CSC 215

(Visual Basic 2013)

Lecture Note

Visual Basic 2013 Lesson 1: Introduction

1.1 Introduction

Visual Basic Express 2013 is the latest version of Visual Basic launched by Microsoft in 2013. Visual Basic Express 2013 is almost similar to Visual Basic Express 2012 but it has added some new features. Like Visual Basic Express 2012, it is now integrated with other Microsoft Programming languages C# and C++ in a package called Visual Studio 2013. Visual Studio Express 2013 now come in five editions, they are:

- Visual Studio Express 2013 for Web
- Visual Studio Express 2013 for Windows
- Visual Studio Express 2012 for Windows Desktop
- Visual Studio Express 2012 for Windows Phone
- Visual Studio Team Foundation Server Express 2013

Right now we shall concentrate on learning Visual Basic 2013 Express for Windows Desktop , so you will nee
software from Microsoft site below: http://www.visualstudio.com/en-us/downloads3

Visual Basic Express 2013 is a full fledged Object-Oriented Programming(OOP) Language, just like other OOP languages such as C++, Java, C# and more.

1.2 Visual Studio 2013 Integrated Development Environment

The Integrated Development Environment when you launch Visual Studio 2013 Express is shown in the diagram below.

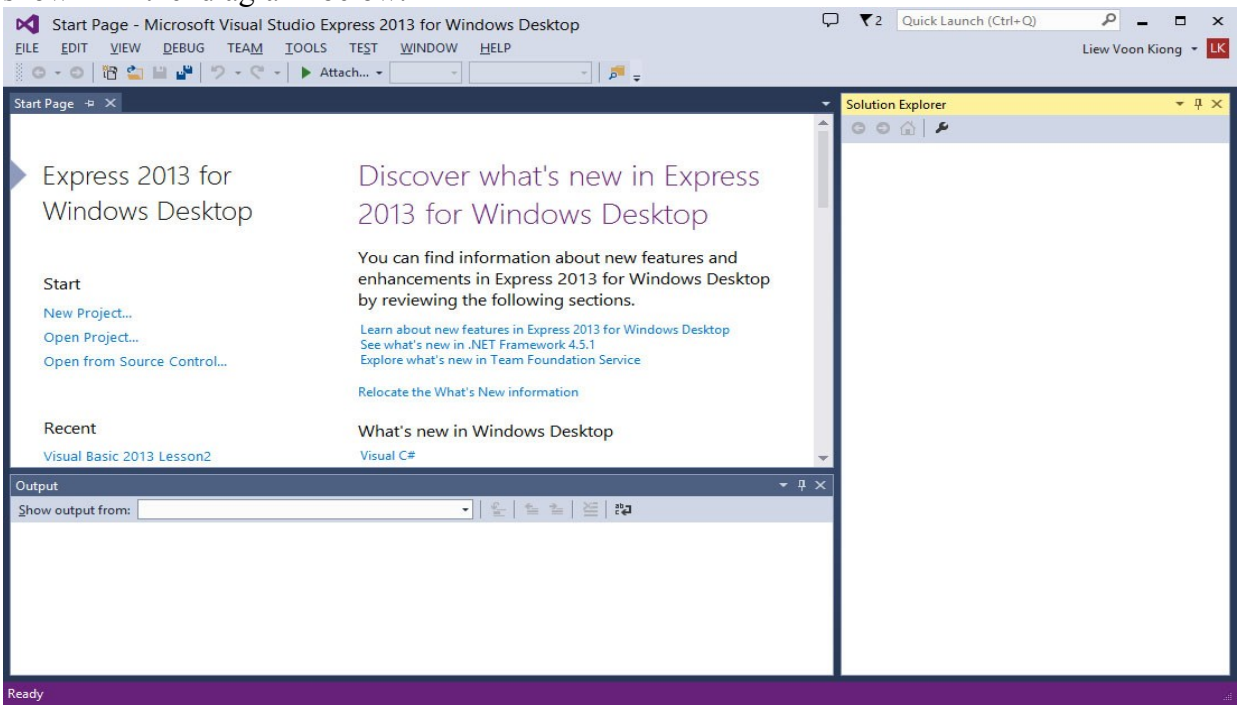


Figure 1.1: Visual Studio 2013 Start Page

1.3 Creating a New Project in Visual Studio 2013

The initial window is the Start Page tab. To start a new Visual Studio Express 2013 project, click on New Project under the Start section to launch the Visual Studio 2013 New Project page as shown in Figure 1.2 below. You can also choose to open a recent project:

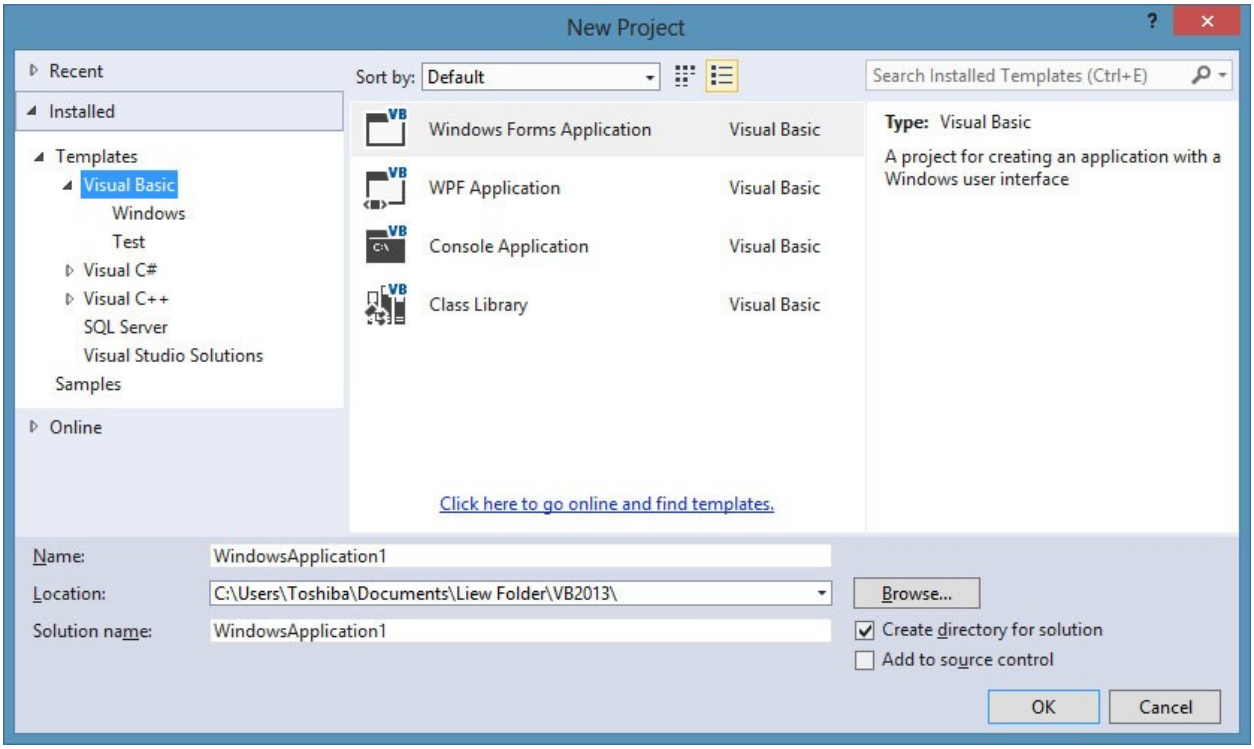


Figure 1.2: Visual Studio 2013 Project Page

The New Project Page comprises three templates, Visual Basic, Visual C# and Visual C++. Since we are going to learn Visual Basic 2013, we shall select Visual Basic. Visual Basic 2013 offers you four types of projects that you can create, they are Windows Forms Application, WPF Application, Console Application and Class Library. Since we are going to learn to create windows Applications, we will select Windows Forms Application.

At the bottom of this dialog box, you can change the default project name WindowsApplication1 to some other name you like, for example, MyFirstProgram. After you have renamed the project, click OK to continue. The following IDE Windows will appear, it is similar to Visual Basic 2012. The Toolbox is not shown until you click on the Toolbox tab. When you click on the Toolbox tab or use the shortcut keys Ctrl+w+x, the common controls Toolbox will appear.

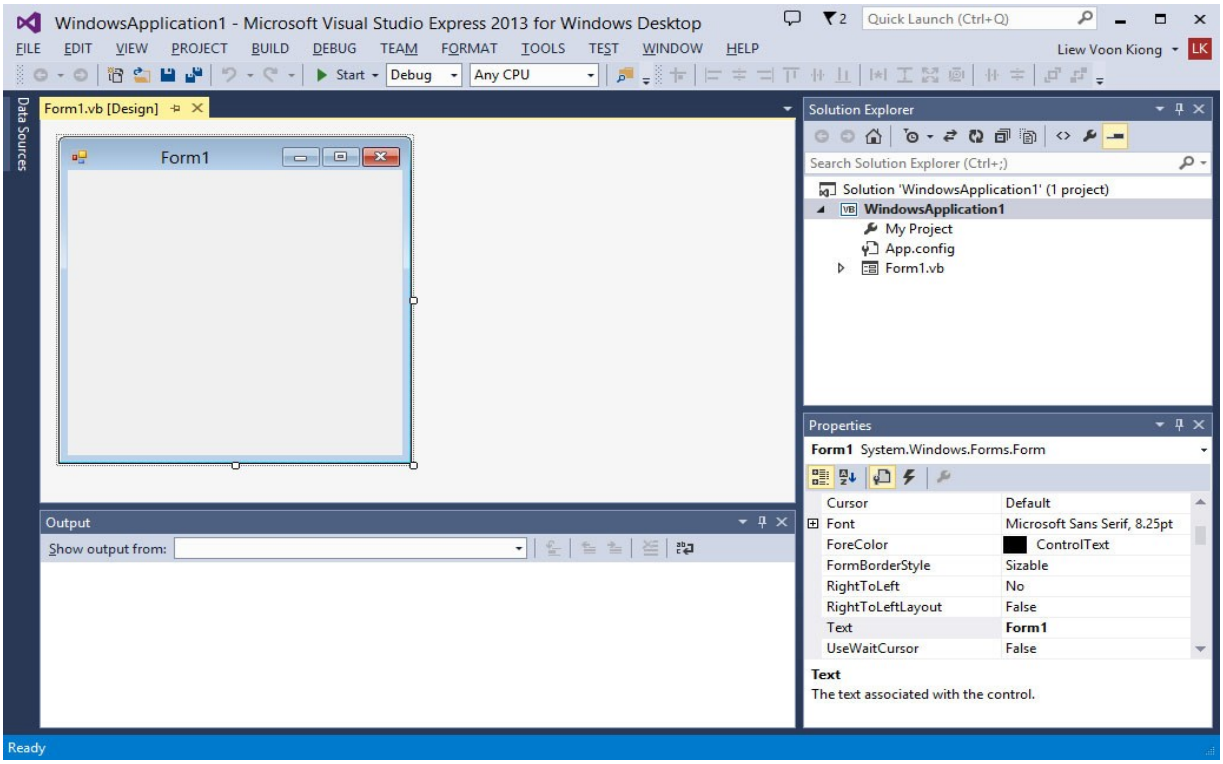


Figure 1.3: Visual Basic 2013 IDE

Visual Basic Express 2013 IDE comprises a few windows, the Form window , the Solution Explorer window and the Properties window . It also consists of a toolbox which contains many useful controls that allows a programmer to develop his or her Visual Basic 2013 programs. The toolbox is shown in Figure 1.4.

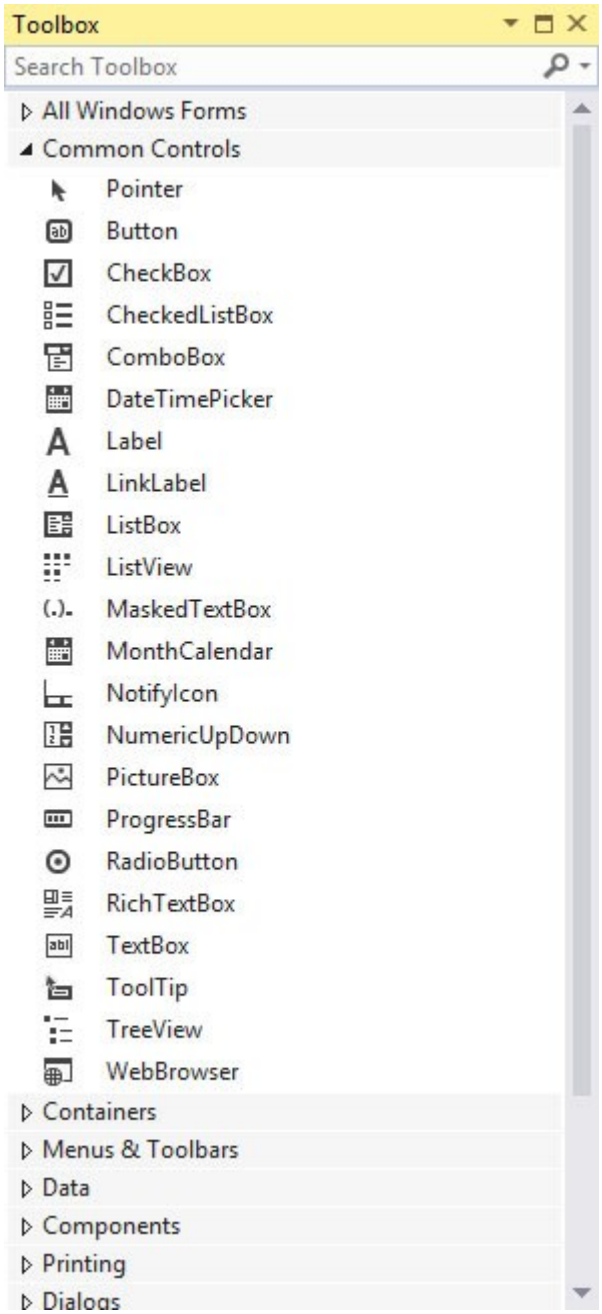


Figure 1.4: Visual Basic 2013 Toolbox

Now, we shall proceed to show you how to create your first program. First, change the text of the form to My First Program in the properties window, it will appear as the title of the program. Next, insert a button and change its text to OK. The design interface is shown in Figure 1.5

