of the loop will be executed at least once.

```
Do
  Print "Loop", nCounter     ◄──┐
  nCounter = nCounter + 1    ◄──┘ statements within the body of the loop
Loop While nCounter <= 5
```

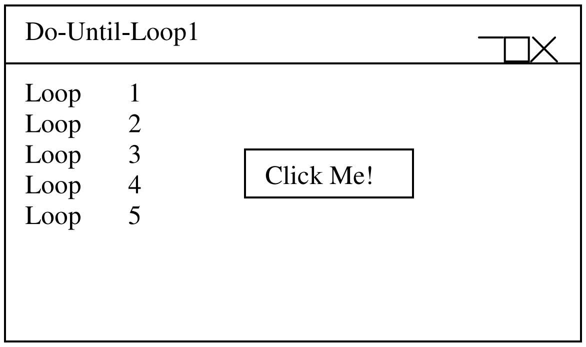
In our next example, we will examine the Do-Until-Loop syntax equivalent solution for the given task above. But before that, it is also good to take note that there is another syntax for the While loop command. Here it is:

```
Private Sub Command1_Click( )
Dim nCounter As Integer
nCounter = 1
While nCounter <= 5
  Print "Loop", nCounter
  nCounter = nCounter + 1
Wend
End Sub
```

See! It's the While-Wend loop statement! This time you can really conclude that we have a lot of options to choose in using looping statements to our application program. Agree?

Example 3:

Design and develop a simple looping statement program that generates the given sequence numbers using the Do-Until-Loop syntax.
Follow the given figure below in designing and developing the application system.



```
Loop     1
Loop     2
Loop     3
Loop     4
Loop     5
```

Figure 5.3 Generating sequence numbers using Do-Until-Loop syntax

Note:

When the user clicks the Click Me! command button, the Loop 1 up to Loop 5 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.

2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Do-Until-Loop1.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 nCounter = 1
 Do Until nCounter > 5
   Print "Loop", nCounter
   nCounter = nCounter + 1
 Loop
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

What is noticeable here in this Do-Until-Loop solution is the way we modify the condition of our loop. We change it from:

nCounter <= 5

to

nCounter > 5

On these changes, we instruct the computer to loop until the nCounter is greater than 5. In other words, the loop will only stop when the nCounter variable contains the value greater than 5. The result was still the same with the previous solution, because the statements within the body of the loop will be executed 5 times.
Here below is an equivalent solution for another variation of the Do-Until-Loop which is the Do-Loop-Until syntax.

Do-Loop-Until Solution:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 nCounter = 1
 Do
   Print "Loop", nCounter
   nCounter = nCounter + 1
 Loop Until nCounter > 5
End Sub
```

You will notice that in the Do-Loop-Until solution, the condition is tested at the last part of our loop. This ensures that even if the condition is evaluated or tested as False, the statements within the body of the loop will be executed at least once. We can observe that this Do-Loop-Until solution works exactly like Do-Loop-While syntax. Their big difference is that we change the conditional expression from:

nCounter <= 5

to

nCounter > 5

to satisfy the given condition. In the Do-Loop-While syntax, we are instructing the computer to execute the statements within the body of the loop as long as the value of the nCounter variable is less than or equal to 5. However, in the Do-Loop-Until syntax, we are instructing the computer to stop executing the statements within the body of the loop if the value of nCounter variable is greater than 5. Remember that the value of nCounter variable increases by one every time the loop iterates. This task is accomplished by this incrementation formula:

nCounter = nCounter + 1

which can be found within the body of the loop.
In the case of the For-Next loop, the incrementation formula is integrated within the For loop statement. So the incrementation happens within this line:

For nCounter = 1 To 5 ⟵ *incrementation happens here*

Putting an incrementation formula within the body of the For loop statement will cause the loop to perform a double incrementation process. One at the For loop line and the next one is performed within the body of the loop caused by the nCounter = nCounter +1 formula. So beware with this kind of mistake. This is a hard to debug program error.

Warning!
This code must not be used:

```
Private Sub Command1_Click()
Dim nCounter As Integer
For nCounter = 1 To 5⟵
  Print "Loop", nCounter
  nCounter = nCounter + 1 ⟵
Next nCounter
End Sub
```

The double incrementation was performed within the For loop line and the other one is in the incrementation formula within the body of the loop: nCounter = nCounter + 1.
This will result to the output:

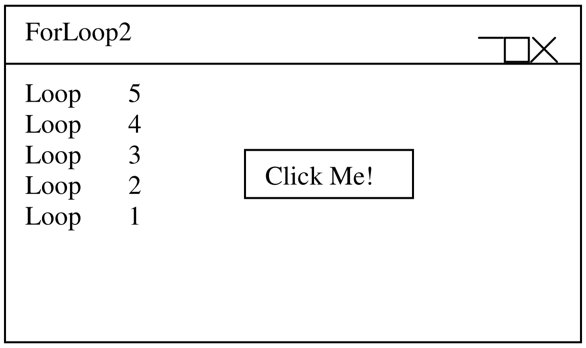
```
Loop 1
Loop 3
Loop 5
```

So that's the end of our lengthy discussions about the different solutions using looping statements .

In our next example, we will learn the inverse number generation. Meaning, the number will start from the highest down to the lowest.

Example 4:

Design and develop a simple looping statement program that generates the given inverse sequence numbers using the For-Next loop syntax.
Follow the given figure below in designing and developing the application system.



```
Loop    5
Loop    4
Loop    3
Loop    2
Loop    1
```

Figure 5.4 Generating an inverse sequence numbers using For-Next syntax.

Note:

When the user clicks the Click Me! command button, the Loop 5 down to Loop 1 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **ForLoop2.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 For nCounter = 5 To 1 Step -1
   Print "Loop", nCounter
 Next nCounter
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

You will notice here in our For- Next loop that we explicitly specify the Step - 1 syntax. This is really a requirement so that our For- Next loop will decrement (decrease) the value of the loop by 1 every time it loops. Without doing so, our For-Next loop will not work the way we want it. Why? Because the default Step syntax for the For-Next loop is +1 (positive one). Meaning that in the absence of the Step syntax, the For-Next loop will increment by 1. So if we want it to decrement (decrease) by 1 (one), we have to explicitly specify the Step -1 syntax, with the following line:

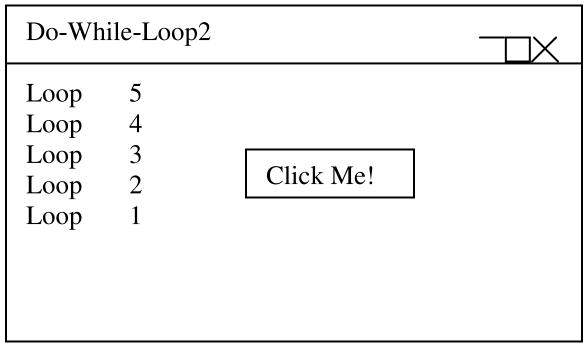
For nCounter = 5 To 1 Step -1

This syntax further means that the loop iterates 5 times downward. Meaning from 5 down to 1. This is still 5 times to loop, isn't it? So the statement within the body of the loop below will be executed five times:

```
Private Sub Command1_Click()
Dim nCounter As Integer
 For nCounter = 5 To 1 Step -1
   Print "Loop", nCounter   ←  statement within the body of the loop
 Next nCounter
End Sub
```

Example 5:

Design and develop a simple looping statement program that generates the given inverse sequence numbers using the Do-While-Loop syntax.
Follow the given figure below in designing and developing the application system.



```
Loop      5
Loop      4
Loop      3
Loop      2
Loop      1
```

Figure 5.5 Generating inverse sequence numbers using While-Do-Loop syntax

Note:

When the user clicks the Click Me! command button, the Loop 5 down to Loop 1 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Do-While-Loop2.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
Do While nCounter >= 1
  Print "Loop", nCounter
  nCounter = nCounter - 1
Loop
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Here in our code for the Do-While-Loop solution, we initialized our nCounter variable with a value of 5. Then our conditions is:

nCounter >= 1

where in as long as the value of nCounter variable is greater than or equal to 1, the statements within the body of the loop will be executed. Here are the statements within the body of the loop:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
Do While nCounter >= 1
  Print "Loop", nCounter
  nCounter = nCounter - 1      statements within the body of the loop
Loop
End Sub
```

Now you will notice also that we include the decrementation formula within the body of the loop. We need this formula to terminate the loop and to decrement (decrease) the value of nCounter variable.
In the For loop syntax, this decrementation formula is not needed (and must not be) since the Step -1 syntax is the one that acts as a decrementation formula. Putting a decrementation formula within the body of the For loop statement causes the looping statement to perform a double decrementation (or incrementation) execution which will result to an undesirable output. Do you want to try? Try it! And see for yourself what I really mean for “must not be included” warning! This time, I am not kidding, buddy.
Here below, we will examine the equivalent solution using the Do-Loop-While syntax. This is another variation of Do-Loop statement.

Do-Loop-While Solution:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
Do
  Print "Loop", nCounter
  nCounter = nCounter - 1
Loop While nCounter >= 1
End Sub
```

You will notice that in the Do-Loop-While solution, the condition is tested at the last part of our loop:

```
.
.
.
nCounter = 5
Do
  Print "Loop", nCounter
  nCounter = nCounter - 1
Loop While nCounter >= 1 ←── last part of the loop
.
.
```

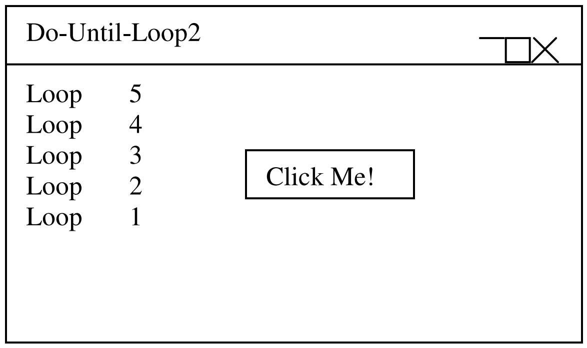
This ensures that even if the condition is evaluated or tested as False, the statements within the body of the loop will be executed at least once.
Next, we will examine the While-Wend equivalent solution. Here is its complete code:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
While nCounter >= 1
  Print "Loop", nCounter
  nCounter = nCounter - 1
Wend
End Sub
```

The While-Wend loop statement works like the Do-While-Loop statement. You can use the While-Wend statement as a substitute to the Do-While-Loop.
Now this time, we have to examine the Do-Until-Loop equivalent solution.

Example 6:

Design and develop a simple looping statement program that generates the given inverse sequence numbers using the Do-Until-Loop syntax.
Follow the given figure below in designing and developing the application system.



```
Loop     5
Loop     4
Loop     3
Loop     2
Loop     1
```

Figure 5.6 Generating inverse sequence numbers using Do-Until-Loop syntax

Note:

When the user clicks the Click Me! command button, the Loop 5 down to Loop 1 numbers will be generated.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Do-Until-Loop2.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
Do Until nCounter < 1
  Print "Loop", nCounter
  nCounter = nCounter - 1
Loop
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

What is noticeable here in this Do-Until-Loop solution is the way we modify the condition of our loop. We change it from:

nCounter >= 1

to

nCounter < 1

On these changes, we instruct the computer to loop until the nCounter is less than 1. In other words, the loop will only stop when the nCounter variable contains the value less

than 1 (this value is zero). This further means that when the nCounter variable holds a value of zero (0), then the loop stops iterating.
The result was still the same with the previous solution, because the statements within the body of the loop will be executed 5 times downward. Meaning, from value 5 down to 1.
Here below is an equivalent solution for another variation of the Do-Until-Loop which is the Do-Loop-Until syntax.

Do-Loop-Until Solution:

```
Private Sub Command1_Click()
Dim nCounter As Integer
nCounter = 5
Do
  Print "Loop", nCounter
  nCounter = nCounter - 1
Loop Until nCounter < 1
End Sub
```

You will notice that in the Do-Loop-Until solution, the condition is tested at the last part of our loop. This ensures that even if the condition is evaluated or tested as False, the statements within the body of the loop will be executed at least once. We can observe that this Do-Loop-Until solution works exactly like Do-Loop-While syntax. Their big difference is that we change the conditional expression from:

nCounter >= 1

to

nCounter < 1

to satisfy the given condition. In the Do-Loop-While syntax, we are instructing the computer to execute the statements within the body of the loop as long as the value of the nCounter variable is greater than or equal to 1. However, in the Do-Loop-Until syntax, we are instructing the computer to stop executing the statements within the body of the loop if the value of nCounter variable is less than 1. Remember that the value of nCounter variable decreases by one every time the loop iterates. This task is accomplished by this decrementation formula:

nCounter = nCounter - 1

which can be found within the body of the loop.
In the case of the For-Next loop, the decrementation formula is integrated within the For loop statement. So the incrementation happens within this line:

For nCounter = 5 To 1 Step -1 ⟵

So the Step - 1 syntax within the For loop line serves as the decrementation formula. If we accidentally include a decrementation formula within the body of the loop such as this equation : nCounter = nCounter - 1, then the decrementation process will be doubled. Once at the For loop line; performed by the Step -1 syntax and the other one is accomplished by the nCounter = nCounter -1 equation, thus results to undesirable output. You can try this one to see it in action!

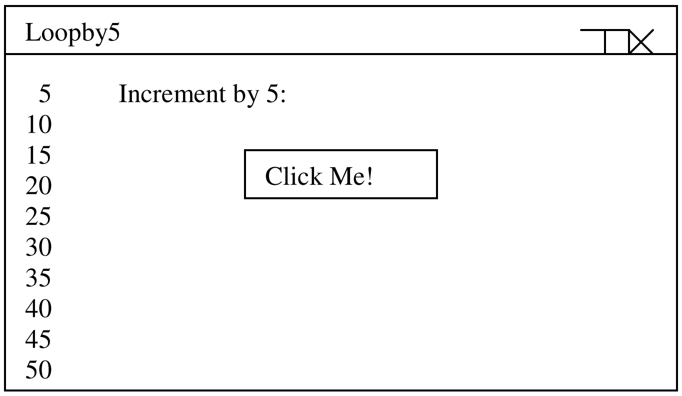
Warning!
This code must not be used:

```
Private Sub Command1_Click()
Dim nCounter As Integer
For nCounter = 5 To 1 Step -1 ←
  Print "Loop", nCounter
  nCounter = nCounter - 1 ←
Next nCounter
End Sub
```

The double decrementation was performed by the For loop statement due to the Step -1 syntax and the other one is in the decrementation formula within the body of the loop performed by the equation: nCounter = nCounter -1.
This will result to the output:

```
  Loop  5
  Loop  3
  Loop  1
```

This time, we are pretty much familiar with how the looping statement works. We are also confident that we can easily convert one looping statement syntax to another, such as converting a *for* loop statement into *while* loop statement.
In our next example, we will explore on how to use looping statement in manipulating objects or controls in a form. Let us start now.
What if we want to generate a sequence numbers by 5 (as in 5, 10, 15, 20 and so on)? Consider the design specification below:



```
 5
10
15
20
25
30
35
40
45
50
```

Figure 5.6a Generating numbers by 5 using loop statement

Our solution is this:

```
Private Sub Command1_Click( )
Dim I As Integer
 For I = 5 To 50 Step 5
   Print I
 Next I
End Sub
```

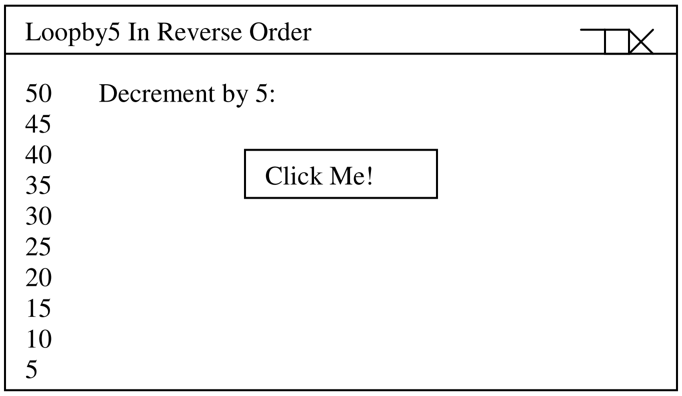
Since we start from 5 and end at 50, we have this For loop line:

```
For I = 5 To 50 Step 5
```

To convert the code to Do While Loop statement equivalent, we will have the following solution:

```
Private Sub Command1_Click( )
Dim I As Integer
I = 5
Do While I <= 50
  Print I
  I = I + 5
Loop
End Sub
```

What if we want to generate the number in reverse order? Consider the following design specification:



```
50
45
40
35
30
25
20
15
10
5
```

Figure 5.6b Generating numbers inversely by 5 using loop statement

The solution is just an opposite from the previous one:

```
Private Sub Command1_Click()
Dim I As Integer
 For I = 50 To 5 Step -5
   Print I
 Next I
End Sub
```

Here, we are using the decrementation process in the For loop line:

```
 For I = 50 To 5 Step -5
```

How can we convert the syntax into the Do While Loop statement? Here is its equivalent code:

```
Private Sub Command1_Click( )
Dim I As Integer
I = 50
Do While I >= 5
  Print I
  I = I - 5
Loop
End Sub
```

Example 7:

Design and develop a simple program that demonstrates how to increase the font size of the displayed message on the form using For loop syntax.
Follow the given figure below in designing and developing the application program.

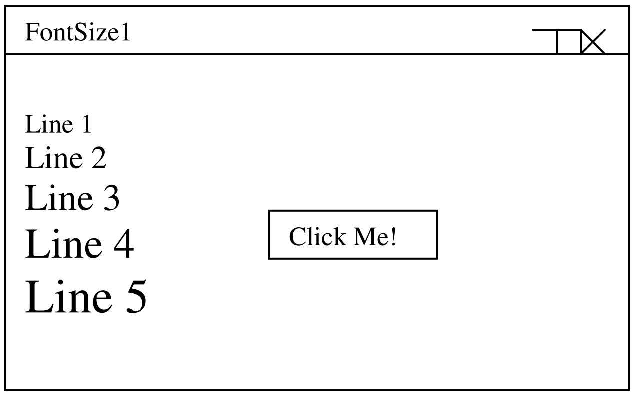


Figure 5.7 Increasing the FontSize of the text.

Note:

When the user clicks the Click Me! command button, the Line 1 through Line 5 will be generated with the font size is increasing as it is displayed.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **FontSize1.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
 For i = 1 To 5
   FontSize = 10 + i
   Print "Line"; i
 Next i
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

You will notice that within our For loop statement we have the equation:

FontSize = 10 + i

This is how we increase the font size of the displayed message “Line” and the corresponding generated number of variable *i.* The constant value of 10 means we increase the displayed message by 10. Finally, we can print the message in the form by the command:

Print "Line"; i

This loop will be performed only 5 times since our *for* loop specifies the starting iteration from 1 up to 5 with the following code:

For i = 1 To 5

In the example above, we learned how to apply the FontSize property of the displayed text or message.

Example 8:

Design and develop a simple program that demonstrates how to generate the first ten screen fonts on the form using For loop syntax.
Follow the given figure below in designing and developing the application program.

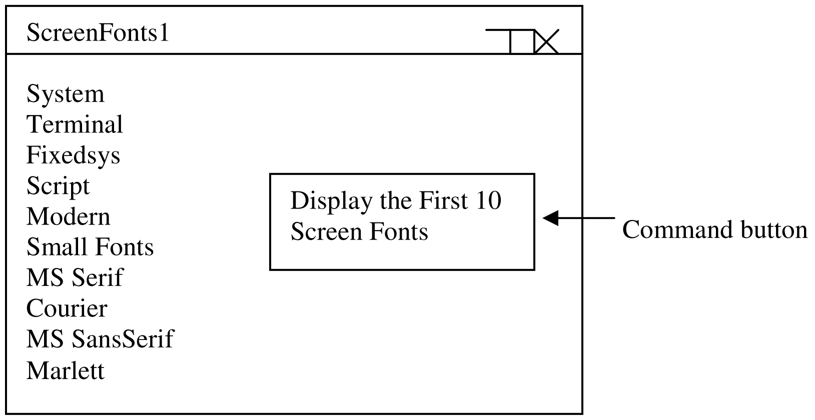


Figure 5.8 Display the First 10 Screen Fonts.

Note:

When the user clicks the Display the First 10 Screen Fonts command button, the ten fonts will be displayed on the form.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **ScreenFonts1.**
3. Double-click a Command button in the Toolbox to add it into the form. Position it properly and adjust its width appropriately. Set its caption to **Display First 10 Screen Fonts!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim I As Integer
 For I = 0 To Screen.FontCount
  If I <= 10 Then
    Print Screen.Fonts(I)
  Else
    Exit For 'Exit from the loop
  End If
 Next
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

As you will notice in our code, our starting iteration number starts with 0 (zero) and ends not with a number but with the object and its property in the syntax of *Object.Property:*

```
Dim I As Integer
 For I = 0 To Screen.FontCount  ←
  If I <= 10 Then
   Print Screen.Fonts(I) ←
  Else
   Exit For 'Exit from the loop
  End If
 Next
```

The next noticeable thing also is in how we displayed the screen fonts using the object and its property with the syntax:

Print Screen.Fonts(I)

wherein we put the iteration variable *I* inside the square brackets of the Fonts property which is connected to the Screen object. This further means that as long as the iteration variable *I* is less than 10 as the loop is iterated, the corresponding screen font will be displayed. In other words, the iteration variable *I* serves as the argument or parameter of the Fonts property. By passing the variable *I* on the Fonts property of the Screen object, we can manipulate the screen fonts. In our case, we simply display the first 10 screen fonts. If we want to display all the screen fonts, we will simply omit the *if* conditional statements here in our program.
This program also demonstrates on how to use the *Exit For* syntax which is our way of forcing to exit within the body of the loop any time we want. In our case, we force the program pointer to exit from the body of the loop as our means to end the iteration process. Remember that the ending iteration here in our program is not an ending iteration number but an object and its corresponding property with the following syntax:

For I = 0 To Screen.FontCount

so we need this *Exit For* command badly to get out from the loop's body, otherwise the loop has no way to end the iteration process (which in turn results to an infinite or endless loop).
We can also use this *Exit* syntax to exit from a Procedure or Function. In other words, since we want to end our loop or the subprogram for this screen fonts once the first 10 screen fonts were displayed, we can replace the

Exit For 'Exit from the loop

with

Exit Sub 'Exit from the Procedure

to exit forcibly from the procedure's body of *Private Sub Command1_Click( ),* which will result to an exit from the *For* loop's body too. So the following syntax will work fine as well:

```
Private Sub Command1_Click()
Dim I As Integer
 For I = 0 To Screen.FontCount
  If I <= 10 Then
    Print Screen.Fonts(I)
  Else
    Exit Sub 'Exit from the Procedure ←
  End If
 Next
End Sub
```

That's the end of our discussion and I hope you have learned well.
In our next example, we will learn how the *For Each* loop statement works. We already knew that the *For Each* loop statement though similar to *For* loop statement, it repeats only a group of statements for each element in a collection of objects or in an array instead of repeating the statements for a specified number of times. Here is now its actual application.

Example 9:

Design and develop a simple program that demonstrates how to use the For Each loop syntax that counts how many objects are there in a Form.
Follow the given figure below in designing and developing the application program.

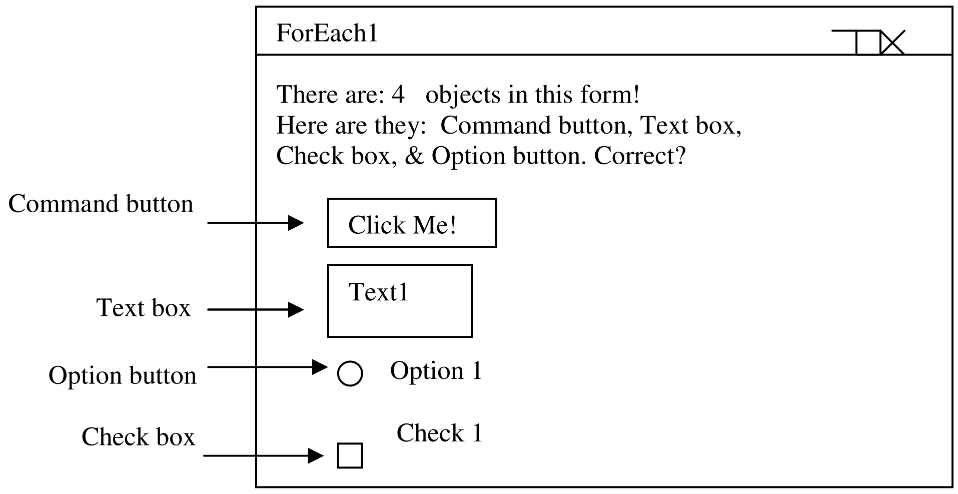

ForEach1
There are: 4 objects in this form!
Here are they: Command button, Text box,
Check box, & Option button. Correct?
Command button
Click Me!
Text box
Text1
Option button
Option 1
Check box
Check 1

Figure 5.9 Display and Count the Objects in the form.

Note:

When the user clicks the Click Me! command button, the message, "There are: 4 objects in this form! Here are they: Command button, Text box, Check box, & Option button. Correct?" should appear on the top right corner of the form.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.

2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **ForEach1.**
3. Double-click the Command button, Text box, Option button and Check box in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately. Set the caption of the command button to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim oObject As Control
Dim nIterate As Integer
nIterate = 0
For Each oObject In Form1.Controls
    nIterate = nIterate + 1
Next oObject
Print "There are:", nIterate, "objects in this form!"
Print "Here are they: Command button"
Print "Text box, Check box, & Option"
Print "button. Correct?"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

First, we declare our oObject as a Control data type, then we simply declare our ordinary variable nIterate as an Integer data type and initialized its value with 0 (zero). Here are their syntax:

Dim oObject As Control
Dim nIterate As Integer
nIterate = 0

Now let us use this oObject control variable into the For Each loop statement syntax with the following syntax:

```
 For Each oObject In Form1.Controls ←
   nIterate = nIterate + 1
Next oObject
```

The nIterate variable will simply increment by 1 (one) each time the loop iterates. After the loop iterates, the nIterate variable contains the latest incremented value which will depend on how many times the loop iterates. In this case, the loop iterates 4 (four) times since there are four objects or controls that are found in our form. In other words, the nIterate variable contains the value of 4 (four). Finally, we displayed the number of objects found in the form with the following statements:

Print "There are:", nIterate, "objects in this form!" ⟵
Print "Here are they: Command button"
Print "Text box, Check box, & Option"
Print "button. Correct?"

where the value 4 is displayed for the nIterate variable which is written within the first *Print* statement.

Example 10:

Design and develop a simple program that demonstrates how to generate all the screen fonts on the List box using For loop syntax.
Follow the given figure below in designing and developing the application program.

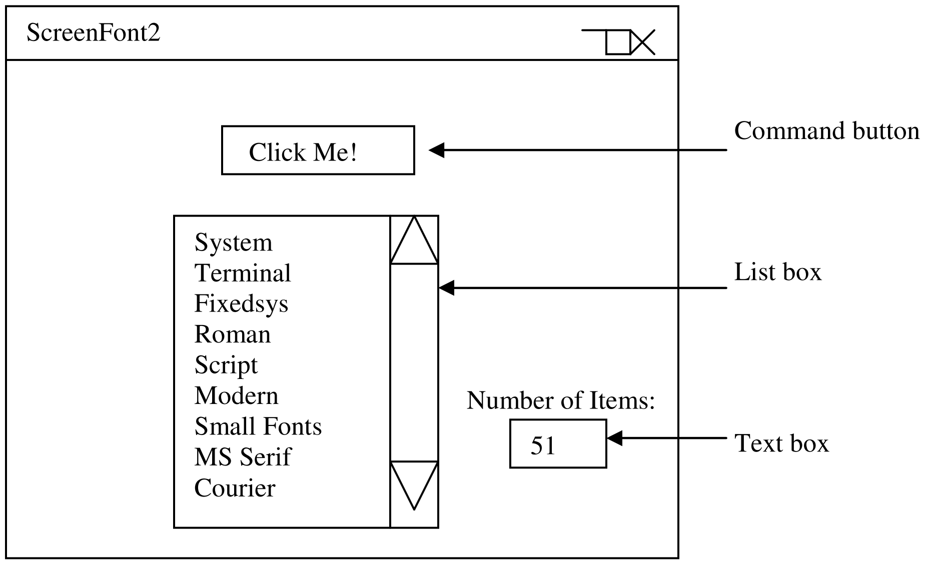


Figure 5.10 Display all the Screen Fonts.

Note:

When the user clicks the command button Click Me!, all the screen fonts should be displayed at the List box and the number of screen fonts must be displayed at the text box.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **ForEach1.**
3. Double-click the Command button, List box , Text box and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately. Set the caption of the command button to **Click Me!.**
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Command1_Click()
Dim I As Integer
 For I = 0 To Screen.FontCount
     List1.AddItem Screen.Fonts(I)
 Next I
 Text1.Text = List1.ListCount
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Here in this case, we can generate all screen fonts since all the items in the List box will be simply wrapped up when they cannot be displayed at once. So there is no need to use the conditional *if-else* statement to control how many screen fonts we would like to display on the form and there is no need also to use the *Exit For* statement in this situation, because there is no way that the List box will be overflowed with items, unlike in a Form wherein it has a limit to how much messages we can print into it.
Here in our code, we simply add all the items within the Screen.Font( ) (object and property) on the List box by passing the variable *I* as its argument or parameter, with the following statement:

List1.AddItem Screen.Fonts(I)

Remember that we started the iteration number with 0 (zero) up to Screen.FontCount which means, as long as there is a screen font within the Screen object, the loop will iterate all over and over again. Now when the last FontCount is reached by the loop iteration, the loop stops iterating. Here is our code to accomplish this task:

For I = 0 To Screen.FontCount

We need exactly how many screen fonts are available, so we have the following code:

Text1.Text = List1.ListCount

to accomplish the task. In this code, we simply count the number of items in the List box using the List1.ListCount statement and store it into Text1.Text statement which results to the display of the number of items into text box 1.
To understand it more, we will delve deeper about this List box object and its similar object too which is the Combo box , in our succeeding example.

Using List Boxes

A list box is an ideal way of presenting a list of data or choices to the user. In fact, the list box is an effective way to present a large number of choices or data to the user in a limited amount of space. The user can browse the data in the list box or select one or more items for processing purposes. The process could be transferring the chosen items from the list box to a cart (or a transfer of items from one list box to another) The transfer can also be from a list box to a text box or to a combo box (we will discuss the combo box, later). In the computerized grocery store system, the items could be the products which the buyer can select and buy.
By default, the items in the list box are displayed vertically in a single column, although we can set up multiple columns too. When the list of items or data is too long for the list box, the Visual Basic system will simply add a vertical scroll bar. With this, the user can then scroll up and down or left to right through the list of items.
Let us now have our next example on list box control.

Example 11:

Design and develop a simple program that demonstrates how to preload a collection of items to a List box at run time. Follow the given figure below in designing and developing the application program.

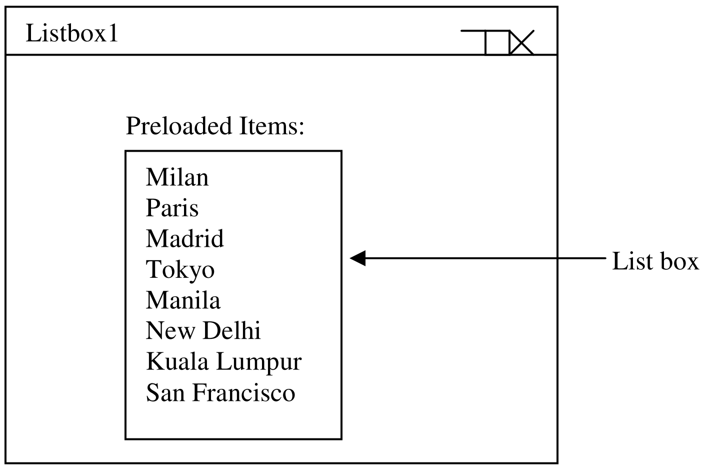


Figure 5.11 Preload the items into the List Boxl

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox1.**
5. Double-click the List box , and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
3. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
  List1.AddItem "Milan"
  List1.AddItem "Paris"
  List1.AddItem "Madrid"
  List1.AddItem "Tokyo"
  List1.AddItem "Manila"
  List1.AddItem "New Delhi"
  List1.AddItem "Kuala Lumpur"
  List1.AddItem "San Francisco"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this simple example about List box, we just simply pre-loaded the List box with items which we want to display. So we just use the following code:

List1.AddItem "Milan"
List1.AddItem "Paris"
List1.AddItem "Madrid"
List1.AddItem "Tokyo"
List1.AddItem "Manila"
List1.AddItem "New Delhi"
List1.AddItem "Kuala Lumpur"
List1.AddItem "San Francisco"

The List box has built-in methods for adding, removing and retrieving collection of items or data. In this case, we applied the built-in method for adding an item into the List box using the AddItem method. Now how about choosing an item from the List box and display it into the text box? Let us have an example about this particular requirement.

Example 12:

Design and develop a simple program that demonstrates how to preload the List box with items. When the user chooses an item at the list box by double-clicking it, that particular item will be displayed at the text box. Follow the given figure below in designing and developing the application program.

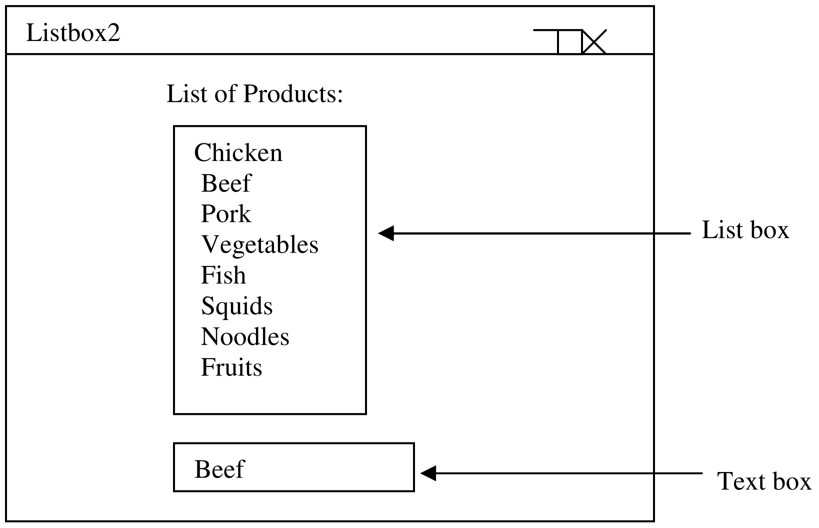


Figure 5.12 Displaying an item from List box into Text box

Note :

When the user double-clicks the Beef at the list box, the item "Beef" will be displayed at the text box.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox2.**
3. Double-click the List box , and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
  List1.Text = ""
  Text1.Text = ""
'Add items to the List box
 List1.AddItem "Chicken"
 List1.AddItem "Beef"
 List1.AddItem "Pork"
 List1.AddItem "Vegetables"
 List1.AddItem "Fish"
 List1.AddItem "Squids"
 List1.AddItem "Noodles"
 List1.AddItem "Fruits"
End Sub
```

```
Private Sub List1_DblClick()
 Text1.Text = ""
 Text1.Text = List1.Text
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this example, the user chooses the Beef product by double-clicking it using the mouse. That is why the product Beef is displayed at text box 1 (Text1).
You will notice that we initialize the list box and text box as empty using the empty string value. Here is the code that accomplishes the task:

```
 List1.Text = ""
 Text1.Text = ""
```

Then we preloaded the list box with the following list of products using the AddItem method:

```
'Add items to the List box
 List1.AddItem "Chicken"
 List1.AddItem "Beef"
 List1.AddItem "Pork"
 List1.AddItem "Vegetables"
 List1.AddItem "Fish"
 List1.AddItem "Squids"
 List1.AddItem "Noodles"
 List1.AddItem "Fruits"
```

And finally, we display the selected item into the text box with the following code:

```
Private Sub List1_DblClick()
 Text1.Text = ""
 Text1.Text = List1.Text  ←
End Sub
```

The Text property of the List box (List1) returns the value of the selected item. This is the reason why the selected item is displayed at text box 1 (Text1).

Example 13:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box and as well as add an item into the List box which is entered by the user from the text box. The program also will be able to remove all the item one by one, starting from the top down to the bottom. Follow the given figure below in designing and developing the application program.