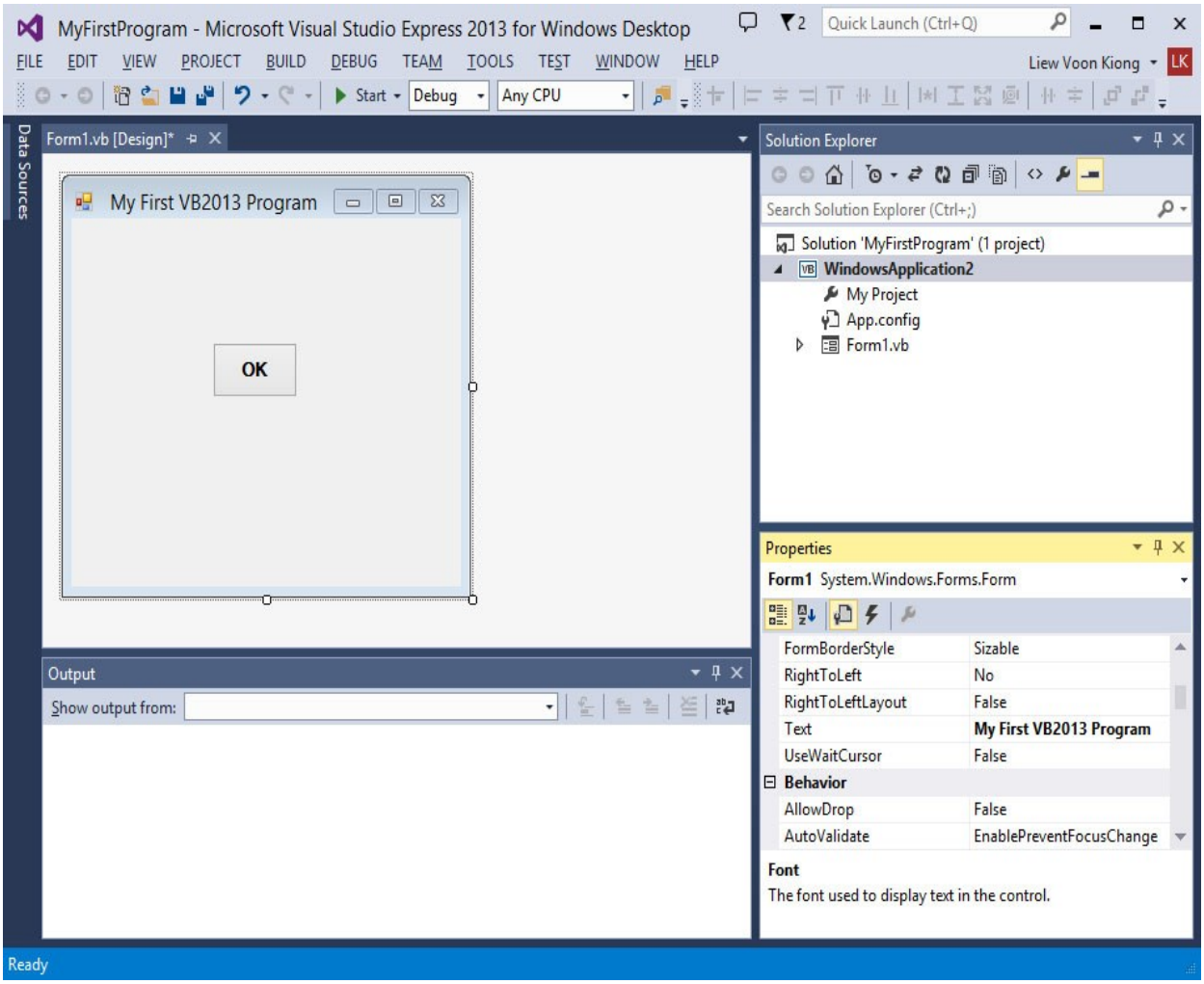


Figure 1.5: The Design Interface

Now click on the OK button to bring up the code window and enter the following statement between Private Sub and End Sub procedure, as shown in Figure 1.6:

MsgBox("My First Visual Basic 2013 Program")

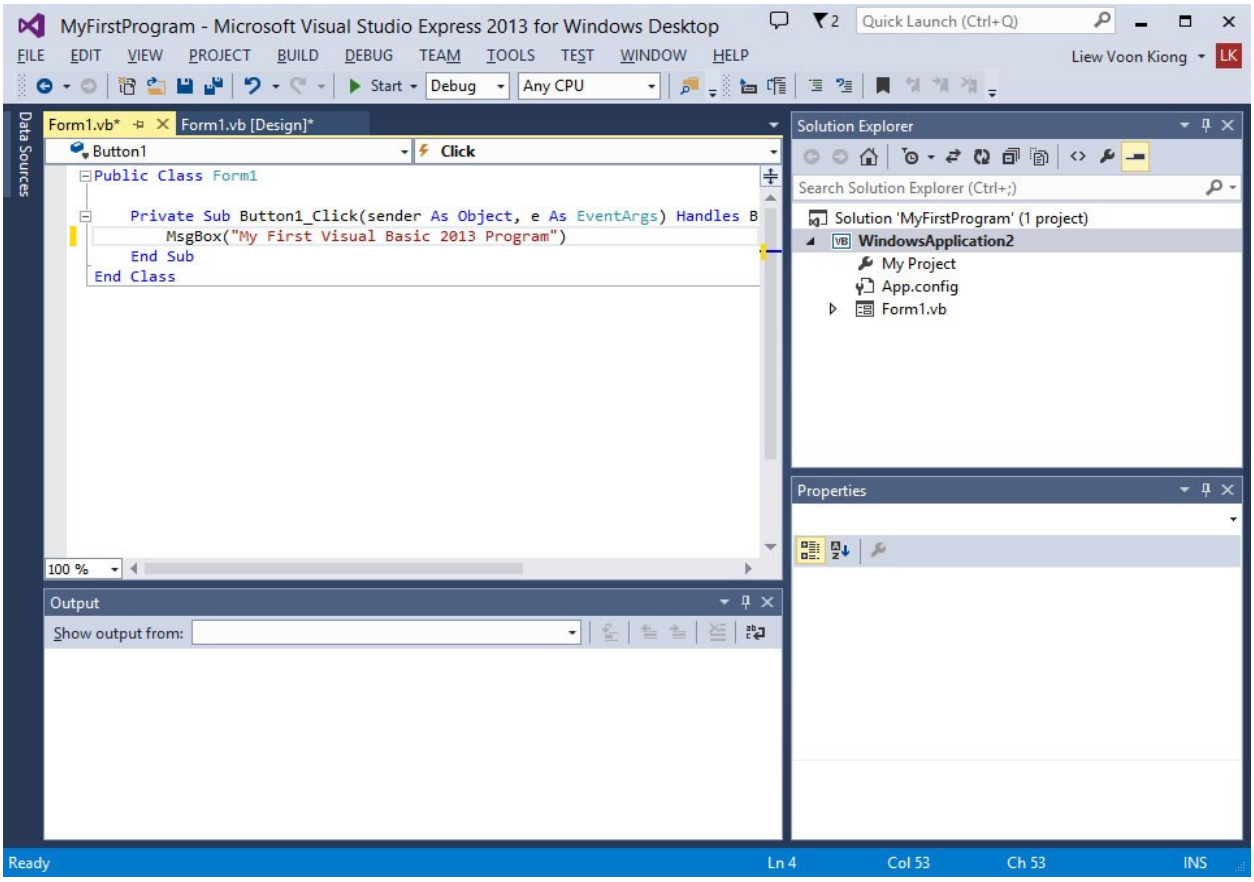


Figure 1.6: Vb2013 Code window

Now click on the Start on the toolbar or press F5 to run the program then click on the OK button, a dialog box that displays the "My First Visual Basic 2013 Program" message will appear, as shown in Figure 1.7:

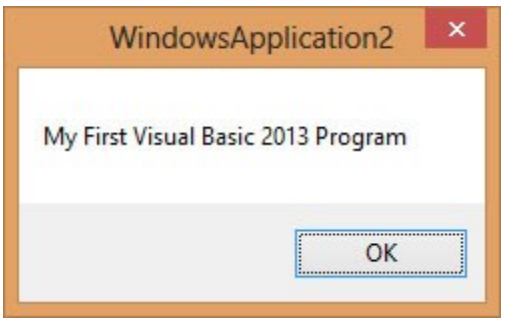


Figure 1.7:The Message Box

The function MsgBox is a built-in function of Visual Basic 2013 and it will display the text enclosed within the brackets.

Now you have created your first program, we shall learn more VB2013 programming techniques in coming lessons.

Visual Basic 2013 Lesson 2: Building the Interface1- Customizing the Form

As Visual Basic 2013 is a GUI-based programming language, the first step in developing an application is to build a graphical user interface. To build a graphical user interface, you need to add controls from the toolbox to the form and then customize their properties. Note that the the default form is also a control by itself and you can change its properties first before adding additional controls. After adding controls to the form, you then need to write code for all the controls so that they can response to events triggered by the user's actions such as clicking the mouse. Therefore, Visual Basic 2013 is also an event-driven programming language. We will learn more about the concept of event-driven programming and coding in later lessons.

2.1 Changing the Properties of the Default-Form using the Properties Window

When you start a new Visual Basic 2013 project, the IDE will display the default form along with the Solution Explorer window and the Properties window for the form as shown in Figure 2.1

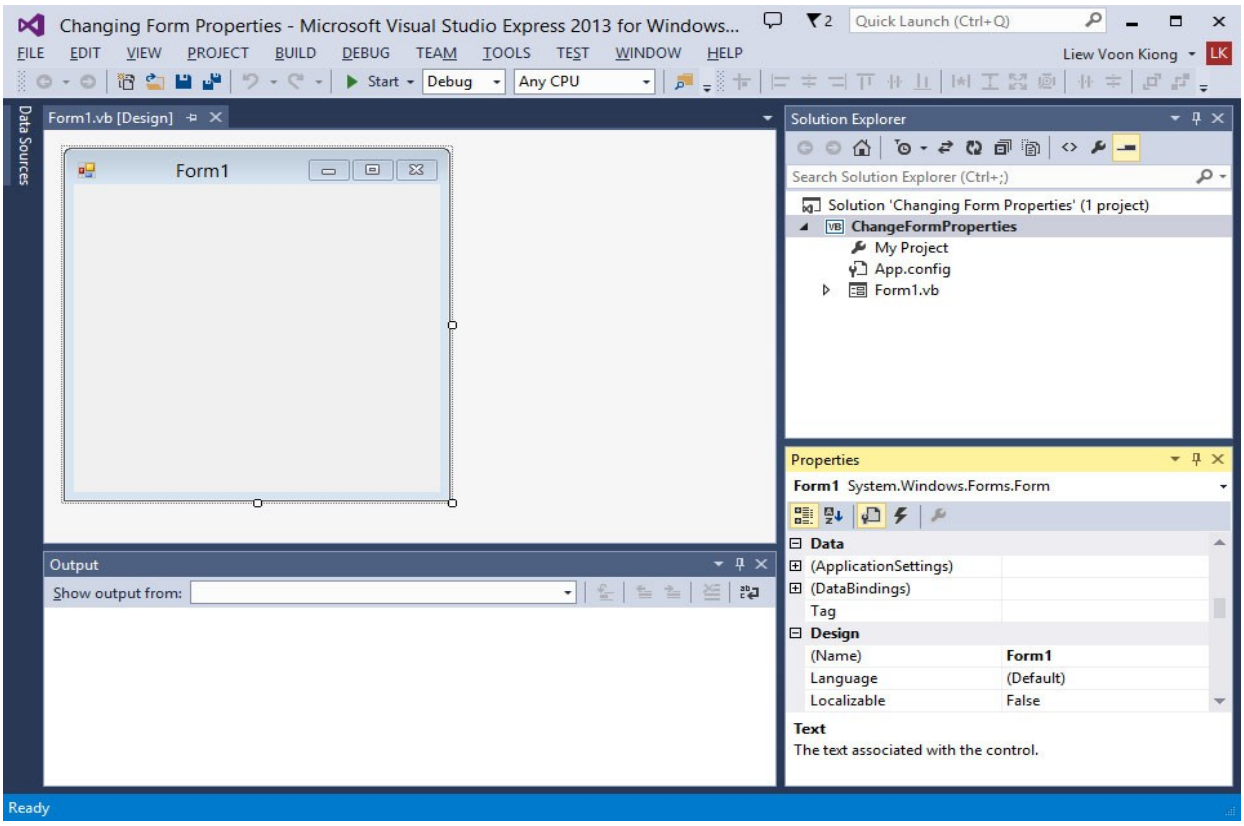


Figure 2.1: Initial VB2013 IDE

The properties window displays all properties related to Form1 and their corresponding attributes or values. You can change the name of the form, the title of the form, the background color, the foreground color , the size and more. Try changing the following properties:

PROPERTY	VALUE
Name	MyForm
Text	My First Program
BackColor	Blue
MaximizeBox	False

The output interface is shown in Figure 2.2. Notice that the title has been changed from Form1 to My First Program, background changed to blue color and the window cannot be maximized.

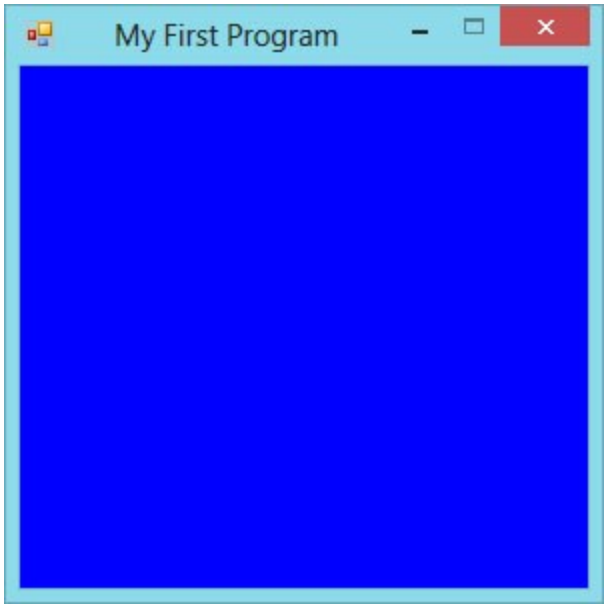


Figure 2.2

2.2 Changing the Properties of the Default-Form at Run-Time

You can also change the properties of the form at run-time by writing the relevant codes. The default form is an object and an instant of the form can be denoted by the name Me. The property of the object can be defined by specifying the object's name followed by a dot or period:

ObjectName.property

For example, we can set the background of the form to blue using the following code

Me.BackColor=Color.Blue

To achieve the same interface as in Figure 2.2, type in the following code by clicking the form to enter the code window:

Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

Me.Text = "My First VB2010 Program"

Me.BackColor = Color.Blue

Me.MaximizeBox=False

Me.MinimizeBox = True

End Sub

End Class

Visual Basic 2013 Lesson 3: Building the Interface 2- Adding Controls to the Form

In previous lesson, we have learned how to build an initial interface in Visual Basic 2013 by customizing the default form. In this lesson, we shall continue to build the interface by adding controls to the form. There are numerous controls that we can add to the form. Among the controls, the most common ones are button, label, text box, list box, combo box, picture box, check box, radio and more. The controls can be made visible or invisible at runtime. However, some controls will only run in the background and cannot be seen at runtime, one such control is the timer.

3.1 Positioning the Toolbox

The Toolbox is usually hidden when you start Visual Basic 2013,you need to click View on the menu bar and then select Toolbox to reveal the tool box, as shown in Figure 3.1. You can also use shortcut keys Ctrl+w+x to bring out the tool box.

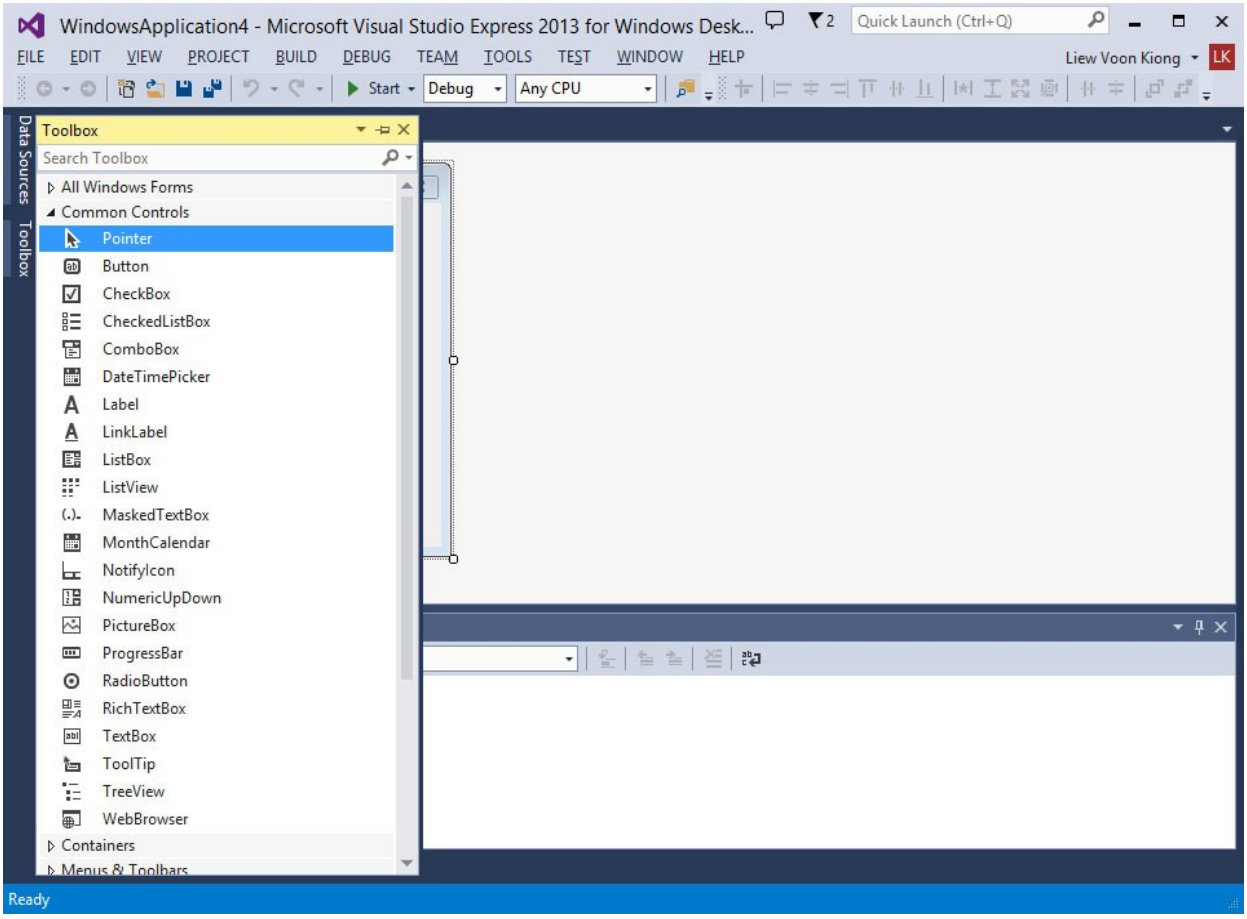


Figure 3.1

You can position the tool box by dragging it anywhere you like while its status is set to float. You can also dock the tool box by right-clicking on the tool box and choose dock from the pop-up menu. The docked tool box that appears side by side with the default form is as shown in Figure 3.2.

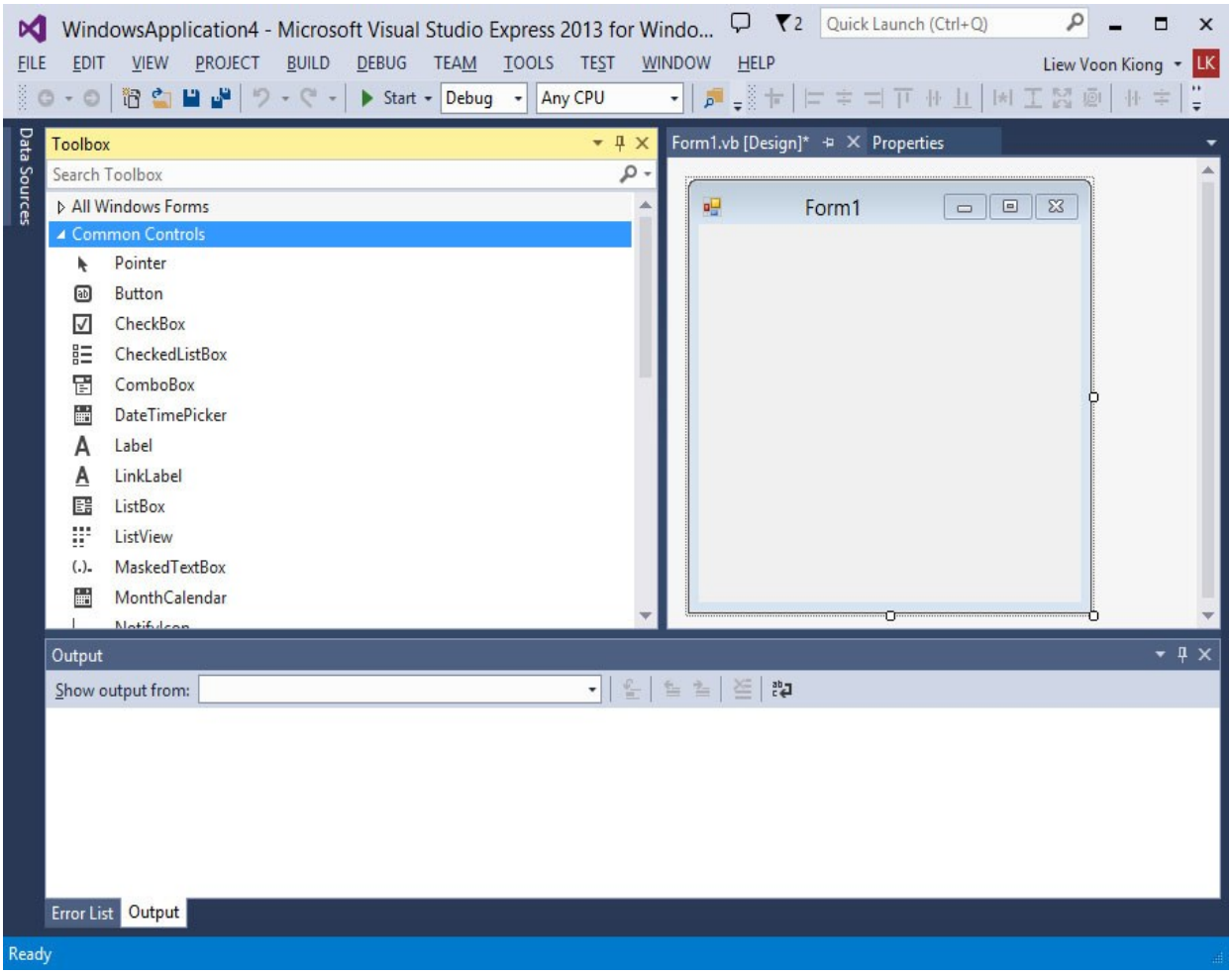

WindowsApplication4 - Microsoft Visual Studio Express 2013 for Windo...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Liew Voon Kiong
Start
Debug
Any CPU
Data Sources
Toolbox
Search Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
Form1.vb [Design]*
Properties
Form1
Output
Show output from:
Error List
Output
Ready

Figure 3.2

You can also dock the tool box at the bottom, below the default form, as shown in Figure 3.3

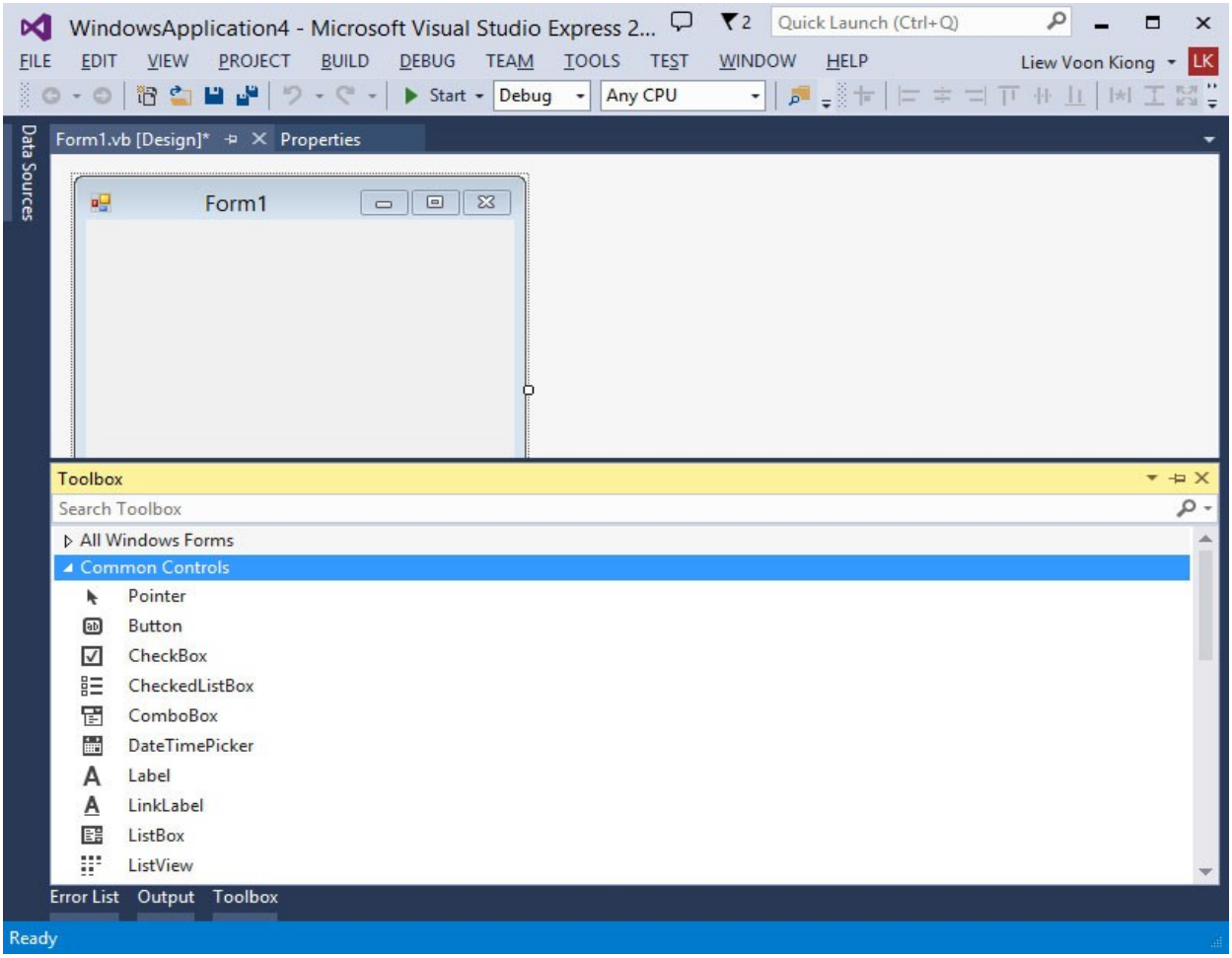


Figure 3.3

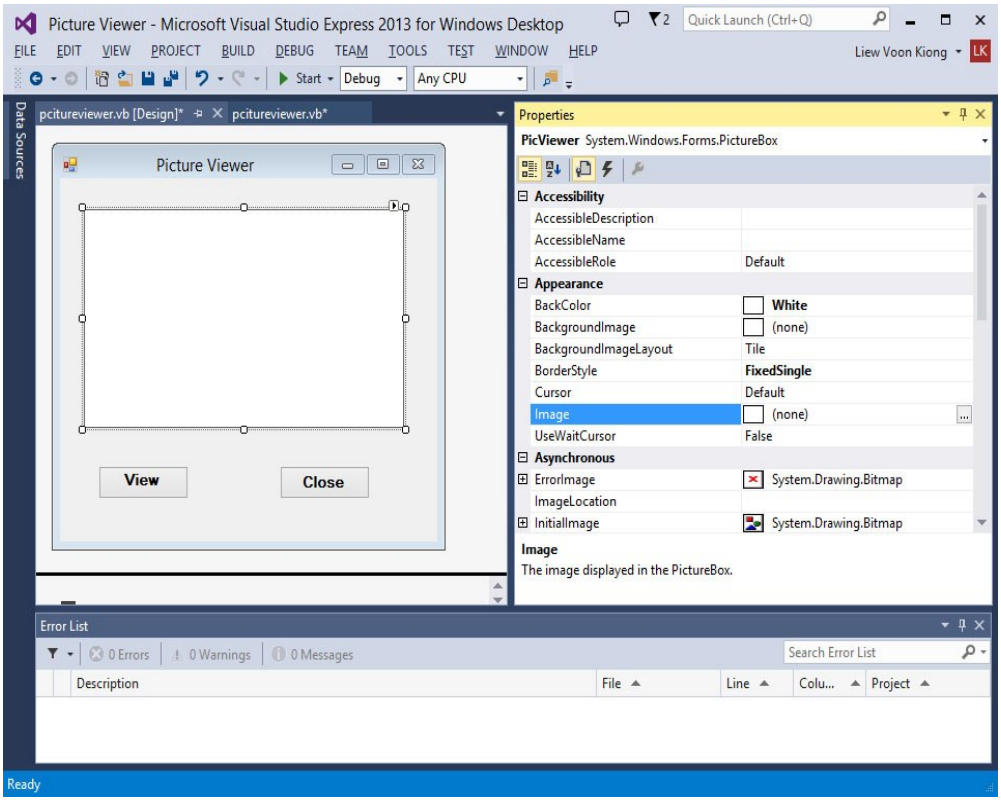
Further, you may also pin the tool box to the side bar or the bottom bar by clicking on the pin icon on the menu bar of the tool box.

How and where you want to position your tool box is entirely up to you but I strongly suggest that you place the tool box along side or at the bottom of the default form so that it is easy for you to add controls from the tool box into the form. You should never cover the form with the tool box because it will be difficult to add controls to the form.

3.2 Adding Controls to the Form and Changing Their Properties

Adding controls to the form is an easy task, what you need to do is drag a control from the tool box and drop it onto the form or draw it on the form. You can drag the control around the form and you can also resize it easily.

To demonstrate how to add controls and then change their properties, we shall design a picture viewer. First, change the title of the default form to Picture Viewer in its properties window. Next, insert a picture box on the form and change its background color to white. To do this, right click the picture box and select properties in the popup menu, then look for the BackColor Property as shown in the properties window below:



Finally, add two buttons to the form and change the text to View and Close in their respective properties windows. Now, we have designed a basic picture viewer. We shall add more features later.