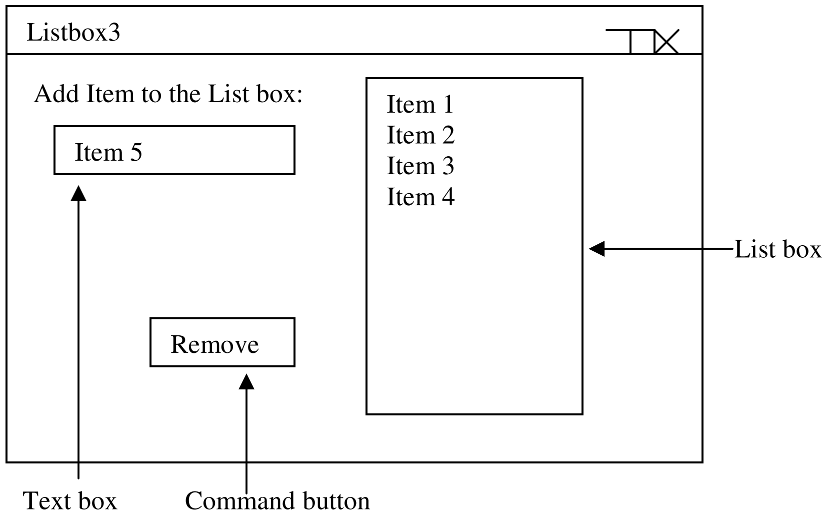


Figure 5.13 Adding and Removing an Item in the List box

Note:

When the user types the “Item 5” at the text box and press the Enter key, the “Item 5” should be added at the list box. Now if the user clicks the Remove command button, the Item 1 will be removed from the list box. And if the user continues to click the Remove command button, the next item that follows will be removed from the list box until the list box becomes empty.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox3.**
3. Double-click the Text box,, List box , Label and Command button in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately. Set the caption of the command button to **Remove**.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub cmdRemove_Click()
If List1.ListCount = 0 Then
  MsgBox "No More Item!"
Else
   List1.RemoveItem (0)
End If
End Sub
```

```
Private Sub Form_Load()
List1.AddItem "Item 1"
List1.AddItem "Item 2"
List1.AddItem "Item 3"
List1.AddItem "Item 4"
Text1.Text = " "
End Sub
```

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'Enter Key
   List1.AddItem Text1.Text
   Text1.Text = " "
   Text1.SetFocus
End If
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this example, we preloaded the list box with four items. It can be accomplished by the following code:

```
Private Sub Form_Load()
    List1.AddItem "Item 1"  ←
   List1.AddItem "Item 2"  ←
  List1.AddItem "Item 3"  ←
  List1.AddItem "Item 4"  ←
 Text1.Text = " "
End Sub
```

We also initialize our text box 1 (Text1) with an empty string so that when we run our program, the text box is empty. This is the reason why we have this code at the Form_Load procedure:

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

You will notice, that in our figure, the user types the “Item 5” to be added into the List box. The user should press the Enter key to confirm the input data. Once the Enter key is pressed, the “Item 5” will be added to the preloaded items of the list box. We can accomplish the task with the following code:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'Enter Key ←
  List1.AddItem Text1.Text ←
  Text1.Text = " "
  Text1.SetFocus
End If
End Sub
```

Remember that after we add the item into the list box, we have to clear out the text box for another input to be typed by the user. So we initialize again the text box 1 with an empty string. Then we have to set the focus of the cursor at the text box, so that the user can type right away the next item to be added. We can accomplish the task with the following code:

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

Now if we want to remove the items in the list box, starting from the top down to the bottom, we will have to click the Remove command button to do it. The code that can accomplish this task is the following:

```
Private Sub cmdRemove_Click( )
If List1.ListCount = 0 Then
 MsgBox "No More Item!"
Else
  List1.RemoveItem (0)←
End If
```

End Sub

You will notice that we remove the item starting from zero (0), so that the item which is located at location zero (means the first item in the list box) will be removed. And if any item which is located on that location is a candidate for removal. When the list box is running out of items, we have to display a warning message using the MsgBox statement. To know if the list box is empty, we have the command:

```
If List1.ListCount = 0 Then ←
  MsgBox "No More Item!"
Else
   List1.RemoveItem (0)
End If
```

because the ListCount is the property we use to determine the number of items in the List box. If it equals to zero, then the list box is already empty. This is the time that we will display the message: “No More Item!”
Removing an item from the top to bottom is not really a good idea. First, when we want to remove an item, we usually select one or two from the list of items presented. So the program above cannot satisfy on what we want. With this reason, we will come up with an example that will accomplish the task. Here it is!

Example 14:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The user should be able to remove a selected item on the list, one at a time. Follow the given figure below in designing and developing the application program.

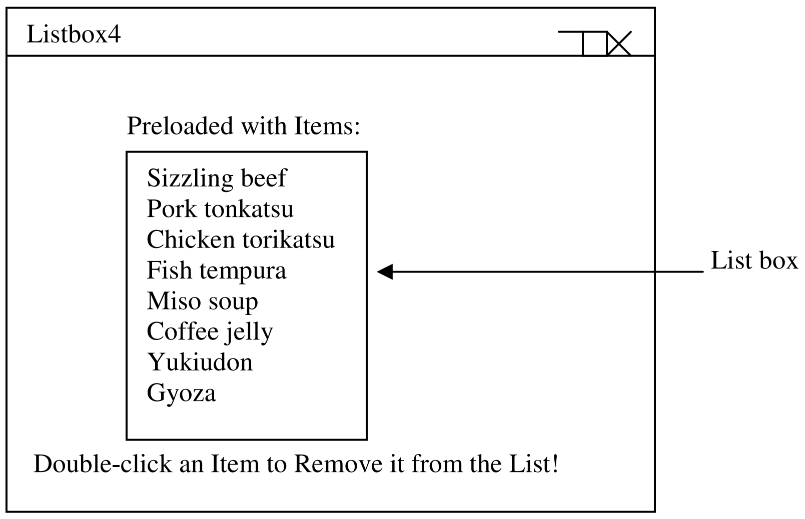


Figure 5.14 Remove an Item from a List box

Note:

When the user selects and double-clicks an item at the List box, that particular item must be deleted from the displayed list.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox4.**
3. Double-click the List box and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
  List1.AddItem "sizzling beef"
  List1.AddItem "pork tonkatsu"
  List1.AddItem "chicken torikatsu"
  List1.AddItem "fish tempura"
  List1.AddItem "miso soup"
  List1.AddItem "coffee jelly"
  List1.AddItem "yakiudon"
  List1.AddItem "gyoza"
End Sub
```

```
Private Sub List1_DblClick()
  List1.RemoveItem List1.ListIndex
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

First, we load the list box with items, using the following code:

Private Sub Form_Load( )
  List1.AddItem "sizzling beef"
  List1.AddItem "pork tonkatsu"
  List1.AddItem "chicken torikatsu"
  List1.AddItem "fish tempura"
  List1.AddItem "miso soup"

```
  List1.AddItem "coffee jelly"
  List1.AddItem "yakiudon"
  List1.AddItem "gyoza"
End Sub
```

Then we put a code in the sub-procedure List1_DblClick( ) with the following:

```
Private Sub List1_DblClick( )
 List1.RemoveItem List1.ListIndex ←
 End Sub
```

It's seems so simple, isn't it? Well, because we know how to do it. Let us dissect the code further. We apply here the RemoveItem method to remove the selected item from the list box. Now to get exactly the index position of the currently selected item, we use the ListIndex property.

What if we want to highlight first an item before confirming to delete it using a command button? That is a good idea to act! Consider the graphical user-interface (GUI) design in our next example.

Example 15:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to remove a selected item on the list, one at a time but with a confirmation using a command button. Follow the given figure below in designing and developing the application program.

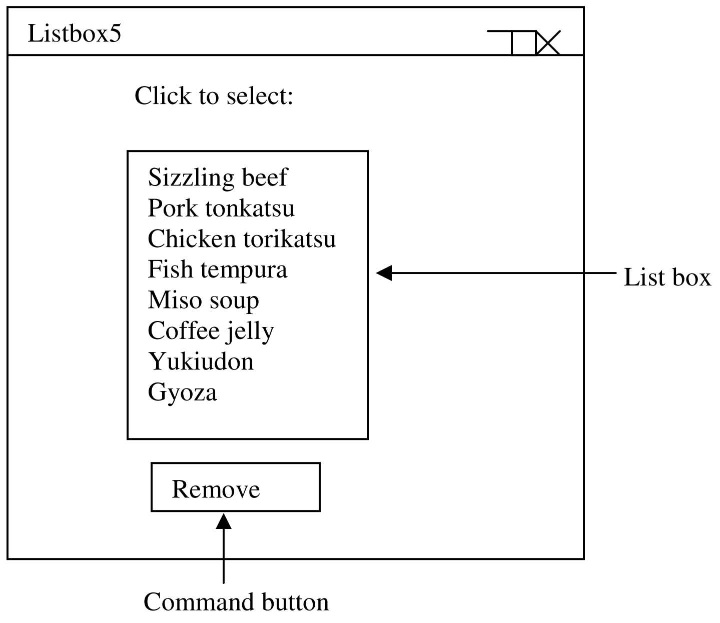


Figure 5.15 Removing a selected item using command button

Note:

The user should select first an item at the List box by highlighting it, before the user can finally remove it by clicking the Remove command button.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox5.**
3. Double-click the Text box,, List box , and Labels in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

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

```
Private Sub Form_Load()
  List1.AddItem "sizzling beef"
  List1.AddItem "pork tonkatsu"
  List1.AddItem "chicken torikatsu"
  List1.AddItem "fish tempura"
  List1.AddItem "miso soup"
  List1.AddItem "coffee jelly"
  List1.AddItem "yakiudon"
  List1.AddItem "gyoza"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

When the user selects an item in the list box by highlighting it through a mouse-click, that item is selected to be deleted. So the user will confirm to remove it finally by clicking the Remove command button. Now what we would do in our program is to transfer simply the code of the procedure List1_DblClick( ) to the Command1_Click( ) procedure this way:

Private Sub Command1_Click( ) ←
  List1.RemoveItem List1.ListIndex ←
End Sub

```
Private Sub Form_Load( )
  List1.AddItem "sizzling beef"
  List1.AddItem "pork tonkatsu"
  List1.AddItem "chicken torikatsu"
  List1.AddItem "fish tempura"
  List1.AddItem "miso soup"
  List1.AddItem "coffee jelly"
  List1.AddItem "yakiudon"
   List1.AddItem "gyoza"
End Sub
```

See? The modification in our program is so easy! Well, if we know how to do it, buddy. What we need is a simple analysis and apply our analysis to the program we create. There are times that when we select an item, we want to display that selected item into another list box. We will do this kind of program in our next example. Are you ready?

Example 16:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to display one or more selected items on another list box. Follow the given figure below in designing and developing the application program.

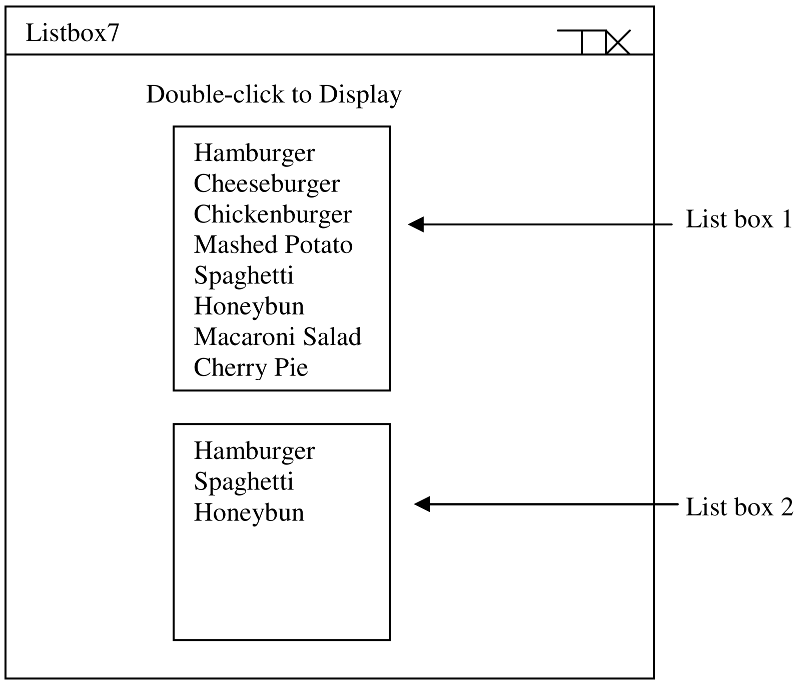


Figure 5.17 Display one or more items to the other list box

Note:

When the user selects and double-clicks an item at the List box 1, that particular item must be displayed at the List box 2.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox7.**
3. Double-click two List boxes and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
  List1.AddItem "Hamburger"
  List1.AddItem "Cheeseburger"
  List1.AddItem "Chickenburger"
  List1.AddItem "Mashed Potatoe"
  List1.AddItem "Spaghetti"
  List1.AddItem "Honeybun"
  List1.AddItem "Macaroni Salad"
  List1.AddItem "Cherry Pie"
End Sub
```

```
Private Sub List1_DblClick()
  'Add item to List box 2
   List2.AddItem List1.Text
End Sub
```

5.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Here in our example, the user selected and double-clicked the items “Cheeseburger”, “Spaghetti”, and “Honeybun” in list box 1. That is why they are displayed at list box 2. When the user selects and double-clicks other collection of items from list box 1, these particular collection of items will be displayed at the list box 2.
As what we have done before, we preloaded our list box 1 with items on it, using the following code:

```
Private Sub Form_Load( )
  List1.AddItem "Hamburger"
  List1.AddItem "Cheeseburger"
  List1.AddItem "Chickenburger"
  List1.AddItem "Mashed Potatoe"
  List1.AddItem "Spaghetti"
  List1.AddItem "Honeybun"
  List1.AddItem "Macaroni Salad"
  List1.AddItem "Cherry Pie"
End Sub
```

Next, we add the selected items from List box 1 into List box 2, using the following code:

```
Private Sub List1_DblClick( )
  'Add item to List box 2
   List2.AddItem List1.Text ◄—
End Sub
```

It seems so simple, isn't it? Well, because we knew how to do it.
There are times that we want to transfer an item from one list box to another. How are we able to do that? In our next example, we will accomplish this particular task:

Example 17:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to transfer one or more selected items on another list box. Follow the given figure below in designing and developing the application program

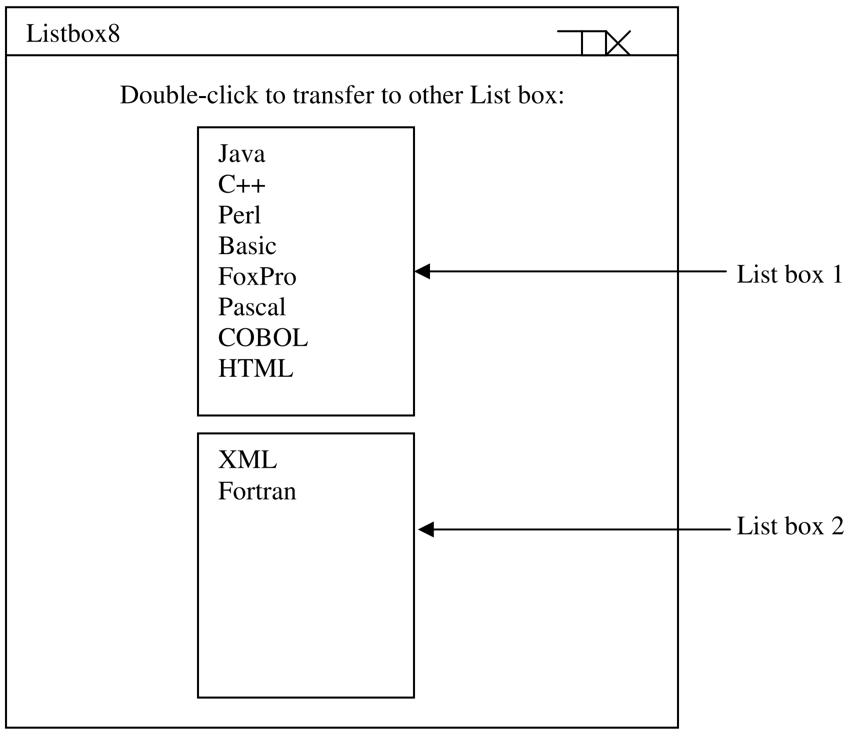


Figure 5.18 Transfer one or more items to the other list box

Note:

When the user selects and double-clicks an item at the List box 1, that particular item must be transferred into List box 2.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox8.**
3. Double-click two List boxes and Label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
  List1.AddItem "Java"
  List1.AddItem "C++"
  List1.AddItem "Perl"
  List1.AddItem "Basic"
  List1.AddItem "FoxPro"
  List1.AddItem "Pascal"
  List1.AddItem "COBOL"
  List1.AddItem "HTML"
  List1.AddItem "XML"
  List1.AddItem "Fortran"
End Sub
```

```
Private Sub List1_DblClick()
  'Add the item to the other List box
  List2.AddItem List1.Text
  'Remove  the item from this List box
  List1.RemoveItem List1.ListIndex
End Sub
```

```
Private Sub List2_DblClick()
  'Add the item to the other List box
  List1.AddItem List2.Text
  'Remove  the item from this List box
  List2.RemoveItem List2.ListIndex
End Sub
```

5.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Here in our example, the user double-clicks the items “XML” and “Fortran” in list box 1. That is why these items are transferred to list box 2, as what we can see in the figure. Now if the user double-clicks again an item in list box 1, this particular item will be transferred to list box 2. In the case where the user double-clicks an item in list box 2, this particular item will be transferred to list box 1. You can try it, to see for yourself how this program works.
As what we have done previously in our program, to preload the list box with items upon running, we use the following code:

```
Private Sub Form_Load( )
 List1.AddItem "Java"
 List1.AddItem "C++"
 List1.AddItem "Perl"
 List1.AddItem "Basic"
 List1.AddItem "FoxPro"
 List1.AddItem "Pascal"
 List1.AddItem "COBOL"
 List1.AddItem "HTML"
 List1.AddItem "XML"
 List1.AddItem "Fortran"
End Sub
```

Then we embed the following code at List box 1 object and in its Double-click event:

```
Private Sub List1_DblClick( )
  'Add the item to the other List box
  List2.AddItem List1.Text ←
  'Remove  the item from this List box
  List1.RemoveItem List1.ListIndex ←
End Sub
```

to add the selected items from list box 1 (List1) into list box 2. At the same time, we have to remove that selected item from the list box 1 using the RemoveItem method and the ListIndex property of the List object.
You will notice that this code is similarly applied to List box 2, as what we can see here in the following code that can be found at List box 2 procedure:

```
Private Sub List2_DblClick( )
  'Add the item to the other List box
  List1.AddItem List2.Text
  'Remove  the item from this List box
  List2.RemoveItem List2.ListIndex
End Sub
```

We did this so that vise-versa, they will work the same such as when we transfer an item from list box 1 into list box 2, the selected item will also be deleted from list box 1. Now if we want to transfer an item from list box 2 into list box 1, the selected item will also be deleted from list box 2.
In the next example, we will know what particular item is currently selected by the user at the List box. Then we will confirm it by displaying a description of that particular item at the text box.

Example 18:

Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to get the value of the currently selected item in the list box. Follow the given figure below in designing and developing the application program:

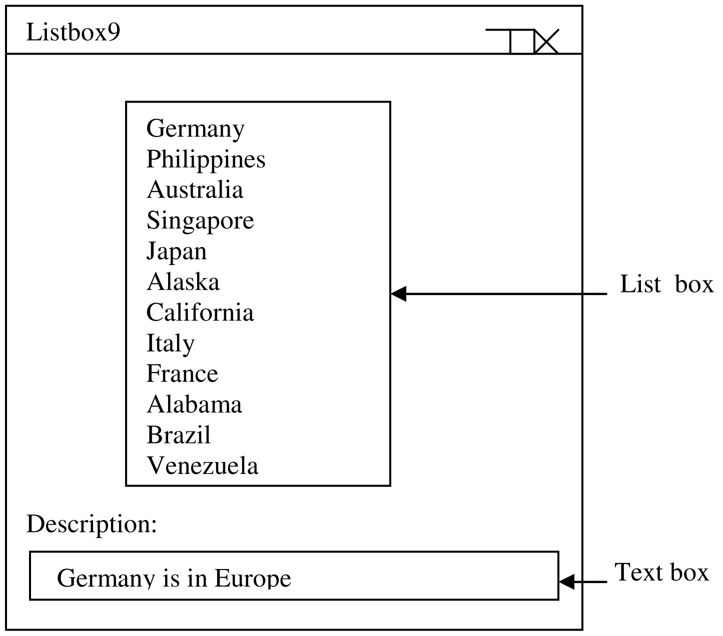


Figure 5.19 Determining what item is currently selected

Note:

When the user clicks one of the item in the List box, the description of that item is displayed at the text box.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox9.**
3. Double-click a List box, a Label and a Text box in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.

4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Form_Load()
 Text1.Text = " "
 List1.AddItem "Germany"
 List1.AddItem "Philippines"
 List1.AddItem "Singapore"
 List1.AddItem "Japan"
 List1.AddItem "Alaska"
 List1.AddItem "California"
 List1.AddItem "Italy"
 List1.AddItem "France"
 List1.AddItem "Alabama"
 List1.AddItem "Brazil"
 List1.AddItem "Venezuela"
End Sub
```

```
Private Sub List1_Click()
If List1.Text = "Germany" Then
   Text1.Text = " "
   Text1.Text = "Germany is in Europe"
ElseIf List1.Text = "Philippines" Then
   Text1.Text = " "
   Text1.Text = "Philippines is in Asia"
ElseIf List1.Text = "Singapore" Then
   Text1.Text = " "
   Text1.Text = "Singapore is in Asia"
ElseIf List1.Text = "Japan" Then
   Text1.Text = " "
   Text1.Text = "Japan is in Asia"
ElseIf List1.Text = "Alaska" Then
   Text1.Text = " "
   Text1.Text = "Alaska is in America"
ElseIf List1.Text = "California" Then
   Text1.Text = " "
   Text1.Text = "California is in America"
ElseIf List1.Text = "Italy" Then
   Text1.Text = " "
   Text1.Text = "Italy is in Europe"
ElseIf List1.Text = "France" Then
   Text1.Text = " "
   Text1.Text = "France is in Europe"
ElseIf List1.Text = "Alabama" Then
   Text1.Text = " "
   Text1.Text = "Alabama is in America"
ElseIf List1.Text = "Brazil" Then
   Text1.Text = " "
   Text1.Text = "Brazil is in Latin America"
ElseIf List1.Text = "Venezuela" Then
   Text1.Text = " "
   Text1.Text = "Venezuela is in Latin America"
End If
End Sub
```

5.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this example, the user selected the item Germany in the list box, thus the description displayed at the text box is “Germany is in Europe”. Now when the user selects another item, its corresponding description will also be displayed at the text box. The code behind how this one happens is here below:

```
Private Sub List1_Click( )
If List1.Text = "Germany" Then
  Text1.Text = " "
  Text1.Text = "Germany is in Europe"
ElseIf List1.Text = "Philippines" Then
  Text1.Text = " "
  Text1.Text = "Philippines is in Asia"
ElseIf List1.Text = "Singapore" Then
  Text1.Text = " "
  Text1.Text = "Singapore is in Asia"
ElseIf List1.Text = "Japan" Then
  Text1.Text = " "
  Text1.Text = "Japan is in Asia"
ElseIf List1.Text = "Alaska" Then
  Text1.Text = " "
  Text1.Text = "Alaska is in America"
ElseIf List1.Text = "California" Then
  Text1.Text = " "
  Text1.Text = "California is in America"
ElseIf List1.Text = "Italy" Then
  Text1.Text = " "
  Text1.Text = "Italy is in Europe"
ElseIf List1.Text = "France" Then
  Text1.Text = " "
  Text1.Text = "France is in Europe"
ElseIf List1.Text = "Alabama" Then
  Text1.Text = " "
  Text1.Text = "Alabama is in America"
ElseIf List1.Text = "Brazil" Then
  Text1.Text = " "
  Text1.Text = "Brazil is in Latin America"
ElseIf List1.Text = "Venezuela" Then
  Text1.Text = " "
  Text1.Text = "Venezuela is in Latin America"
End If
End Sub
```

using the conditional *Elseif* statement. The Text property of the list box item (List1) always corresponds to a list item a user selects at run time. In other words, the Text property contains the currently selected item in the List1 list box. The *Elseif* code above checks to see if "Germany" has been selected by the user and, if so, displays the description in the text box.
You will notice also that every time we display another description at the text box, we will initialize it with an empty string. We did this to clear entirely the text box, before another description will be displayed. Otherwise, the previous string which has a longer length than the present one might display the last part of it (as though it is concatenated to the new string).
In our next example, we will present a better idea in presenting a list of items and in how to add or remove them easily with confirmation using command buttons. Here it is now!

Example 19:

Design and develop a simple program that should apply the AddItem, RemoveItem, and Clear methods with the ListIndex and ListCount properties to add, remove and clear the list entries in the List box at run time. Follow the given figure below in designing and developing the application program:

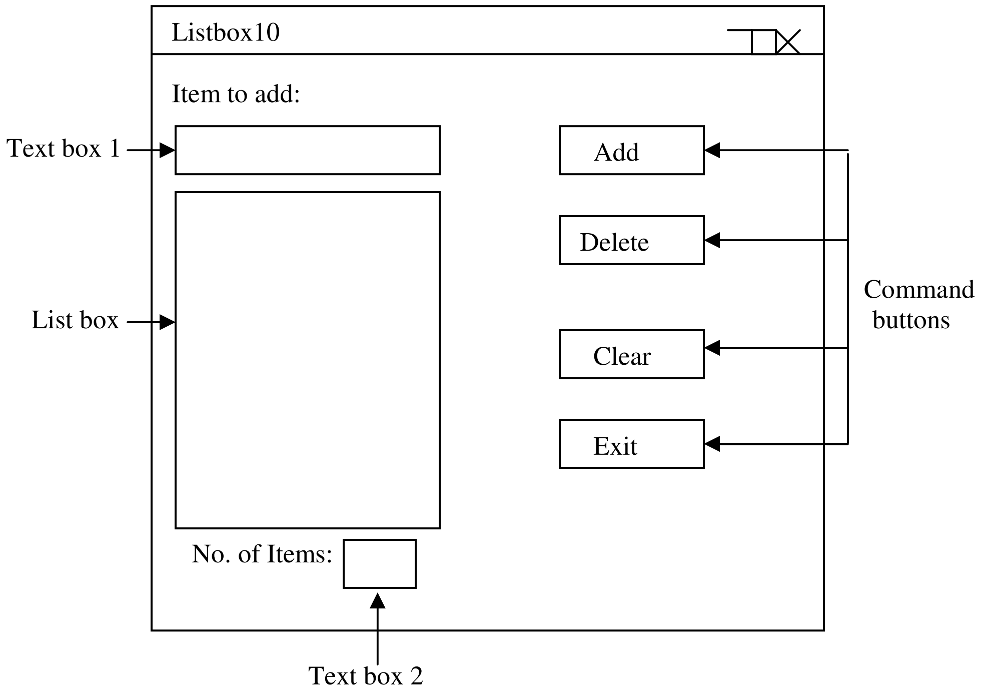


Figure 5.20 Adding, Removing and Clearing an item in the list box

Note:

The user will enter an item first at text box 1 and then clicks the Add command button to add the inputted item in the List box. Now when the user wants to remove an item from the list, the user should highlight it first, then clicks the Remove command button to finally delete it. Clicking the Clear command button will wipe out all the items in the List box.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox10.**
3. Double-click a text box, a list box, labels, and four command buttons in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately. The following table lists the property settings for the four command buttons in the application program to track them easily.

Command button objects	**Property**	**Setting**
Add command button	Name	cmdAdd
	Caption	Add
Delete command button	Name	cmdRemove
	Caption	Remove
Clear command button	Name	cmdClear
	Caption	Clear
Exit command button	Name	cmdExit
	Caption	Exit

It is very hard to track what command button is currently in use in our program because there are four of them. So we have to assign an object name for each command button to easily identify them inside our code. You will assign the object name and its respective caption at the Property box.

4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub cmdAdd_Click()
 'Add to the List box
 List1.AddItem Text1.Text
 'Clear the text box
 Text1.Text = " "
 Text1.SetFocus
 'Display the number of items
 'to text box 2
 Text2.Text = List1.ListCount
End Sub
```

```
Private Sub cmdClear_Click()
 'Empty the List box
 List1.Clear
 'Disable Delete command button
 cmdDelete.Enabled = False
 'Display the number of items
 'to the text box 2
 Text2.Text = List1.ListCount
End Sub
```

```
Private Sub cmdDelete_Click()
 Dim Ind As Integer
 'Get the index
 Ind = List1.ListIndex
 If Ind >= 0 Then
   'Remove it from the List box
   List1.RemoveItem Ind
   'Display the number of items
   'to the text box 2
   Text2.Text = List1.ListCount
 Else
    'Generate a warning sound
    Beep
 End If
 'Disable (dimmed) button if no items
 'on the List box
 cmdDelete.Enabled = (List1.ListIndex <> -1)
End Sub
```

```
Private Sub cmdExit_Click()
 Unload Me
End Sub
```

```
Private Sub Form_Load()
 Text1.Text = " "
 Text2.Text = " "
End Sub
```

```
Private Sub List1_Click()
 cmdDelete.Enabled = List1.ListIndex <> -1
End Sub
```

```
Private Sub Text1_Change()
 'Enable the Add button if at least
 'one character in the text box 1
 cmdAdd.Enabled = (Len(Text1.Text) > 0)
End Sub
```

5.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Our program is a little bit long and complex. To understand it demands patience and tenacity. We will dissect it so that we can divide the complexity into manageable pieces and hopefully we can conquer it piece by piece. Let us start it now. First, we will learn how the code behind the Command button works, because this is the first object that we will click with after entering an item at the text box and finally add it into a List box. Here is its code below:

```
Private Sub cmdAdd_Click( )
   'Add to the List box
   List1.AddItem Text1.Text ←
  'Clear the text box
  Text1.Text = " "  ←
 Text1.SetFocus   ←
 'Display the number of items
 'to text box 2
 Text2.Text = List1.ListCount ←
End Sub
```

You will notice that we will simply add the item entered at text box 1 into the List box with the following code:

```
List1.AddItem Text1.Text
```

After adding the item into the list box, we will clear the text box for another item to be entered by the user. This can be accomplished simply with the following code:

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

Now we need to set the focus of the cursor at text box 1, so that the user can keep on typing any items she wants to enter (otherwise the cursor will position itself at the next object that follows the text box, which we don't want to happen). Here is the code that can accomplish that task:

```
Text1.SetFocus
```

Finally, we can determine how many items are already added at the List box using the ListCount property of List box object. We will display the number of items at text box 2 using the following code:

```
   Text2.Text = List1.ListCount
```

This time, we will dissect the Delete command button and study how the code behind it works. Here is its code:

```
Private Sub cmdDelete_Click( )
 Dim Ind As Integer
 'Get the index
 Ind = List1.ListIndex ←
 If Ind >= 0 Then
   'Remove it from the List box
   List1.RemoveItem Ind ←
   'Display the number of items
   'to the text box 2
   Text2.Text = List1.ListCount ←
 Else
    'Generate a warning sound
    Beep
 End If
 'Disable (dimmed) button if no items
 'on the List box
 cmdDelete.Enabled = (List1.ListIndex <> -1)
End Sub
```

If we want to know the position of the selected item in a list box, we use the ListIndex property. This property sets or returns the index of the currently selected item in the List box and is available only at run time. The value of this property is 0 if the first or top item in the List box is selected. It is 1 if the next item down is selected, and so on. The value of ListIndex is -1 (negative one) if no item is selected.
Here we test if the list box contains any item with the following code:

```
   Ind = List1.ListIndex
  If Ind >= 0 Then
```

We know that the first or top item has a ListIndex number of 0, therefore if the ListIndex of a list box is greater than or equal to zero, it has an item or items on it. So we can delete it. We can delete the selected item using the following code:

```
   List1.RemoveItem Ind
```

where Ind is the ListIndex number and at the same time the currently selected item in the List box. This is actually the item selected by the user by highlighting it.

After removing an item at the list box, we need to update the number of items remaining to text box 2 with the following code:

```
  Text2.Text = List1.ListCount
```

Remember that the ListCount property returns the number of items in a list box, so that's what we are using here to be stored at text box 2 (Text2).
Now if the list box is running out of items inside it, we should generate a warning sound using the *Beep* command. And finally we have to dim the Delete command button, meaning it is disabled so that the user can no longer click it or use it, using the following code:

```
 cmdDelete.Enabled = (List1.ListIndex <> -1)
                                          ↑
                                      expression
```

If the above expression is False, then the command button Delete will be disabled. In other words, the Delete command button is like being hard-coded with the following code:

```
  cmdDelete.Endabled = False
```

Next, we will examine the code behind the Clear command button. Here is it!

```
Private Sub cmdClear_Click( )
 'Empty the List box
 List1.Clear ←
 'Disable Delete command button
 cmdDelete.Enabled = False ←
 'Display the number of items
 'to the text box 2
 Text2.Text = List1.ListCount ←
End Sub
```

Using the Clear property of the list box, we can wipe out all the items in the list box. Since the list box is now empty, we have to disable the command button Delete so that the user is restricted to use or click it. At the same time, we have to update text box 2 (Text2) with the number of items in the list box. In this case, since all the items are wiped out, the number 0 is displayed at text box 2 (Text2). Try it!
This time, we will go to the code behind the Exit command button. We have here its code below:

```
Private Sub cmdExit_Click( )
   Unload Me ←
End Sub
```

It's very simple, isn't it? This Unload Me statement simply frees up the computer's memory for all the objects we have used in this application program. All the objects that we have used in this program consumed more memory space, so we need to free up the memory of the computer. This memory is commonly known as RAM (Random Access Memory).
Now we will examine the code behind the text box 1 (Text1) object. Here is its code:

```
  Private Sub Text1_Change( )
    'Enable the Add button if at least
   'one character in the text box 1
   cmdAdd.Enabled = (Len(Text1.Text) > 0) ←
End Sub
```

Here, we simply enable the command button Add if the text box 1 (Text1) contains at least one character. Meaning the user is at least entering something in text box 1 which triggers the program to enable the Add command button. Enabling the Add command button makes it available for the user to click it.
Now we need to know also that the user has selected an item in the list box. To determine it, we put the following code in the application program:

```
Private Sub List1_Click( )
   cmdDelete.Enabled = List1.ListIndex <> -1 ←
End Sub                                 ↑

                                   expression
```

The expression above must be evaluated to True for the command button Delete to be enabled, otherwise that command button is dimmed (meaning disabled),thus the user cannot use or click it.
 Now remember that in our previous discussion, the -1 (negative one) means there is no currently selected item in the list box. So if the ListIndex property is not equal to -1 (negative one), then the user has selected an item.
Here in our code below, we simply initialize the Text box and List box with an empty string so that at run time, they contain nothing.