The picture viewer is not functional until we write code to response to events triggered by user. We will deal with the programming part in the coming lessons.

Visual Basic 2013 Lesson 4: Writing the Code

In previous lesson, we have learned how to design the user interface by adding controls to the form and changing their properties. However, the user interface alone will not work without adding code to them. In this lesson, we shall learn how to write code for all the controls so that they can interact with events triggered by the users. Before learning how to write Visual Basic 2013 code, let us dwell into the concept of event-driven programming

4.1 The Concept of Event-Driven Programming

Visual Basic 2013 is an event driven programming language because we need to write code to response to certain events triggered randomly by the user via the controls on the form. These events do not occur in a certain order. The events usually comprises but not limited to the user's inputs. Some of the events are load, click, double click, drag and drop, pressing the keys and more.

Every form and and every control you place on the form has a set of events related to them. Some of the events are load, click, double click, drag and drop, pressing the keys and more. To view the events, double-click the control (object) on the form to enter the code window. The default event will appear at the top part on the right side of the code window. You need to click on the default event to view other events associated with the control. The code appears on the left side is the event procedure associated with the load event. Figure 4.1 illustrates the event procedure load associated with the default form .

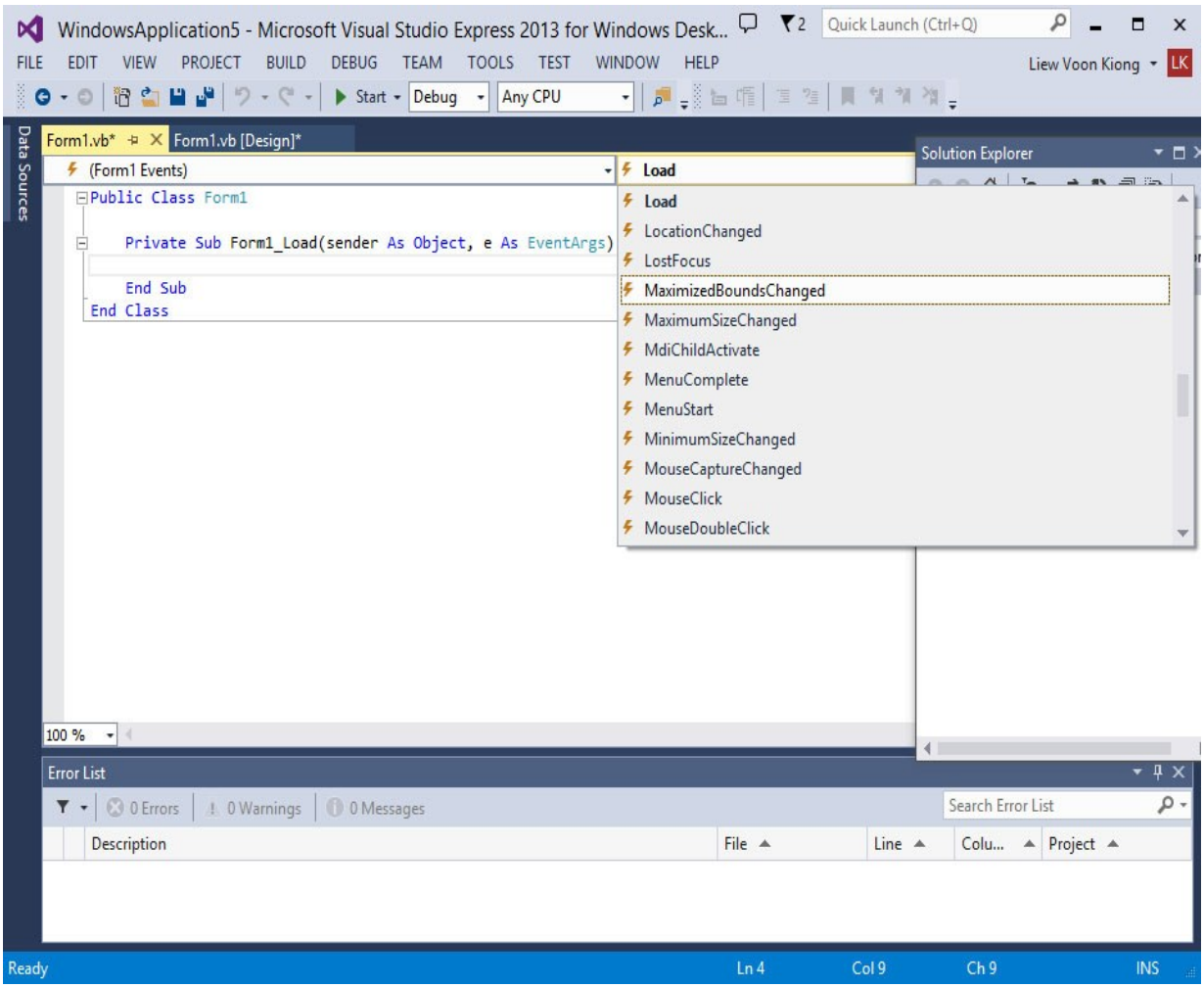


Figure 4.1: Events associated with Form

Figure 4.2 shows the events associated with button

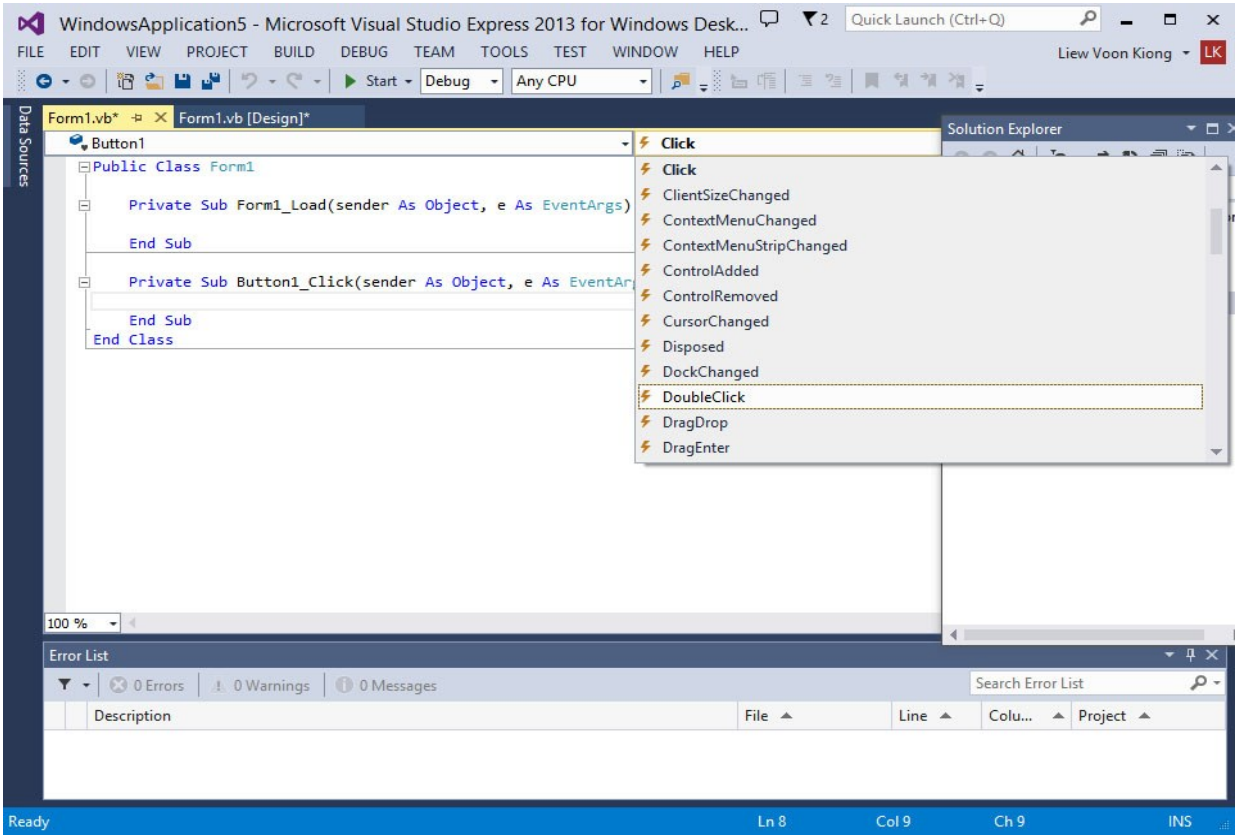


Figure 4.2

2.2 Writing the Code

To start writing code in Visual Basic 2013, click on any part of the form to go into the code window as shown in Figure 4.1. This is the structure of an event procedure. In this case, the event procedure is to load Form1 and it starts with **Private Sub** and end with **End Sub**. This procedure includes the Form1 class and the event **Load**, and they are bind together with an underscore, i.e. Form_Load. It does nothing other than loading an empty form. To make the load event does something, insert the statement

MsgBox "Welcome to Visual Basic 2013″

Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

MsgBox “My First Visual Basic 2013 Program”

End Sub

End Class

When you run the program, a message box display the text “My First Visual Basic 2013 Program” will appear, as show in Figure 4.3. MsgBox is a built-in function in Visual Basic 2013 that display a message in a pop-up message box.

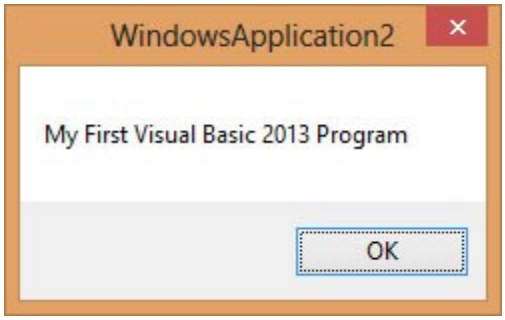


Figure 4.3

* You will notice that above Private Sub structure there is a preceding keyword Public Class Form1. This is the concept of an object oriented programming language. When we start a windows application in Visual Basic 2013, we will see a default form with the name Form1 appears in the IDE, it is actually the Form1 Class that inherits from the Form class **System.Windows.Forms.Form.** A class has events as it creates an instant of a class or an object.

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.LoadMsgBox(“2″ & “+” & “5” & “=” & 2 + 5)

End Sub

*The symbol & (*ampersand)* is to perform string concatenation.

The output is as shown in Figure 4.4

You can also write code to perform arithmetic calculation. For example, you can use the MsgBox and the arithmetic operator plus to perform an addition of two numbers, as shown below:

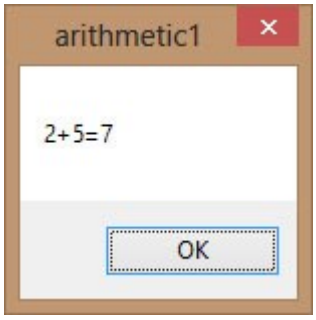


Figure 4.4

If you wish to close the window after the message, you can add the statement **Me.Close(),** as follows:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

MsgBox("2″ & "+" & "5" & "=" & 2 + 5)
Me.Close

End Sub
```

We will learn more about code writing in coming lessons

Visual Basic 2013 Lesson 5: Working with Controls

In previous lesson, we have learned how to write simple Visual Basic 2013 code. In this lesson, we will learn how to work with some common controls and write codes for them. Some of the commonly used controls are label, text box, button, list box and combo box. However, in this lesson, we shall only deal with the text box and the label, we shall deal with other controls later.

5.1 Text Box

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

Example 5.1

In this program, you add two text boxes and a button on the form. The two text boxes are used to accept inputs from the user . Besides, a button is also programmed to calculate the sum of the two numbers using the plus operator. The value enter into a text box is stored using the syntax **Textbox1.Text** , where Text is the one of the properties of text box.

The following program will add the value in text box 1 and value in text box 2 and output the sum in a message box.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
MsgBox(The sum is & Val(TextBox1.Text) + Val(TextBox2.Text))
End Sub
```

The output window:

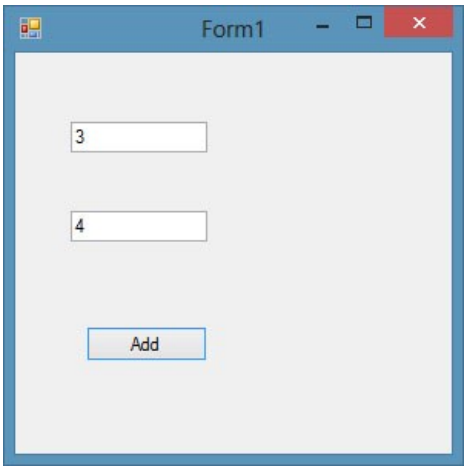


Figure 5.1

After clicking Add button, you can get the answer in a message box, as shown in Figure 5.2:

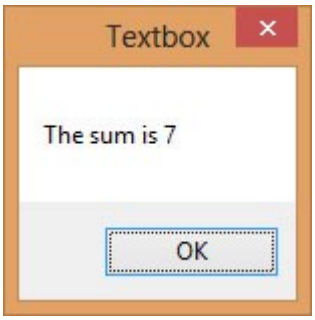


Figure 5.2

5.2 The Label

The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. It is different from text box because it can only display static text, which means the user cannot change the text. Using the syntax Label.Text, it can display text and numeric data . You can change its text in the properties window and also at runtime.

Example 5.2

Based on Example 5.1, you now add two labels, one is to display the text Sum= and the other label is to display the answer of the Sum. For the first label, change the text property of the label by typing Sum= over the default text Label1. Further, change its font to bold and font size to 10. For the second label, delete the default text Label2 and change its font to bold and font size to 10. Besides that, change its background color to white.

In this program, instead of showing the sum in a message box, we wish to display the sum on the label.