```
Private Sub Form_Load( )
 Text1.Text = " "
 Text2.Text = " "
End Sub
```

The code above will be executed upon loading the application program. Finally, we have ended our lengthy and very challenging discussion. I hope you enjoyed it, buddy!
What we have is a program that can select an item at the List box, one at a time. How about a program that can select two or more items in the list box at once. That will be our topic in the next example. Are you excited to learn it?

Example 20:

Design and develop a simple program that should be able to select two or more items in the List box at once and be able to transfer the collection of selected items to another list box simultaneously. Follow the given figure below in designing and developing the application program:

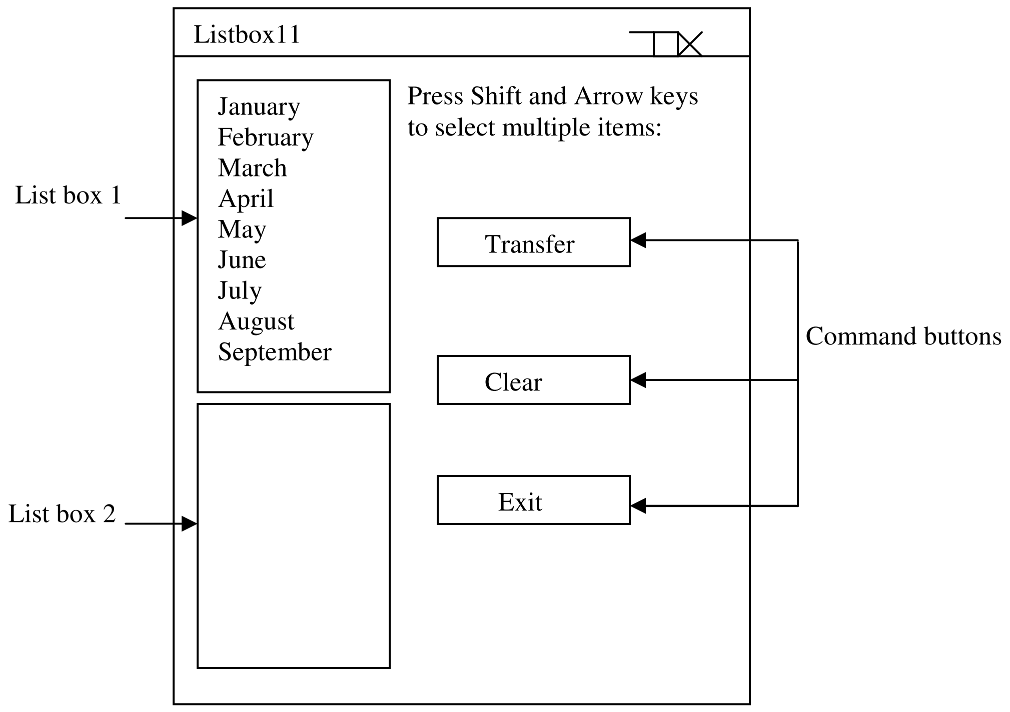


Figure 5.21 Selecting and Transferring multiple items at the same time

Note:

The user will first click the first item to select, then use the Shift and Arrow keys to select other multiple items at the list box 1. Then clicking the Transfer command button will copy the collection of selected items into the list box 2.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Listbox11.**
3. Double-click two list boxes, a label, and three command buttons in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately. The following table lists the property settings for the three

command buttons and of the list box in the application program to easily track them.

Objects	Property	Setting
List box 1	Name	List1
	MultiSelect	2-Extended
Transfer command button	Name	cmdTransfer
	Caption	Transfer
Clear command button	Name	cmdClear
	Caption	Clear
Exit command button	Name	cmdExit
	Caption	Exit

It is very hard to track what command button is currently in use in our program because there are three of them. So we have to assign an object name for each command button to easily identify them inside our code. You will assign the object name and its respective caption at the Property box.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub cmdClear_Click()
 List2.Clear
 cmdClear.Enabled = False
 End Sub
```

```
Private Sub cmdExit_Click()
 Unload Me
End Sub
```

```
Private Sub cmdTransfer_Click()
  For i = 0 To (List1.ListCount - 1)
  'If selected, add to the List box
    If List1.Selected(i) = True Then
       List2.AddItem List1.List(i)
    End If
  Next
  cmdClear.Enabled = True
End Sub
```

```
Private Sub Form_Load()
 List1.AddItem "January"
 List1.AddItem "February"
 List1.AddItem "March"
 List1.AddItem "April"
 List1.AddItem "May"
 List1.AddItem "June"
 List1.AddItem "July"
 List1.AddItem "August"
 List1.AddItem "September"
 List1.AddItem "October"
 List1.AddItem "November"
 List1.AddItem "December"
 'Pre-select a collection of items
 List1.Selected(0) = True
 List1.Selected(1) = True
 List1.Selected(2) = True
End Sub
```

```
Private Sub List1_DblClick()
  'Click the transfer button
  cmdTransfer.Value = False
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Warning!

Don't forget to set the Multi-Select property of list box 1 with the 2-Extended setting. It is very important, otherwise our program will not run on our intended purpose.

Explanation:

In this program, we preselected three items, they are the first three items at the top of the list in our list box. To accomplish this task, we use the following code:

```
Private Sub Form_Load( )
 List1.AddItem "January"
 List1.AddItem "February"
 List1.AddItem "March"
 List1.AddItem "April"
 List1.AddItem "May"
 List1.AddItem "June"
 List1.AddItem "July"
 List1.AddItem "August"
 List1.AddItem "September"
 List1.AddItem "October"
 List1.AddItem "November"
```

```
  List1.AddItem "December"
  'Pre-select a collection of items
  List1.Selected(0) = True ←
  List1.Selected(1) = True ←
  List1.Selected(2) = True ←
End Sub
```

We are using here the Selected( ) property which is a Boolean array containing the selection status of a list box - to determine which items are preselected. Each entry in the array corresponds to a list item and is set to True if the item is selected, or False if it is not selected. After the user selects items from the list box 1, each array entry is checked to see if it is set (meaning it is True). If so, the entry is added to the list box 2. You can try clicking the Transfer command button to see the effect of these preselected items. You will see that the top three items in the list box 1 will be transferred or copied into list box 2.
Next, we will examine the code behind the Transfer command button:

```
Private Sub cmdTransfer_Click()
  For i = 0 To (List1.ListCount - 1) ←
  'If selected, add to the List box
    If List1.Selected(i) = True Then
      List2.AddItem List1.List(i) ←
    End If
  Next
  cmdClear.Enabled = True ←
End Sub
```

In the code above, we apply the For loop statement to determine if the items in list box 1 are selected. This loop statement will navigate the entire content of list box 1 and check if the item or items are selected. If so, then they will be copied to list box 2 (List2). We also set the Clear command button to True so that the user can click it. Clicking the Clear command button will wipe out all the items in list box 2.
Now let us examine the code behind the Clear command button:

```
Private Sub cmdClear_Click()
 List2.Clear ←
 cmdClear.Enabled = False ←
 End Sub
```

Using the Clear property of the list box, we can wipe out all the items loaded at the list box at once. We need also to disable the Clear command button after we wiped out all the items, so that the user is restricted to click it. We can accomplish this by setting its value to False.
What if the user double-clicks an item or group of items at the list box? We need to assign a False value to the Transfer command button with the following code:

```
Private Sub List1_DblClick( )
  'Click the transfer button
  cmdTransfer.Value = False ←
End Sub
```

so that the user is prevented to copy an item from list box 1 into list box 2. In other words, the user has no other option to transfer an item but by using the Transfer command button only.
Lastly, we will examine the code behind the Exit command button:

```
Private Sub cmdExit_Click( )
   Unload Me
End Sub
```

This Unload Me statement simply frees up the computer's memory for all the objects that we have used in this application program. All the objects that we have used in this program consumed more memory space, so we need to free up the memory of the computer. This memory is commonly known as RAM (Random Access Memory).

Using Combo Boxes

The words **combo box** are derived from the words "combination box", meaning it combines the features of both a text box and a list box. This control allows the user to select an item either by typing text into the combo box, or by selecting it from the list. In other words, it allows us to select a predefined item from a list or to enter a new item which is not in the list. Like a list box, a combo box present a list of choices to the user. If the number of items exceeds what can be displayed in the combo box, scroll bars will automatically appear on the control. We can then scroll up and down or left to right through the list to navigate it.

Advantages of the Combo Box over List Box

A potential problem with a list box is that in some situations, you are stuck with the entries displayed. You can't directly edit an item in the list or select an entry that's not already there. Though its big advantage compared to combo box is that if you want to restrict the user, then a list box is best suited in this situation. While the strength of a combo box is that it takes less room on a form than a normal list box, because the full list is not displayed until the user clicks the down arrow. In other words, a combo box can easily fit in a small space where a list box would not fit. Moreover, a combo box contains an edit field, so choices not on the list can be typed in this field by the user.
In most program development situations, a combo box is appropriate when there is a list of suggested choices, and a list box is appropriate when you want to limit input to what is on the list.

Example 21:

Design and develop a simple application system that accepts and lists the items inputted by the user in the combo box. Follow the given figure below in designing and developing the application program:

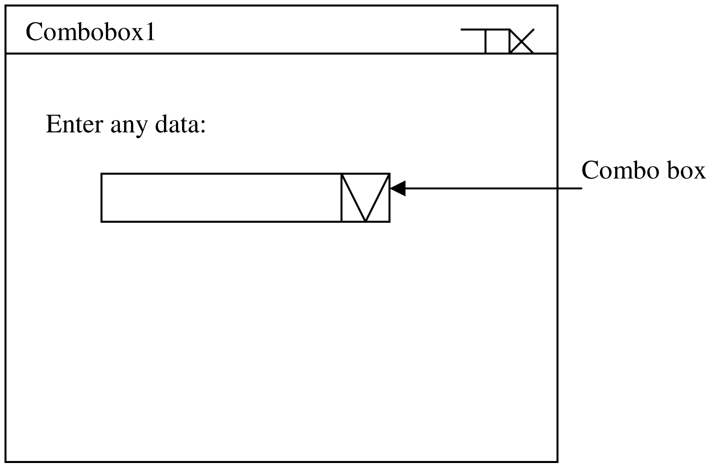


Figure 5.22 Entering and displaying the items on the Combo box

Note:

When the user inputs an item at the Combo box, this item will be added on it. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously entered by the user.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox1.**
3. Double-click a combo box, and a label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'When Enter Key is pressed
  Combo1.AddItem Combo1.Text
  Combo1.Text = " "
End If
End Sub
```

```
Private Sub Form_Load()
Combo1.Text = " "
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Try to input the data: Item1, then Item 2, Item 3, and Item 4 and so on. Now, inspect all the items that you have entered by clicking the down arrow of the Combo box. Can you see now all the items you have entered? Now try to enter Item 1 and Item 2 again. You will notice that when you click the down arrow, Item 1 and Item 2 appears twice.
There are times, that you would like to design and develop a Combo box which will accept only those data that are not yet inputted at the Combo box. With this design, we can only have unique items in our combo box. Well, in this case, it demands more programming skills. We will apply here the looping statement and conditional statement (which we have learned in our previous examples) to accomplish such requirement.
This time we will dissect the code in our application system. Here are the first part:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'When Enter Key is pressed ←
  Combo1.AddItem Combo1.Text ←
  Combo1.Text = " " ←
End If
End Sub
```

We applied here the *if* conditional statement to test if the Enter key was pressed by the user. And if so, the item entered at the combo box will be added on it. Then after adding the data into the combo box, we have to store an empty string into the combo box to clear it, so that the user can easily enter another data into it.
Now let us dissect the last sub-procedure:

```
Private Sub Form_Load( )
Combo1.Text = " " ←
End Sub
```

Every time the form is loaded (every time the application system is run), we will clear the combo box by storing an empty string on it. Seems so easy, isn't it?

Example 22:

Design and develop an application system that accepts and lists the unique items inputted by the user in the combo box. Follow the given figure below in designing and developing the application program:

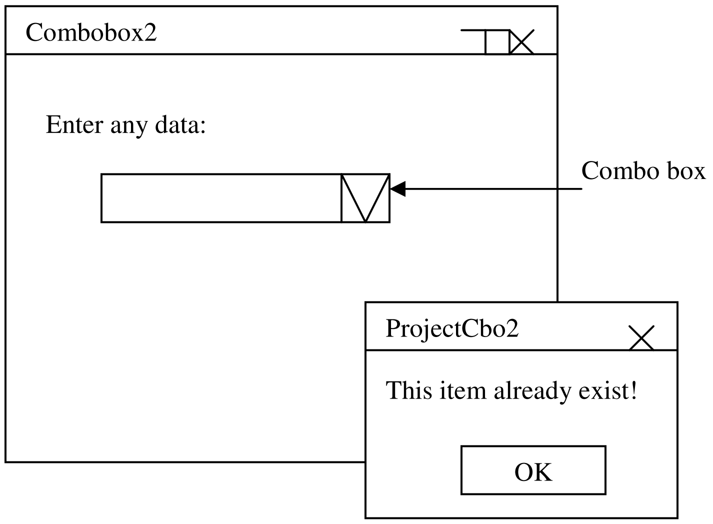


Figure 5.23 Entering and displaying unique items on the Combo box

Note:

When the user inputs an item at the Combo box, this item will be added on it, as long as it is unique from the items already entered on the list. Otherwise, the message box will appear informing the user that the item she entered already exists. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously entered by the user.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox2.**
3. Double-click a combo box, and a label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private lExist As Boolean
Private Sub Combo1_KeyPress(KeyAscii As Integer)
For i = 0 To (Combo1.ListCount - 1)
```

```
  If Combo1.List(i) = Combo1.Text Then
     lExist = True
     MsgBox "This item already exist!"
     Combo1.Text = ""
     lExist = False
     Exit Sub
  End If
Next
If Not lExist And KeyAscii = 13 Then  'Enter Key
  Combo1.AddItem Combo1.Text
  'This initialization ("") must have
  ' no space in between, otherwise
  'the item 1 will be duplicated
  Combo1.Text = ""
End If
End Sub
```

```
Private Sub Form_Load()
'Making the text box of the Combo box
' empty or blank
 Combo1.Text = ""
'Assuming the value isn't in the list
  lExist = False
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Try to input Item 1, then Item 2, Item3, Item 4 and so on. Now, inspect all the items that you have inputted by clicking the down arrow of the Combo box. Can you see now all the items?
This time, try to input the same item such as Item 2 or Item 3. See? The program had informed you that the item had already existed. With this design, we can be sure that all the items are unique from each other. This is very helpful if you want to input data such as employee number, account number, student number, or SSS number. These numbers should be unique for each person or applicant.
Okay, we have to dissect the code of this application program:

Private lExist As Boolean
Private Sub Combo1_KeyPress(KeyAscii As Integer)
For i = 0 To (Combo1.ListCount - 1) ⟵
 If Combo1.List(i) = Combo1.Text Then ⟵
 lExist = True
 MsgBox "This item already exist!"

```
    Combo1.Text = ""
    lExist = False
    Exit Sub ←
  End If
Next
If Not lExist And KeyAscii = 13 Then  'Enter Key ←
  Combo1.AddItem Combo1.Text
  'This initialization ("") must have
  ' no space in between, otherwise
  'the item 1 will be duplicated
  Combo1.Text = ""
End If
End Sub
```

Here in the code above, we had used two private variables: the first one is a variable that will store a Boolean (logical) value which can be True or False.The name of the variable is *lExist*. The second one is a variable that will control the looping statement. We name it simply as *i*. You will notice that this variable *i* is not formally declared at the top of our program and yet it didn't produce a syntax error during Visual Basic compilation process. This means that we can use a variable without declaring it formally. However, this kind of programming style is prone to logical errors which are very hard to detect or debug. So do your very best to declare all variables used in the program formally. The ListCount property of the Combo box control is our way of determining the maximum items that are already entered in the Combo box. This is the very reason why we make it as our ending iteration number, while we use the value 0 as our starting iteration number for our loop. We use the ListCount -1 because in Visual Basic, the first item in the combo box (as well as also in list box) is stored in location 0, therefore the last item's location is always -1 (minus one).
The List property of the combo box is our way of determining the current item (as the loop iterates) if it is equal to the entered data. If it is equal, then we have to store the Boolean value True to the logical variable lExist, so that the conditional statement that is placed after the *for* loop will not be executed. Remember that the associated statement of this *if* conditional statement (which is outside the *for* loop statement) will add the item into the Combo box, once it is evaluated to True. It can only be evaluated to True if the lExist logical variable contains the value of False and the Enter key was pressed.
The Exit Sub statement is our way to force the program pointer to jump out from the whole procedure if the *if* conditional statement (inside the *for* loop statement) is evaluated to True, meaning there is a similar item already existed on the list. In this way, the remaining code at the procedure can no longer be executed.
The next short procedure will only initialize the value of our combo box object:

```
Private Sub Form_Load( )
'Making the text box of the Combo box
' empty or blank
 Combo1.Text = ""
'Assuming the value isn't in the list
```

```
  lExist = False ←
End Sub
```

upon running our program. Our first assumption is that the item which is entered by the user in the combo box is not yet on the list. This is the reason why we stored the False value into the logical variable lExist.
We are now pretty much involved to the idiosyncrasies of our program. Well, it is enjoyable, isn't it? I beat, it is!

Example 23:

Design and develop a simple application system that accepts and lists the items inputted by the user in the combo box, together with the preloaded items. Follow the given figure below in designing and developing the application program:

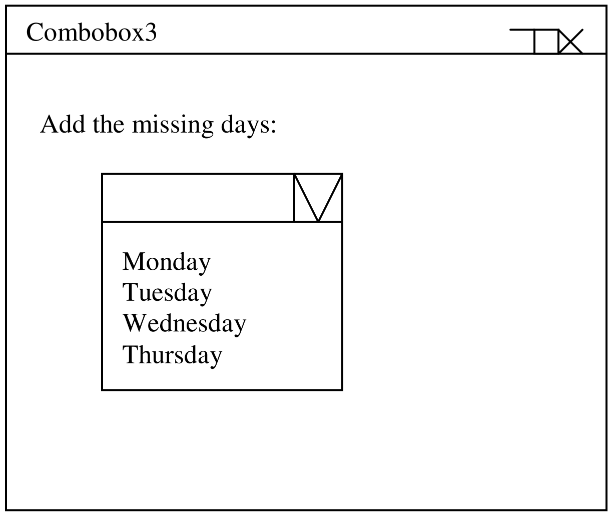


Figure 5.24 Entering and displaying items on the Combo box

Note:

When the user inputs an item at the Combo box, this item will be added on it. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously entered by the user.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox3.**

3. Double-click a combo box, and a label in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'When Enter Key is pressed
  Combo1.AddItem Combo1.Text
  Combo1.Text = " "
End If
End Sub
```

```
Private Sub Form_Load()
 Combo1.Text = " "
 Combo1.AddItem "Monday"
 Combo1.AddItem "Tuesday"
 Combo1.AddItem "Wednesday"
 Combo1.AddItem "Thursday"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

You can see here in our figure that the days: Friday, Saturday and Sunday are missing. So the user can input them at the combo box to add them to the existing days of the week. Try to input these days, then click the down arrow to see if they are added to the list. Can you see them now?
See? This is the biggest advantage of the combo box over the list box. Since in the list box, we are restricted to add an item to the list directly.
Let us now dissect the code of this application system:

Private Sub Form_Load( )
 Combo1.Text = " "
 Combo1.AddItem "Monday" ⟵
 Combo1.AddItem "Tuesday" ⟵
 Combo1.AddItem "Wednesday" ⟵
 Combo1.AddItem "Thursday" ⟵
End Sub

You can observe that we had preloaded the combo box with items from Monday through Thursday. This is the reason why the combo box has these items when we click its down arrow. We also initialize the text box of the combo box with an empty string with the following syntax :

```
  Combo1.Text = " "
```

so that when the Visual Basic system compiler runs our program, it is empty for the user to input an item.
Below is the code that executes when the user enters an item:

```
Private Sub Combo1_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then 'When Enter Key is pressed ←
  Combo1.AddItem Combo1.Text ←
  Combo1.Text = " " ←
End If
End Sub
```

We applied here the *if* conditional statement to test if the Enter key was pressed by the user. And if so, the item entered at the combo box will be added on it. Then after adding the data into the combo box, we have to store an empty string into the combo box to clear it, so that the user can easily enter another data into it.

Example 24:

Design and develop a simple application system that preloads a combo box with items. The user can select an item from the list to be displayed at the text box. Follow the given figure below in designing and developing the application program:

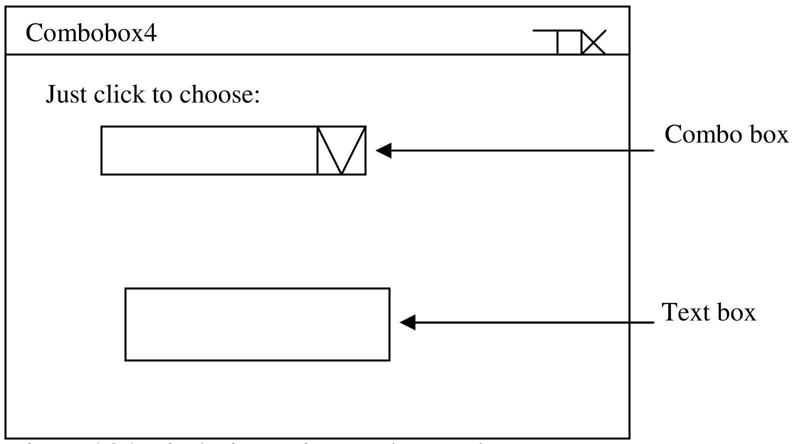


Figure 5.25 Displaying an item at the Text box

Note:

When the user selects an item at the Combo box by clicking on it, this item will be displayed at the text box. The user must click the down arrow (found at the right corner of the combo box) to be able to see the preloaded list of items .

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox4.**
3. Double-click a combo box, a label and a text box in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Combo1_Click()
 Text1.Text = ""
 Text1.Text = Combo1.Text
End Sub
```

```
Private Sub Form_Load()
 Combo1.Text = ""
 Text1.Text = ""
 'Add items to the Combo box
 Combo1.AddItem "Chicken"
 Combo1.AddItem "Beef"
 Combo1.AddItem "Pork"
 Combo1.AddItem "Vegetables"
 Combo1.AddItem "Fish"
 Combo1.AddItem "Squids"
 Combo1.AddItem "Noodles"
 Combo1.AddItem "Fruits"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this simple example about combo box, we just simply pre-loaded the Combo box with items which we want to view, select and finally display into the text box. So we just use the following code:

```
Private Sub Form_Load( )
 Combo1.Text = ""    ←
 Text1.Text = ""     ←
 'Add items to the Combo box
 Combo1.AddItem "Chicken"
 Combo1.AddItem "Beef"
 Combo1.AddItem "Pork"
 Combo1.AddItem "Vegetables"
 Combo1.AddItem "Fish"
 Combo1.AddItem "Squids"
 Combo1.AddItem "Noodles"
 Combo1.AddItem "Fruits"
End Sub
```

You will notice that we initialize the combo box and text box as empty by storing an empty string into them. Then we preloaded the combo box with the list of items using the AddItem method.

And finally, we display the selected item into the text box with the following code:

```
Private Sub Combo1_Click( )
   Text1.Text = ""              ←
  Text1.Text = Combo1.Text←
End Sub
```

We need to reinitialize the text box (Text1) with an empty string every time another item is selected, so that the previous displayed item will be completely cleared before the new item would be displayed. Otherwise, some parts of the previous item with longer string will appear as though it is concatenated to the new item displayed.
The Text property of the Combo box (Combo1) control returns the value of the selected item. This is the reason why the selected item is displayed at text box 1 (Text1).

Example 25:

Design and develop a simple program that demonstrates how to preload a collection of items into the combo box. The program should be able to display one or more selected items on a list box. Follow the given figure below in designing and developing the application program.

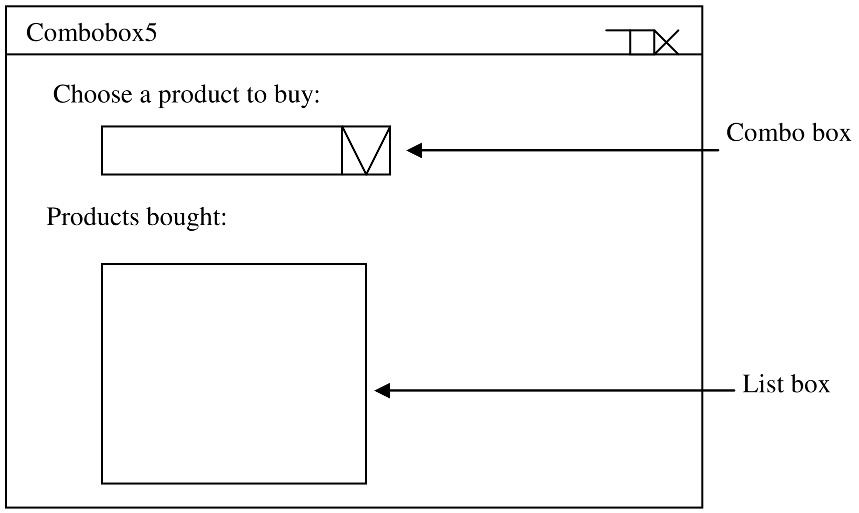


Figure 5.26 Display one or more items to the other list box

Note:

When the user selects an item at the Combo box by clicking on it, this item will be displayed at the list box. The user must click the down arrow (found at the right corner of the combo box) to be able to see the preloaded list of items.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox5.**
3. Double-click a combo box, two labels and a list box in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.
4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub Combo1_Click()
 List1.AddItem Combo1.Text
End Sub

Private Sub Form_Load()
 Combo1.Text = ""
 List1.Text = ""
 Combo1.AddItem "Banana Split"
 Combo1.AddItem "Blueberry"
 Combo1.AddItem "Cherry"
 Combo1.AddItem "Mud Pie"
 Combo1.AddItem "Nestle Crunch"
 Combo1.AddItem "Oreo Cookies"
 Combo1.AddItem "Choco Chip"
 Combo1.AddItem "Chocolate Covered"
 Combo1.AddItem "Rocky Road"
 Combo1.AddItem "Strawberry"
 Combo1.AddItem "Walnut Fudge"
 Combo1.AddItem "Choco Almond"
 Combo1.AddItem "Choco Mallows"
 Combo1.AddItem "Brownies"
 Combo1.AddItem "Butterfinger"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

In this simple example about combo box, we just simply pre-loaded the Combo box with items which we want to view, select and finally display into the list box. So we just use the following code:

```
Private Sub Form_Load( )
 Combo1.Text = ""          ←
 List1.Text = ""           ←
 Combo1.AddItem "Banana Split"
 Combo1.AddItem "Blueberry"
 Combo1.AddItem "Cherry"
 Combo1.AddItem "Mud Pie"
 Combo1.AddItem "Nestle Crunch"
 Combo1.AddItem "Oreo Cookies"
 Combo1.AddItem "Choco Chip"
 Combo1.AddItem "Chocolate Covered"
 Combo1.AddItem "Rocky Road"
 Combo1.AddItem "Strawberry"
 Combo1.AddItem "Walnut Fudge"
 Combo1.AddItem "Choco Almond"
 Combo1.AddItem "Choco Mallows"
 Combo1.AddItem "Brownies"
 Combo1.AddItem "Butterfinger"
End Sub
```

You will notice that we initialize the combo box and list box as empty by storing an empty string into them. Then we preloaded the combo box with the list of items using the AddItem method.
And finally, we display the selected item into the list box with the following code:

```
Private Sub Combo1_Click( )
  List1.AddItem Combo1.Text ←
End Sub
```

The Text property of the Combo box (Combo1) control returns the value of the selected item. This is the reason why the selected item is displayed at the list box.

There is no need for us to clear the list box, since we just simply add an item into it. That is the reason why there is no reinitialization every time a user selects an item at the combo box (unlike using a text box to display the item selected).

Example 26:

Design and develop a simple program that demonstrates how to preload a collection of items into the combo box. The program should be able to display one or more selected items on a list box with a confirmation using the command button. Follow the given figure below in designing and developing the application program.

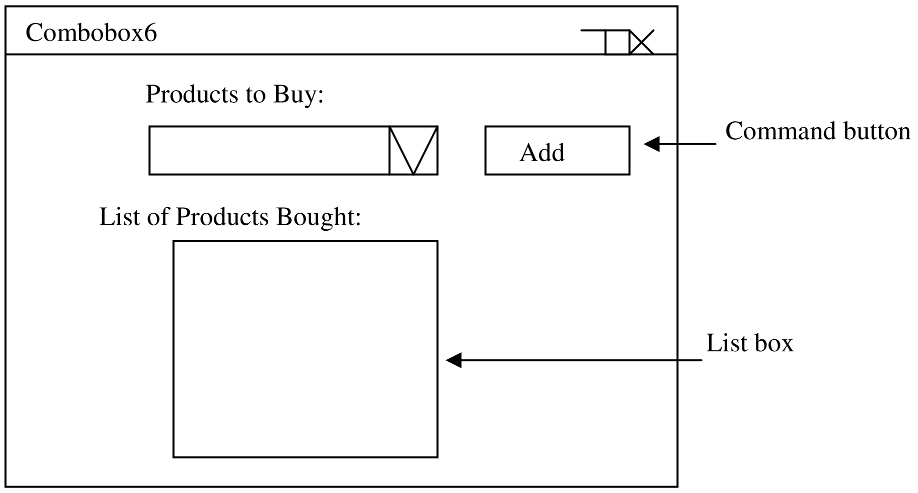


Figure 5.27 Display one or more items to the other list box

Note:

When the user selects an item at the Combo box and would like to buy it, she has to click the Add command button to confirm it. Clicking the Add command button will display the selected items at the list box.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Combobox5.**
3. Double-click a combo box, a command button with an **Add** caption, two labels and a list box in the Toolbox to add them into the form. Position them properly and adjust their respective width appropriately.

4. Double-click now the form to display the Code editor. Embed the following code to its respective procedures:

```
Private Sub cmdAdd_Click()
 List1.AddItem Combo1.Text
End Sub
```

```
Private Sub Form_Load()
 Combo1.Text = ""
 'Add items to the Combo box
 Combo1.AddItem "Chicken"
 Combo1.AddItem "Beef"
 Combo1.AddItem "Pork"
 Combo1.AddItem "Vegetables"
 Combo1.AddItem "Fish"
 Combo1.AddItem "Squids"
 Combo1.AddItem "Noodles"
 Combo1.AddItem "Fruits"
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Again, we preloaded our combo box with items using the following code:

```
Private Sub Form_Load( )
 Combo1.Text = ""
 'Add items to the Combo box
 Combo1.AddItem "Chicken"
 Combo1.AddItem "Beef"
 Combo1.AddItem "Pork"
 Combo1.AddItem "Vegetables"
 Combo1.AddItem "Fish"
 Combo1.AddItem "Squids"
 Combo1.AddItem "Noodles"
 Combo1.AddItem "Fruits"
End Sub
```

When the user selects an item at the combo box and clicks the Add command button, that particular item will be displayed at the list box. The code that accomplishes this task is the following:

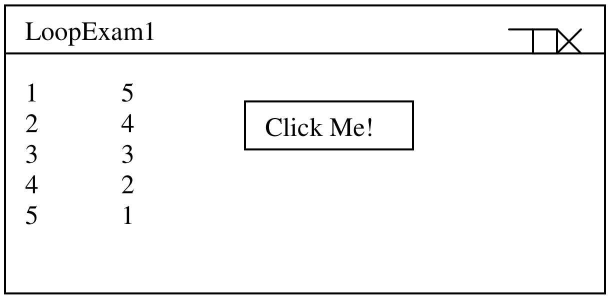
```
Private Sub cmdAdd_Click( )
  List1.AddItem Combo1.Text ←
```

End Sub

I think we have accomplished a lot of things in the previous examples, and it’s time for us to go to the next chapter for another topic to learn more. Are you ready for the next chapter? But before that, we will have first our Lab Activity Test that will tests our ability to comprehend and understand the examples presented in this chapter.

LAB ACTIVITY TEST 4

1. Design and develop a simple looping statement program that generates the given sequence numbers. Follow the given figure below in designing and developing the application program:

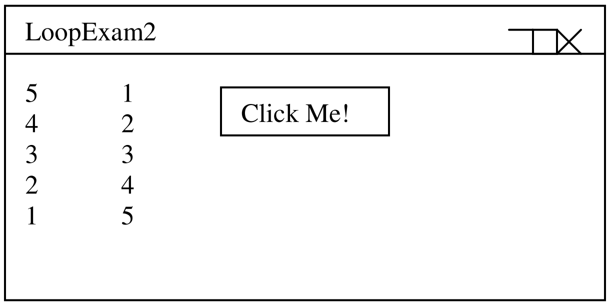


```
1     5
2     4
3     3
4     2
5     1
```

Note:

When the user clicks the Click Me! command button, the sequence numbers will be generated.

2. Design and develop a simple looping statement program that generates the given sequence numbers. Follow the given figure below in designing and developing the application program:

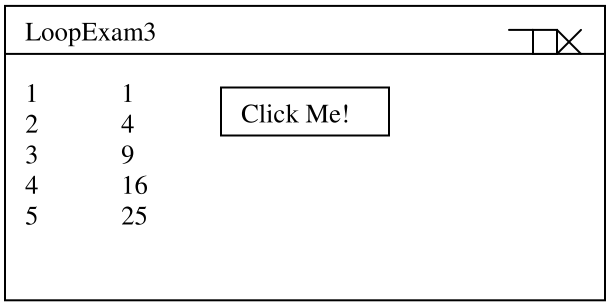


```
5     1
4     2
3     3
2     4
1     5
```

Note:

When the user clicks the Click Me! command button, the sequence numbers will be generated.

3. Design and develop a simple looping statement program that generates the given sequence numbers. Follow the given figure below in designing and developing the application program:

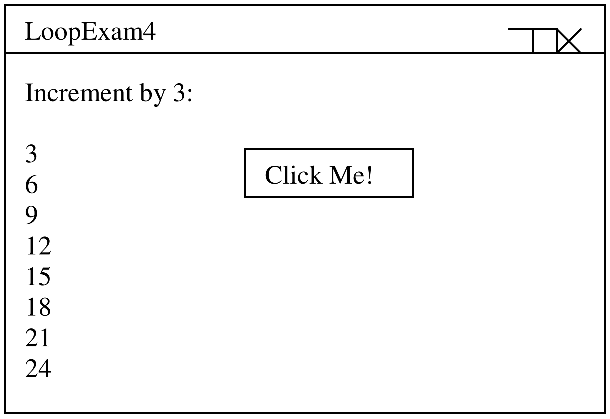


```
1          1
2          4
3          9
4          16
5          25
```

Note:

When the user clicks the Click Me! command button, the sequence and squared numbers will be generated.

4. Design and develop a simple looping statement program that generates the given sequence numbers. Follow the given figure below in designing and developing the application program:



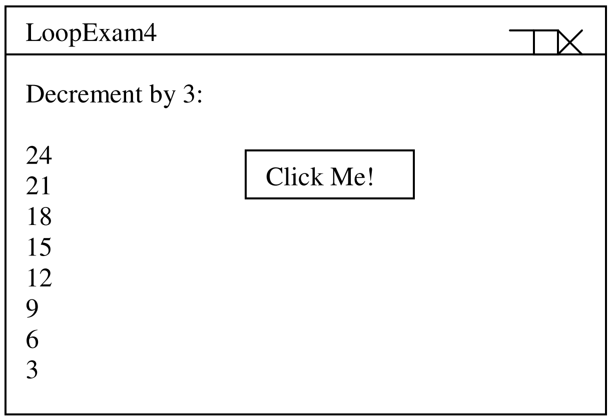
```
Increment by 3:

3
6
9
12
15
18
21
24
```

Note:

When the user clicks the Click Me! command button, the sequence numbers will be generated.

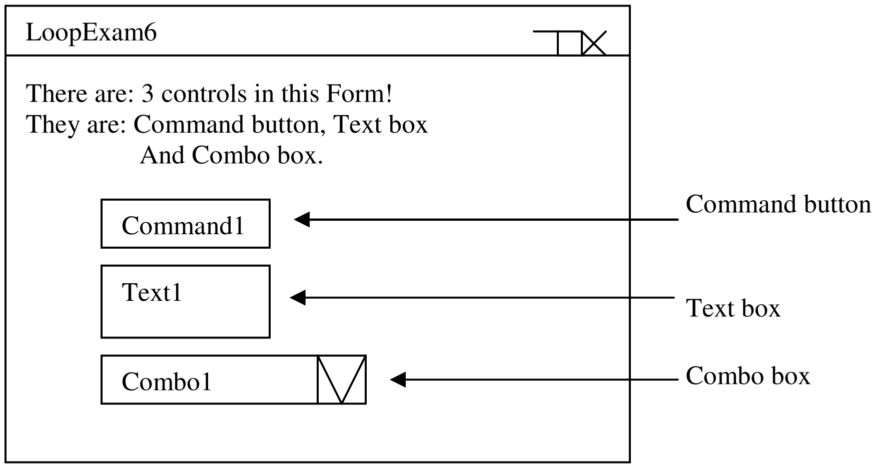
5. Design and develop a simple looping statement program that generates the given sequence numbers. Follow the given figure below in designing and developing the application program:



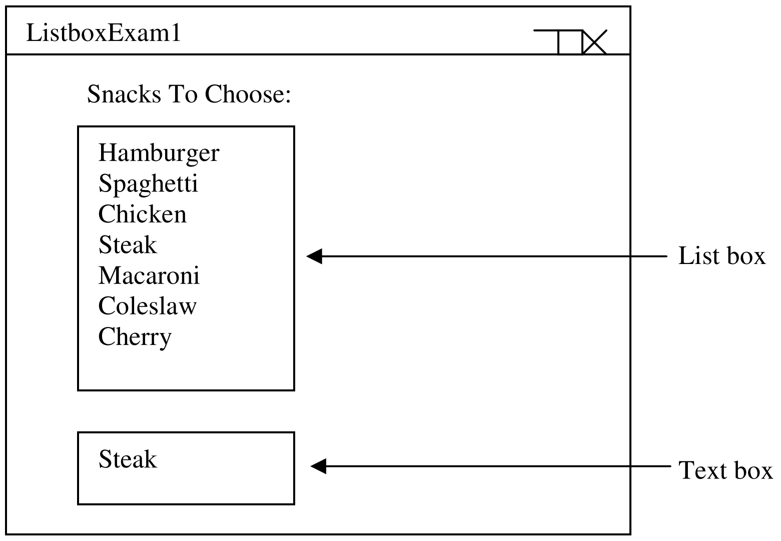
Note:

When the user clicks the Click Me! command button, the sequence numbers will be generated.

6. Design and develop a simple program that demonstrates how to use the For Each loop syntax that counts how many objects are there in a Form. Follow the given figure below in designing and developing the application program:



7.Design and develop a simple program that demonstrates how to preload the List box with items. When the user chooses an item at the list box by double-clicking it, that particular item will be displayed at the text box. Follow the given figure below in designing and developing the application program:



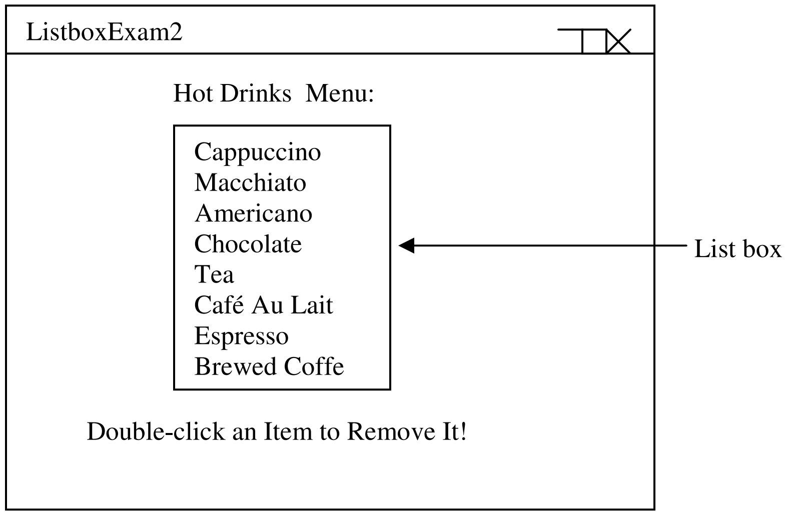
Note:

Here in the figure above, the Steak was selected by the user through double-clicking it. That is why the Steak is displayed at the text box.

Add the following items into the List box using the AddItem method at the Form_Load procedure:

Hamburger
Spaghetti
Chicken
Steak
Macaroni
Coleslaw
Cherry

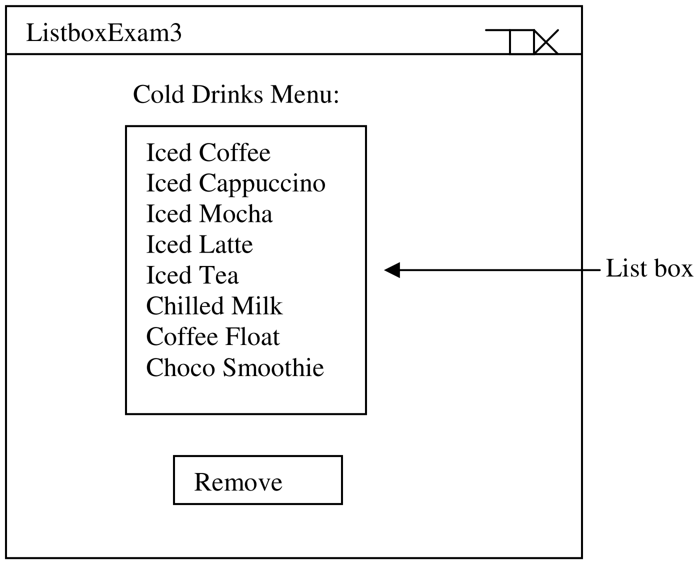
8. Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The user should be able to remove the selected item on the list by double-clicking it. Follow the given figure below in designing and developing the application program:

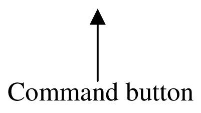


Add the following items into the List box using the AddItem method at the Form_Load procedure:

Cappuccino
Macchiato
Americano
Chocolate
Tea
Café Au Lait
Espresso
Brewed Coffe

9. Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to remove the selected item on the list with a confirmation using a command button. Follow the given figure below in designing and developing the application program:





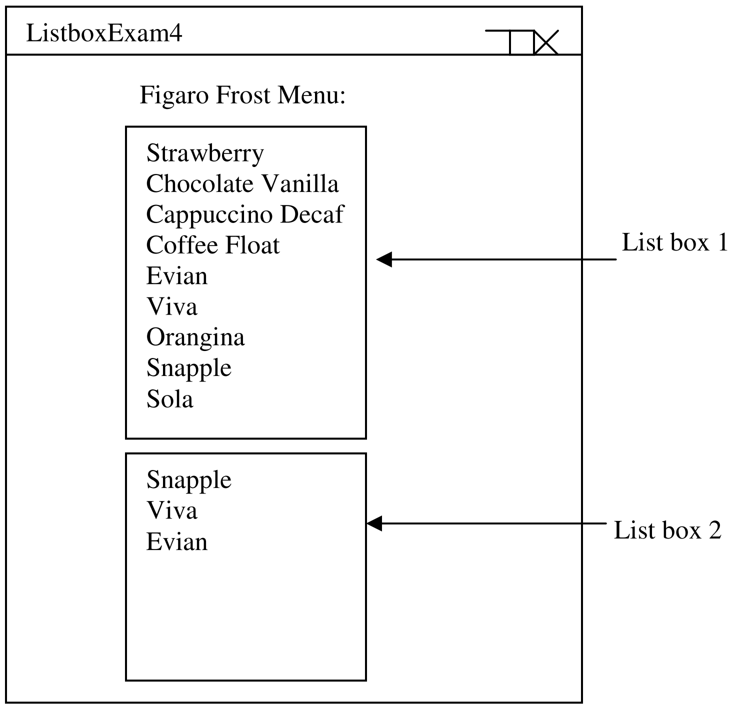
Note:

The user should select first an item at the List box by highlighting it, before the user can finally remove it by clicking the Remove command button.

Add the following items into the List box using the AddItem method at the Form_Load procedure:

Iced Coffee
Iced Cappuccino
Iced Mocha
Iced Latte
Iced Tea
Chilled Milk
Coffee Float
Choco Smoothie

10. Design and develop a simple program that demonstrates how to preload a collection of items into List box 1. The program should be able to display one or more selected items on another list box. Follow the given figure below in designing and developing the application program:

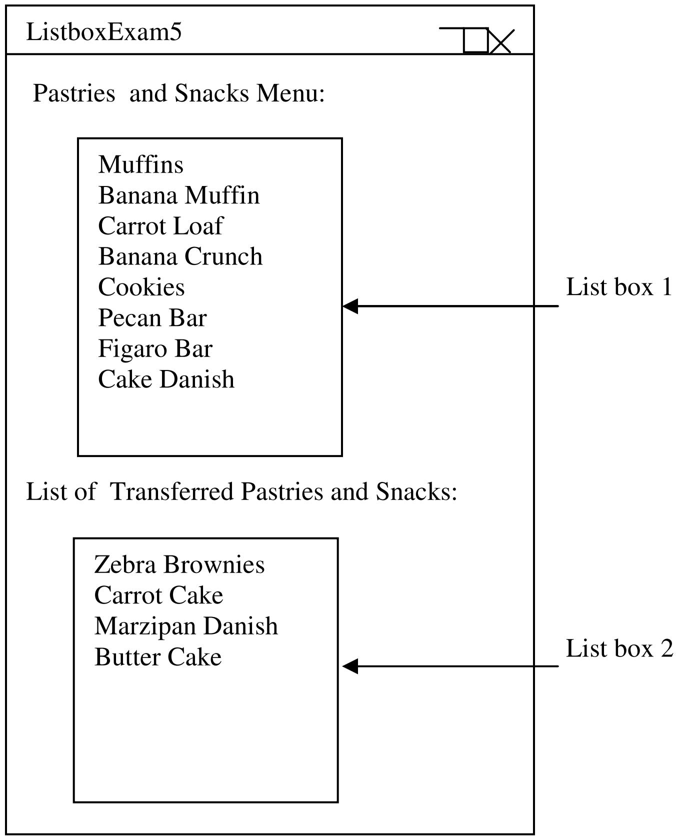


Note:

When the user selects and double-clicks an item at the List box 1, that particular item must be displayed at the List box 2.
Add the following items into the List box 1 using the AddItem method at the Form_Load procedure:

Strawberry
Chocolate Vanilla
Cappuccino Decaf
Coffee Float
Evian
Viva
Orangina
Snapple
Sola

11.Design and develop a simple program that demonstrates how to preload a collection of items into List box 1. The program should be able to transfer one or more selected items on another list box. Follow the given figure below in designing and developing the application program:



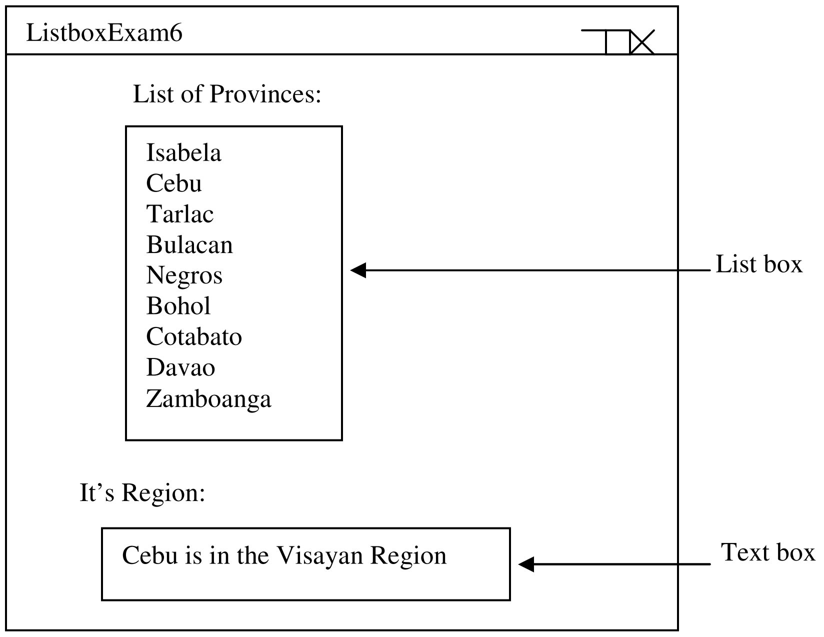
Note:

When the user selects and double-clicks an item at List box 1, that particular item must be transferred into List box 2.

Add the following items into the List box 1 using the AddItem method at the Form_Load procedure:

Muffins
Banana Muffin
Carrot Loaf
Banana Crunch
Cookies
Pecan Bar
Figaro Bar
Cake Danish
Zebra Brownies
Carrot Cake
Marzipan Danish
Butter Cake

12.Design and develop a simple program that demonstrates how to preload a collection of items into the List box. The program should be able to get the value of the currently selected item in the list box. Follow the given figure below in designing and developing the application program:



Note:

The user selects Cebu by double-clicking it at the list box. This is also the reason why its region was displayed at the text box. Here are the group regions for each provinces:

NCR Region:
  Isabela
  Tarlac
  Bulacan

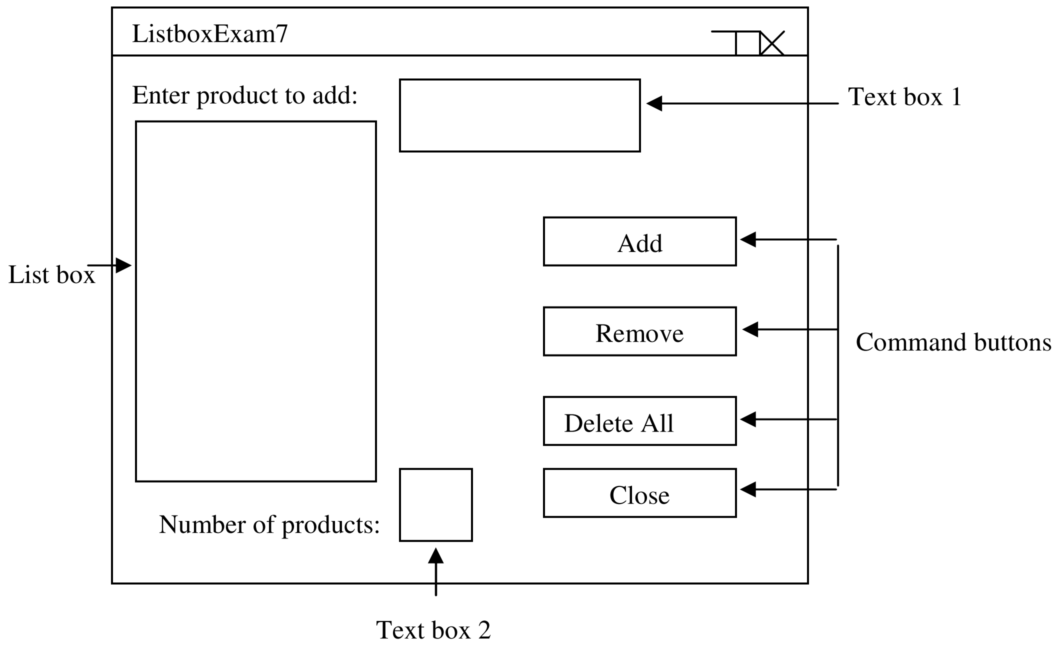
Visayan Region:
  Cebu
  Bohol
  Negros

Mindanao Region:
  Davao
  Cotabato
  Zamboanga

Add the following items into the List box using the AddItem method at the Form_Load procedure:

Isabela
Cebu
Tarlac
Bulacan
Negros
Bohol
Cotabato
Davao
Zamboanga

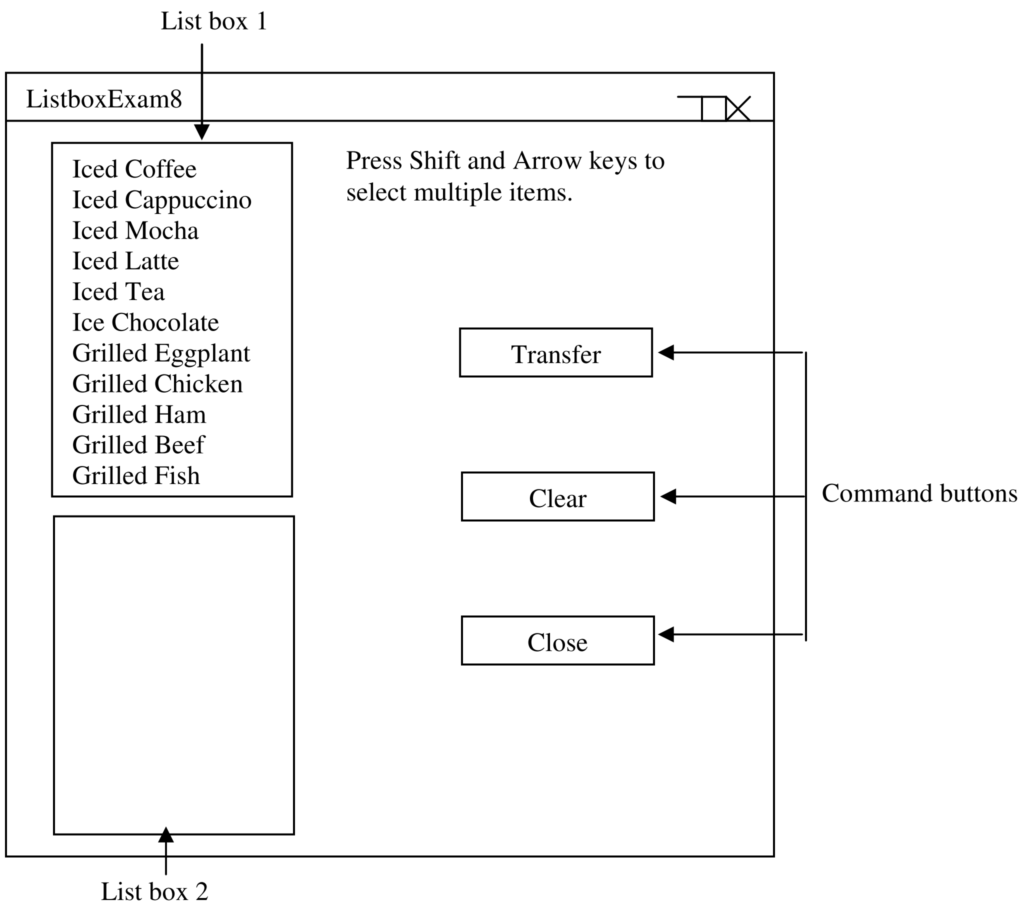
13. Design and develop a simple program that should apply the AddItem, RemoveItem and Clear methods with the ListIndex and ListCount properties to add, remove and clear the list entries in the list box at run time. Follow the given figure below in designing and developing the application program:



Note:

The user will enter an item first at text box 1 and then clicks the Add command button to add the inputted item in the List box. Now when the user wants to remove an item from the list, the user should highlight it first, then clicks the Remove command button to finally delete it. Clicking the Delete All command button will wipe out all the items in the list box.

14. Design and develop a simple program that should be able to select two or more items in the list box at once and be able to transfer the collection of selected items to another list box simultaneously. Follow the given figure below in designing and developing the application program:



Note:

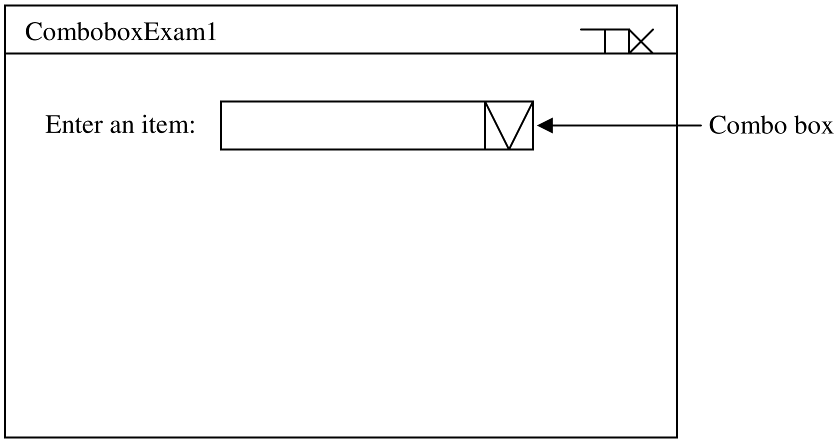
The user will click first an item located at the top portion of the list box, then use the Shift and Arrow keys to select down other multiple items at list box 1. Clicking the Transfer command button will copy the collection of selected items into the list box 2. Preselect the first two items at the list box 1 using the Selected statement of the List1 control.

Add the following items into the List box 1 using the AddItem method at the Form_Load procedure:

Iced Coffee
Iced Cappuccino
Iced Mocha
Iced Latte
Iced Tea
Ice Chocolate
Grilled Eggplant
Grilled Chicken
Grilled Ham

Grilled Beef
Grilled Fish

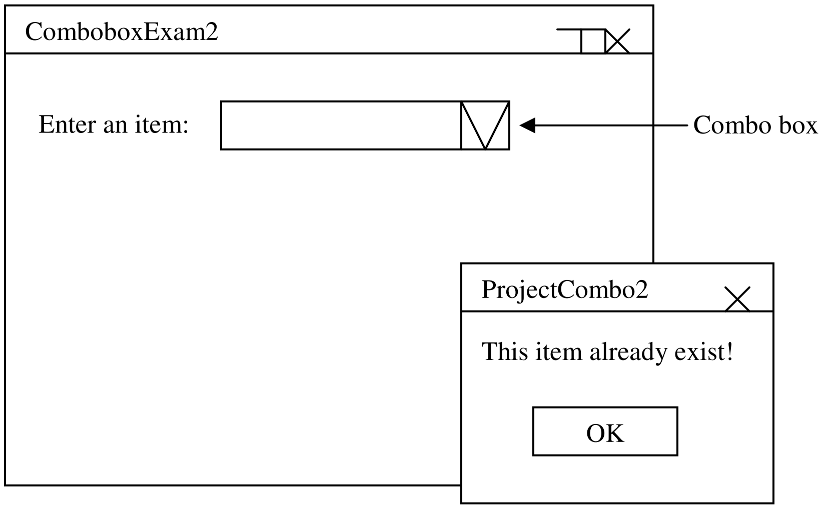
15. Design and develop a simple application system that accepts and lists the items inputted by the user in the combo box. Follow the given figure below in designing and developing the application program:



Note:

When the user inputs an item at the Combo box, this item will be added on it. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously entered by the user.

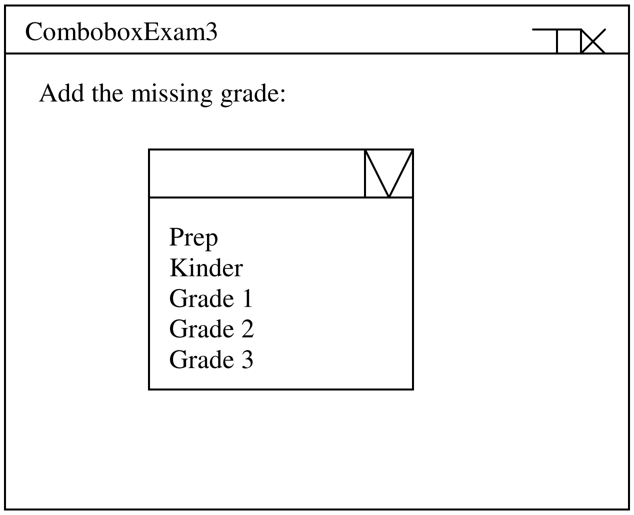
16. Design and develop an application system that accepts and lists the unique items inputted by the user in the combo box. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously entered by the user. Follow the given figure below in designing and developing the application program:



Note:

When the user inputs an item at the Combo box, this item will be added on it, as long as it is unique from the items already entered on the list. Otherwise the message box will appear informing the user that the item she entered already existed. To test if your program is correct, try to enter the same item which you have entered previously. If the Message box appears, then your program is right.

17. Design and develop a simple application system that accepts and lists the items inputted by the user in the combo box, together with the preloaded items. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of items which are previously and presently entered by the user. Follow the given figure below in designing and developing the application program:

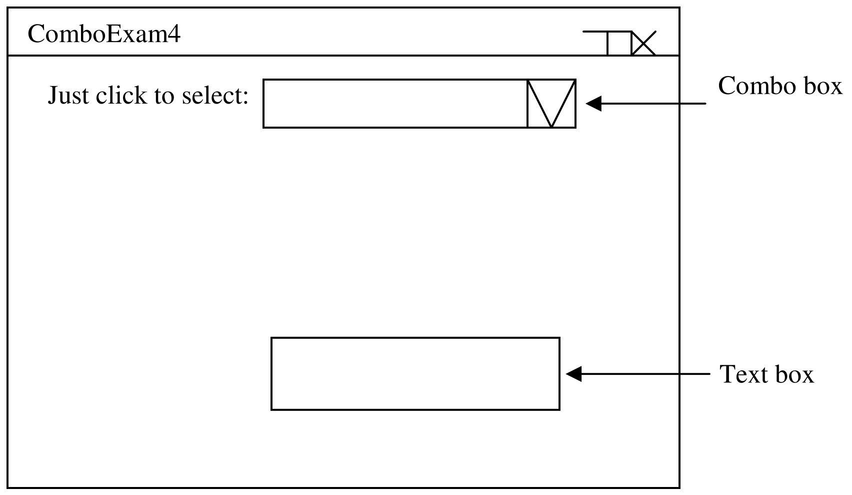


Note:

When the user inputs an item at the Combo box, this item will be added on it.
Add the following items into the Combo box using the AddItem method at the Form_Load procedure:

Prep
Kinder
Grade 1
Grade 2
Grade 3

18. Design and develop a simple application system that preloads a Combo box with items. The user can select an item from the list to be displayed at the text box. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of preloaded items. Follow the given figure below in designing and developing the application program:

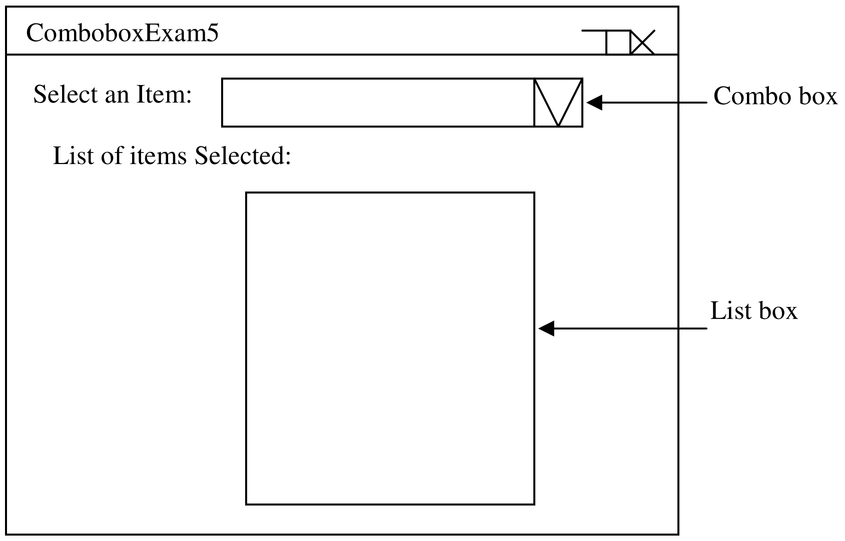


Note:

When the user selects an item at the Combo box by clicking it, this item will be displayed at the text box.
Add the following items into the Combo box using the AddItem method at the Form_Load procedure:

Dilly Bar
Plain Dog
Cheese Dog
Chilimelt Dog
Relish Dog
Bacon n Cheese
Fudge Brownie
Banana Split
Peanut Buster

19. Design and develop a simple application system that preloads a Combo box with items. The user can select an item from the list to be displayed at the list box. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of preloaded items. Follow the given figure below in designing and developing the application program:

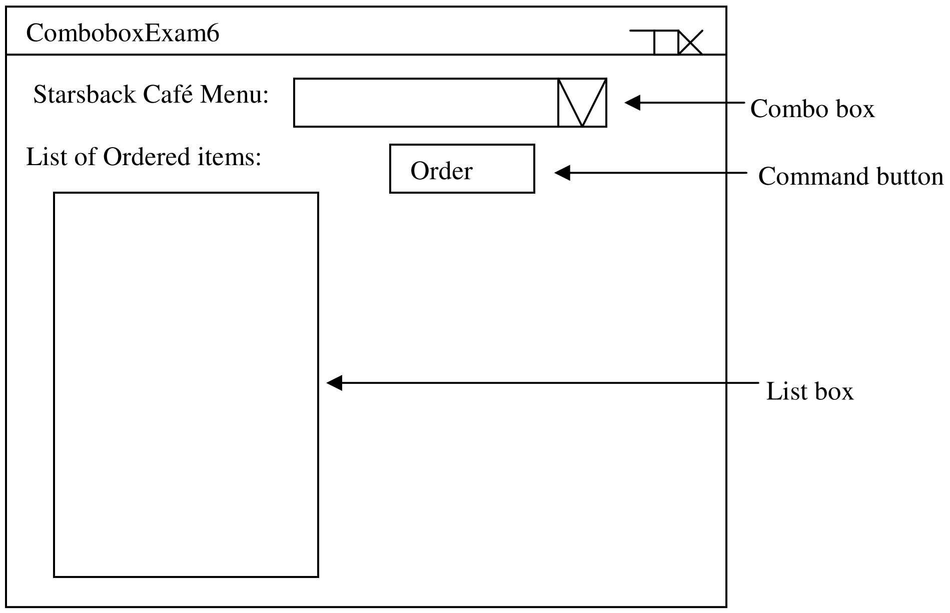


Note:

When the user selects an item at the Combo box by clicking it, this item will be displayed at the list box.
Add the following items into the Combo box using the AddItem method at the Form_Load procedure:

Al Tonno
Chorizo
Arrabiata
Pesto
Carbonara
Mushroom Sauce
Pasta
Mexicali
Veggies Heaven
Mushroom Galore
Neptune's Fancy

20. Design and develop a simple application system that preloads a Combo box with items. The user can select an item from the list to be displayed at the list box with a confirmation using the command button. The user must click the down arrow (found at the right corner of the combo box) to be able to see the list of preloaded items. Follow the given figure below in designing and developing the application program:



Note:

When the user selects an item at the Combo box and would like to order it, she has to click the Order command button to display the selected item into the list box.

Add the following items into the Combo box using the AddItem method at the Form_Load procedure:

Cheese Tempura
Caesar's Salad
Cajun Burger
Citrus Pork
Burger Steak
Café Mocha
Café Latte
Cappuccino
Steamed Milk
Espresso
Brewed Coffee

Chapter 6

Designing and Developing Menus

When we create a large application system, we need to use menus and toolbars to organize each subsystems effectively and efficiently. In designing and developing menus and toolbars, we have to make it sure that they conform to the Windows graphical user-interface (GUI) standards, so that the user can easily adapt them for their daily use. In other words, we have to mimic how the Microsoft Office software are designed and created. For example, the File menu must be the first menu and the Help menu must be the last menu of our application system, the way the Microsoft Word works. And the File menu of our application system should contain the menu items such as New, Open, Close, Save and Print, while the Help menu should contain menu items such as What's This?, and About the System. Though not exactly as what is mentioned above, but at least similarly arranged or grouped by.

Using Menu Editor to Create Menus

The Menu Editor is where we will do most of the design and development of the menus and menu bars of our application system. Furthermore, we can use the Menu Editor to create new menus, add new commands to existing menus, replacing existing menu commands with our own commands, and change and delete existing menus and menu bars.
Let us now have a working example about this menu design and development.

Example 1:

Design and develop a simple menu-based application system that demonstrates how to create a simple menu system with the following menu title: Menu1, Menu2, and Menu3. Follow the given figures below in designing and developing the application program:

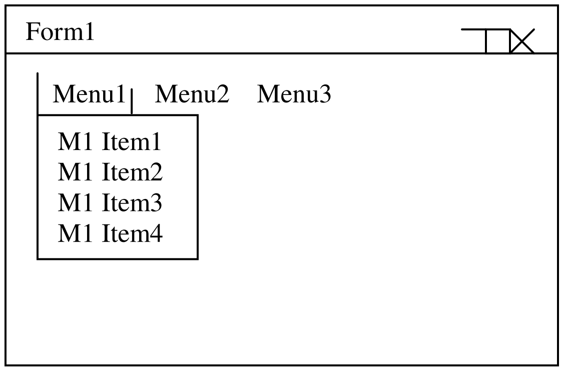


Figure 6.1 Menu items of the Menu1 menu

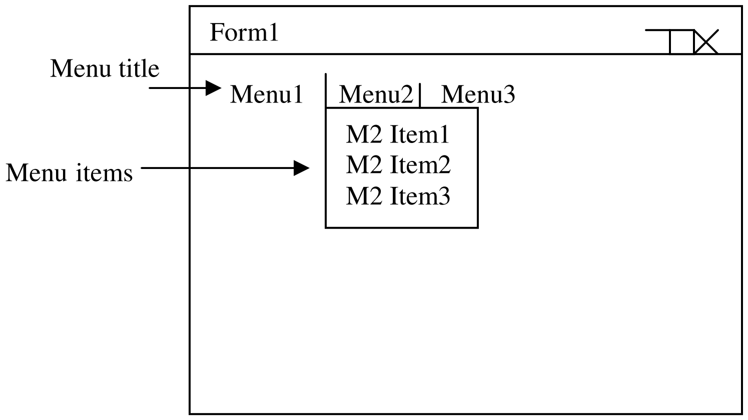


Figure 6.2 Menu items of the Menu2 menu

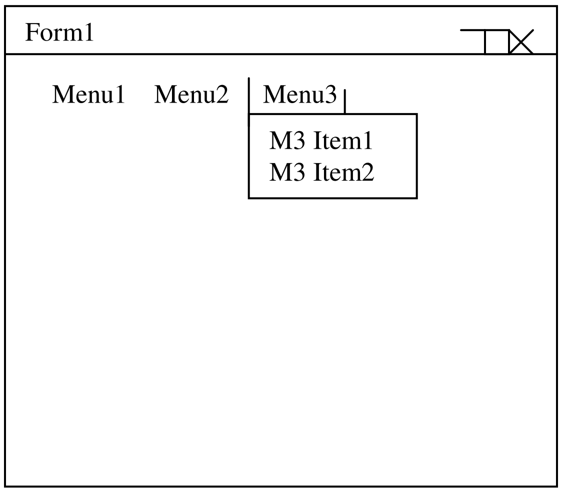


Figure 6.3 Menu items of the Menu3 menu

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button.
3. Open the Menu Editor by selecting Tools at the Visual Basic main menu.

4. Create the Menu1 menu by typing &**Menu1** in the Caption field, and **mnuMenu1** in the Name field.
5. Click the Next button to start the next menu item. Type **&M1 Item1** in the Caption field, and **mnuMenu1Item1** in the Name field. We want to make this a menu item of the Menu1 menu, so click the right arrow button to indent this menu item. You can see here the four dots before the menu item's caption, signifying that it is a menu item of the Menu1 menu.
6. Now click the Next button again to type the second menu item: &**M1 Item2** in the Caption field and **mnuMenu1Item2** in the Name field.
7. Click the Next button and type the third menu item of the Menu1 menu: &**M1 Item3** in the Caption field and **mnuMenu1Item3** in the Name field.
8. Click the Next button and type the fourth menu item (which is also the last menu item of the Menu1 item): **&M1 Item4** in the Caption field and **mnuMenu1Item4** in the Name field.
9. Click the Next button and this time click the left arrow to outdent the new menu item which we will name: Menu2. You will see that the four dots had disappeared, signifying that the thing we will type is a new menu item.Type now the &**Menu2** in the Caption field and **mnuMenu2** in the Name field.
10. Now click the Next button to type the first menu item under the Menu2 menu. Again we have to click the right arrow to indent this menu item. Did you click it now? If so, you can see the four dots displayed. Type &**M2 Item1** in the Caption field and **mnuMenu2Item1** in the Name field.
11. Click again the Next button to type the next menu item. Type &**M2 Item2** in the Caption field and **mnuMenu2Item2** in the Name field.
12. Click again the Next button to type the next menu item. Type **&M2 Item3** in the Caption field and **mnuMenu2Item3** in the Name field.
13. This time, we have to click the Next button and click the left arrow to outdent the new menu item which we will name: Menu3. You will see by now that the four dots had disappeared, signifying that the word we are about to type is a menu item. Type now the &**Menu3** in the Caption field and **mnuMenu3** in the Name field.
14. Now click the Next button to type the first menu item under the Menu3 menu. Again we have to click the right arrow to indent this menu item. Type &**M3 Item1** in the Caption field and **mnuMenu3Item1** in the Name field.
15. Finally click the Next button to type the last menu item. Type &**M3 Item2** in the Caption field and **mnuMenu3Item2** in the Name field.

After we are done creating these menu objects, our design must resemble like the following setup of the Menu Editor:

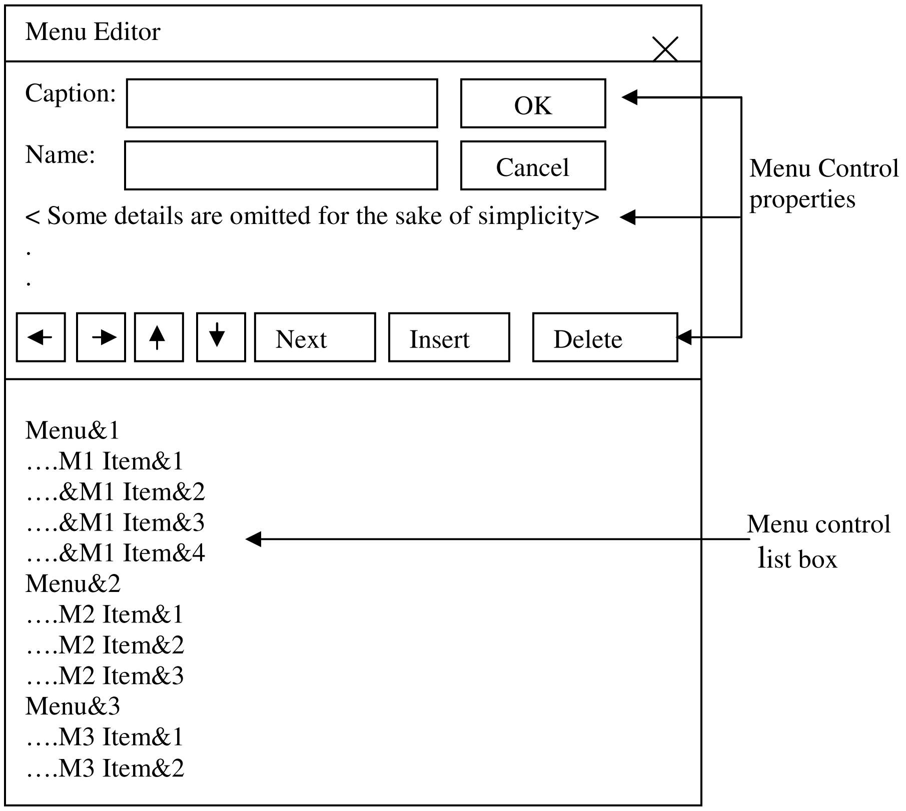


Figure 6.4 Menu Editor setup for the simple menu

16. This time click the OK button to run the menu we have created.
17. Embed the following code at Form1 so that when we click each individual item in each menu, a message box will appear confirming that we indeed click that particular menu item in a particular menu title.

```
Private Sub mnuMenu1Item1_Click()
 MsgBox "Menu1 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu1Item2_Click()
 MsgBox "Menu1 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu1Item3_Click()
 MsgBox "Menu1 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu1Item4_Click()
  MsgBox "Menu1 Item4 clicked"
```

```
End Sub
```

```
Private Sub mnuMenu2Item1_Click()
  MsgBox "Menu2 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu2Item2_Click()
 MsgBox "Menu2 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu2Item3_Click()
 MsgBox "Menu2 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu3Item1_Click()
 MsgBox "Menu3 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu3Item2_Click()
 MsgBox "Menu3 Item2 clicked"
End Sub
```

18. Now run the simple application menu system by clicking the Run icon at the toolbar. Then try to click each menu item in each menu title and see for yourself how the menu is responding. I hope it responds positively.

Explanation:

There are two most important Menu control properties in the Menu Editor; they are the Caption and the Name. The Caption is the text that appears on the control ,while the Name is the name we use to reference the menu control from the program.
The menu control list box which is located at the lower portion of the Menu Editor will list all the menu controls for the current form that we are designing. When we type a menu item in the Caption text box, that item also appears in the menu control list box. By selecting an existing menu control from the list box, we can change or modify the properties for that control.
The position of the menu control in the menu control list box determines whether the control is a menu title, menu item, or submenu item. So a menu control that appears at the extreme left in the list box is displayed on the menu bar as a menu title (in our example, they are the Menu&1, Menu&2 and Menu&3 menu titles), while a menu control that is indented in the list box (with 4 dots) is displayed on the menu when the user clicks the preceding menu title. In other words, it is among the menu items of a particular menu title. In our example, under the Menu1 menu title, the menu items are: ….M1 Item&1, ….M1 Item&2, ….M1 Item&3 and ….M1 Item&4. Under the &Menu2 menu title, the menu items are: ….M2 Item&1, ….M2 Item&2, and….M2 Item&3. While under the &Menu3 menu title, the menu items are: ….M3 Item&1 and ….M3 Item&2.

Here, we are using the MsgBox method to display a message when the user clicks a particular menu item. The message tells what particular menu item was clicked and in what particular menu title that menu item belonged to.

Example 2:

Design and develop a simple menu-based application system that demonstrates how to create a simple menu system with the following menu title: Control, Calculate, and About. Follow the given figures below in designing and developing the application program:

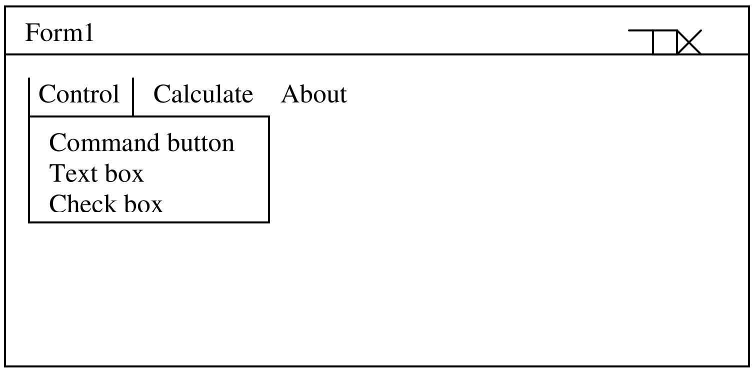


Figure 6.5 Menu items of the Control menu

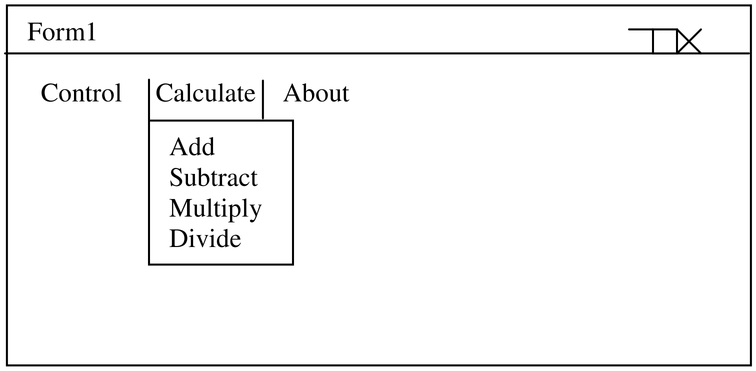


Figure 6.6 Menu items of the Calculate menu

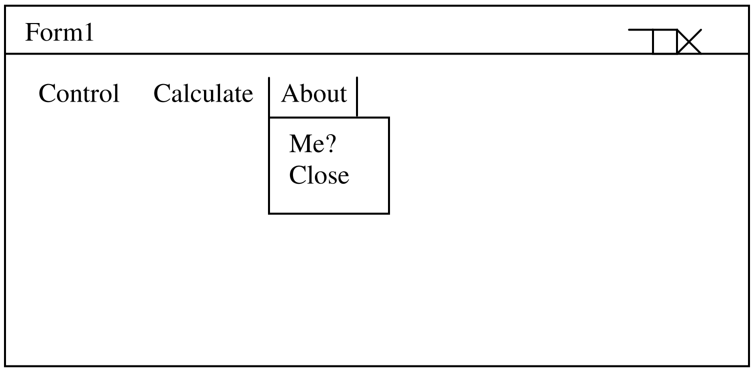
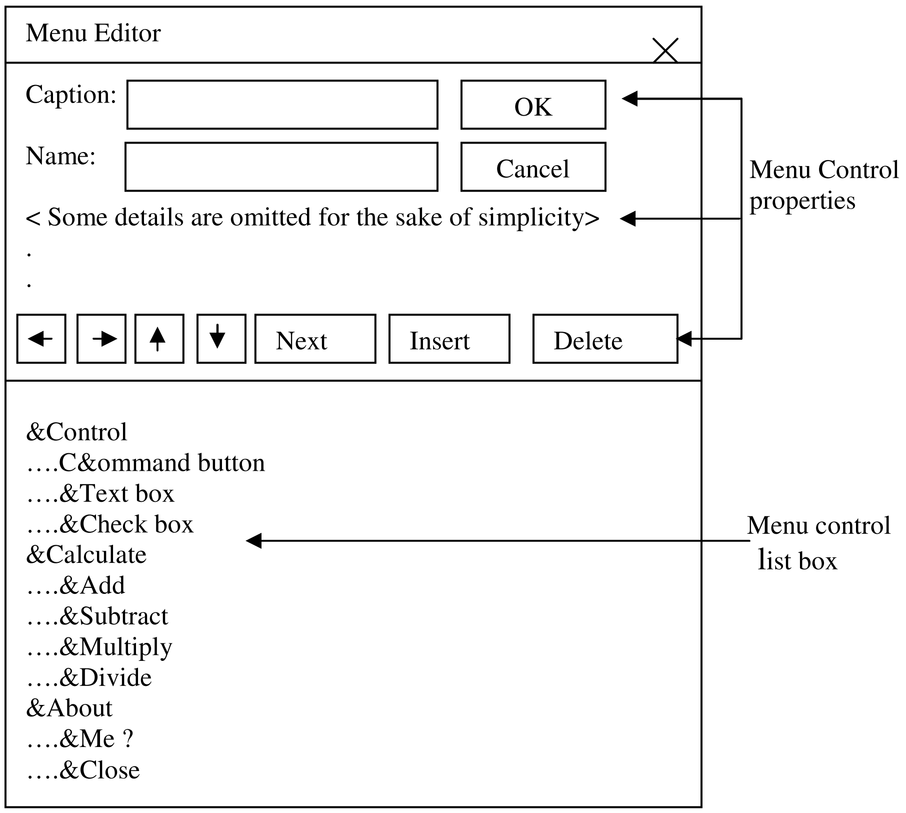


Figure 6.7 Menu items of the About menu

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button.
3. Open the Menu Editor by selecting Tools at the Visual Basic main menu.
4. Create the Control menu by typing &**Control** in the Caption field, and **mnuControl** in the Name field.
5. Click the Next button to start the next menu item. Type **C&ommand button** in the Caption field, and **mnuControlCButton** in the Name field. We want to make this a menu item of the Control menu, so click the right arrow button to indent this menu item. You can see here the four dots before the menu item's caption, signifying that it is a menu item of the Control menu.
6. Now click the Next button again to type the next menu item: &**Text box** in the Caption field and **mnuControlTBox** in the Name field.
7. Finally click the Next button and type the last menu item of the Control menu: &**Check box** in the Caption field and **mnuControlCBox** in the Name field.
8. Click the Next button and this time click the left arrow to outdent the new menu item which we will name: Calculate. You will see that the four dots had disappeared, signifying that the thing we will type is a new menu item.Type now the &**Calculate** in the Caption field and **mnuCalculate** in the Name field.
9. Now click the Next button to type the first menu item under the Calculate menu. Again we have to click the right arrow to indent this menu item. Did you click it now? If so, you can see the four dots displayed. Type &**Add** in the Caption field and **mnuCalculateAdd** in the Name field.
10. Click again the Next button to type the next menu item. Type &**Subtract** in the Caption field and **mnuCalculateSubtract** in the Name field.
11. Click again the Next button to type the next menu item. Type &**Multiply** in the Caption field and **mnuCalculateMultiply** in the Name field.
12. Finally click again the Next button to type the next menu item. Type &**Divide** in the Caption field and **mnuCalculateDivide** in the Name field.
13. This time, we have to click the Next button and click the left arrow to outdent the new menu item which we will name: About. You will see by now that the four dots had disappeared, signifying that the word we are about to type is a menu item. Type now the &**About** in the Caption field and mnuAbout in the Name field.
14. Now click the Next button to type the first menu item under the About menu. Again we have to click the right arrow to indent this menu item. Type &**Me ?** in the Caption field and **mnuAboutMe** in the Name field.
15. Finally click the Next button to type the last menu item. Type &**Close** in the Caption field and **mnuAboutClose** in the Name field.

After we are done creating these menu objects, our design must resembles like the following setup of the Menu Editor:



< Some details are omitted for the sake of simplicity>

&Control
….C&ommand button
….&Text box
….&Check box
&Calculate
….&Add
….&Subtract
….&Multiply
….&Divide
&About
….&Me ?
….&Close

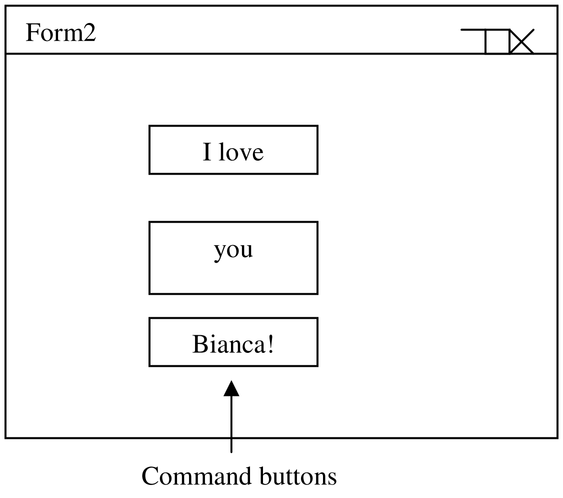
Figure 6.8 Menu Editor setup for Menu application system

16. This time click the OK button to run the menu we have created.

17. We have to put a code for each menu item in each control. What we have to do is to add a form. Click now the Add Form at the Visual Basic's Tool bar. Then click the Open button. We are now in Form2.

Since our menu objects are in Form1, we have to put the Command button menu item of the Control menu title in Form2, while the Text box menu item of the Control menu title should be in Form3 and the Check box is in Form4.
The Add menu item should be in Form5, the Subtract menu item is in Form6, the Multiply menu item is in Form7 and the Divide menu item should be in Form8. Let us do it now.

18. Design the following specification in Form2 for our Command button menu item:



19. After designing the above specs, you have to embed the following code into Form1 procedure, so that we can call the Form2 from our menu system:

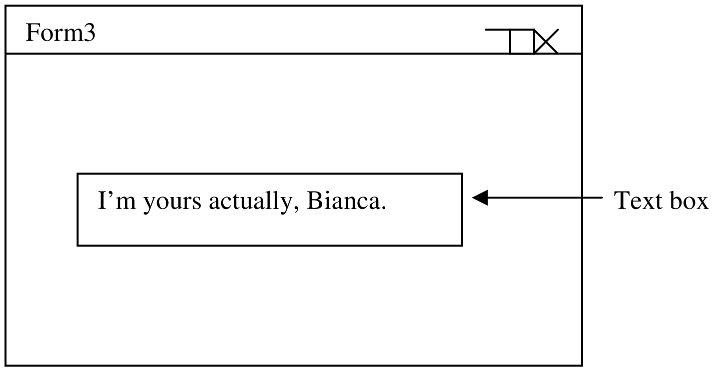
```
Private Sub mnuControlCButton_Click()
  Form2.Show
End Sub
```

The procedure above can be found by clicking the Command button menu item at design time.

20. Try to run the menu program by clicking the Run icon at the toolbar.

Does it work? I hope so.

21.This time, add another form. Click the Add Form at the toolbar, then click the Open button. This must be Form3, right? Now design the given specification below for our Text box menu item:



22. Embed the following code into Form3 procedure, so that the message is displayed at the text box automatically upon loading the Form3:

```
Private Sub Form_Load()
 Text1.Text = "I'm yours actually, Bianca"
End Sub
```

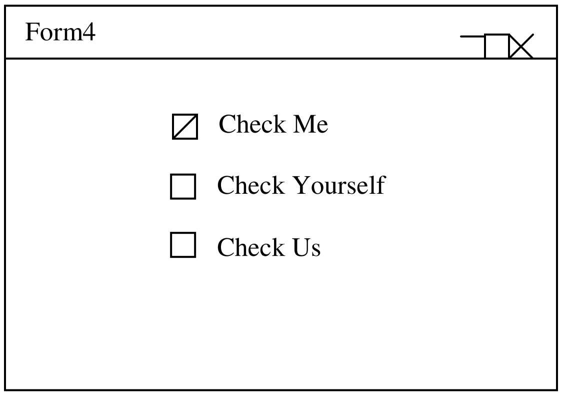
23. To call Form3 in our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuControlTBox_Click()
  Form3.Show
End Sub
```

24. Try to run the menu program by clicking the Run icon at the toolbar.

I hope it's working well.

25. Add another form. Click the Add Form at the toolbar, then click the Open button. This must be Form4. Now design the given specification below for our Text box menu item:

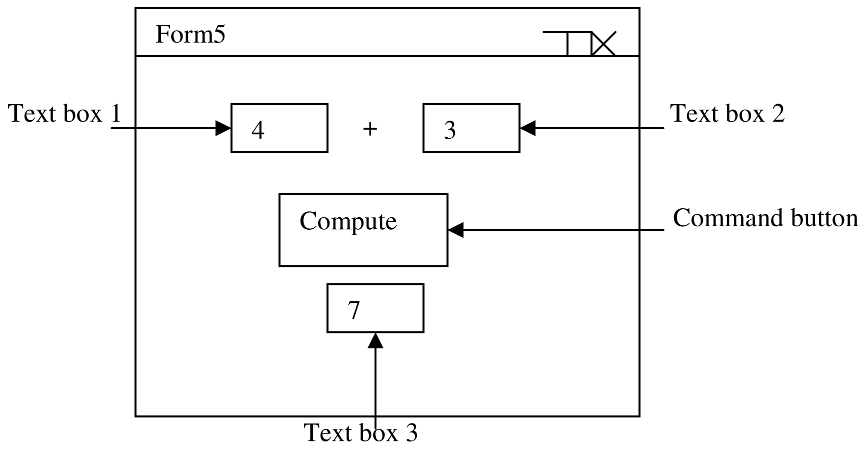


26. To call the Form4 in our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuControlCbox_Click()
 Form4.Show
End Sub
```

27. Run again the menu program by clicking the Run icon at the toolbar.

28. In this juncture, under the Calculate menu, let us add another form. Click the Add Form at the toolbar, then click the Open button. This must be Form5. Now design the given specification below for our Add menu item:



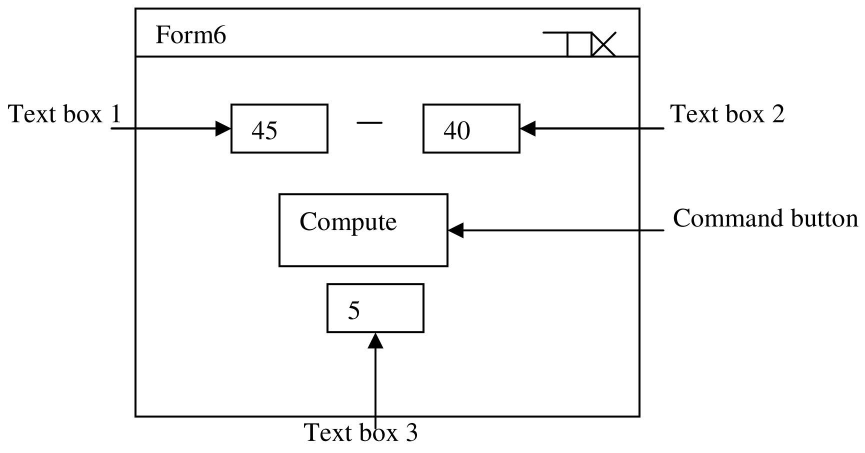
29. Then embed the following code at Form5 procedure:

```
Private txtNum1, txtNum2, txtSum As Integer
______________________________________________
Private Sub Command1_Click()
 txtSum = txtNum1 + txtNum2
 Text3.Text = Str(txtSum)
End Sub
_____________________________________________
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtSum = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
 Text3.Text = nSum
End Sub
_____________________________________________
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
_____________________________________________
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

30. To call the Form5 from our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuCalculateAdd_Click()
  Form5.Show
End Sub
```

31. Let us add another form under the Calculate menu. Click the Add Form at the toolbar, then click the Open button. This must be Form6. Now design the given specification below for our Subtract menu item:



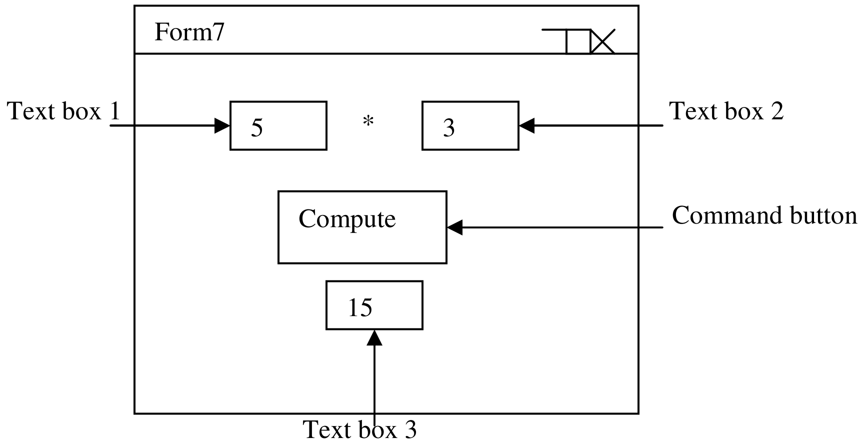
32. Then embed the following code at Form6 procedure:

```
Private txtNum1, txtNum2, txtDiff As Integer
_________________________________
Private Sub Command1_Click()
 txtDiff = txtNum1 - txtNum2
 Text3.Text = Str(txtDiff)
End Sub
_________________________________
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtDiff = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
 Text3.Text = nDiff
End Sub
_________________________________
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
_________________________________
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

33. To call Form6 in our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuCalculateSubtract_Click()
  Form6.Show
End Sub
```

34. Again, let us add another form under the Calculate menu. Click the Add Form at the toolbar, then click the Open button. This must be Form7. Now design the given specification below for our Multiply menu item:



35. Then embed the following code at Form7 procedure:

```
Private txtNum1, txtNum2, txtProd As Long
```

```
Private Sub Command1_Click()
 txtProd = txtNum1 * txtNum2
 Text3.Text = Str(txtProd)
End Sub
```

```
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtProd = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
 Text3.Text = nProd
End Sub
```

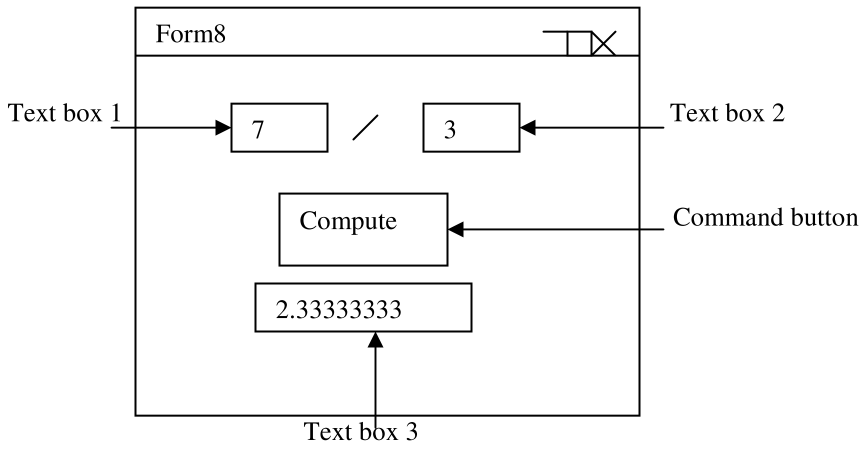
```
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

36. To call Form7 in our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuCalculateMultiply_Click()
  Form7.Show
End Sub
```

37. Let us add the last form under the Calculate menu. Click the Add Form at the toolbar, then click the Open button. This must be Form8. Now design the given specification below for our Divide menu item:



38. Then embed the following code at Form8 procedure:

```
Private txtNum1, txtNum2 As Integer
Private txtQuot As Double
```

```
Private Sub Command1_Click()
 txtQuot = txtNum1 / txtNum2
 Text3.Text = Str(txtQuot)
End Sub
```

```
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtQuot = 0
 Text1.Text = nNum1
 Text2.Text = nNum2
 Text3.Text = nQuot
End Sub
```

```
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

39. Now this time, let us put a ready-made Form for our Me? menu item under About menu title. Where can we find it? It is very easy. Click the Add Form icon now at the toolbar of Visual Basic, then select the About Dialog and finally click the Open button. Select the Description part of the About form and write the following paragraph at its Caption property:

This program will demonstrate how to design and develop simple menu system. This features the basic controls, simple calculator, and About menu. This program is not copyrighted by any International laws.

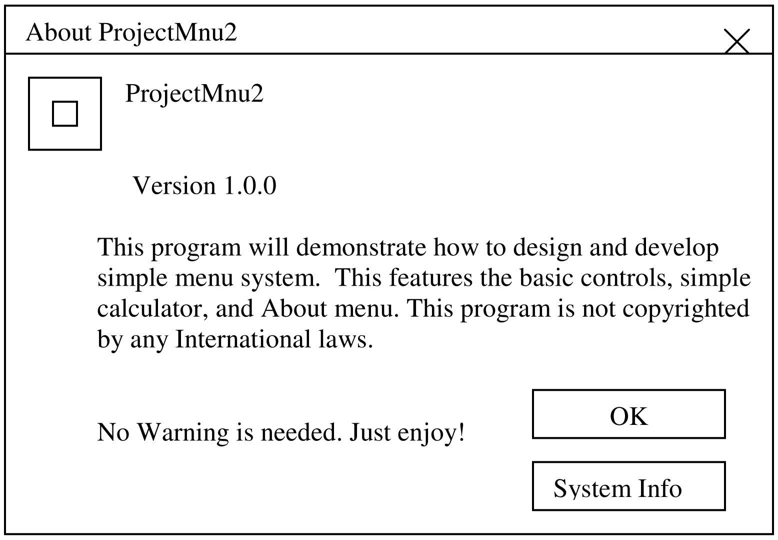
Next, select the Warning part of the About form and write the following sentence at its Caption property:

No Warning is needed. Just enjoy!

40. To call the ready-made frmAbout in our menu system, embed the following code into Form1 procedure:

```
Private Sub mnuAboutMe_Click()
 frmAbout.Show
End Sub
```

After designing and modifying the About Form, it should look like the following figure when you run the menu system:


About ProjectMnu2

ProjectMnu2

Version 1.0.0

This program will demonstrate how to design and develop simple menu system. This features the basic controls, simple calculator, and About menu. This program is not copyrighted by any International laws.

No Warning is needed. Just enjoy!

41. Lastly, embed the following at Form1 procedure for the Close menu item under About menu:

```
Private Sub mnuAboutClose_Click()
 Unload Me
End Sub
```

42. Now run the simple application menu system by clicking the Run icon at the toolbar.

Then try to click each menu item in each menu title and see for yourself how the menu is responding. I hope it responds positively.

Explanation:

There are two most important Menu control properties in the Menu Editor; they are the Caption and the Name. The Caption is the text that appears on the control ,while the Name is the name we use to reference the menu control from the program.
The menu control list box which is located at the lower portion of the Menu Editor will list all the menu controls for the current form that we are designing. When we type a menu item in the Caption text box, that item also appears in the menu control list box. By selecting an existing menu control from the list box, we can change or modify the properties for that control.
The position of the menu control in the menu control list box determines whether the control is a menu title, menu item, or submenu item. So a menu control that appears at the extreme left in the list box is displayed on the menu bar as a menu title (in our example, they are the &Control, &Calculate and &About menu titles), while a menu control that is indented in the list box (with 4 dots) is displayed on the menu when the user clicks the preceding menu title. In other words, it is among the menu items of a particular menu title. In our example, under the Control menu title, the menu items are: ….C&ommand button, ….&Text box and ….&Check box. Under the &Calculate menu title, the menu items are: ….&Add, ….&Subtract,….&Multiply and ….&Divide. While under the &About menu title, the menu items are: ….&Me? and ….&Close.

First, we have to embed the following code at the **(General)** and (**Declarations**) sections in our program. Select now the General item at the Object listbox and the Declaration item at the Procedure listbox.

(General)	**(Declarations)**

```
Private txtNum1, txtNum2, txtSum As Integer
```

The procedure listbox for a General section of our module contains a single selection - the Declarations section, where we place module-level variable, a constant, or even the DLL (Dynamic Link Library) declaration.

Here in our example, we declare all our variables as Private because we make it sure that only the procedures within the Sum1 module can see them. Other procedures in other modules cannot see them, therefore these private variables cannot affect nor interfere the operation of other outside modules.
In our procedure Private Sub Form_Load( ), we initialize the variables txtNum1, txtNum2, and txtSum with a value of zero(0) as what we can see in the following code:

```
  Private Sub Form_Load()
→  txtNum1 = 0
→  txtNum2 = 0
→  txtSum = 0
   Text1.Text = nNum1
   Text2.Text = nNum2
   Text3.Text = nSum
  End Sub
```

Our purpose for the initialization of values for these variables is to ensure that the moment the form is loaded, they contain a zero value so that the resulting calculation in our equation is correct. Forgetting to do so, will result to undesirable output. You will notice also that we put the value of the variable nNum1 to Text1.Text as well as the value of nNum2 to Text2.Text and the value of nSum to Text3.Text as we can see in the following code:

```
Private Sub Form_Load()
 txtNum1 = 0
 txtNum2 = 0
 txtSum = 0
 Text1.Text = nNum1 ←
 Text2.Text = nNum2 ←
 Text3.Text = nSum  ←
End Sub
```

The implicit declarations of variable nNum1, nNum2, and nSum are intentional in our part, because we want it to produce a blank text box. The “implicit” means we didn’t formally declare a variable at the top of our program before using it. This is acceptable in Visual Basic, Visual FoxPro or Visual C++ but not in older programming languages. You have to take note that this one is not a recommended practice nor a good programming style. We just did this to force our text box to become blank or empty when we run our program.

In this code:

```
Private Sub Command1_Click()
 txtSum = txtNum1 + txtNum2
 Text3.Text = Str(txtSum)
End Sub
```

we formulate the equation txtSum = txtNum1 + txtNum2 to sum up the two input numbers in text box 1 and text box 2 respectively. Then we convert the computed value that is stored in txtSum variable into a string value before we display it at the text box 3 with this code:

```
Text3.Text = Str(txtSum)
```

This is needed to display the number properly. Forgetting to do so will result to undesirable output. You will notice also that we convert the input numbers at text box 1 and text box 2 into its equivalent numeric value using the Val( ) function to properly calculate them in our equation. We can see the conversion syntax on the following code:

```
Private Sub Text1_Change()
  txtNum1 = Val(Text1.Text)
End Sub
```

```
Private Sub Text2_Change()
  txtNum2 = Val(Text2.Text)
End Sub
```

When we input the numbers in our text box, they are treated as string data. Therefore, converting them to numeric data are needed, otherwise they cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.
We use the Integer data type in declaring our variable such as txtNum1, txtNum2, and txtSum, since we are sure that these three ordinary variables will contain whole numbers. In the case of a variable that holds a number with a fractional part, we have to use the Double data type declaration to accommodate the required capacity of bigger numbers to store them in the computer’s memory or storage device. Remember that numbers with decimal points need longer bytes to fit in the storage capacity requirements.
You will notice that the explanation here at the Add procedure is applicable to Subtract, Multiply and Divide procedures too. So there is no need for me to repeat the explanation, buddy. Or should I?

Designing a Toolbar

The toolbar is located right below the menu titles in almost all application software such as Microsoft Word, Microsoft Excel, or Microsoft PowerPoint. They are represented by icons that signify their use such as a blank paper sheet icon for creating new document, an open folder icon for opening a new document, a picture of a diskette for saving your document, and a picture of a printer to print your presently typed document.
Through these icons or pictures, a toolbar can provide us a quick access to the most frequently used menu commands in an application such as opening, saving or printing a document. This is in the case of the application software. We can imitate this strategy also in our business application system which we would like to design and develop. That is why it is important to follow a certain standard in designing menus and toolbars. As much as possible, the menu and toolbar that we will design and develop should mimic the existing icons used in most popular software application. In this way, the target user of our application can easily adapt to it without the need of a manual or an online help. Let us have our example now on how to design and develop this so called "toolbar".

Example 3:

Design and develop a simple menu-based application system that demonstrates how to create a simple toolbar with the following icons: blank paper, open folder, diskette, and printer. Follow the given figures below in designing and developing the application program:

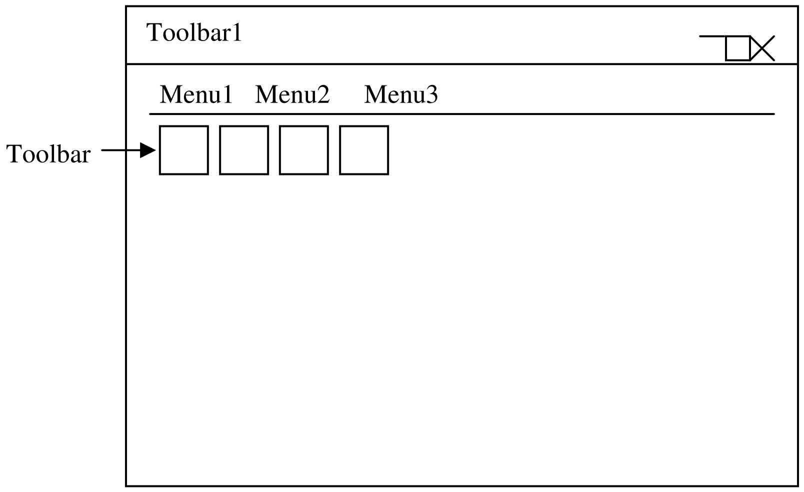


Figure 6.9 Creating a toolbar.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button.
3. Open the Menu Editor by selecting Tools at the Visual Basic main menu.
4. Create the Menu1 menu by typing &**Menu1** in the Caption field, and **mnuMenu1** in the Name field.
5. Click the Next button to start the next menu item. Type **&M1 Item1** in the Caption field, and **mnuMenu1Item1** in the Name field. We want to make this a menu item of the Menu1 menu, so click the right arrow button to indent this menu item. You can see here the four dots before the menu item's caption, signifying that it is a menu item of the Menu1 menu.
6. Now click the Next button again to type the second menu item: &**M1 Item2** in the Caption field and **mnuMenu1Item2** in the Name field.
7. Click the Next button and type the third menu item of the Menu1 menu: &**M1 Item3** in the Caption field and **mnuMenu1Item3** in the Name field.
8. Click the Next button and type the fourth menu item (which is also the last menu item of the Menu1 item): **&M1 Item4** in the Caption field and **mnuMenu1Item4** in the Name field.
9. Click the Next button and this time click the left arrow to outdent the new menu item which we will name: Menu2. You will see that the four dots had disappeared, signifying that the thing we will type is a new menu item.Type now the &**Menu2** in the Caption field and **mnuMenu2** in the Name field.
10. Now click the Next button to type the first menu item under the Menu2 menu. Again we have to click the right arrow to indent this menu item. Did you click it now? If so, you can see the four dots displayed. Type &**M2 Item1** in the Caption field and **mnuMenu2Item1** in the Name field.
11. Click again the Next button to type the next menu item. Type &**M2 Item2** in the Caption field and **mnuMenu2Item2** in the Name field.
12. Click again the Next button to type the next menu item. Type **&M2 Item3** in the Caption field and **mnuMenu2Item3** in the Name field.
13. This time, we have to click the Next button and click the left arrow to outdent the new menu item which we will name: Menu3. You will see by now that the four dots had disappeared, signifying that the word we are about to type is a menu item. Type now the &**Menu3** in the Caption field and **mnuMenu3** in the Name field.
14. Now click the Next button to type the first menu item under the Menu3 menu. Again we have to click the right arrow to indent this menu item. Type &**M3 Item1** in the Caption field and **mnuMenu3Item1** in the Name field.
15. Finally click the Next button to type the last menu item. Type &**M3 Item2** in the Caption field and **mnuMenu3Item2** in the Name field.
16. This time click the OK button to run the menu we have created.

17. Embed the following code at Form1 so that when we click each individual item in each menu, a message box will appear confirming that we indeed click that particular menu item in a particular menu title.

```
Private Sub mnuMenu1Item1_Click()
 MsgBox "Menu1 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu1Item2_Click()
 MsgBox "Menu1 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu1Item3_Click()
 MsgBox "Menu1 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu1Item4_Click()
  MsgBox "Menu1 Item4 clicked"
End Sub
```

```
Private Sub mnuMenu2Item1_Click()
  MsgBox "Menu2 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu2Item2_Click()
 MsgBox "Menu2 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu2Item3_Click()
 MsgBox "Menu2 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu3Item1_Click()
 MsgBox "Menu3 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu3Item2_Click()
 MsgBox "Menu3 Item2 clicked"
End Sub
```

Let us create a toolbar to enhance the menu we just created in example number 1. Before we can work on toolbar designing, we need to be sure it is added to the Visual Basic Toolbox.

18. If we need to add the Toolbar control, we right-click the Toolbox, and click the Components menu item. When the Components dialog box appears, mark a check at the box next to the Microsoft Windows Common Controls 6. Next, click the

Apply command button. Can you see them now (as they appear at the lower part of the Toolbox)? The toolbar control and its related controls are now added at the bottom part of the Visual Basic Toolbox temporarily. Finally, click the Close command button to close the Components dialog box.

19. Now let us add a Toolbar control to our form.

You can observe that it creates a blank space, below the menu titles. This is where the toolbar icons are positioned.

20. From the Properties window (of the Toolbar object), select and click the Custom property.
21. When the Property Pages dialog box appears, select the Buttons tab.
22. Click the Insert Button to place the first button on the toolbar. Set the Key to **Paper** and leave other default values as is.
23. Click again the Insert Button to place the second button on the toolbar. Set the Key to **Folder**.
24. Next, click again the Insert Button to place the third button on the toolbar. Set the Key to **Diskette**.
25. Click again the Insert Button to place the fourth button on the toolbar. Set the Key to **Printer**.
26. Now click the OK button.

This time, we need to add an image(also called icon or picture) to the toolbar.

27. Now add an Image List to our Form. You will notice that this Image List control appears at the center of our form. This means that we can now add an icon to our toolbar. Don’t worry about this ImageList object which appear right at the center of our form. It is displayed there temporarily, because the moment we run our program, this ImageList object will disappear from our form. Believe me, buddy, it’s true. Its appearance will only guide us that we can now add a list of image in our toolbar. That is the only purpose why it appears right at the center of our form, temporarily.
28. Let us now open up the Property Pages (of the ImageList object) by selecting and clicking the Custom property from the Properties window.
29. When the Property Pages dialog box appears, click the Images tab so you can add some images.

The first image or icon that we want to add is the blank paper page, which is used to represent a New file.

30. Click the Insert Picture now.

Now we are instructed to select a picture. How can we find the picture of a blank paper page? It is easy. Let us go now at the Windows Start menu and click the Search menu item for Files and Folders. Let us now search for the Folder which name is **Graphics**. Type the word: Graphics at the text box, then click the Search Now button. Can you see now the Graphics folder at the Result at the Search Results window?