The Code

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Label2.Text = Val(TextBox1.Text) + Val(TextBox2.Text)
End Sub
```

*The function Val is to convert text to numeric value. Without using Val, you will see that two numbers are joined together without adding them.

The output is as shown in Figure 5.3

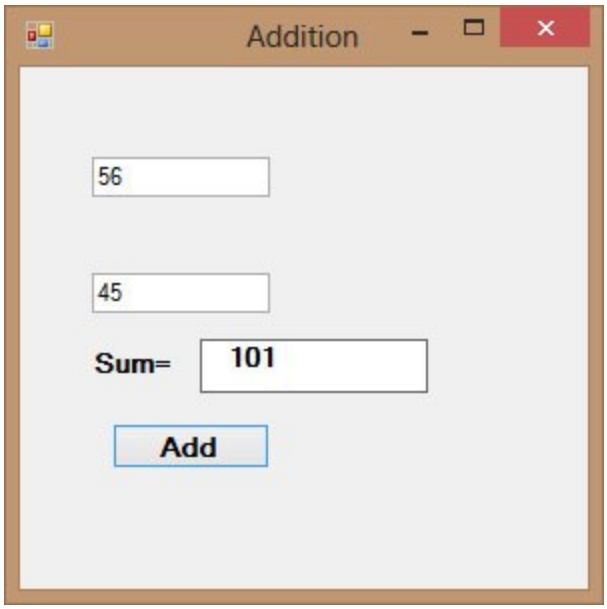


Figure 5.3

Visual Basic 2013 Lesson 6: Working with List Box and Combo Box

In previous lesson, we have just mastered the usage of text boxes and labels in Visual Basic 2013. In this lesson, we shall learn two more important controls, the list box and the combo box. Both controls are used to display a list of items. However, they differ slightly in the ways they display the items. The list box displays the items all at once in a text area whilst combo box displays only one item initially and the user needs to click on the handle of the combo box to view the items in a drop-down list.

6.1 List Box

The function of the List Box is to present a list of items where the user can click and select the items from the list. Items can be added at design time and at runtime. The items can also be removed at design time and also at runtime.

6.1.1 Adding Items to a List Box

To demonstrate how to add items at design time, start a new project and insert a list box on the form. Right-click on the list box to access the properties window. Next, click on collection of the Item property, you will be presented with String Collection Editor whereby you can enter the items one by one by typing the text and press the Enter key, as shown in Figure 6.1:

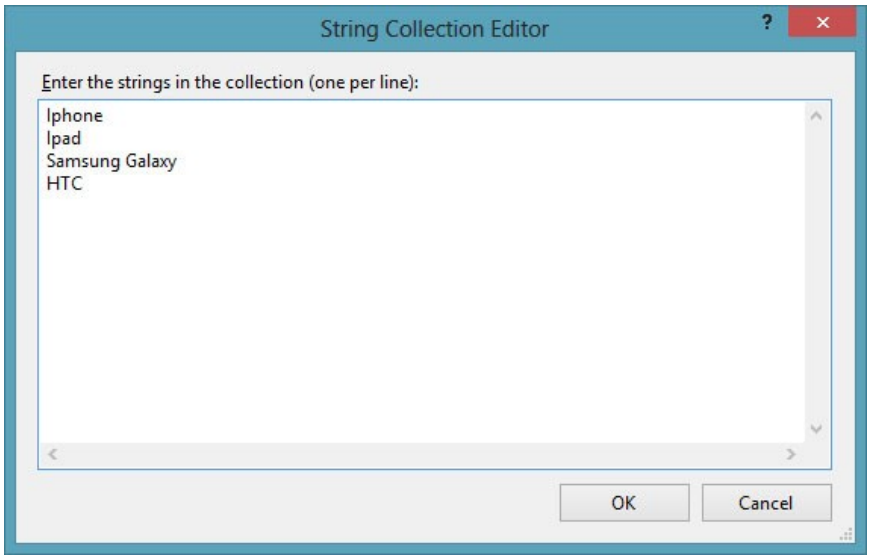


Figure 6.1: String Collection Editor

After clicking on the OK button, the items will be displayed in the text box, as shown in Figure 6.2

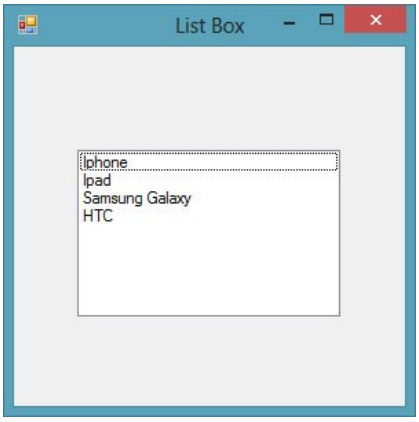


Figure 6.2

Items can also be added at runtime using the **Add( ) method**. Before we proceed further, we need to know that Visual Basic 2013 is a full fledged object oriented programming language. Therefore, visual basic 2013 comprises objects. All objects have methods and properties, and they can are differentiated and connected by hierarchy. For a list box, Item is an object subordinated to the object ListBox . Item comprises a method call Add() that is used to add items to the list box. To add an item to a list box, you can use the following syntax:

ListBox.Item.Add("Text")

For example, if you wish to add a new item to ListBox1 above, you can key-in the following statement

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
ListBox1.Items.Add("Nokia")
End Sub
```

The item "Nokia" will be added to the end of the list, as shown in Figure 6.3

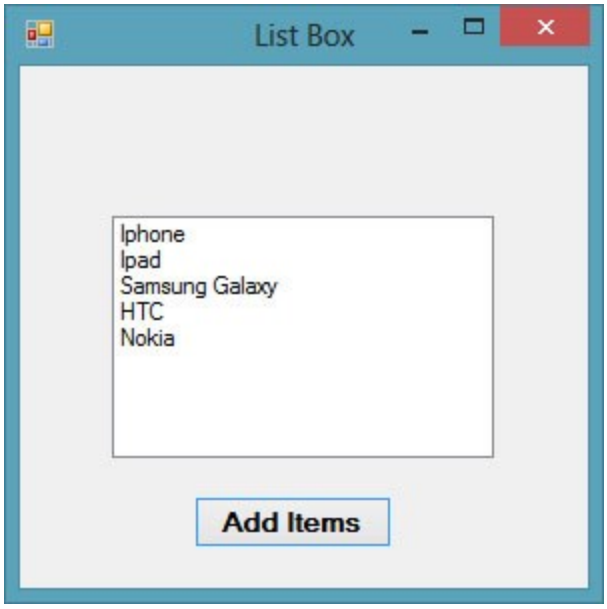


Figure 6.3

You can also allow the user to add their own items using the InputBox function, as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim myitem
myitem = InputBox("Enter your Item")
ListBox1.Items.Add(myitem)
End Sub
```

* The keyword Dim is to declare the variable myitem. You will learn more about variables in coming lessons

6.1.2 Removing Items from a List Box

To remove items at design time, simply open the String Collection Editor and delete the items at line by line or all at once using the Delete key.

To remove the items at runtime, you can use the **Remove** method, as illustrated in the following example. In this example, add a second button and label it "Remove Items". Click on this button and enter the following code:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
ListBox1.Items.Remove("Ipad")
End Sub
```

The item "Ipad" will be removed after running the program. You can also let the user choose which item to delete.

To clear all the items at once, use the clear method, as illustrated in the following example. In this example, add a button and label it "Clear Items"

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button2.Click
ListBox1.Items.Clear()
End Sub
```

6.2 Combo Box

In Visual Basic 2013, the function of the Combo Box is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the handle(small arrowhead) on the right of the combo box to see the items which are presented in a drop-down list.

6.2.1 Adding Items to a Combo Box

In order to add items to the list at design time, you can also use the String Collection Editor. You will also need type an item under the text property in order to display the default item at runtime.

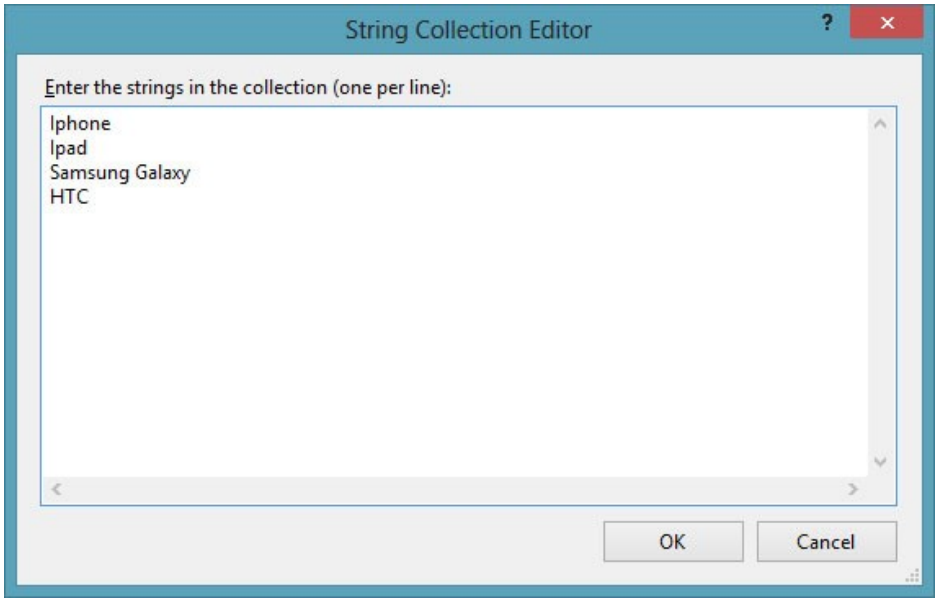


Figure 6.4

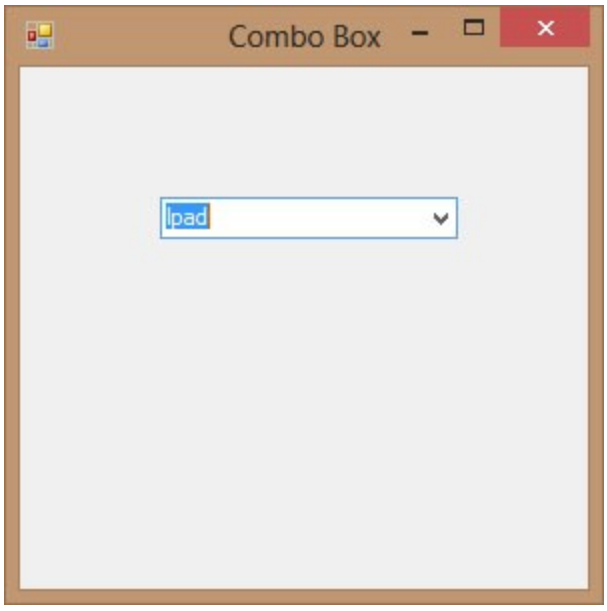


Figure 6.5

After clicking the handle of the right side of the combo box, the user will be able to view all the items.

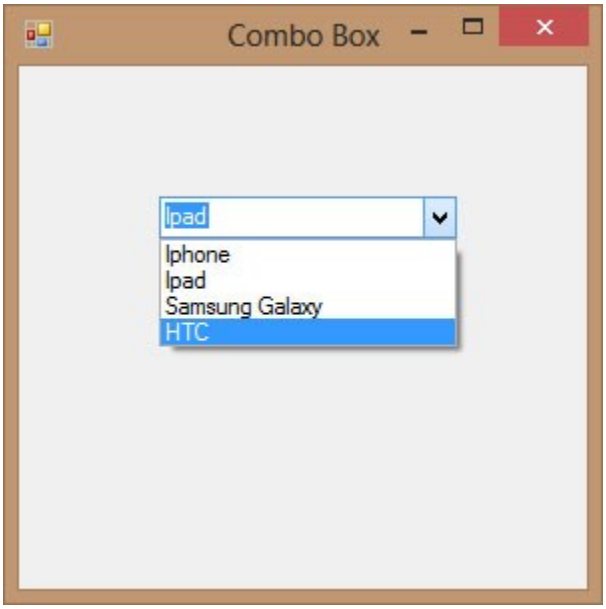


Figure 6.6

Besides, you may add items using the **Add() method**. For example, if you wish to add a number of items to Combo box 1, you can key in the following statement

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
ComboBox1.Items.Add("Nokia")
End Sub

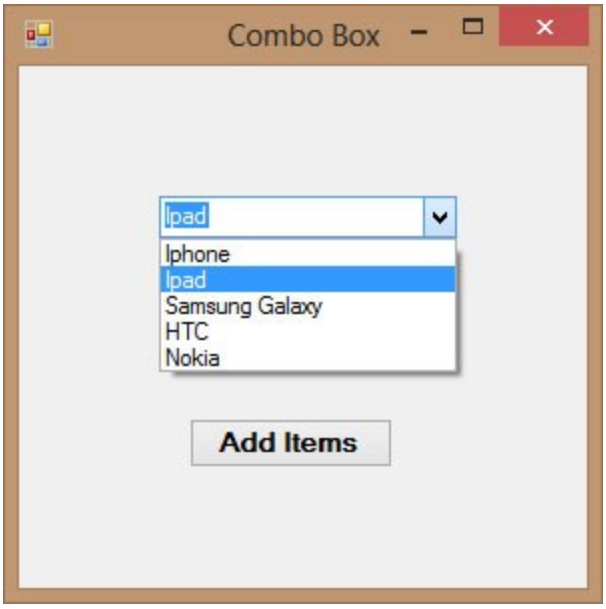


Figure 6.7

You can also allow the user to add their own items using the InputBox function, as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim myitem
myitem = InputBox("Enter your Item")
ComboBox1.Items.Add(myitem)
End Sub
```

6.2.2 Removing Items from a Combo Box

To remove items at design time, simply open the String Collection Editor and delete the items at line by line or all at once using the Delete key.

To remove the items at runtime, you can use the **Remove** method, as illustrated in the following example. In this example, add a second button and label it "Remove Items". Click on this button and enter the following code:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
ComboBox1.Items.Remove("Ipad")
End Sub
```

The item "Ipad" will be removed after running the program. You can also let the user choose which item to delete.

To clear all the items at once, use the clear method, as illustrated in the following example. In this example, add a button and label it "Clear Items"

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button2.Click
ComboBox1.Items.Clear()
End Sub
```

Visual Basic 2013 Lesson 7: Displaying Images in the Picture Box

In lesson 3, we have learned how to insert a picture box on the form in Visual Basic 2013. However, we have not learned how to load a picture in the picture box yet. In this lesson, we shall learn how to load an image into the picture box at design time and at runtime. Besides that, we shall also learn how to using a common dialog control to browse for image files in your local drives and then select and load a particular image in the picture box.