31. This time, you have to open the Graphics folder and select the New, Open, Save, and Print bmps for copying them into the Graphics folder under the VB98 folder. But first, you have to Create a New Folder named: Graphics. This Graphics folder should contain the New, Open, Save and Print bmps (the icons you have copied from other Graphics folder somewhere out there in your hard disks resources).
32. This time, select the picture New bmp at the Graphics folder and click the Open button. You will see that the blank paper page appears in the Property Pages.
33. Click again the Insert Picture to insert the next icon. This time select the Open bmp and click the Open button You will see that the open folder is added in the Property Pages.
34. Click the Insert Picture again to insert the two remaining icons. This time select the Save, then the Print bmps so that they can be added in the Property Pages. After seeing the four icons, click the OK button to close the Property Pages dialog box.
35. Select the Toolbar1 object and open its property pages again by selecting the Custom property from the Properties window.
36. On the General tab, set the ImageList property to **ImageList1.**
37. Now click the Buttons tab of the Property Pages.
38. Set the Index property to 1 (which is the default value) and set the Image property to 1. This will place the image (icon) in index position 1, the blank paper page on the first button which is also located at position 1 in the toolbar.
39. Next, set the Index property to 2 and set the Image property to 2. This places the Open folder icon on the second button on the toolbar.
40. Now set the Index property to 3 and set the Image property to 3. This places the Diskette icon on the third button on the toolbar. Finally, set the Index property to 4 and the Image property to 4. This places the Printer icon on the fourth button on the toolbar.
41. After we are done with all the settings of values for Index and Image properties, we have to click the OK button.
42. Now add the following code to the Toolbar1_ButtonClick( ) event (by double-clicking the Toolbar at the Form; double-clicking it will bring us to this event procedure):

```
 Private Sub Toolbar1_ButtonClick(ByVal Button As
MSComctlLib.Button)
Select Case Button.Key
 Case Is = "Paper"
  MsgBox "Paper icon is clicked!"
  'Form2.Show can be called from here too
 Case Is = "Folder"
  MsgBox "Folder icon is clicked!"
  'Form3.Show can be called from here too
 Case Is = "Diskette"
  MsgBox "Diskette icon is clicked!"
  'Form4.Show can be called from here too
 Case Is = "Printer"
  MsgBox "Printer icon is clicked!"
```

```
   'Form5.Show can be called from here too
  End Select
End Sub
```

43. Now run the simple application menu system by clicking the Run icon at the toolbar.

Then try to click each menu item in each menu title and see for yourself how the menu is responding. This time, try to click also the toolbar icons if they too are responding. I hope they respond positively.
Whew!!! What a long and winding code! But its worth the effort, isn't it?

Explanation:

Here, we are using the MsgBox method to display a message when the user clicks a particular menu item. The message tells what particular menu item was clicked and in what particular menu title that menu item belonged to.
Notice that we used a Select…Case conditional statement here in our code to determine which icon in our toolbar was clicked. We are able to manipulate each icon in the toolbar through setting the Key property. For example, we set the Key property of the first button as Paper, so once the user clicks the Paper button, the code that we would like to execute can be inserted within the following Select…Case statement:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
Select Case Button.Key
 Case Is = "Paper"
  MsgBox "Paper icon is clicked!"             ←
  'Form2.Show can be called from here too     ←
 Case Is = "Folder"
  MsgBox "Folder icon is clicked!"
  'Form3.Show can be called from here too
 Case Is = "Diskette"
  MsgBox "Diskette icon is clicked!"
  'Form4.Show can be called from here too
 Case Is = "Printer"
  MsgBox "Printer icon is clicked!"
  'Form5.Show can be called from here too
 End Select
End Sub
```

The same thing goes to other manipulation of icons in the toolbar. By using the Key property, we can simply insert the code within the Case…Select statement to whatever we want that icon to respond or execute. It seems so easy, isn't it?

Example 4:

Design and develop a simple menu-based application system that demonstrates how to create a simple toolbar with the following icons: blank paper, open folder, diskette, and printer. This time, once the user clicks the icon in the toolbar, another form will appear. For the blank paper icon, a Form that contains a command button will appear. At the folder icon, a Form that contains a text box will appear, then at the diskette icon, a Form that contains an check box will appear, and finally at the printer icon, a Form that contains a option button will appear. Follow the given figures below in designing and developing the application program:

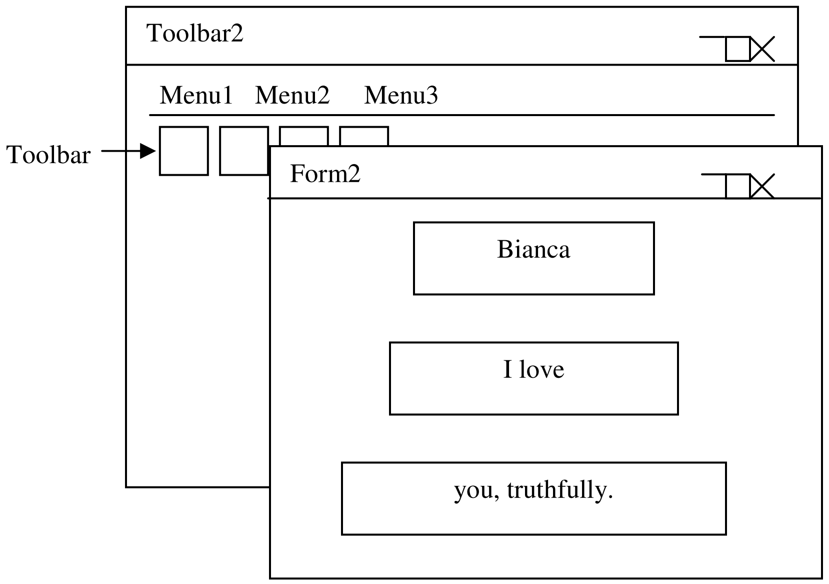


Figure 6.10 Creating a toolbar that calls multiple forms.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button.
3. Open the Menu Editor by selecting Tools at the Visual Basic main menu.
4. Create the Menu1 menu by typing &**Menu1** in the Caption field, and **mnuMenu1** in the Name field.
5. Click the Next button to start the next menu item. Type **&M1 Item1** in the Caption field, and **mnuMenu1Item1** in the Name field. We want to make this a menu item of the Menu1 menu, so click the right arrow button to indent this menu item. You can see here the four dots before the menu item's caption, signifying that it is a menu item of the Menu1 menu.
6. Now click the Next button again to type the second menu item: &**M1 Item2** in the Caption field and **mnuMenu1Item2** in the Name field.

7. Click the Next button and type the third menu item of the Menu1 menu: &**M1 Item3** in the Caption field and **mnuMenu1Item3** in the Name field.
8. Click the Next button and type the fourth menu item (which is also the last menu item of the Menu1 item): **&M1 Item4** in the Caption field and **mnuMenu1Item4** in the Name field.
9. Click the Next button and this time click the left arrow to outdent the new menu item which we will name: Menu2. You will see that the four dots had disappeared, signifying that the thing we will type is a new menu item.Type now the &**Menu2** in the Caption field and **mnuMenu2** in the Name field.
10. Now click the Next button to type the first menu item under the Menu2 menu. Again we have to click the right arrow to indent this menu item. Did you click it now? If so, you can see the four dots displayed. Type &**M2 Item1** in the Caption field and **mnuMenu2Item1** in the Name field.
11. Click again the Next button to type the next menu item. Type &**M2 Item2** in the Caption field and **mnuMenu2Item2** in the Name field.
12. Click again the Next button to type the next menu item. Type **&M2 Item3** in the Caption field and **mnuMenu2Item3** in the Name field.
13. This time, we have to click the Next button and click the left arrow to outdent the new menu item which we will name: Menu3. You will see by now that the four dots had disappeared, signifying that the word we are about to type is a menu item. Type now the &**Menu3** in the Caption field and **mnuMenu3** in the Name field.
14. Now click the Next button to type the first menu item under the Menu3 menu. Again we have to click the right arrow to indent this menu item. Type &**M3 Item1** in the Caption field and **mnuMenu3Item1** in the Name field.
15. Finally click the Next button to type the last menu item. Type &**M3 Item2** in the Caption field and **mnuMenu3Item2** in the Name field.
16. This time click the OK button to run the menu we have created.
17. Embed the following code at Form1 so that when we click each individual item in each menu, a message box will appear confirming that we indeed click that particular menu item in a particular menu title.

```
Private Sub mnuMenu1Item1_Click()
 MsgBox "Menu1 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu1Item2_Click()
 MsgBox "Menu1 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu1Item3_Click()
 MsgBox "Menu1 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu1Item4_Click()
  MsgBox "Menu1 Item4 clicked"
End Sub
```

```
Private Sub mnuMenu2Item1_Click()
  MsgBox "Menu2 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu2Item2_Click()
 MsgBox "Menu2 Item2 clicked"
End Sub
```

```
Private Sub mnuMenu2Item3_Click()
 MsgBox "Menu2 Item3 clicked"
End Sub
```

```
Private Sub mnuMenu3Item1_Click()
 MsgBox "Menu3 Item1 clicked"
End Sub
```

```
Private Sub mnuMenu3Item2_Click()
 MsgBox "Menu3 Item2 clicked"
End Sub
```

Let us create a toolbar to enhance the menu we just created in example number 1. Before we can work on toolbar designing, we need to be sure it is added to the Visual Basic Toolbox.

18. If we need to add the Toolbar control, we right-click the Toolbox, and click the Components menu item. When the Components dialog box appears, mark a check at the box next to the Microsoft Windows Common Controls 6. Next, click the Apply command button. Can you see them now (as they appear at the lower part of the Toolbox)? The toolbar control and its related controls are now added at the bottom part of the Visual Basic Toolbox temporarily. Finally, click the Close command button to close the Components dialog box.

19. Now let us add a Toolbar control to our form.

    You can observe that it creates a blank space, below the menu titles. This is where the toolbar icons are positioned.

20. From the Properties window (of the Toolbar object), select and click the Custom property.
21. When the Property Pages dialog box appears, select the Buttons tab.
22. Click the Insert Button to place the first button on the toolbar. Set the Key to **Paper** and leave other default values as is.
23. Click again the Insert Button to place the second button on the toolbar. Set the Key to **Folder**.
24. Next, click again the Insert Button to place the third button on the toolbar. Set the Key to **Diskette**.

25. Click again the Insert Button to place the fourth button on the toolbar. Set the Key to **Printer**.
26. Now click the OK button.

This time, we need to add an image(also called icon or picture) to the toolbar.

27. Now add an Image List to our Form. You will notice that this Image List control appears at the center of our form. This means that we can now add an icon to our toolbar. Don't worry about this ImageList object which appear right at the center of our form. It is displayed there temporarily, because the moment we run our program, this ImageList object will disappear from our form. Believe me, buddy, it's true. Its appearance will only guide us that we can now add a list of image in our toolbar. That is the only purpose why it appears right at the center of our form, temporarily.
28. Let us now open up the Property Pages (of the ImageList object) by selecting and clicking the Custom property from the Properties window.
29. When the Property Pages dialog box appears, click the Images tab so you can add some images.

The first image or icon that we want to add is the blank paper page, which is used to represent a New file.

30. Click the Insert Picture now.

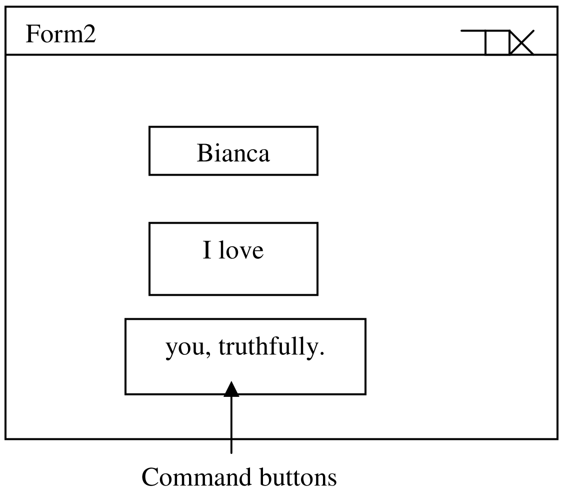
Now we are instructed to select a picture. How can we find the picture of a blank paper page? It is easy. Let us go now at the Windows Start menu and click the Search menu item for Files and Folders. Let us now search for the Folder which name is **Graphics**. Type the word: Graphics at the text box, then click the Search Now button. Can you see now the Graphics folder at the Result at the Search Results window?

31. This time, you have to open the Graphics folder and select the New, Open, Save, and Print bmps for copying them into the Graphics folder under the VB98 folder. But first, you have to Create a New Folder named: Graphics. This Graphics folder should contain the New, Open, Save and Print bmps (the icons you have copied from other Graphics folder somewhere out there in your hard disks resources)
32. This time, select the picture New bmp at the Graphics folder and click the Open button. You will see that the blank paper page appears in the Property Pages.
33. Click again the Insert Picture to insert the next icon. This time select the Open bmp and click the Open button You will see that the open folder is added in the Property Pages.
34. Click the Insert Picture again to insert the two remaining icons. This time select the Save, then the Print bmps so that they can be added in the Property Pages. After seeing the four icons, click the OK button to close the Property Pages dialog box.
35. Select the Toolbar1 object and open its property pages again by selecting the Custom property from the Properties window.
36. On the General tab, set the ImageList property to **ImageList1.**
37. Now click the Buttons tab of the Property Pages.

38. Set the Index property to 1 (which is the default value) and set the Image property to 1. This will place the image (icon) in index position 1, the blank paper page on the first button which is also located at position 1 in the toolbar.
39. Next, set the Index property to 2 and set the Image property to 2. This places the Open folder icon on the second button on the toolbar.
40. Now set the Index property to 3 and set the Image property to 3. This places the Diskette icon on the third button on the toolbar. Finally, set the Index property to 4 and the Image property to 4. This places the Printer icon on the fourth button on the toolbar.
41. After we are done with all the settings of values for Index and Image properties, we have to click the OK button.
42. We have to put a code for each icon in the toolbar of our menu system application. What we have to do is to add a form. Click now the Add Form at the Visual Basic toolbar. Then click the Open button. We are now in Form2.

Since our menu objects are in Form1, the Form2 that contains the Command buttons will be called by clicking the blank paper page icon while the Form3 that contains the Text box will be called by clicking the folder icon. The Form4 that contains the Check boxes will be called by clicking the diskette icon while the Form5 that contains the Option button will be called by clicking the printer icon in the toolbar of our menu application system.

43. Design the following specification in Form2 for our Command button menu item:

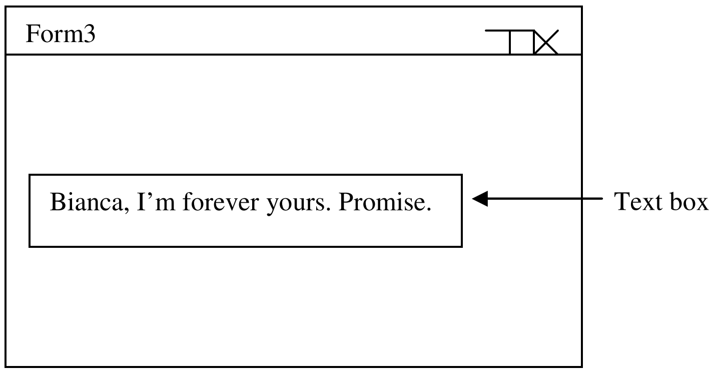


The caption that you will assign for the first button is “Bianca”, the caption of the second button is “I love” and the third button’s caption is “you, truthfully!”.

Try to run the menu program by clicking the Run icon at the toolbar.

Does it work? I hope so.

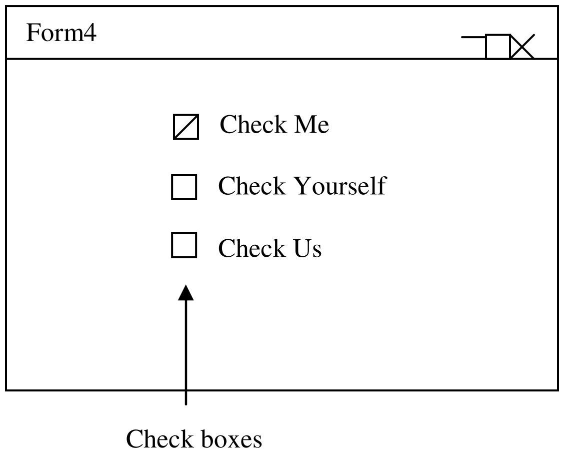
44.This time, add another form. Click the Add Form at the Visual Basic toolbar, then click the Open button. This must be Form3, right? Now design the given specification below for our Text box menu item:



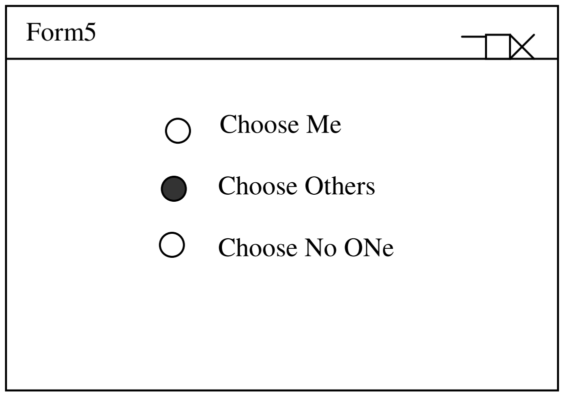
45. Embed the following code into Form3 procedure, so that the message is displayed at the text box automatically upon loading the Form3:

```
Private Sub Form_Load()
 Text1.Text = "Bianca,I'm forever yours.Promise."
End Sub
```

46. Add another form. Click the Add Form at the Visual Basic toolbar, then click the Open button. This must be Form4. Now design the given specification below for our Text box menu item:



47. Add another form. Click the Add Form at the toolbar, then click the Open button. This must be Form5. Now design the given specification below for our Text box menu item:



48.Now add the following code to the Toolbar1_ButtonClick( ) event (by double-clicking the Toolbar at the Form; double-clicking it will bring us to this event procedure):

```
Private Sub Toolbar1_ButtonClick(ByVal Button As
MSComctlLib.Button)
Select Case Button.Key
 Case Is = "Paper"
  'Calling the Command Buttons form
  Form2.Show
 Case Is = "Folder"
  'Calling the Text box form
  Form3.Show
 Case Is = "Diskette"
  'Calling the Check box form
  Form4.Show
 Case Is = "Printer"
  'Calling the Option button form
  Form5.Show
 End Select
End Sub
```

49. Now run the simple application menu system by clicking the Run icon at the toolbar.

Then try to click each menu item in each menu title and see for yourself how the menu is responding. This time, try to click also the toolbar icons if they too are responding. I hope they respond positively.

Explanation:

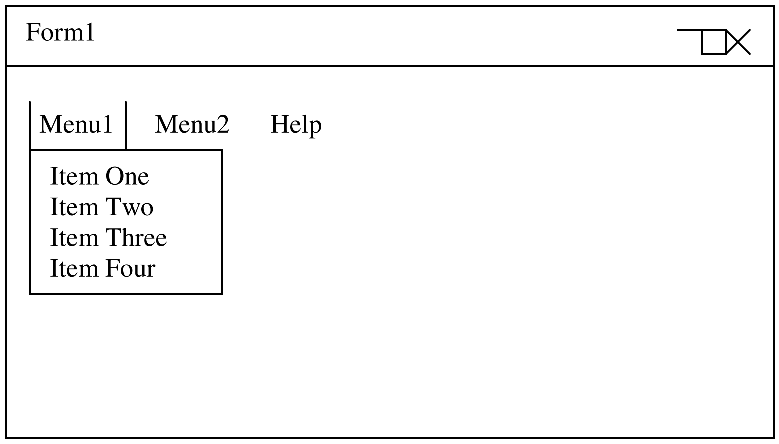
Here, we are using the MsgBox method to display a message when the user clicks a particular menu item. The message tells what particular menu item was clicked and in what particular menu title that menu item belonged to.
Notice that we used a Select…Case conditional statement here in our code to determine which icon in our toolbar was clicked. We are able to manipulate each icon in the toolbar through setting the Key property. For example, we set the Key property of the first button as Paper, so once the user clicks the Paper button, the code that we would like to execute can be inserted within the following Select…Case statement:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
Select Case Button.Key
 Case Is = "Paper"
  'Calling the Command Buttons form
  Form2.Show   ←
 Case Is = "Folder"
  'Calling the Text box form
  Form3.Show
 Case Is = "Diskette"
  'Calling the Check box form
  Form4.Show
 Case Is = "Printer"
  'Calling the Option button form
  Form5.Show
 End Select
End Sub
```

The same thing goes to other manipulation of icons in the toolbar. By using the Key property, we can simply insert the code within the Case…Select statement to whatever we want that icon to respond or execute. In this example, once the user clicks an icon at the toolbar, a particular Form object was being executed or called (or displayed). It seems so easy, isn't it?

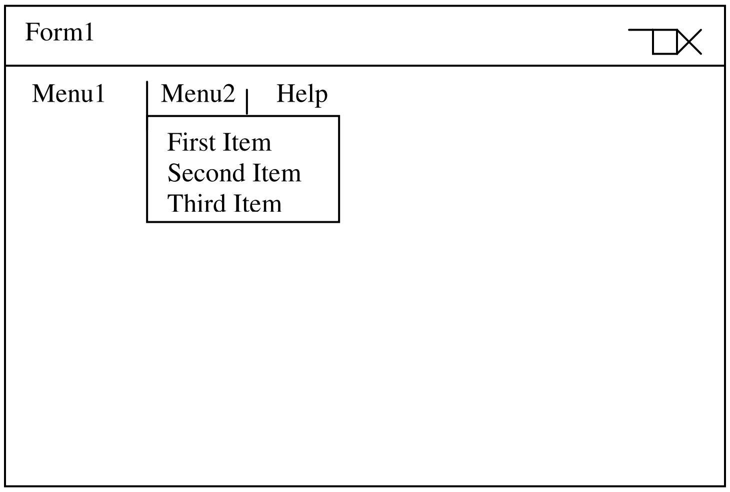
LAB ACTIVITY
TEST 5

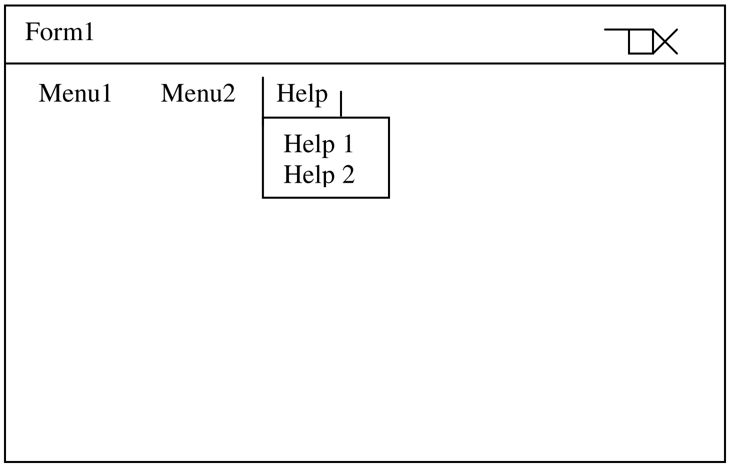
1. Design and develop a simple menu-based application system that demonstrates how to create a simple menu system with the following menu title: Menu1, Menu2, and Help. Follow the given figures below in designing and developing the application program:



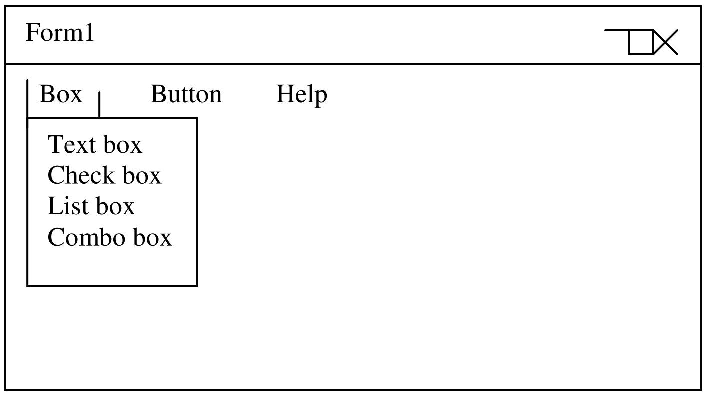
Note:

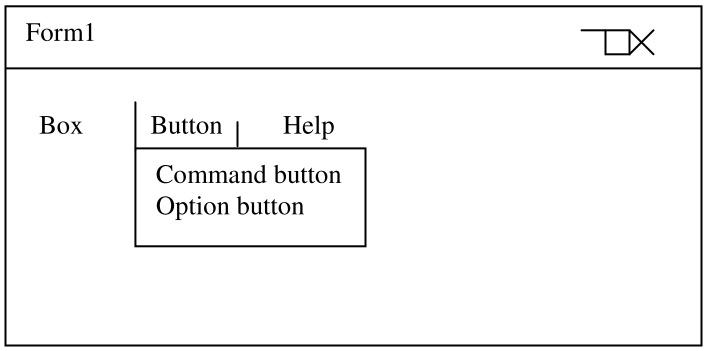
When the user clicks the Item One, the Message box will display “Item One clicked!”. When the user clicks the Item Two, the Message box will display “Item Two clicked!” and so on. When the user clicks the First Item, the Message box will display “First Item clicked!”. When the user clicks the Second Item, the Message box will display “Second Item clicked!” and so on.
When the user clicks the Help 1, the Message box will display “Help 1 clicked!”. And when the user clicks the Help 2, the Message box will display “Help 2 clicked!”.

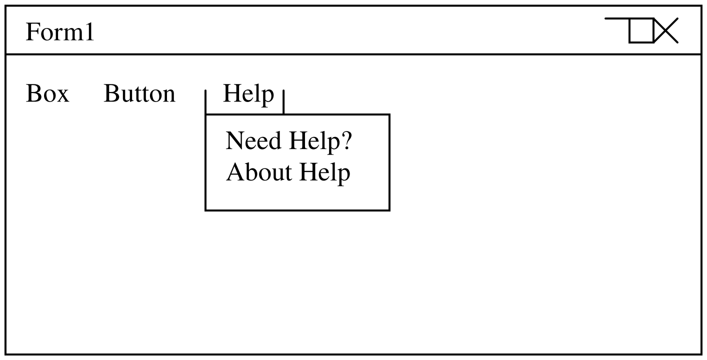




2. Design and develop a simple menu-based application system that demonstrates how to create a simple menu system with the following menu title: Box, Button, and Help. Follow the given figures below in designing and developing the application program:

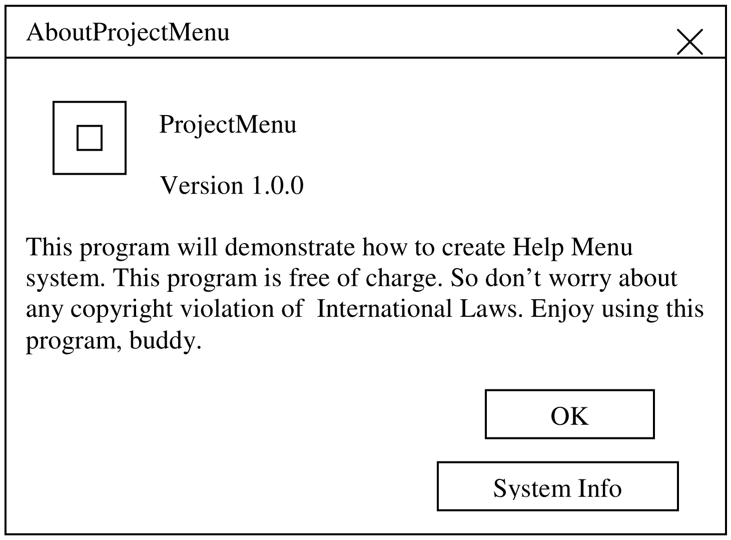






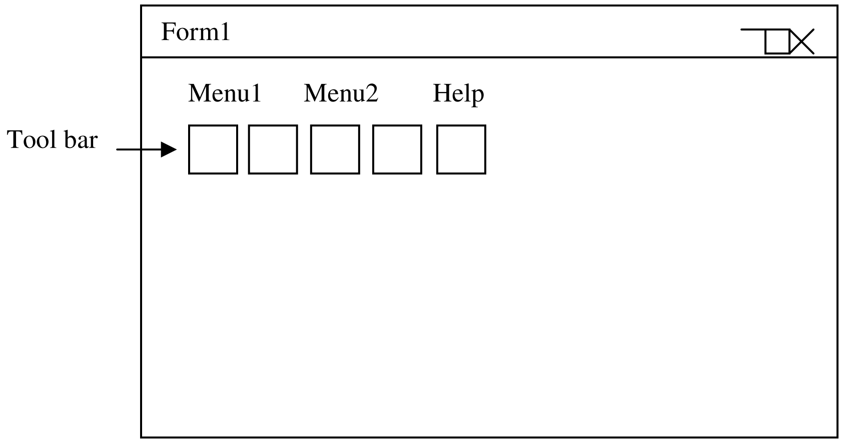
Note:

When the user clicks the Text box, the Form2 that contains Text boxes will appear. When the user clicks Check box, the Form3 that contains Check boxes will appear, and so on. It's up to you what kind of box design you want, as long as that particular box or button, will appear when it is selected or clicked!
When the user clicks the Need Help, the Message box will display "Need help? Call 911!" When the user selects About Help, the following About dialog box should appear:



This program will demonstrate how to create Help Menu system. This program is free of charge. So don't worry about any copyright violation of International Laws. Enjoy using this program, buddy.

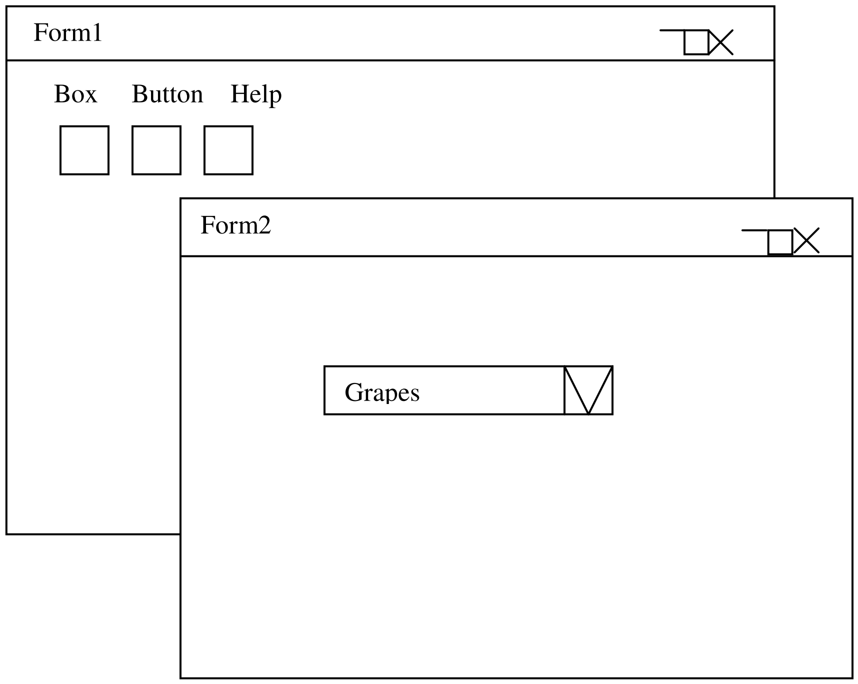
3. Design and develop a simple menu-based application system that demonstrates how to create a simple toolbar with the following icons: blank paper, open folder, diskette, printer and a filter. Follow the given figures below in designing and developing the application program:



Note:

When the user clicks the first button at the toolbar of our simple menu system, the Message box that says "First button clicked!" will be displayed. When the user clicks the second button at the toolbar, the Message box that says "Second button clicked!" will be displayed and so on. The icon of the first button (in the toolbar) is a blank paper page, the icon of the second button is an open folder, the icon of the third button is a diskette, the icon of the fourth button is a printer, and the icon of the fifth button is a filter.

4. Design and develop a simple menu-based application system that demonstrates how to create a simple toolbar with the following icons: blank paper, open folder, diskette, and printer. This time, once the user clicks the icon in the toolbar, another form will appear. Follow the given figure below in designing and developing the application program:



Note:

When the user clicks the first button, the Form2 that contains a Combo box (preloaded with 5 or more items) will be displayed. When the user clicks the second button, the Form3 that contains a List box (preloaded with 5 or more items) will be displayed. When the user clicks the third button, the Form4 that contains both Combo box and List box (both are preloaded with 5 or more items) will be displayed. It's up to you what kind of items you would like to put at the list box and combo box. These 5 or more items could be a list of your favorite fruits such as Banana, Apple, Orange, Grapes, Cherries, or Pears. Or a list of your favorite foods: pizza, burger, sandwich, hotdog, or pie. Indulge now with these foods (joke). Satisfy your hunger (joke, again). I think I'm hungry now (what do you think?).

Chapter 7

Other Basic Controls

In this chapter, we will learn how to use some other important basic controls of Visual Basic language. We already know that the Toolbox contains the tools we use to draw controls on our forms. You will notice that there are still controls we haven't yet used or included in our previous application system. The reason for these is that they are not yet needed in our simple application system and putting them is not appropriate for the requirements of our system.
Surely, we will not cover these other controls in this chapter. However, we will try our very best to choose those controls which we think are much needed in a real-world business application system that we would like to design and develop in the near future. So let us start studying them now.

Using the Timer Control

The Timer control can check and display the computer system clock. Some actual application of a timer control is to check the system clock if it is time to display an important message such as, "It's 12 noon, time to go to lunch!". Or a time to accomplish a certain task such as giving a list of a burger chain products to be sold at 30 percent discount because they are not sold within 30 minutes after they were cooked. You can think of some real-world application too. What we will do now is to come up with a simple example of this timer control.

Example 1:

Design and develop a simple timer-control application system that demonstrates how it works. Follow the given figure below in designing and developing the application program

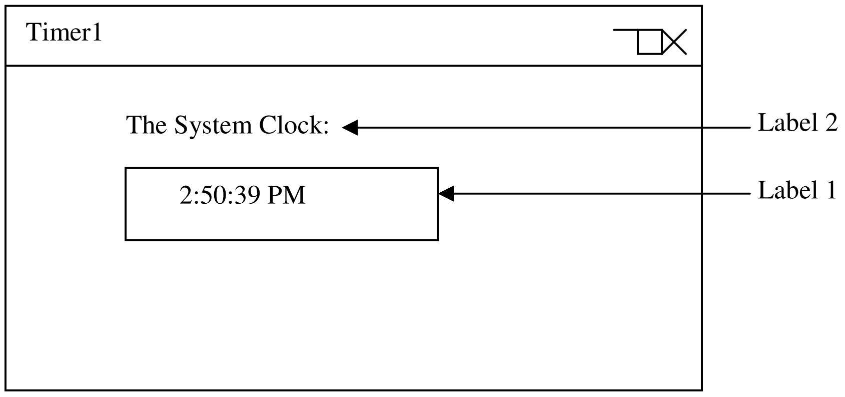


Figure 7.2 Using Timer control within a Label control:

Note:

The Timer control should be drawn at the right side of Label 1.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Timer1.**
3. Double-click two Labels and a Timer control in the Toolbox to add them into the form. Position them properly side by side.
4. Follow the property settings for each respective control below:

Control	**Property**	**Setting**
Label1	BorderStyle	Fixed Single
Timer1	Interval	500
Timer1	Enabled	True

5. Double-click now the Timer control on the form to display the Code editor. Embed the following code to this only one procedure of the Timer control:

```
Private Sub Timer1_Timer()
 If Label1.Caption <> CStr(Time) Then
     Label1.Caption = Time
 End If
End Sub
```

6. Now run the simple Timer application system by clicking the Run icon at the toolbar.

Explanation:

The Timer1_Timer( ) procedure simply displays the system time at the Label 1 control by calling the Time function with the following code:

```
Private Sub Timer1_Timer()
 If Label1.Caption <> CStr(Time) Then
   Label1.Caption = Time ←
 End If
End Sub
```

The Time function returns a Variant data type that contains the current time of the system clock of the computer as a date/time value. When we assign it to a string variable or property, which in our case this is a Caption property of the Label 1, the Visual Basic converts it to a string using the time format specified in the Control Panel of our Windows 2000 operating system.
Furthermore, the Timer1_Timer( ) event takes place every time an interval elapses. The interval is determined by the Interval property. Once an interval has elapsed, the timer generates its own Timer event. It accomplished this by checking the system clock at frequent intervals.
The 500 interval set up simply means half a second. The Interval property is measured in milliseconds. Meaning, if we want to count in seconds, we have to multiply the number of seconds by 1,000. So 1 second is equivalent to 1,000 milliseconds (therefore, 500 is a half second, isn't it?).

Example 2:

Design and develop a simple timer-control application system that demonstrates how it works with a bigger font size of the clock timer. Follow the given figure below in designing and developing the application program

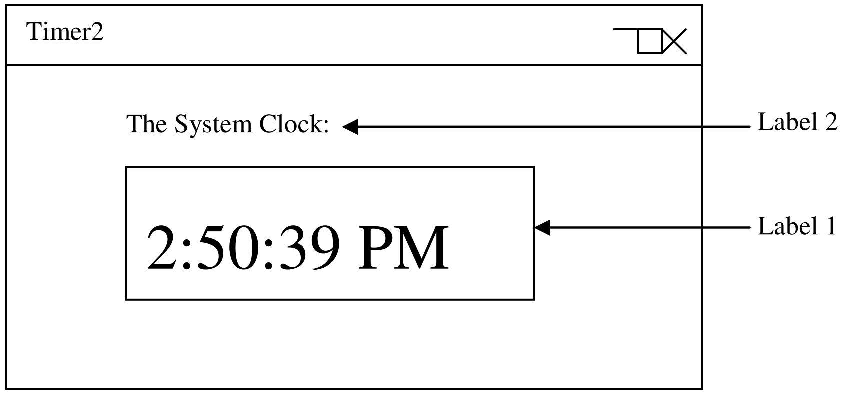


Figure 7.1 Using Timer control with a bigger Label control font size

Note:

The Timer control should be drawn at the right side of Label 1.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.