7.1 Loading an Image in a Picture Box

7.1.1 Loading an Image at Design Time

First, insert a picture box on the form and change its border property to FixedSingle and its background to white. You might also want to change the size mode of the image to stretchable so that the image can fit in the picture box. Now right-click on the picture box to bring out its properties window. In the properties window, scroll to the Image property , as shown in Figure 7.1

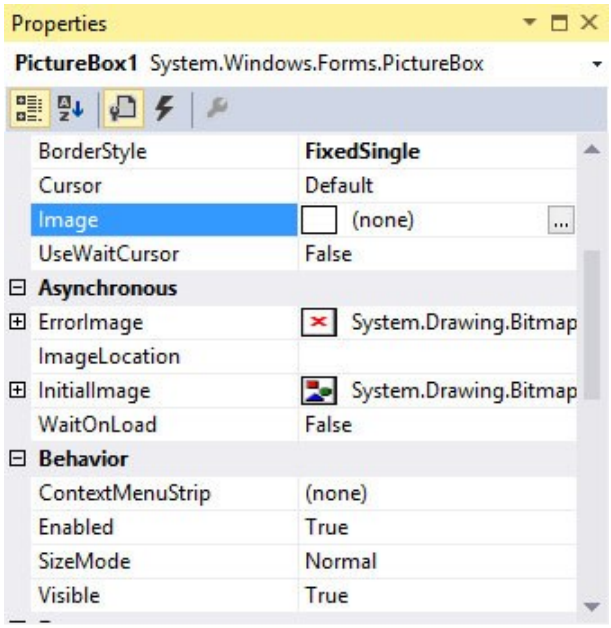


Figure 7.1

Next, click on the grey button on its right to bring out the "Select Source" dialog box , as shown in Figure 7.2

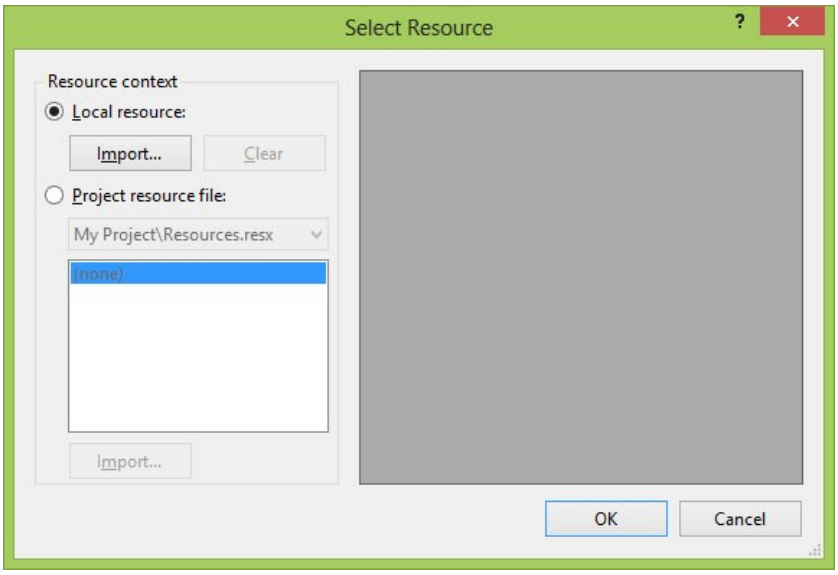
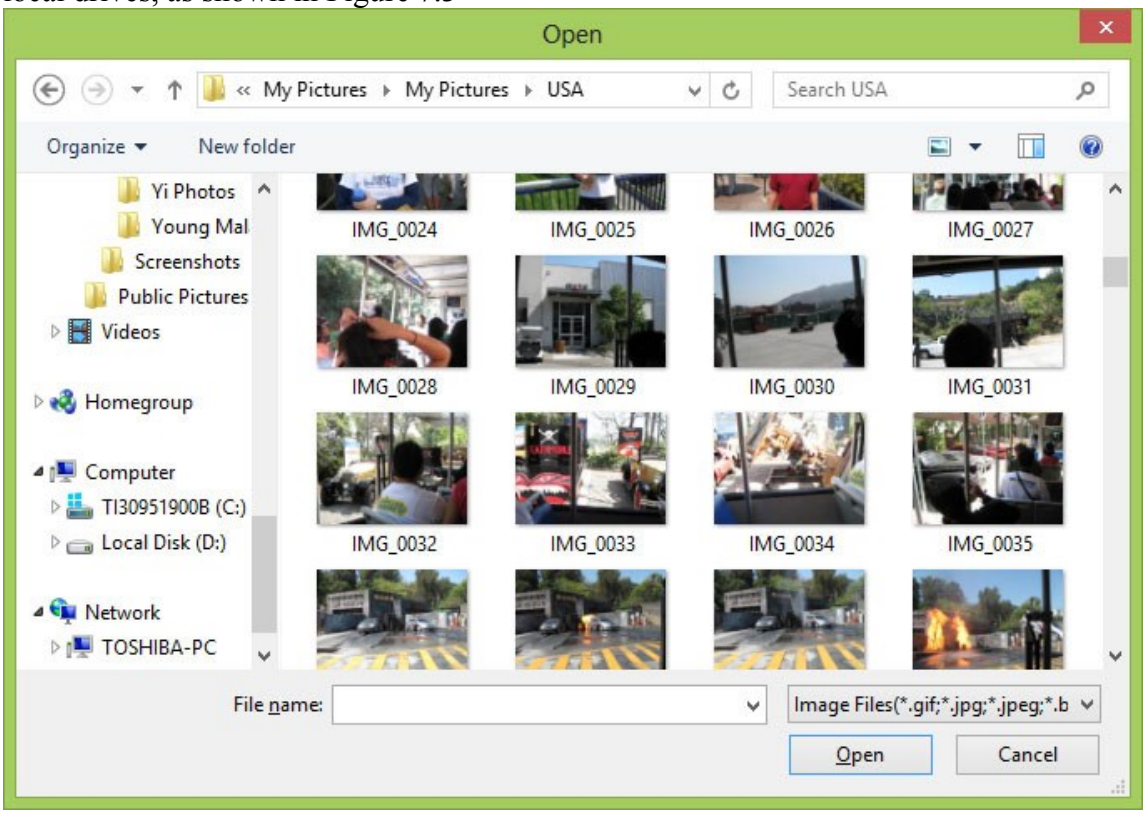


Figure 7.2

Now select local source and click on the Import button to view the available image files in your local drives, as shown in Figure 7.3



Finally, select the image you like and then click the open button, the image will be displayed in the picture box, as shown in Figure 7.4

Figure 7.4

7.1.2 Loading an Image at Runtime

In Visual Basic 2013, an image can also be loaded at runtime, using the code as follows:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

PictureBox1.Image = Image.FromFile("C:\Users\Toshiba\Pictures\My Pictures\USA\Chicago 2012.jpg")

End Sub
```

* You need to search for an image in your local drive and determine its path.

Running the program will display the same image in the picture box as in Figure 7.4

7.2 Loading an Image in a Picture Box using Open File Dialog Control

We have designed the picture viewer interface in lesson 3. Now we shall write code so that the user can browse for the image files in his or her local drives then select a particular image to display in the picture box.

First, we need to add the OpenFileDialog control on the form. This control will be invisible during runtime but it facilitates the process of launching a dialog box and let the user browse his or her local drives and then select and open a file. In order for the OpenFileDialog to display all types of image files, we need to specify the types of image files under the Filter property. Before that, rename OpenFileDialog as OFGSelectImage. Next, right-click on the OpenFileDialog control to access its properties window. Beside the Filter property, specify the image files using the format:
JPEG Files| *.JPG|GIF Files|*.GIF|WIndows Bitmaps|*.BMP
as shown in Figure 7.5. These are the common image file formats. Besides that, you also need to delete the default Filename.

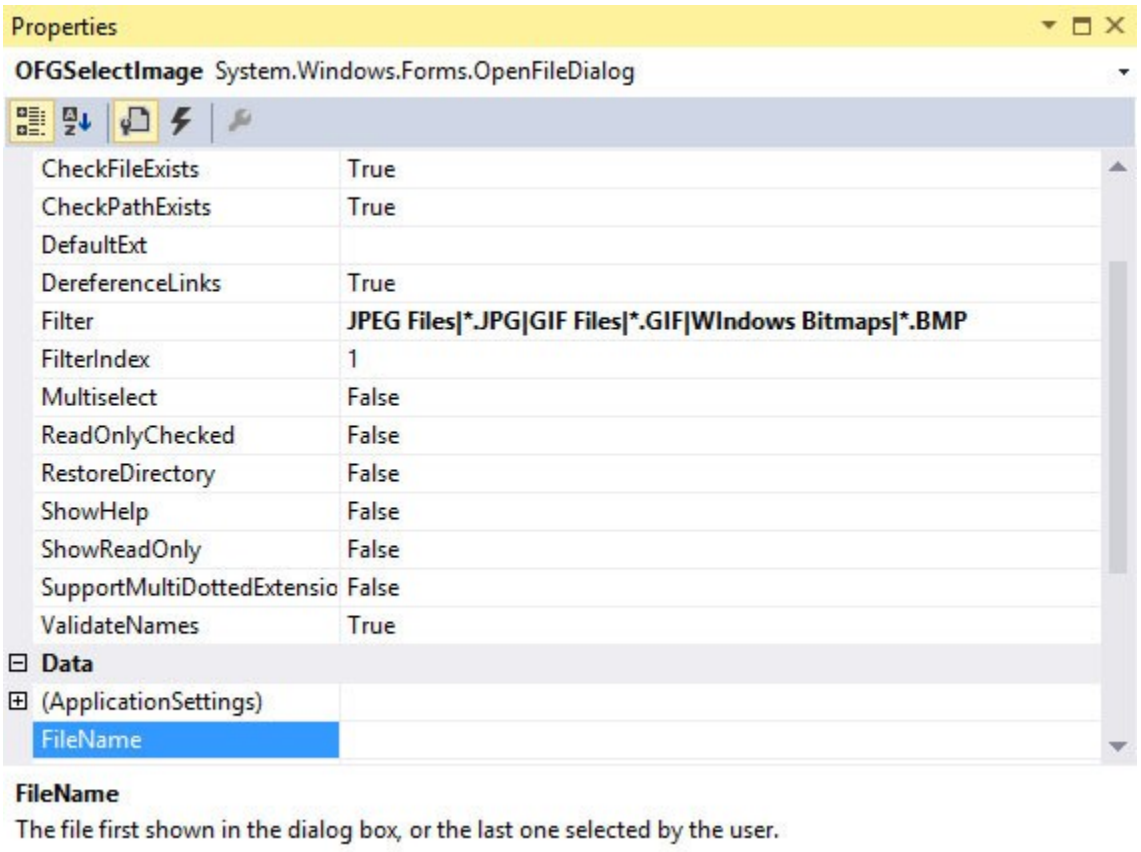


Figure 7.5

Next, double-click on the **View** button and enter the following code:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

If OFGSelectImage.ShowDialog = Windows.Forms.DialogResult.OK Then
PictureBox1.Image = Image.FromFile(OFGSelectImage.FileName)

End If
End Sub
```

Press F5 to run the program and click the View button, a dialog box showing all the image files will appear, as shown in Figure 7.6

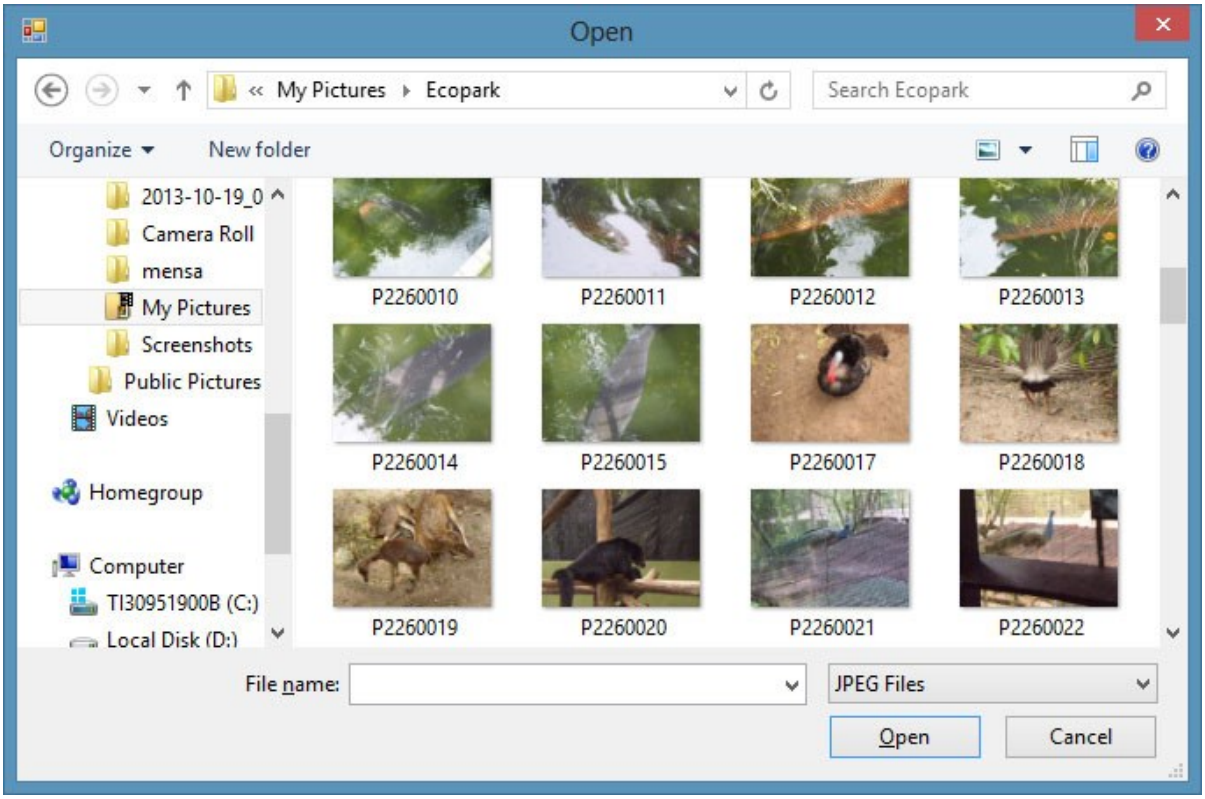


Figure 7.6

Please notice that that the default image file is JPEG as we have placed it in the first place in the Filter property. Selecting and opening an image file will load it in the picture box, as shown in Figure 7.7

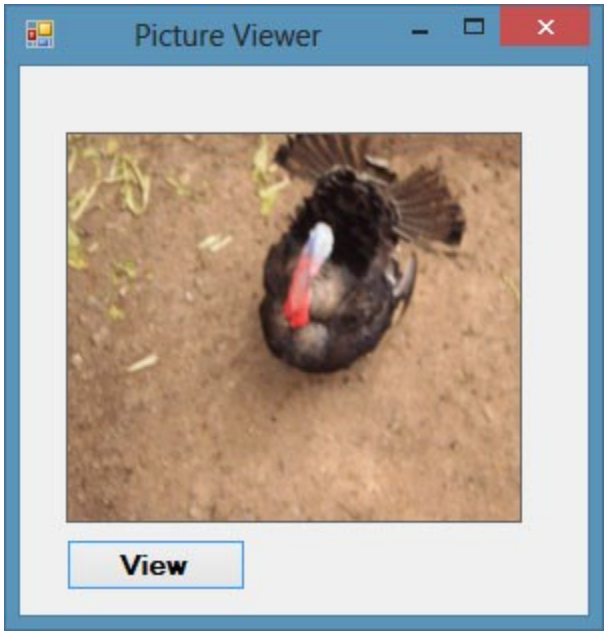


Figure 7.7

Visual Basic 2013 Lesson 8: Understanding Data

We deal with many kinds of data in our daily life. For example, we need to handle data like names, phone number, addresses, money, date, stock quotes, statistics and other data every day. Similarly in Visual Basic 2013, we have to deal with all sorts of of data, some of them can be mathematically calculated while some are in the form of text or other non-numeric forms. In Visual Basic 2013, data can be stored as variables, constants or arrays. The values of data stored as variables always change, just like the contents of a mail box or the storage bin while the value of a constant remains the same throughout. (We shall deal with variables, constants and arrays in coming lessons)

8.1 Visual Basic 2013 Data Types

Visual Basic 2013 classifies information into two major data types, the numeric data types and the non-numeric data type

8.1.1 Numeric Data Types

Numeric data types are types of data comprises numbers that can be calculated mathematically using various standard operators such as addition, subtraction, multiplication, division and more. Examples of numeric data types are examination marks, height, body weight, number of students in a class, share values, price of goods, monthly bills, fees , bus fares and more. In Visual Basic 2013, numeric data are divided into seven types based on the range of values they can store. Calculations that only involve round figures or data that do not need high precision can use Integer or Long integer . Programs that require high precision calculation need to use Single and Double precision data types, they are also called floating point numbers. For currency calculation , you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve many decimal points, we can use the decimal data types. These data types are summarized in Table 8.1

Table 8.1: Numeric Data Types

TYPE	STORAGE	RANGE
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

8.1.2 Non-numeric Data Types

Non-numeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .They are summarized in Table 8.2

Table 8.2: Non-numeric Data Types

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

8.1.3 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data. The suffixes are summarized in Table 8.3.