2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Timer2.**
3. Double-click two Labels and a Timer control in the Toolbox to add them into the form. Position them properly side by side.
4. Follow the property settings for each respective control below:

Control	Property	Setting
Label1	BorderStyle	Fixed Single
Label1	Font	24 (Size:)
Label1	AutoSize	True
Timer1	Interval	500
Timer1	Enabled	True

5. Double-click now the Timer control on the form to display the Code editor. Embed the following code to this only one procedure of the Timer control:

```
Private Sub Timer1_Timer()
 If Label1.Caption <> CStr(Time) Then
     Label1.Caption = Time
 End If
End Sub
```

6. Now run the simple Timer application system by clicking the Run icon at the toolbar.

Explanation:

By setting the font size to 24 of the Font property of Label 1 and set its AutoSize property to True, we can display the clock timer bigger in size. It becomes so visible enough even in a far distance.
The Timer1_Timer( ) procedure simply displays the system time at the Label 1 control by calling the Time function with the following code:

```
Private Sub Timer1_Timer( )
 If Label1.Caption <> CStr(Time) Then
    Label1.Caption = Time ◄—
 End If
End Sub
```

The Time function returns a Variant data type that contains the current time of the system clock of the computer as a date/time value. When we assign it to a string variable or property, which in our case this is a Caption property of the Label 1, the Visual Basic converts it to a string using the time format specified in the Control Panel of our Windows 2000 operating system.
Furthermore, the Timer1_Timer( ) event takes place every time an interval elapses. The interval is determined by the Interval property. Once an interval has elapsed, the timer

generates its own Timer event. It accomplishes this by checking the system clock at frequent intervals.
The 500 interval set up simply means half a second. The Interval property is measured in milliseconds. Meaning, if we want to count in seconds, we have to multiply the number of seconds by 1,000. So 1 second is equivalent to 1,000 milliseconds (therefore, 500 is a half second, isn't it?).

Example 3:

Design and develop a simple timer-control application system that demonstrates how it works with a clock timer at the caption of the Form control. Follow the given figure below in designing and developing the application program:

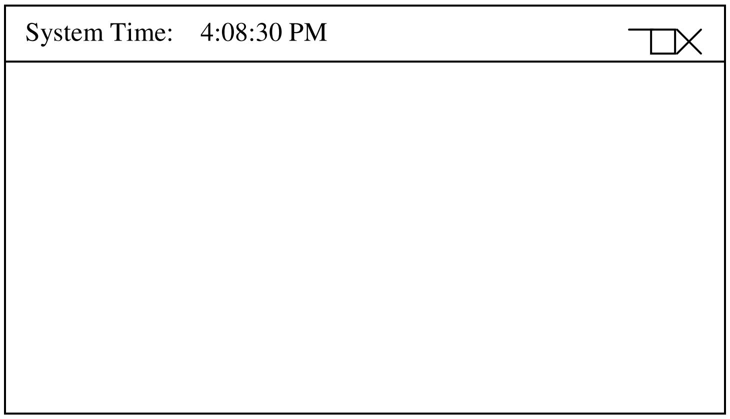


Figure 7.3 Using Timer control as the caption of the Form

Note:

The Timer control should be drawn at the center of the form.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **System Time:.**
3. Double-click a Timer control in the Toolbox to add it into the form. Position it at the center of the form.
4. Follow the property settings for each respective control below:

Control	Property	Setting

Form1	BorderStyle	Fixed Single
Timer1	Interval	500
Timer1	Enabled	True

5. Double-click now the form to display the Code editor. Embed the following code to its respective procedure:

```
Option Explicit
Private FormCaption As String
________________________________________________________
Private Sub Form_Load()
 FormCaption = Me.Caption
End Sub
________________________________________________________
Private Sub Timer1_Timer()
   Dim strMessage As String
   strMessage = FormCaption & " " & Time
   If strMessage <> Caption Then
     Caption = strMessage
   End If
End Sub
```

6. Now run the simple Timer application system by clicking the Run icon at the toolbar.

Explanation:

The Timer1_Timer( ) procedure simply displays the system time as the caption of the Form control by calling the Time function with the following code:

```
Private Sub Timer1_Timer( )
  Dim strMessage As String
  strMessage = FormCaption & " " & Time  ←—
  If strMessage <> Caption Then
    Caption = strMessage  ←—
  End If
End Sub
```

You will notice here in the above code that we simply concatenate the Time function to the caption of the form (separated by a space using the " " (double quote) symbol). Then we simply store them at the string variable called strMessage. After that, we evaluate if strMessage is not equal to the Caption. If it is, then we store the strMessage at the Form's caption. This is the reason why if there are changes in the form's caption which is generated by a half second (500) time interval, the value of the strMessage (which is the string: System Time plus the clock time) will be redisplayed through storing it to the Caption of the form. This is the technique to minimize the flicker caused by a constant update of the timer control.

The Timer1_Timer( ) event takes place every time an interval elapses. The interval is determined by the Interval property. Once an interval has elapsed, the timer generates its own Timer event. It accomplishes this by checking the system clock at frequent intervals. The 500 interval set up simply means half a second. The Interval property is measured in milliseconds. Meaning, if we want to count in seconds, we have to multiply the number of seconds by 1,000. So 1 second is equivalent to 1,000 milliseconds (therefore, 500 is a half second, isn't it?).

Using Scroll Bars

A scroll bar is typically used to increase or decrease a value such as when we want to change the color setting of our computer's monitor through the Control Panel's Display object or to increase or decrease the volume of our computer's digital audio speaker. The scroll bar acts as a sliding scale with a starting point and ending point, including the values in between.

Example 4:

Design and develop a simple Horizontal scroll-bar application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:

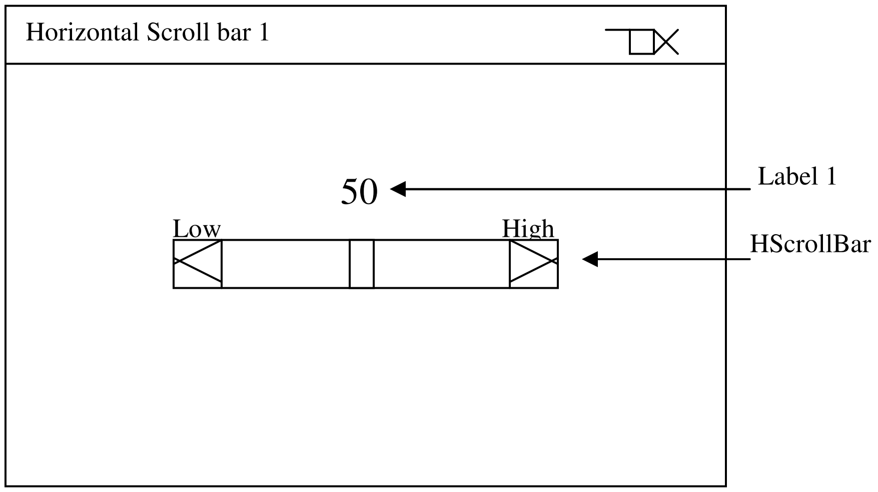


Figure 7.4 Using Horizontal scroll bar

Note:

The Caption of the Label 1 should be set to 50.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Horizontal Scroll bar 1.**
3. Double-click the Horizontal Scroll bar and Labels in the Toolbox to add them into the form. Position them properly at the form.

4. Set the **Max** property of Hscroll1 to 100, its **Value** property to 50 and its **LargeChange** property to 10.
5. Click the Font property of Label1 and set the Font Size property to 24 and its AutoSize to True.

6. Double-click the scroll bar to open the Code editor and add the following code:

```
Private Sub HScroll1_Change()
 Label1.Caption = Trim$(Str$(HScroll1.Value))
End Sub
```

7. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Since we set the Horizontal scroll bar Max (maximum) property to 100, the ending point of our scroll bar is 100 and its starting point is 0 (since 0 is the default Min property). We set the Value property to 50 so that the scroll bar's current position upon running the application will be at the center. Changing the font size of the Label1 makes its caption larger and more visible.
The Trim$ function simply trims the leading and trailing spaces from a string. When we convert a number to a string, the string result will contain leading spaces. So to keep our string formatting neat, we used the following code in our program:

Private Sub HScroll1_Change( )
Label1.Caption = Trim$(Str$(HScroll1.Value))⟵
End Sub

The Str$ function converts a numeric value to a string. We need to make this conversion because the Caption property of Label1 control expects a string value, for it to display correctly and appropriately. Since the value of the scroll bar is an integer value, so we need to convert it into a string value using the Str$ function.
You will notice that the value of the scroll bar will only change if the user stops dragging it. This means that the value can only be generated when the user stops dragging the

scroll bar. What if we want to update the value of the scroll bar while the user is dragging it? This will be our topic in the next example.

Example 5:

Design and develop a simple Horizontal scroll-bar application system that demonstrates how it works. The value of the scroll bar will be updated automatically while the user drags it. Follow the given figure below in designing and developing the application program:

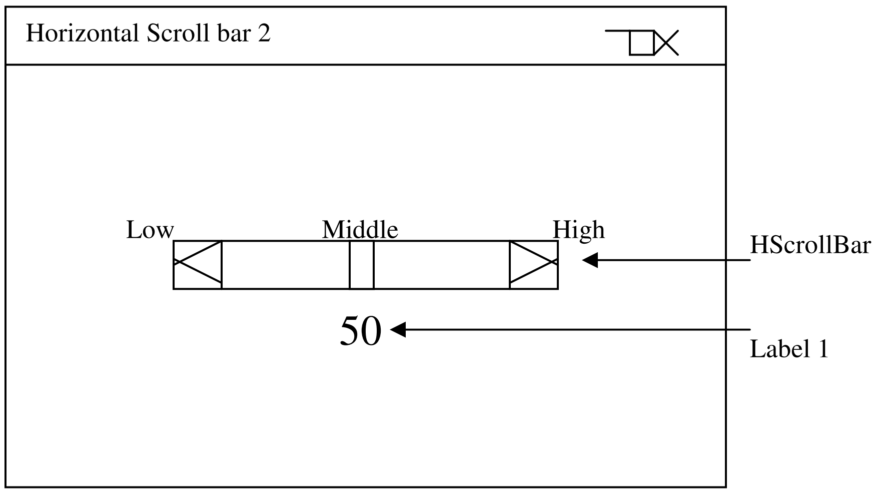


Figure 7.5 Using Horizontal scroll bar with automatic update.

Note:

The Caption of the Label 1 should be set to 50.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Horizontal Scroll bar 2.**
3. Double-click the Horizontal Scroll bar and Labels in the Toolbox to add them into the form. Position them properly at the form.

4. Set the **Max** property of Hscroll1 to 100, its **Value** property to 50 and its **LargeChange** property to 10.
5. Click the Font property of Label1 and set the Font Size property to 24 and its AutoSize to True.
6. Double-click the scroll bar to open the Code editor and add the following code:

```
Private Sub HScroll1_Scroll()
 Label1.Caption = Trim$(Str$(HScroll1.Value))
End Sub
```

7. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Do you now notice the difference between our previous example of Scroll bar to this present example? As the user drags the scroll box, the value of the scroll bar is automatically updated, isn't it?

Explanation:

Since we set the Horizontal scroll bar Max (maximum) property to 100, the ending point of our scroll bar is 100 and its starting point is 0 (since 0 is the default Min property). We set the Value property to 50 so that the scroll bar's current position upon running the application will be at the center. Changing the font size of the Label1 makes its caption larger and more visible.
The Trim$ function simply trims the leading and trailing spaces from a string. When we convert a number to a string, the string result will contain leading spaces. So to keep our string formatting neat, we use the following code in our program:

Private Sub HScroll1_Scroll( ) ⟵
Label1.Caption = Trim$(Str$(HScroll1.Value)) ⟵
End Sub

If we want to generate the Change( ) event as the user drags the scroll box, then we must call the Scroll( ) event. This is what we have done here in this example, because the Scroll( ) event is continually triggered as the user drags the scroll box.
The Str$ function converts a numeric value to a string. We need to make this conversion because the Caption property of Label1 control expects a string value, for it to display correctly and appropriately. Since the value of the scroll bar is an integer value, so we need to convert it into a string value using the Str$ function.

Example 6:

Design and develop a simple Horizontal scroll-bar application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:

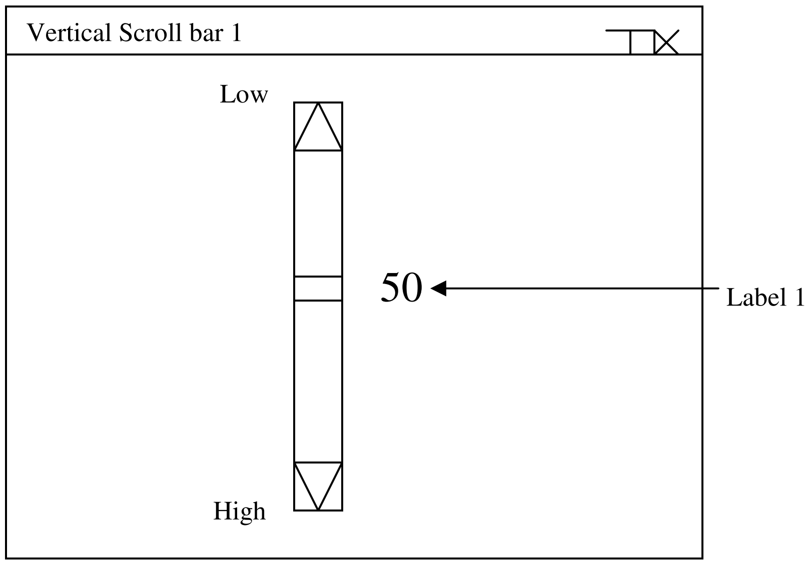


Figure 7.6 Using Horizontal scroll bar with automatic update.

Note:

The Caption of the Label 1 should be set to 50.

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
4. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Vertical Scroll bar 1.**
5. Double-click the Horizontal Scroll bar and Labels in the Toolbox to add them into the form. Position them properly at the form.

4. Set the **Max** property of Hscroll1 to 100, its **Value** property to 50 and its **LargeChange** property to 10.
5. Click the Font property of Label1 and set the Font Size property to 24 and its AutoSize to True.
6. Double-click the scroll bar to open the Code editor and add the following code:

```
Private Sub VScroll1_Scroll()
  Label1.Caption = Trim$(Str$(VScroll1.Value))
End Sub
```

7. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Since we set the Vertical scroll bar Max (maximum) property to 100, the ending point of our scroll bar is 100 and its starting point is 0 (since 0 is the default Min property).
We set the Value property to 50 so that the scroll bar's current position upon running the application will be at the center. Changing the font size of the Label1 makes its caption larger and more visible.
The Trim$ function simply trims the leading and trailing spaces from a string. When we convert a number to a string, the string result will contain leading spaces. So to keep our string formatting neat, we use the following code in our program:

```
Private Sub VScroll1_Scroll( ) ←
  Label1.Caption = Trim$(Str$(VScroll1.Value))←
End Sub
```

If we want to generate the Change( ) event as the user drags the scroll box, then we must call the Scroll( ) event. This is what we have done here in this example, because the Scroll( ) event is continually triggered as the user drags the scroll box.
The Str$ function converts a numeric value to a string. We need to make this conversion because the Caption property of Label1 control expects a string value, for it to display correctly and appropriately. Since the value of the scroll bar is an integer value, so we need to convert it into a string value using the Str$ function.

Using Image Control

The Image control is used primarily to display simple graphics in the following formats: .bmp .jpeg, gif, or an icon. It can even respond to the Click event and can be used as a substitute for command button or as an item in a toolbar. This image control can even be used to create simple graphical animation in your application.
Though Image control and Picture box control share common functionalities such as they both display graphics and support the same graphic formats; the former is a lightweight equivalent of the later. Like for example, the image control uses fewer system resources and repaints faster than a picture box. Plus you can stretch the graphics in the image control to fit the control's size, while in the picture box , this is not possible. However, the picture box has more capabilities compared to image control. We will learn that one later.

Note:
BMP stands for bitmaps, JPEG stands for Joint Photographic Experts Group, and GIF stands for Graphic Interchange Format. An icon is a special type of bitmap. Its maximum size is 32 pixels by 32 pixels and has a file name extension of .ico.

Example 7:

Design and develop a simple Image control application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:

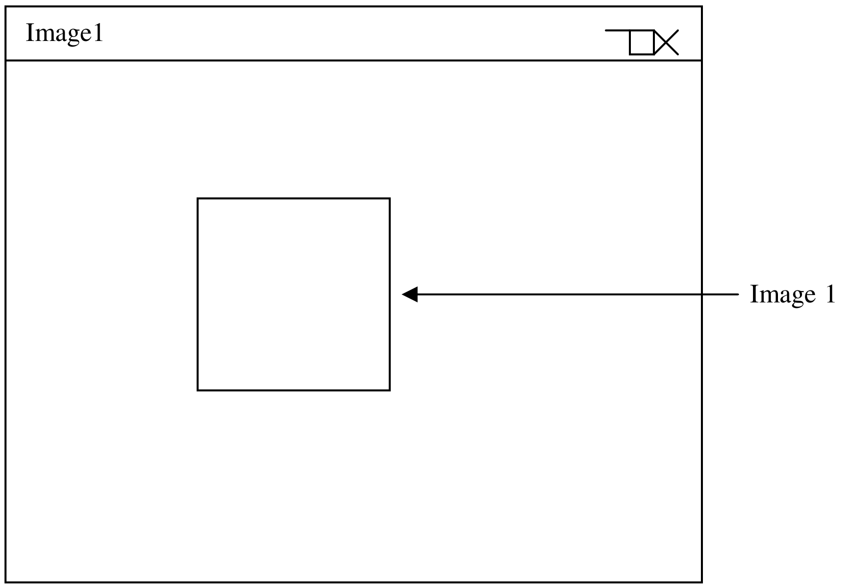


Figure 7. 7 Using Image control

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Image1.**
3. Double-click an Image control in the Toolbox to add it into the form. Position it at the center of the form.
4. Double-click the Image control in the form to open the Code editor and add the following code:

```
Private Sub Form_Load()
```

```
 Image1.Picture = LoadPicture("C:\WINNT\Coffee Bean.bmp")
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

We can easily find some samples of .bmp files at the Windows 2000 (or higher) operating system by just searching it using the Search command at the Start menu. Just make it sure that the exact location (its disk drive letter and folders are correct) so that no errors will result upon running your application system.
The syntax of loading the picture in our image control is so simple. Here is its code below:

```
Private Sub Form_Load( )
  Image1.Picture = LoadPicture("C:\WINNT\Coffee Bean.bmp") ←
End Sub
```

The Coffee Bean.bmp image can be found at the disk drive C:\ under the WINNT folder. Using the LoadPicture function, we can display the image at Image1 object and in its corresponding Picture property. In our next example, we will apply the Picture box control in our application system. You will notice that the Picture box control is similar to Image control in most of its application.

Using Picture Box Control

Like an image control, a picture box control is also used for displaying graphics with the following formats: .bmp, jpeg, gif and even an icon. However a picture box can contain other controls. In this capacity, we can place a command button within a picture box.
If we just want to show a picture or image, then an image control is usually a better choice compared to a picture box. It's because an image control takes up less memory and repaints the image or graphic faster than the picture box control. However, if we have plans to move the image or picture around the form, then a picture box control is a better choice because it produces smoother display. Moreover, we can design text and graphics methods in a picture box at run time. The graphics methods will enable us to draw lines, rectangles, and circles on a picture box. This makes the picture box control more powerful and flexible than image control. Are you convinced by now?

Example 8:

Design and develop a simple Picture box control application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:

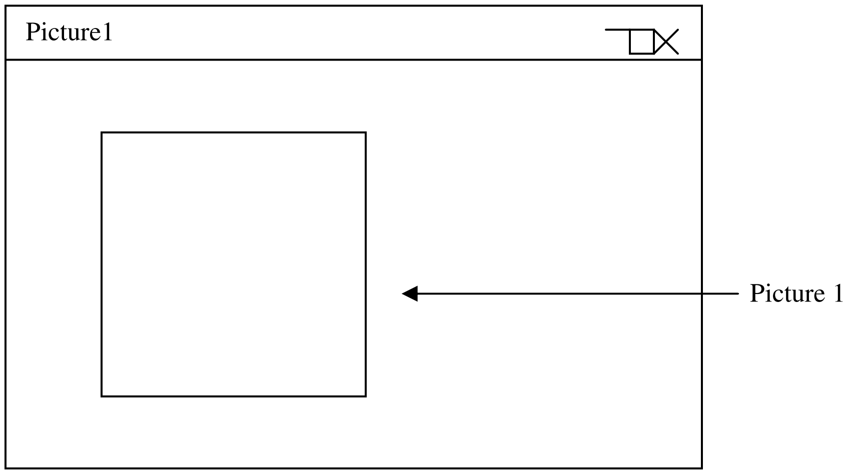


Figure 7. 7 Using the Picture box control

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Picture1.**
3. Double-click a Picture box control in the Toolbox to add it into the form. Set its AutoSize property to True. Position it at the center of the form.
4. Double-click the Picture box control in the form to open the Code editor and add the following code:

```
Private Sub Form_Load()
 Picture1.Picture = LoadPicture("Chateau.jpg")
End Sub
```

5. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

We can easily find some samples of .jpg files at the Windows 2000 (or higher) operating system by just searching it using the Search command at the Start menu. Just make it sure that the exact location (its disk drive letter and folders are correct) so that no errors will result upon running your application system.
The syntax of loading the picture in our Picture box control is so simple. Here is its code below:

Private Sub Form_Load( )
 Picture1.Picture = LoadPicture("Chateau.jpg") $\longleftarrow$
End Sub

The Chateau.jpg image can be found in the Windows operating system. Just search it using the Search command at the Start menu, then copy it to your Visual Basic folder so that you have no problem upon running your application. This will make it easy for the Visual Basic compiler to search for Chateau.jpg file because it is located right in its own directory or folder. Using the LoadPicture function, we can display the image at Picture1 object and in its corresponding Picture property.
To load the Chateau.jpg picture from its original WINNT folder, we have to specify its complete directory. Here it is below:

```
C:\WINNT\Web\Wallpaper\Chateau.jpg
```

And to put it in our code, we have the following complete syntax:

```
Picture1.Picture = LoadPicture("C:\WINNT\Web\Wallpaper\Chateau.jpg")
```

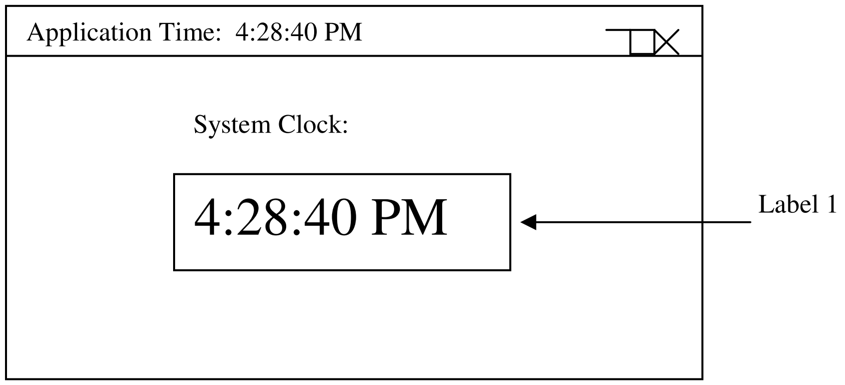
Setting the property of AutoSize to True (in our Picture object) causes the picture box to resize to match the size of the image or graphic. This is important because if the image is too big for the picture box that we have drawn on the form, the image is clipped. We can empty the picture box by using the LoadPicture function with no parameter on it. Here is its syntax:

```
Picture1.Picture = LoadPicture( )
```

This is the end of our discussion about the other basic controls of Visual Basic. I hope that you have learned a lot from these examples.

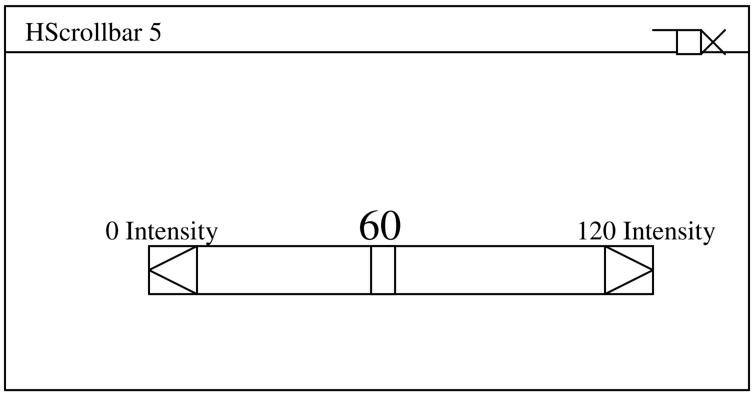
LAB ACTIVITY
TEST 6

1. Design and develop a simple timer-control application system that demonstrates how it works with a bigger font size of the clock timer and at the same time a second clock timer as the caption of the form. Follow the given figure below in designing and developing the application program:



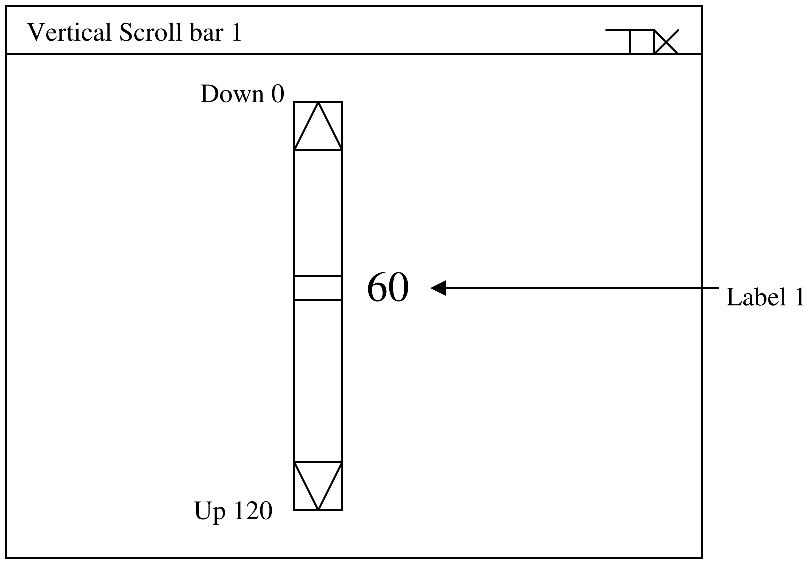
Note:
Set the Font Size property of the Label 1 to 20.

2. Design and develop a simple Horizontal scroll-bar application system that demonstrates how it works. The value of the scroll bar will be updated automatically while the user drags it. Follow the given figure below in designing and developing the application program:



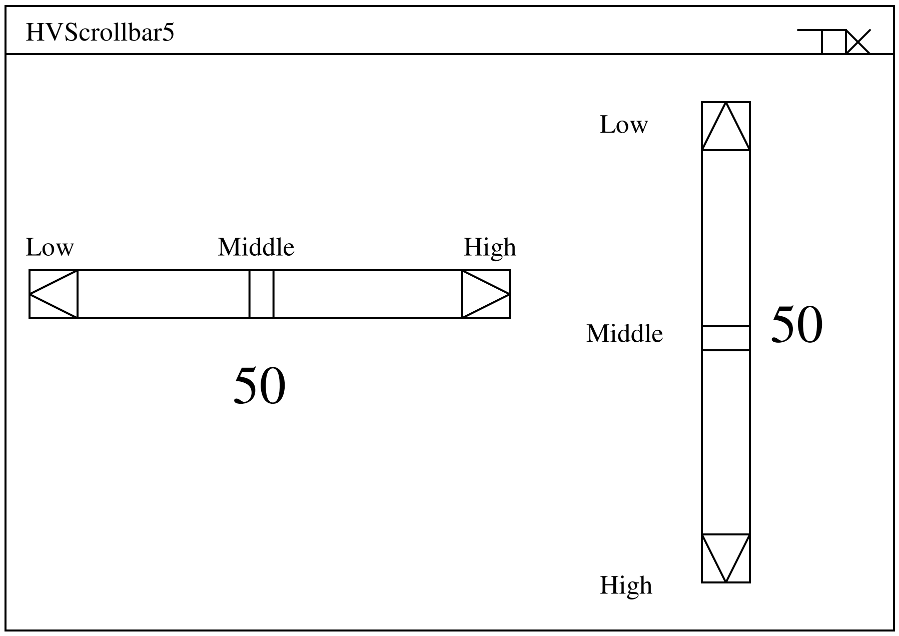
Note:
Set the Max property of the Horizontal Scroll bar to 120.

3. Design and develop a simple Horizontal scroll-bar application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:



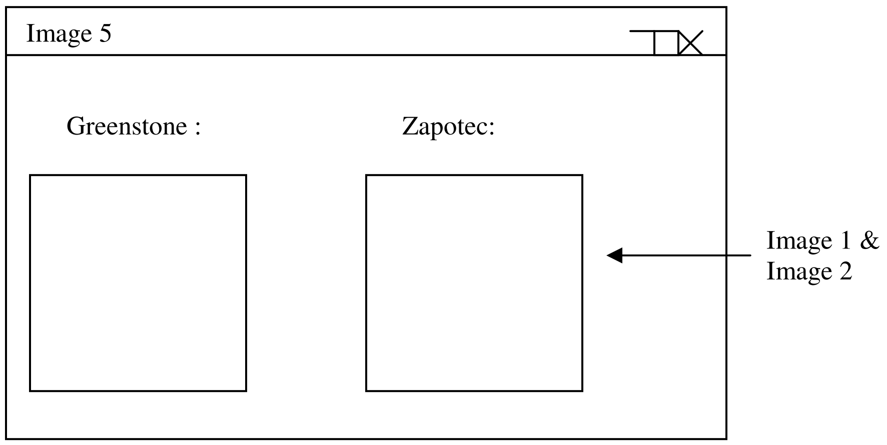
Note:
Set the Max property of the Vertical Scroll bar to 120.

4. Design and develop the Horizontal and Vertical scroll bars application system that demonstrate how they work. Follow the given figure below in designing and developing the application program:



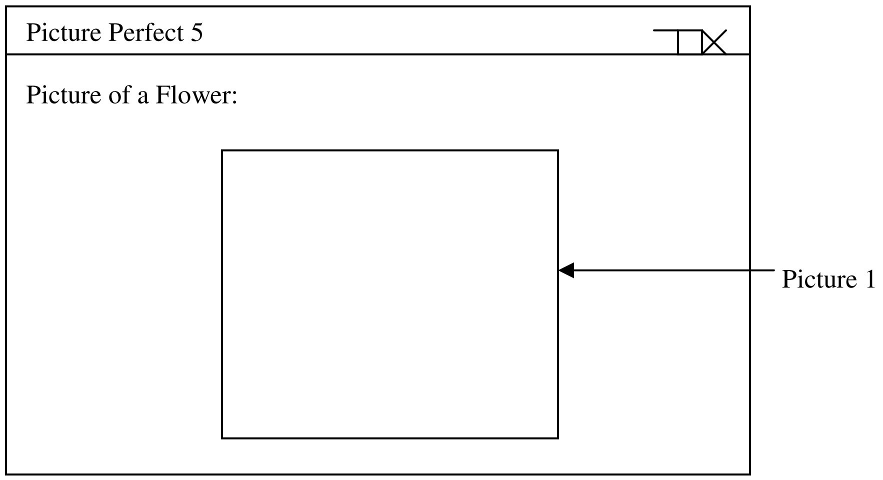
Note:
Set the Max property of the Horizontal and Vertical Scroll bars to 100.

5. Design and develop a simple Image control application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:



Note:
The images should be loaded from the following directory:
"C:\WINNT\Greenstone.bmp"
"C:\WINNT\Zapotec.bmp"

6. Design and develop a simple Picture box control application system that demonstrates how it works. Follow the given figure below in designing and developing the application program:



Note:
The picture should be loaded from the following directory:
"C:\WINNT\Web\Wallpaper\Gold Petals.jpg"

Chapter 8

Accessing Databases

A database is a file that contains tables, indexes, and queries. A table is a collection of data that can be represented in rows and columns. In database application, a row is called a **record**, and a column is called a **field**. An index file has a linked list pointer to records within a table. Through an index file, we can rapidly search a particular data in a table. A query is a structured query language (SQL) statements that we write to extract a particular data from a table or tables. A query is also used to insert, update, add, or change the data in the table. It's like asking a question about a particular data or group of data, and the database system will simply answer back.
The popular database system software that we commonly heard are ORACLE, Microsoft's SQL Server, IBM's DB2, Microsoft FoxPro and Microsoft Access.

Using Data Control

There are many programmers in Visual Basic who keep their data in Microsoft SQL Server or Microsoft Access database system software. The Visual Basic programming language has a capability to access data in a database files of MS Access, MS SQL Server and even in ORACLE Database Server. We can access the database through the use of Data control of Visual Basic. This Data control implements data access by using the Microsoft Jet Database engine, the same database engine that powers the Microsoft Access. This technology gives us seamless access to many standard database formats and allows us to create data-aware application system without writing any code or just a minimal code required to accomplish the task.
Specifically, we can use the Data control to design and develop an application system that display, edit, add, or update the data from many existing popular databases such as MS Access, MS FoxPro, Paradox, and Dbase. Moreover, we can use the Data control to access and manipulate the remote Open Database Connectivity (ODBC) databases such as Microsoft SQL Server and ORACLE.

Example 1:

Design and develop an application system using Data Control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:

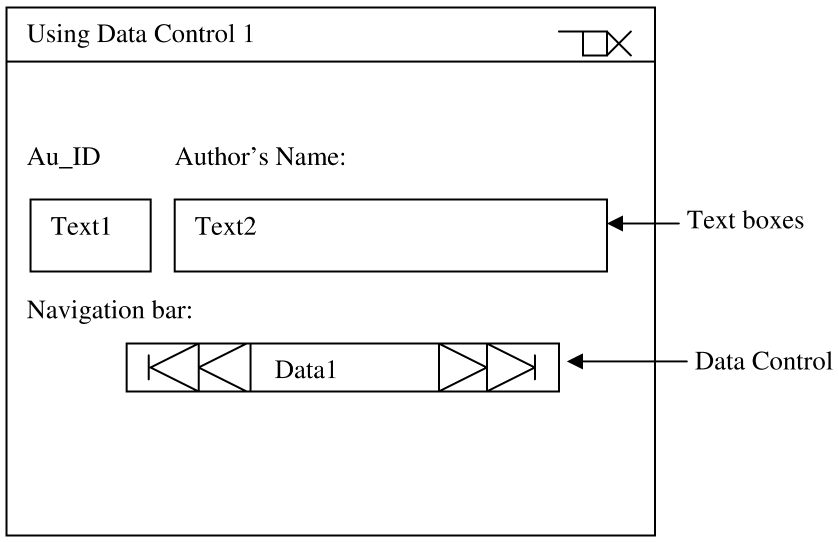


Figure 8.1 Using Data Control to access a database

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using Data Control 1.**
3. Double-click two text boxes and a Data control in the Toolbox to add them into the form.
4. Widen the Data control so that we can see its caption.
5. Set the properties for the Data control as follows:

DatabaseName: **C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB**
RecordSource: **Authors**

6. Set the properties for text box 1:

DataSource: **Data1**
DataField: **Au_ID**

7. Set the properties for text box 2:

DataSource: **Data1**
DataField: **Author**

8. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Now scroll through the records with the navigation buttons on the data control. Take note that any modifications or changes you make in the text boxes are written back to the database. We can easily add a new record to our database. And to do it is - we will simply set the **EOFAction** property of the data control to **2 - Add New**.
The RecordSource property corresponds to the table in the database we want to access. In this example it is the Authors table.
We have to remember always that the Data control does not display the data. To do that, we must link another type of control to the Data control. In our example, we used two text boxes. For each text box, we set the DataSource property to the name of the data control. In this example, it is the Data1. Then next, we set the DataField property equal to the name of a field in the recordset returned by the data control's RecordSource property. In this way, we can display the data to the text boxes and navigate them through the Data control

Example 2:

Design and develop an application system using Data Control that demonstrates how it is used to access a database. This time, add two command buttons for the Delete and Add a record task. Follow the given figure below in designing and developing the application program:

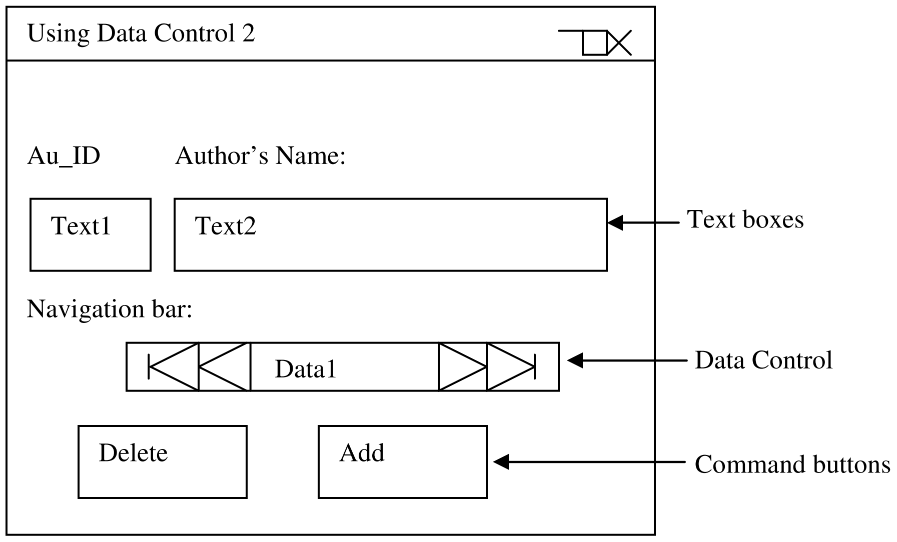


Figure 8.2 Using Data Control with Delete and Add buttons

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using Data Control 2.**
3. Double-click two text boxes, a Data control and two command buttons in the Toolbox to add them into the form. Set the Caption of Command button 1 to **Delete** and the Caption of the Command button 2 to **Add**.
4. Widen the Data control so that we can see its caption.
5. Set the properties for the Data control as follows:

DatabaseName: **C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB**
RecordSource: **Authors**

6. Set the properties for text box 1:

DataSource: **Data1**
DataField: **Au_ID**

7. Set the properties for text box 2:

DataSource: **Data1**
DataField: **Author**

8. Double-click the Command button 1 (Delete button) to open the Code editor and add the following code:

```
Private Sub Command1_Click()
 With Data1.Recordset
   .Delete
   .MovePrevious
 End With
End Sub
```

9. Double-click the Command button 2 (Add button) to open the Code editor and add the following code:

```
Private Sub Command2_Click()
 With Data1.Recordset
  .MoveLast
  .AddNew
 End With
 End Sub
```

10. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

Now scroll through the records with the navigation buttons on the data control. Take note that any modifications or changes you make in the text boxes are written back to the database.
When we want to delete the currently displayed record, we have the following code:

```
Private Sub Command1_Click( )
 With Data1.Recordset
  .Delete  ←
  .MovePrevious  ←
 End With
End Sub
```

After deleting the record, we will position the record pointer to the previous record, so that if the deleted record happens to be the last record, the record before the last record is the one that will be displayed.
Take note that before we can add a new record in our table, we need to position the record pointer to the end of file (EOF), so that we can input the data in the blank text boxes. This is the reason why we have the following code:

```
Private Sub Command2_Click( )
With Data1.Recordset
   .MoveLast  ←
   .AddNew
End With
End Sub
```

Remember also that we can easily add a record to our table without utilizing the code above by simply setting the *EOFAction* property of the Data control to **2 - Add New**.
The RecordSource property corresponds to the table in the database we want to access. In this example it is the Authors table.
Actually the Data control does not display the data. To do that, we must link another type of control to the Data control. In our example, we used two text boxes. For each text box, we set the DataSource property to the name of the data control. In this example, it is the Data1. Then next, we set the DataField property equal to the name of a field in the recordset returned by the data control's RecordSource property. In this way, we can display the data to the text boxes and navigate them through the Data control

Using the ActiveX Data Object (ADO) Control

The ADO data control is the new Microsoft's seamless universal data-access framework. It combines the power and functionality of the old and obsolete Remote Data Object (RDO) and Database Access Object (DAO) data controls. Using the ADO data control, we can quickly create connections between data-bound controls and data providers. The data-bound controls are any controls that feature a DataSource property, while the data providers can be any source written to the OLEDB specification. Client-server databases such as SQL Server and ORACLE has an OLEDB provider or ODBC provider.
Several controls found in the Toolbox can be data-bound, including ComboBox, ListBox, TextBox, CheckBox, Image, Label, and PictureBox. Moreover, the Visual Basic system includes other data-bound Active-X controls such as the DataList, DataCombo,DataGrid, and Chart controls.

Example 3:

Design and develop an application system using ADO data control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:

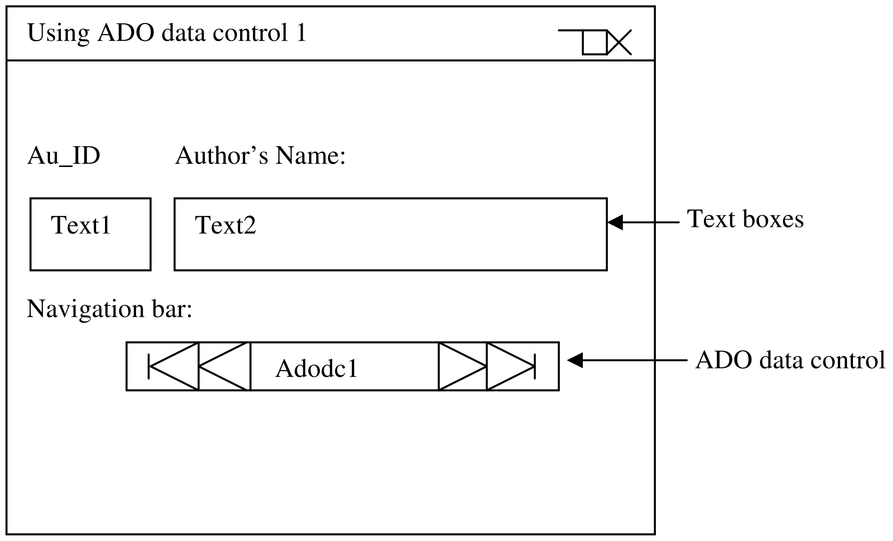


Figure 8.3 Using ADO data control to access a database

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using ADO data control 1.**
3. Double-click two text boxes and a Data control in the Toolbox to add them into the form.

Before we can use the ADO data control, we must add it to the Toolbox.

4. Right-click the Toolbox and select the Components from the pop-up menu.
5. In the Components dialog box, select Microsoft ADO Data Control 6.0 (OLEDB), and click the OK button. This process will add the ADO data control to the Toolbox. Have you noticed it?
6. Add the ADO data control to the form and widen it, so that we can see its caption.
7. Instead of using a Database property, the ADO connects to databases through a Connection object. Click the ConnectionString property to bring up the property page for the control.

Note:
The Use Data Link File and Use ODBC Data Source Name options and fields are used when we want to connect to an external data source, such as ORACLE database server or SQL Server database. In this example, we just want to connect to a Microsoft Access table.

8. Click the Build button to start building the connection string.
9. From the Data Link Properties dialog box, select and click the Microsoft Jet 3.51 OLE DB provider to use this driver to connect to the Access database. Then click the Next button.
10. Click the small button to the right (with three small dots) of the Select or Enter a Database Name field. Select the BIBLIO.MDB database (from the \Program Files\Microsoft Visual Studio\VB98 directory). Then click the Open button to select the database. You will be brought to the Connection tab.
11. Now click the OK button to create the connection string.
12. Click the OK button to close the property page. The ConnectionString is now set.
13. Double-click the RecordSource property of the ADO data control to specify the table we want to look at.
14. In the Command Type field, select **2-adCmdTable**.
15. When the Table or Stored Procedure Name field appears, select Authors.
16. Click the OK button to close the dialog box.
17. Click now the Text2 control to make it the active control. Set its DataSource property to **Adodc1**. Set its DataField property to **Author**.

18. Click the Text1 control and set its DataSource property to **Adodc1**. Then set its DataField property to **Au_ID**.
19. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

You will notice that the ADO data control looks and acts like the Data Control that we have used in our previous example. However, the ADO technology is more powerful and flexible, which means it can scale easily from single-user databases such as Microsoft Access to multi-user client-server systems such as Microsoft SQL Server or ORACLE database server.
Try now to scroll through the records with the navigation buttons on the data control. Take note that any modifications or changes we make in the text boxes are written back to the database. We can easily add a new record to our database. And to do it is - we will simply set the **EOFAction** property of the ADO data control to **2 - Add New**. The RecordSource property corresponds to the table in the database we want to access. In this example it is the Authors table.
We have to remember always that the ADO data control does not display the data. To do that, we must link another type of control to the ADO data control. In our example, we use two text boxes. For each text box, we set the DataSource property to the name of the data control. In this example, it is the Adodc1. Then next, we set the DataField property equal to the name of a field in the recordset returned by the ADO data control's RecordSource property. In this way, we can display the data to the text boxes and navigate them through the ADO data control.

Using the DataGrid Control

The DataGrid control displays data like Microsoft Excel spreadsheet software do.
It displays a series of rows and columns representing records and fields from a Recordset object. Using the DataGrid, we can easily create an application system that can allow the users to read and write to the tables. Meaning, they can add column header or adjust its width, or even delete or rearrange the grid's column. We can use the DataGrid to view and edit data whether remotely or in a local database.

Example 4:

Design and develop an application system using DataGrid control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:

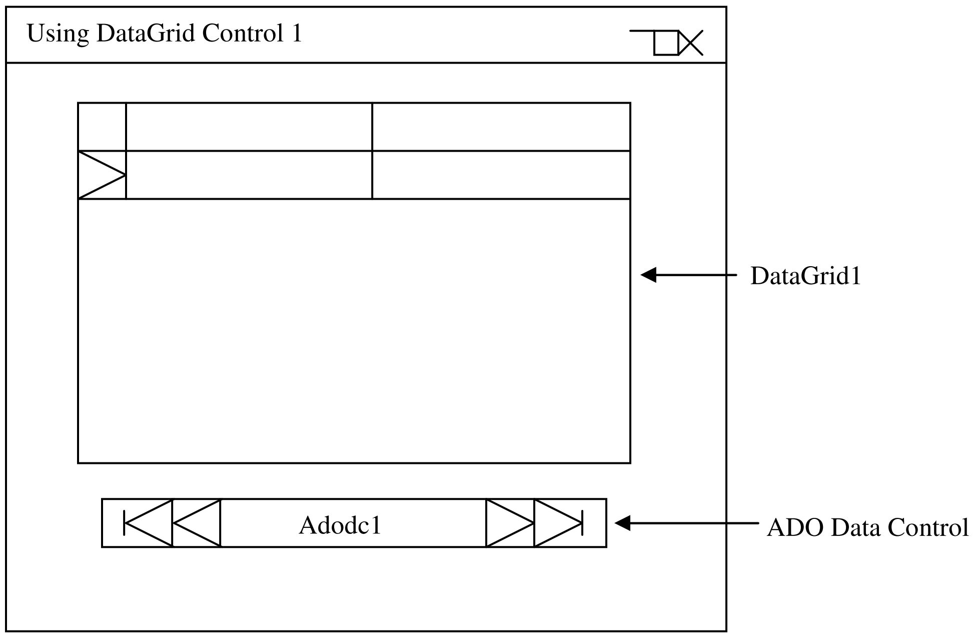


Figure 8.4 Using DataGrid control

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using DataGrid control 1.**

Before we can use the DataGrid and ADO data controls, we must add them to the Toolbox.

3. Right-click the Toolbox and select the Components from the pop-up menu.
4. In the Components dialog box, select the Microsoft ADO Data Control 6.0 (OLEDB) and Microsoft DataGrid Control 6.0 (OLEDB). Then click the OK button. This process will add the ADO and DataGrid data controls to the Toolbox. Have you noticed them?
5. Add the DataGrid and ADO data controls to the form.

6. Instead of using a Database property, the ADO connects to databases through a Connection object. Click the ConnectionString property of the ADO data control to bring up the property page for the control.

Note:
The Use Data Link File and Use ODBC Data Source Name options and fields are used when we want to connect to an external data source, such as ORACLE database server or SQL Server database. In this example, we just want to connect to a Microsoft Access table.

7.Click the Build button to start building the connection string.
8. From the Data Link Properties dialog box, select and click the Microsoft Jet 3.51 OLE DB provider to use this driver to connect to the Access database. Then click the Next button.
9.Click the small button to the right (with three small dots) of the Select or Enter a Database Name field. Select the BIBLIO.MDB database (from the \Program Files\Microsoft Visual Studio\VB98 directory). Then click the Open button to select the database. You will be brought to the Connection tab.
10. Now click the OK button to create the connection string.
11.Click the OK button to close the property page. The ConnectionString is now set.
12.Click the OK button to close the dialog box.
13. Set the ADO data control's RecordSource by typing a SQL statement to populate the DataGrid control. In this example, type in:

```
select *  from Authors where Au_ID <=100
```

14.Set the DataGrid Control's DataSource property to Adodc1.
15.Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

The RecordSource property corresponds to the table in the database we want to access. In this example it is the Authors table. Here we issue the SQL statement command:

*select * from Authors where Au_ID <=100*

so that only the list of authors whose author's ID is less than or equal to 100 will be displayed. Now since the author's id starts from 1 up to 16,099; only the first 100 authors will be displayed at the DataGrid.
We have to remember always that the ADO data control does not display the data. To do that, we must link another type of control to the ADO data control. In our example, we used a DataGrid. So we set the DataSource property of the DataGrid to Adodc1. In this way, we can display the data into the DataGrid control and navigate them through ADO data control.

Example 5:

Design and develop an application system using ADO hard code. Follow the given figure below in designing and developing the application program:

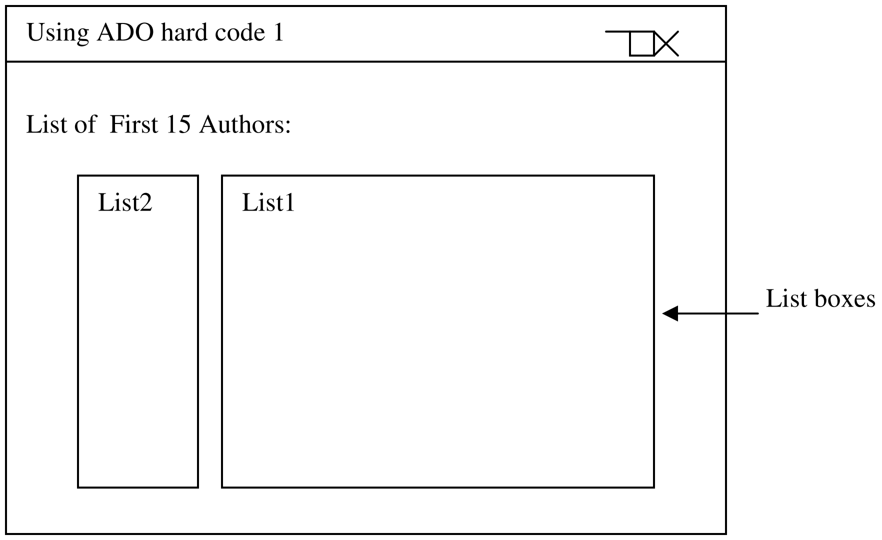


Figure 8.5 Using ADO hard code

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using ADO hard code 1.**
3. Double-click two List boxes and a Label in the Toolbox to add them into the form.
4. Select the Project menu of the Visual Basic and click the References menu item to open the References dialog box.
5. Reference the ADO type library by checking the box next to Microsoft ActiveX Data Objects 2.0 library.
6. Now click the OK button to reference the library and close the References dialog box. You can now access the ADO Connection and Recordset objects in our code.
7. Double-click the Form and add the following code at the **General** object and **Declarations** procedure section:

```
Private cn As ADODB.Connection
Private rs As ADODB.Recordset
```

8. Then, add the following code to the Form_Load( ) event of the form:

```
Private Sub Form_Load()
Dim cmd As String
Dim sql As String
Dim cn As ADODB.Connection
Dim rs As ADODB.Recordset

'Create a connection string
cmd = "Provider=microsoft.jet.OLEDB.3.51;Data
Source=C:\Program Files\Microsoft Visual
Studio\VB98\Biblio.mdb"

'Establish a connection with the database
Set cn = New ADODB.Connection
With cn
   .ConnectionString = cmd
   .Open
End With

'Create a query
sql = "select * from Authors where Au_ID <=15"

 'Open the Recordset
Set rs = New ADODB.Recordset
With rs
 .Open sql, cn, adOpenForwardOnly, adLockReadOnly
 Do While Not rs.EOF
   'Add the author to the list
    List1.AddItem rs("Author")
    List2.AddItem rs("Au_ID")
   'Move to the next record
   rs.MoveNext
 Loop

 'Close the Recordset
 .Close
 End With
 Set rs = Nothing

 'Close the connection
 cn.Close

 'Destroy the connection object
 Set cn = Nothing
```

```
End Sub
```

9. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

Explanation:

We could notice here in our code the two variables: cmd and sql. The variable cmd is used to hold the ADO connection string which in turn used to connect to the database. While the sql variable is used to store the SQL query. Let us analyze the following code:

```
   Private Sub Form_Load( )
   Dim cmd As String ←
   Dim sql As String ←
   Dim cn As ADODB.Connection
   Dim rs As ADODB.Recordset

   'Create a connection string
→ cmd = "Provider=microsoft.jet.OLEDB.3.51;Data Source=C:\Program Files\Microsoft
   Visual Studio\VB98\Biblio.mdb"

   'Establish a connection with the database
   Set cn = New ADODB.Connection ←
   With cn
     .ConnectionString = cmd
     .Open
   End With

   'Create a query
→ sql = "select * from Authors where Au_ID <=15"
   .
   .
   .
```

The Provider= construct creates a connection to BIBLIO.MDB database of Microsoft Access using the Microsoft's new OLEDB provider.
To establish the actual connection to the database, we accomplish this by instantiating an ADO.Connection object, and then setting its Connection-String property to cmd. Then we issue the .Open method for the object. This .Open method took the data stored in the ConnectionString property and used it to connect to the database successfully.
After a successful connection to a database, we can now open an ADO.Recordset object to retrieve the records from the table. But before we can open a recordset, we have to issue a query using the complete SQL statement to instruct it what records to retrieve. That is why we have this SQL statement:

```
'Create a query
sql = "select * from Authors where Au_ID <=15"
```

And to open the Record set, we have the following code:

```
'Open the Recordset
Set rs = New ADODB.Recordset ←
With rs
 .Open sql, cn, adOpenForwardOnly, adLockReadOnly ←
 Do While Not rs.EOF ←
   'Add the author to the list
    List1.AddItem rs("Author")
    List2.AddItem rs("Au_ID")
   'Move to the next record
   rs.MoveNext
 Loop ←

 'Close the Recordset
 .Close

 End With
 Set rs = Nothing

 'Close the connection
 cn.Close

 'Destroy the connection object
 Set cn = Nothing

End Sub
```

In the With block statement, we instruct the ADO to open the recordset using the command:

```
With rs
 .Open sql, cn, adOpenForwardOnly, adLockReadOnly
```

The Open sql passes the SQL query to the recordset while the cn instructs the recordset to use the connection object cn to get to the database. The adOpenForwardOnly and adLockReadOnly are the type of database cursor which we use for the recordset. The adOpenForwardOnly cursor type is used to move forward through the recordset. This option is fast and useful when we want to rapidly populate a list box (or combo box). We use the AdLockReadOnly as the lock mode when we do not want to allow any additions, updates, or deletions from the recordset. The Do While loop instructs the application program to loop through every record in the table until the EOF (end of file) is reached.

As the cursor moves to each record, we add the data in the Au_ID and Author fields to the List box 2 and List box 1 respectively using the following code:

```
List1.AddItem rs("Author")
List2.AddItem rs("Au_ID")
```

After adding the data to the List boxes, we have to instruct the cursor to move to the next record using the .MoveNext method, otherwise the loop will get stuck because the EOF will not be reached.
We have to close any objects we used here in this program and destroy them after using them. This is the proper way of ending or terminating our application program. We accomplish these tasks by calling the .Close method for the Recordset and the Connection objects. Finally, we set these objects equal to Nothing to prevent memory leakage in our application program.

Creating Databases Using Visual Data Manager

The Database provides us the way to organize the data in an orderly fashion. We can store or retrieve the data using database files. We can view, update, insert, delete, or append the data in the database. The database file contains tables, indexes, and queries that make record searching easier and faster.
The Visual Data Manager is a Visual Basic tool for creating databases. Let us start creating a database now.
Here are the steps:

1. Select Add-Ins menu of the Visual Basic, then click the Visual Data Manager menu item.
2. Now select the File menu of the VisData (Visual Data Manager) and select New… menu item.
3. Now select the Microsoft Access Version 7.0 MDB to see the Select Microsoft Access Database to Create dialog box.
4. Type in a name of the database we want to create and name it: SALES. (There is no need to supply the .MDB filename extension, because the Microsoft Access does this automatically).
5. Now, right-click the Database window and select New Table to begin a new table.
6. Enter **STOCKS** at the field of the Table Name. Then click the Add Field button and type **ItemNumber** as the field name. Then input **3** for its size. Now click the OK button to complete the process.
7. Now enter the second field name. Name it: **Description** and input **15** for its size. Click the OK button to complete the process.
8. This time name the third field name as **Price** and its Type as Currency. Then click the OK button. To close this Field dialog box, just click now the Close button.
9. Now click the Build the Table button in the Table Structure dialog box to have the Data Manager add the table structure to the database.

10. Now to enter the data in our table, we simply open the SALES database. Select now the Add-Ins menu of the Visual Basic and click the Visual Data Manager.
11. Click the File menu of the VisData (Visual Data Manager) and select the Open Database… menu item.
12. Now select the Microsoft Access menu item. Then select the SALES database file and click the Open button.
13. Now double-click the STOCKS table.
14. Now you can enter the data for ItemNumber, the Description, and the Price. Enter the following data:

ItemNumber	**Description**	**Price**
01	Hamburger	20
02	French Fries	25
03	Spaghetti	30
04	Coke	15
05	Sprite	15

Just click the Add button to add the above data. After entering each data, just click the Update button.
You will notice that there is a navigation bar below. Just click the right and left arrow keys to view the data you entered. This navigation bar also displays how many data you stored to the table and the current record you are in. The 1/5 means you are currently at record number 1 and there are 5 records in the table.

In our previous examples, we simply manipulate the built-in BIBLIO.MDB database file of Visual Basic. This time we have already created our own database which we named as SALES.MDB. Let us make an application system that will access these SALES database and STOCKS table.

Note:
You can create a database easily using database management system software such as Microsoft Access, Microsoft SQL Server and ORACLE. What we have done here is to use the Visual Data Manager because there are cases that incompatible version arises. For example, we cannot access the database created by Microsoft Access 2000 in Visual Basic 6. Maybe there are ways to convert it to make it compatible, but as of the moment, I do not know how to do it. The best way is to create the database in Visual Data Manager to make it sure that it will work in our program.

Example 6:

Design and develop an application system using Data Control that demonstrates how it is used to access a database. The database should be created using the Visual Data Manager. Follow the given figure below in designing and developing the application program:

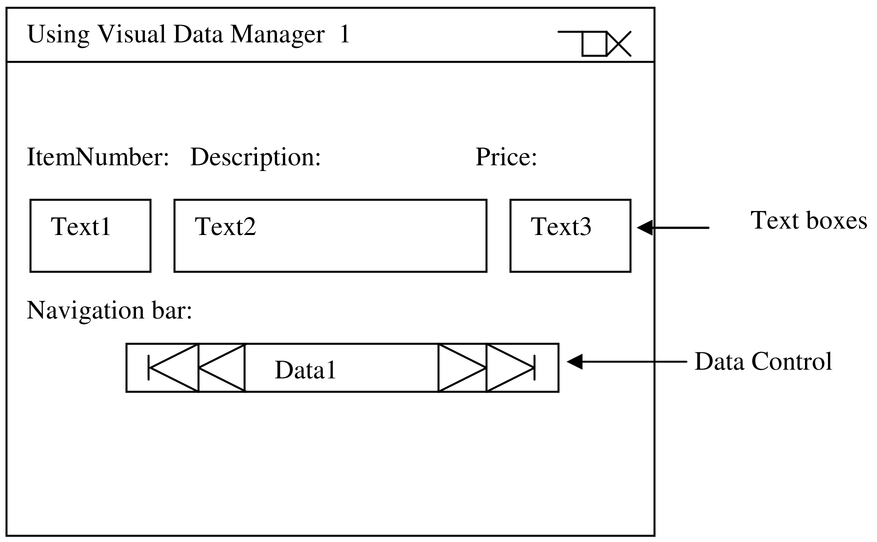


Figure 8.6 Using Visual Data Manager to create a database

Solution:

1. Select and click the Visual Basic under the Microsoft Visual Studio at the Start of the Taskbar and Programs submenu of the operating system.
2. Start a new project and select the Standard EXE on the given displayed items, then click the Open command button. Set the caption of the Form to **Using Visual Data Manager 1.**
3. Double-click three text boxes and a Data control in the Toolbox to add them into the form. Add also the specific labels for each control in the form.
4. Widen the Data control so that we can see its caption.
5. Set the properties for the Data control as follows:

DatabaseName: **C:\Program Files\Microsoft Visual Studio\VB98\SALES.MDB**
RecordSource: **STOCKS**

6. Set the properties for text box 1:

DataSource: **Data1**
DataField: **ItemNumber**

7. Set the properties for text box 2:

DataSource: **Data1**
DataField: **Description**

8. Set the properties for text box 3:

DataSource: **Data1**
DataField: **Price**

9. Run the application system by selecting the Run menu at the menu bar and click the Start item (or click the Run icon at the tool bar).

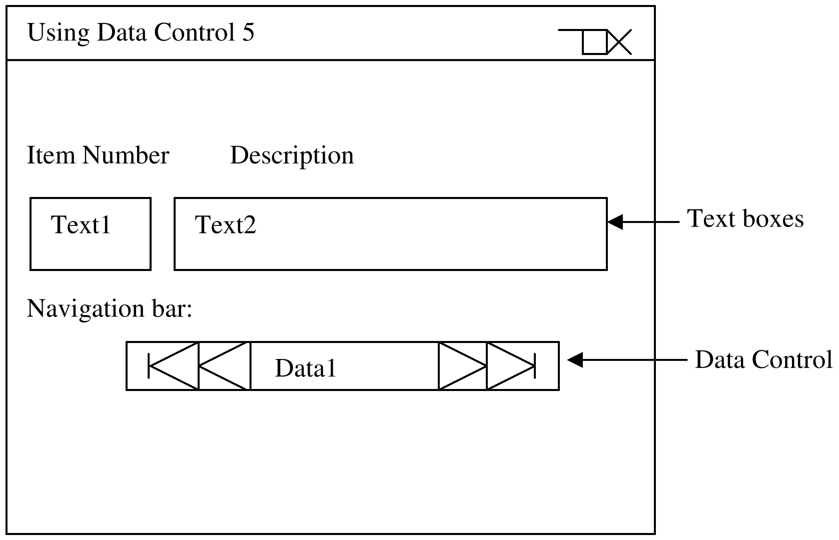
Did it run as expected? I hope so, buddy. Now you know how to create a database using the Visual Data Manager. And you know how to manipulate it using Data Control. In this particular example, we have all the control on what our database looks like and what data we want to put on it. This is better because in the real-world application system, we are the one who will create the needed databases and tables that mimics the real system requirements of the company.
Congratulations for being a Visual Basic programmer and developer now. God bless.

Explanation:

Now scroll through the records with the navigation buttons on the data control. Take note that any modifications or changes you make in the text boxes are written back to the database. We can easily add a new record to our database. And to do it is - we will simply set the **EOFAction** property of the data control to **2 - Add New**.
The RecordSource property corresponds to the table in the database we want to access. In this example it is the Authors table.
We have to remember always that the Data control does not display the data. To do that, we must link another type of control to the Data control. In our example, we used two text boxes. For each text box, we set the DataSource property to the name of the data control. In this example, it is the Data1. Then next, we set the DataField property equal to the name of a field in the recordset returned by the data control's RecordSource property. In this way, we can display the data to the text boxes and navigate them through the Data control.

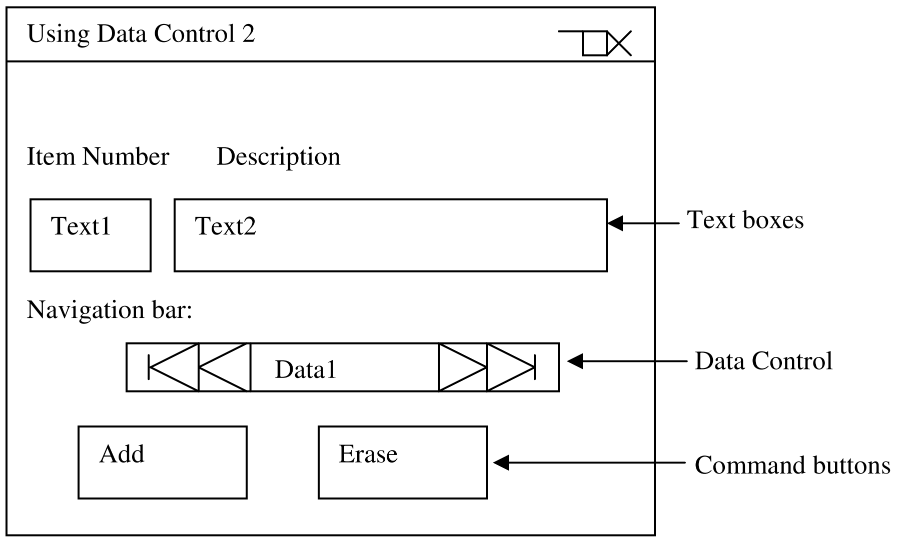
LAB ACTIVITY
TEST 7

1. Design and develop an application system using Data Control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:



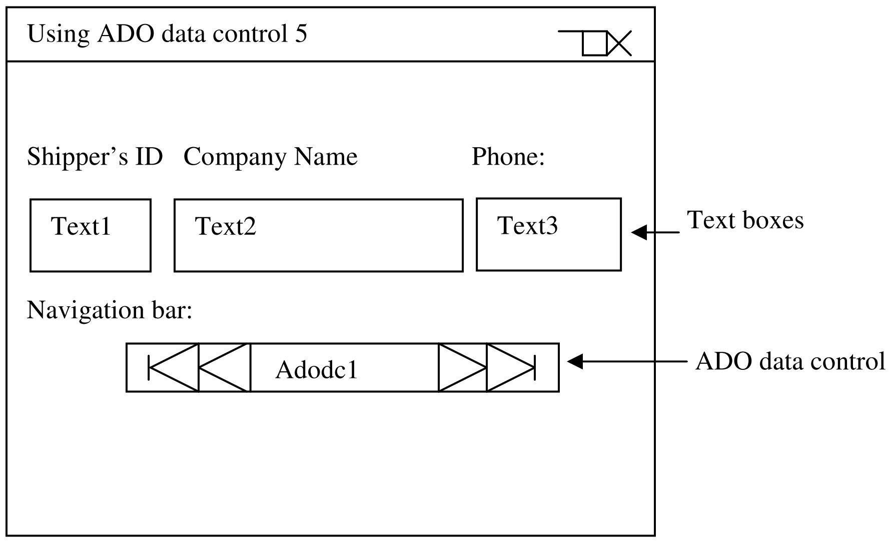
Note:
Use the SALES database and its STOCKS table that we have created in Example 6.

2. Design and develop an application system using Data Control that demonstrates how it is used to access a database. This time, add two command buttons for And and Erase a record task. Follow the given figure below in designing and developing the application program:



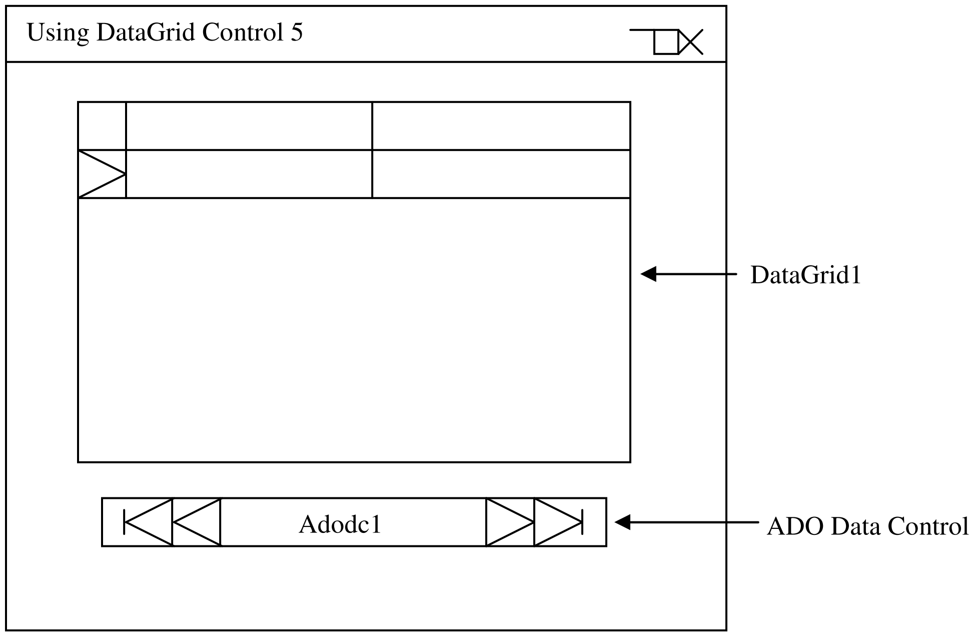
Note:
Use the SALES database and its STOCKS table that we have created in Example 6.

3. Design and develop an application system using ADO data control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:



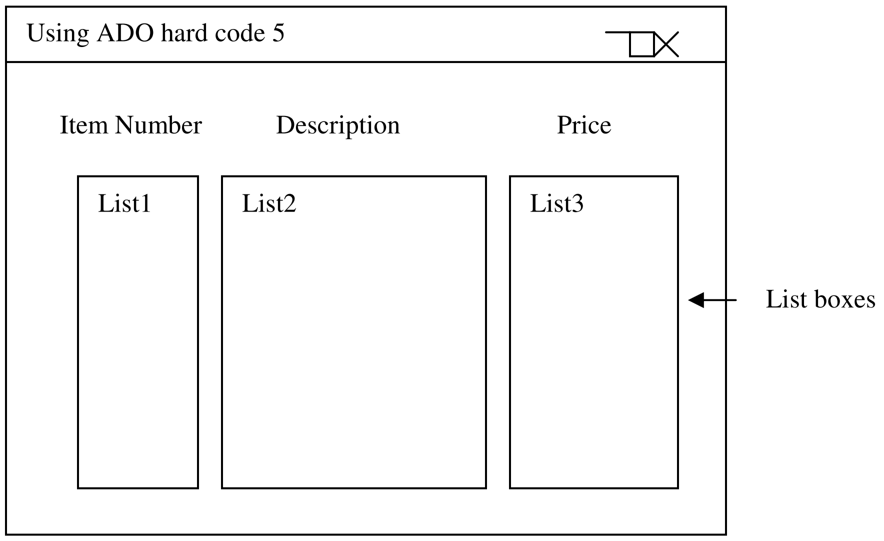
Note:
Use the NWIND database and its SHIPPERS table.

4. Design and develop an application system using DataGrid control that demonstrates how it is used to access a database. Follow the given figure below in designing and developing the application program:



Note:
Use the NWIND database and its EMPLOYEES table.

5. Design and develop an application system using ADO hard code. Follow the given figure below in designing and developing the application program:



Note:

Use the SALES database and its STOCKS table that we have created in Example number 6.

6.Create a database using the Visual Data Manager. Follow the table structure below:

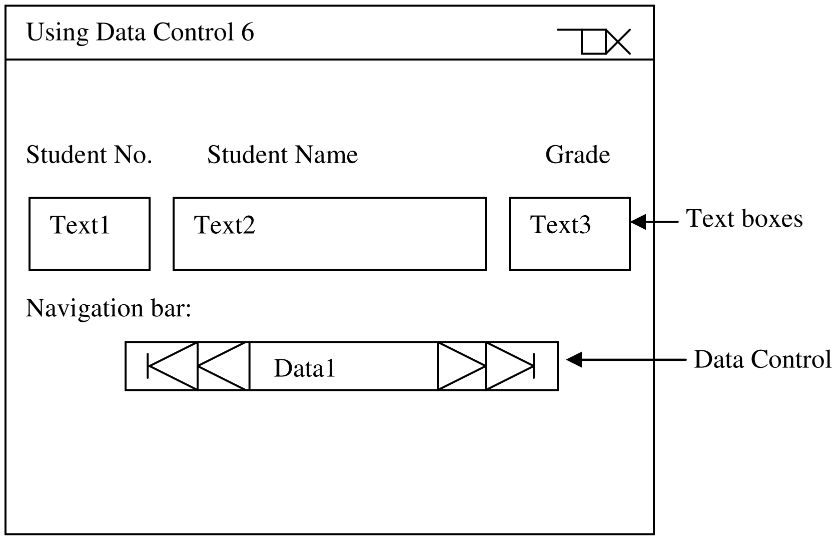
Database name: **GRADES** Table name: **STUDENTS**

StudentNumber	**StudentName**	**Grade**
00001	Castillo, Eddie	90
00002	Dela Rey, Roy	95
00003	Pureza, Gavy	80
00004	Bandolon, Lita	82
00005	Hermoso, Nestie	80
00006	Balolot, Ely	75
00007	Gimongala, Jose	78

Set the values of the following:

StudentNumber (Size): 5
StudentName (Size): 15
Grade (Type): Number

Now design and develop the application system below to manipulate the data of the database GRADES and table STUDENTS:


Using Data Control 6
Student No.
Student Name
Grade
Text1
Text2
Text3
Text boxes
Navigation bar:
Data1
Data Control