Table 5.3

SUFFIX	DATA TYPE
&	Long
!	Single

#	Double
@	Currency

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

memberName=”Turban, John.”

TelNumber=”1800-900-888-777″

LastDay=#31-Dec-00#

ExpTime=#12:00 am#

Visual Basic 2013 Lesson 9: Variables and Constants

In previous lesson, we have learned about data and various data types in Visual Basic 2013. Data can be stored as a variable or as a constant. Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In Visual Basic 2013, variables are areas allocated by the computer memory to hold data.

9.1 Variable Names

Like the mail boxes , each variable must be given a name. To name a variable in Visual Basic 2013, you have to follow a set of rules. The following are the rules when naming the variables in Visual Basic It must be less than 255 characters
No spacing is allowed
It must not begin with a number
Period is not permitted

Examples of valid and invalid variable names are displayed in Table 9.1

Table 9.1:

VALID NAMES	INVALID NAME
My_Computer	My.COmputer
Smartphone123	123Smartphone
Long_Name_Can_beUSE	LongName&Canbe&Use *& is not acceptable

9.2 Declaring Variables

In Visual Basic 2013, we have to declare the variables before using them by assigning names and data types. If you fail to do so, the program will show an error. Variables are usually declared in the general section of the code windows using the Dim statement.
The syntax is as follows:

Dim VariableName As Data Type

If you want to declare more variables, you can declare them in separate lines or you may also combine more in one line , separating each variable with a comma, as follows:

Dim VariableName1 As DataType1, VariableName2 As DataType2, VariableName3 As DataType3

Example 9.1

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.LoadDim password As String

Dim yourName As String

Dim firstnum As Integer

Dim secondnum As Integer

Dim total As Integer
```

Dim doDate As Date

End Sub

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

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

For string declaration, there are two possible forms, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same syntax as Example 9.1

Example 9.2

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim YourMessage As String
YourMessage = "Happy Birthday!"
MsgBox(YourMessage)
End SubWhen you run the program, a message box that shows the text "Happy Birthday!" will appear, as shown in Figure 9.1

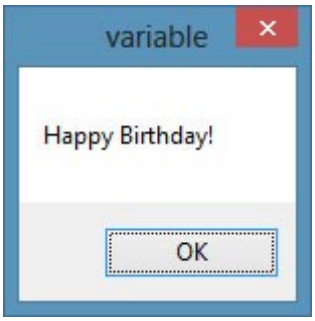


Figure 9.1

For the fixed-length string, you have to use the syntax as shown below:

Dim VariableName as String * n

where n defines the number of characters the string can hold.

Example 9.3

Dim yourName as String * 10

yourName can holds no more than 10 Characters.

9.3 Assigning Values to Variables

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

Variable=Expression

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

```
firstNumber=100
secondNumber=firstNumber-99
userName=”John Lyan”
userpass.Text = password
Label1.Visible = True
Command1.Visible = false
Label4.text = textbox1.Text
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber+ThirdNumber
MeanScore% = SumScores% / NumSubjects%
X=sqr (16)
TrimString= Ltrim (“ Visual Basic”, 4)
Num=Int(Rnd*6)+1
```

An error occurs when you try to assign a value to a variable of incompatible data type. For example, if you have declared a variable as an integer but you assigned a string value to it, an error occurred, as shown in Example 9.4:

Example 9.4

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim YourMessage As Integer
YourMessage = “Happy Birthday!”
MsgBox(YourMessage)
End Sub
```

When you run the program, the following error messages will appear in a dialog box, as shown in Figure 9.2

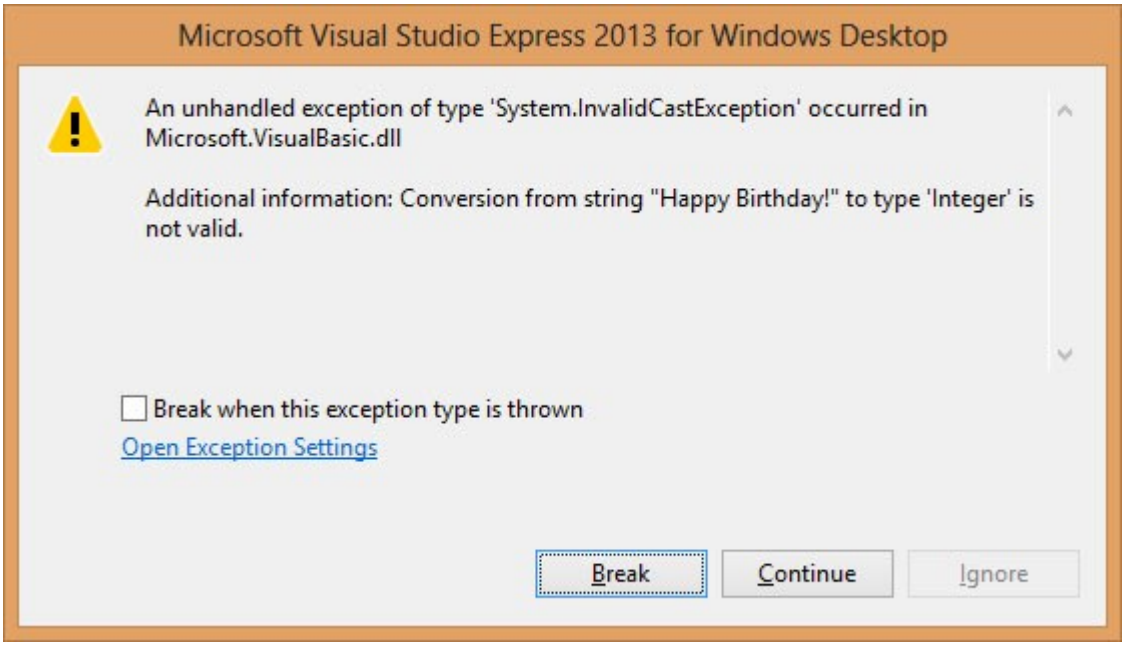


Figure 9.2

You can either break the program and continue to run the program.

9.4 Scope of Declaration

Other than using the Dim keyword to declare the data, you can also use other keywords to declare the data. Three other keywords are private ,static and public. The forms are as shown below:

Private VariableName as Datatype
Static VariableName as Datatype
Public VariableName as Datatype

The above keywords indicate the scope of declaration. **Private** declares a local variable, or a variable that is local to a procedure or module. However, Private is rarely used, we normally use Dim to declare a local variable. The **Static** keyword declares a variable that is being used multiple times, even after a procedure has been terminated. Most variables created inside a procedure are discarded by Visual Basic when the procedure is finished, static keyword preserve the value of a variable even after the procedure is terminated. **Public** is the keyword that declares a global variable, which means it can be used by all the procedures and modules of the whole program.

9.5 Declaring Constants

Constants are different from variables in the sense that their values do not change during the running of the program. The syntax to declare a constant is

Const Constant Name As Data Type = Value

Example 9.5

P Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

```
Const Pi As Single = 3.142
Dim R As Single = 10
Dim AreaCircle As Single
AreaCircle = Pi * R ^ 2

MsgBox("Area of circle with " & "radius" & R & "=" & AreaCircle)

End Sub
```

Press F5 to run the program and clicking the button will produce the following message:

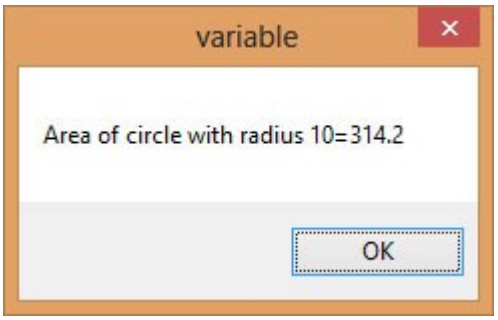


Figure 9.3

Visual Basic 2013 Lesson 10 : Arrays

Name(0,0)	Name(0,1)	Name(0,2)	Name(0,3)

10.1 Introduction to Arrays

In visual Basic 2013, an array is a group of variables (elements) with the same data type . When we work with a single item, we only need to use one variable. However, if we have a list of items which are of similar type to deal with, we need to declare an array of variables instead of using a variable for each item

For example, if we need to enter one hundred names, it is very tedious to declare one hundred different names, this is a waste of time and efforts. So, instead of declaring one hundred different variables, we need to declare only one array. We differentiate each item in the array by using subscript, the index value of each item, for example name(1), name(2),name(3) …….etc. , which will make declaring variables streamline and much more systematic.

10.2 Dimension of an Array

An array can be one dimensional or multidimensional. One dimensional array is like a list of items or a table that consists of one row of items or one column of items.

A two dimensional array is a table of items that is made up of rows and columns. The way to reference an element in a one dimensional array is ArrayName(x), where x is the index or position number of the element. The way to reference an element in a two dimensional array is ArrayName(x,y) , where (x,y) is the index or position number of the element. Usually it is sufficient to use one dimensional and two dimensional array ,you only need to use higher dimensional arrays if you need to deal with more complex problems. Let me illustrate the arrays with tables.

Table 10.1: One Dimensional Array

Student Name	Name(0)	Name(1)	Name(2)	Name(3)	Name(4)	Name(5)	Name(6)

Table 10.2: Two Dimensional Array

Name(1,0)	Name(1,1)	Name(1,2)	Name(1,3)
Name(2,0)	Name(2,1)	Name(2,2)	Name(2,3)
Name(3,0)	Name(3,1)	Name(3,2)	Name(3,3)

10.2 Declaring Arrays

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

The statement to declare a one dimensional array is as follows:

Dim arrayName(n) as dataType

where n indicates the last index in the array. Please note that n does not indicate the number of elements in the array, it is one less than the number of elements (n-1) because the first element is always the zeroth element. The first element is arrayName(0), the second element is arrayName(1), the third element is arrayName(2) and so on. The number of elements in an array is also known as length, we can retrieve the length of an array using the syntax **arrayName.length**

For example,

Dim CusName(10) as String

will declare an array that consists of 11 elements starting from CusName(0) through to CusName(10)

To find out the length of the array, you can write the following code:

Example 10.1

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Dim CusName(10) As String
MsgBox(CusName.Length)

End Sub
```

Running the program will produce a message box that displays the length of the array i.e 11, as shown in Figure 10.1

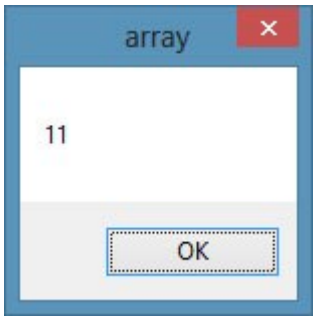


Figure 10.1

You might also declare an array with a non-zero starting index buy initialize an index vale other than zero, as follows:

Dim arrayname As DataType()

arrayName=New String(){1,2,3,….,n)

This array will consists of n elements, starting with arrayName(1)

Example 10.2

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Dim CusName As String()
CusName = New String() {1, 2, 3}
MsgBox(CusName.Length)

End Sub
```

The message box will display the length as 3

The statement to declare a two dimensional array is as follow:

Dim ArrayName(m,n) as dataType

where m and n indicate the last indices in the array. The number of elements or the length of the array is (m+1)x(n+1)

Example 10.3

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Dim CusName(5,6) As String
MsgBox(CusName.Length)

End Sub
```

Running the program and the message box will display a length of 42, as shown in Figure 10.2

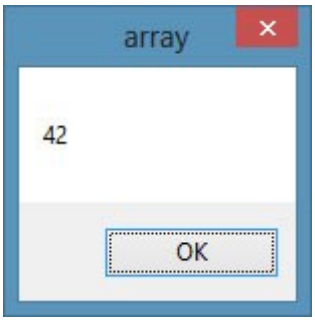


Figure 10.2

Example 10.4

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
Dim num As Integer
Dim CusName(5) As String

For num = 0 To 5
CusName(num) = InputBox("Enter the customer name", "Enter Name")
ListBox1.Items.Add(CusName(num))

Next
End Sub
```

This program will prompt the user to enter names in an input box for a total of 6 times and the names will be entered into a list box, as shown in Figure 10.3 and Figure 10.4

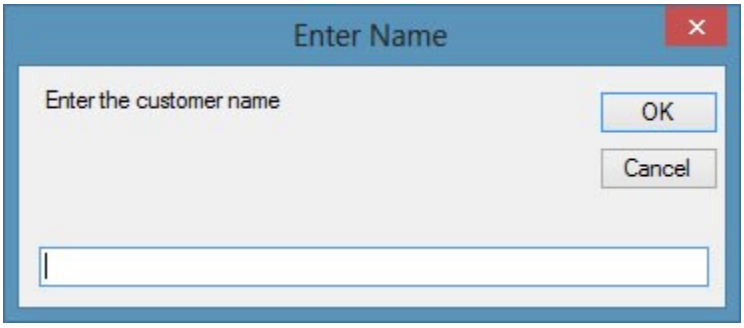


Figure 10.3

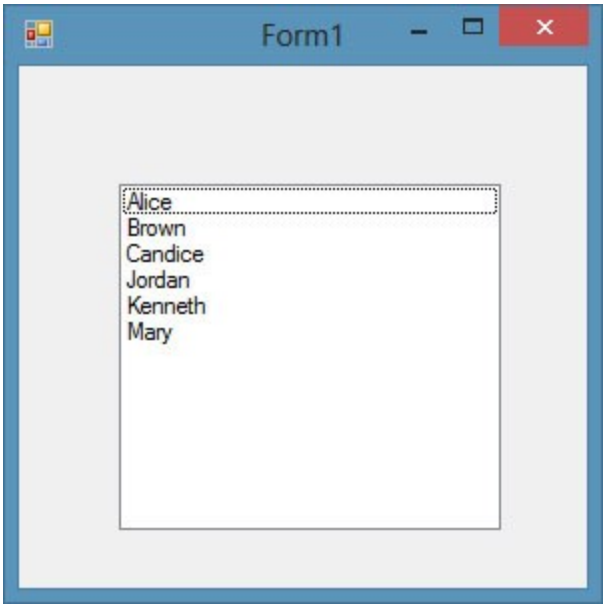


Figure 10.4

Visual Basic 2013 Lesson 11: Performing Mathematical Operations

Computer can perform mathematical calculations much faster than human beings do. However, computer itself will not be able to perform any mathematical calculations without receiving instructions from the user. In Visual Basic 2013, we can write code to instruct the computer to perform mathematical calculations such as addition, subtraction, multiplication, division and other kinds of mathematical operations. In order for Visual Basic 2013 to carry out arithmetic calculations, we need to write code that involve the use of various mathematical operators. The Visual Basic 2013 mathematical operators are very similar to the normal arithmetic operators, only with slight variations. The plus and minus operators are the same while the multiplication operator use the * symbol and the division operator use the / symbol. The list of Visual Basic 2013 mathematical operators are shown in table 11.1 below:

Table 11.1: Mathematical Operators

OPERATOR	MATHEMATICAL FUNCTION	EXAMPLE
+	Addition	1+2=3
-	Subtraction	10-4=6
^	Exponential	3^2=9
*	Multiplication	5*6=30
/	Division	21/7=3
Mod	Modulus(returns the remainder of an integer division)	15 Mod 4=3
\	Integer Division(discards the decimal places)	19/4=4

Example 11.1

In this program, you need to insert two text boxes, four labels and one button. Click the button and enter the code as shown below. When you run the program, it will perform the four basic arithmetic operations and displays the results on the four labels.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

  Dim num1, num2, difference, product, quotient As Single
  num1 = TextBox1.Text
```

```
num2 = TextBox2.Text
sum=num1+num2
difference=num1-num2
product = num1 * num2
quotient=num1/num2
Label1.Text=sum
Label2.Text=difference
Label3.Text = product
Label4.Text = quotient
End Sub
```

Example 11.2

The program can use Pythagoras Theorem to calculate the length of hypotenuse c given the length of the adjacent side a and the opposite side b. In case you have forgotten the formula for the Pythagoras Theorem, it is written as
c^2=a^2+b^2

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

Dim a, b, c As Single
a = TextBox1.Text
b = TextBox2.Text
c=(a^2+b^2)^(1/2)
Label3.Text=c

End Sub
```

Example 11.3: BMI Calculator

A lot of people are obese now and it could affect their health seriously . Obesity has proven by the medical experts to be a one of the main factors that brings many adverse medical problems, including the the cardiovascular disease. If your BMI is more than 30, you are considered obese. You can refer to the following range of BMI values for your weight status.

Underweight = <18.5
Normal weight = 18.5-24.9
Overweight = 25-29.9
Obesity = BMI of 30 or greater

In order to calculate your BMI, you do not have to consult your doctor, you can just use a calculator or a home made computer program, this is exactly what I am showing you here. The BMI calculator is a Visual Basic program that can calculate the body mass index, or BMI of a person based on the body weight in kilogram and the body height in meter. BMI can be calculated using the formula weight/(height)^2, where weight is measured in kg and height in

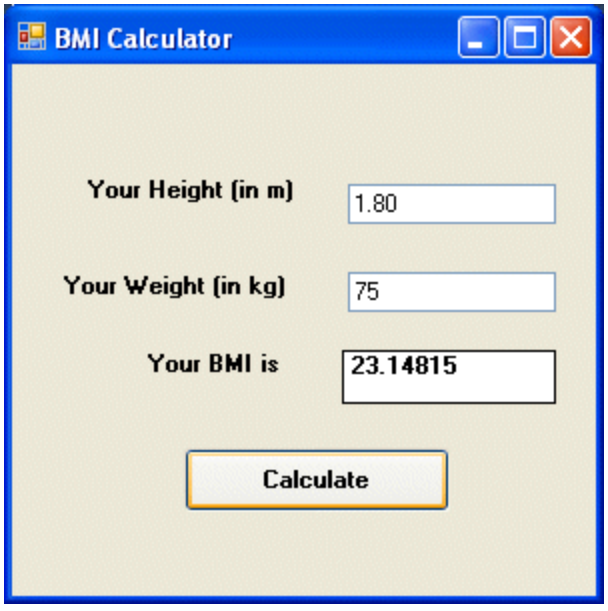
meter. If you only know your weight and height in lb and feet, then you need to convert them to the metric system .

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArsgs)
Handles Button1.Click

Dim height, weight, bmi As Single
height = TextBox1.Text
weight = TextBox2.Text
bmi = (weight) / (height ^ 2)
Label4.Text = bmi

End Sub
```

The output is shown in the Figure 11.1 below. In this example, your height is 1.80m(about 5 foot 11),your weight is 75 kg(about 168lb), and your BMI is about 23.14815. The reading suggests that you are healthy. (Note; 1 foot=0.3048, 1 lb=.45359237 kilogram)



rom the above examples, you can see that writing code that involve arithmetic operations is relatively easy. Here are more arithmetic projects you work on:

Area of a triangle
Area of a rectangle
Area of a circle
Volume of a cylinder
Volume of a cone
Volume of a sphere

Compound interest
Future value
Mean
Variance
Sum of angles in polygons
Conversion of lb to kg
Conversion of Fahrenheit to Celsius

Visual Basic 2013 Lesson 12: String Manipulation

In this lesson, we shall learn how to manipulate string. String manipulation means writing code to process characters like names, addresses, gender, cities, book titles, sentences, words, text alphanumeric characters (@,#,$,%,^,&,*, etc) and more. String manipulation is best demonstrated in the area of word processing which deals with text editing.

In Visual Basic 2013, a string is a single unit of data that made up of a series of characters that includes letters, digits, alphanumeric symbols and more. It is treated as the String data type and therefore it is non-numeric in nature which means it cannot be manipulated mathematically though it might consists of numbers.

12.1 String Manipulation Using + and & signs.

In Visual Basic 2013, strings can be manipulated using the & sign and the + sign, both perform the string concatenation which means combining two or more smaller strings into larger strings. For example, we can join “Visual” ,”Basic” and “2013″ into “Visual Basic 2013″ using “Visual”&”Basic” or “Visual “+”Basic”, as shown in the Examples below:

Example 12.1

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

  Dim text1, text2, text3, text4 As String
  text1 = “Visual”
  text2 = “Basic”
  text3=”2013″
  text4 = text1 + text2+text3
  MsgBox (text4)

   End Sub
```

The line text4=text1+ text2 + text3 can be replaced by text4=text1 & text2 &text3 and produces the same output. However, if one of the variables is declared as numeric data type, you cannot use the + sign, you can only use the & sign.

The output is shown in Figure 12.1:

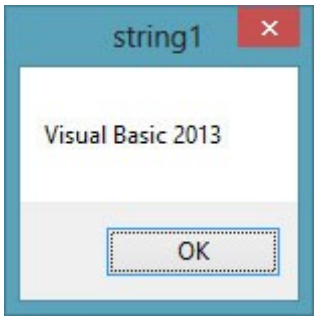


Figure 12.1

Example 12.2

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

Dim text1, text3 as string
Dim Text2 As Integer
text1 = “Visual”
text2=22
text3=text1+text2
MsgBox(text3)

End Sub
```

This code will produce an error because of data mismatch. The error message appears as follows:

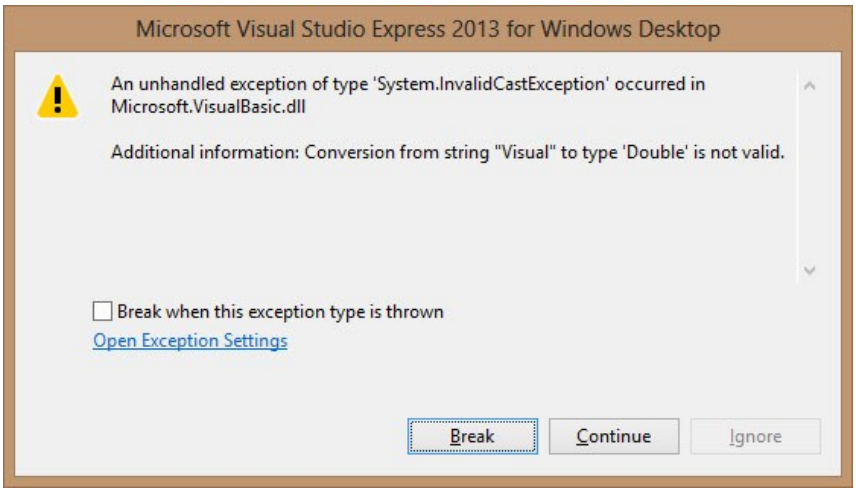


Figure 12.2

However, using & instead of + will be all right.

```
Dim text1, text3 as string
Dim Text2 As Integer
text1 = "Visual"
text2=22
text3=text1 & text2
MsgBox(text3)
```

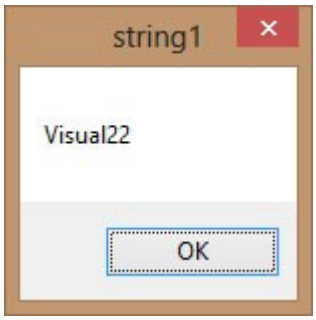


Figure 12.3

12.2 String Manipulation Using VB2013 Built-in Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input and return a value which is passed on to the main program to finish the execution. There are numerous string manipulation functions that are built into Visual Basic 2013.

12.2 (a) The Len Function

The Len function returns an integer value which is the length of a phrase or a sentence, including the empty spaces. The syntax is
Len ("Phrase")

For example,
Len (Visual Basic) = 12 and Len ("welcome to VB tutorial") = 22

Example 12.3

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

Dim MyText as String
MyText="Visual Basic 2013″
MsgBox(Len(MyText))
End Sub
```

The output:

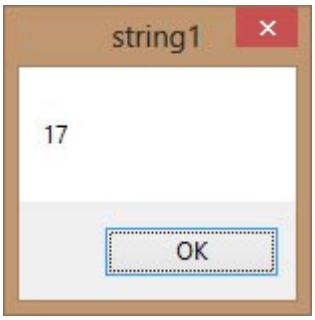


Figure 12.4

12.2(b) The Right Function

The Right function extracts the right portion of a phrase. The syntax for Visual Basic 6 is

Right ("Phrase", n)

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

Right("Visual Basic", 4) = asic

However, this syntax is not applicable in Visual Basic 2013. In VB2013, we need to use the following syntax

Microsoft.VisualBasic.Right("Phrase",n)

The reason of using the full reference is because many objects have the Right properties so using Right on its own will make it ambiguous to Visual Basic 2013.

Example 12.4

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim MyText As String

MyText="Visual Basic"

MsgBox(Microsoft.VisualBasic.Right(MyText, 4))

End Sub
```

The above program returns four right most characters of the phrase entered into the textbox.

The Output:

Figure 12.5

12.2(c)The Left Function

The Left function extract the left portion of a phrase. The syntax is

Microsoft.VisualBasic.Left("Phrase",n)

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

Microsoft.VisualBasic.Left ("Visual Basic", 4) = Visu .

12.2 (d) The Mid Function

The Mid function is used to retrieve a part of text form a given phrase. The syntax of the Mid Function is

Mid(phrase, position,n)

where

phrase is the string from which a part of text is to be retrieved.

position is the starting position of the phrase from which the retrieving process begins.

n is the number of characters to retrieve.

Example 12.5:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim myPhrase As String
myPhrase = InputBox("Enter your phrase")
```

```
LblPhrase.Text = myPhrase
LblExtract.Text = Mid(myPhrase, 2, 6)
End Sub
```

* In this example, when the user clicks the button, an input box will pop up prompting the user to enter a phrase. After a phrase is entered and the OK button is pressed, the label will show the extracted text starting from position 2 of the phrase and the number of characters extracted is 6.

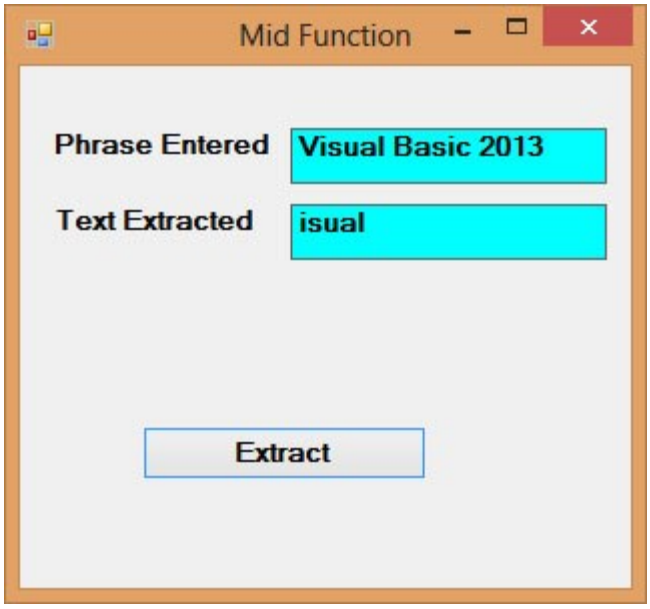


Figure 12.6

12.2(e) The Trim Function

The Trim function trims the empty spaces on both side of the phrase. The syntax is

Trim("Phrase")

.For example, Trim (" Visual Basic ") = Visual basic

Example 13.4

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim myPhrase As String
myPhrase = InputBox("Enter your phrase")
Label1.Text = Trim(myPhrase)
End Sub
```

12.2(f) The Ltrim Function

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

Ltrim("Phrase")

.For example,

Ltrim (" Visual Basic 2013″)= Visual basic 2013

12.2(g)The Rtrim Function

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

Rtrim("Phrase")

.For example,

Rtrim ("Visual Basic 2013 ") = Visual Basic 2013

12.2(h) The InStr function

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

Instr (n, original phase, embedded phrase)

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

Instr(1, "Visual Basic 2013 ","Basic")=8

*The function returns a numeric value.

You can write a program code as shown below:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

Label1.Text = InStr(1, "Visual Basic", "Basic")
End Sub
```

12.2(i) The Ucase and the Lcase Functions

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

The syntaxes are

Microsoft.VisualBasic.UCase(Phrase)

Microsoft.VisualBasic.LCase(Phrase)

For example,

Microsoft.VisualBasic.Ucase("Visual Basic") =VISUAL BASIC

Microsoft.VisualBasic.Lcase("Visual Basic") =visual basic

12.2(j)The Chr and the Asc functions

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

Chr(charcode)

and the format of the Asc function is

Asc(Character)

The following are some examples:

Chr(65)=A, Chr(122)=z, Chr(37)=% ,

Asc("B")=66, Asc("&")=38

* We shall learn more about functions in later lessons

Visual Basic 2013 Lesson 13: Making Decisions Using If…Then…Else

In the previous lessons, we have learned how to write code that accepts input from the user and displays the output without controlling the program flow. In this lesson, we shall learn how to write Visual Basic 2013 code that can make decisions and control the program flow in the process.

Decision making process is an important part of programming in Visual Basic 2012 because it can solve practical problems intelligently and provide useful output or feedback to the user. For example, we can write a Visual Basic 2013 program that can ask the computer to perform certain task until a certain condition is met, or a program that will reject non-numeric data. In order to control the program flow and to make decisions, we need to use the conditional operators and the logical operators together with the If..Then…Else control structure.

13.1 Conditional Operators

The conditional operators are powerful tools that resembles mathematical operators . These operators allow a Visual Basic 2013 program to compare data values and then decides what actions to take, whether to execute a program or terminate the program and more. They are also known as numerical comparison operators which are used to compare two values to see whether they are equal or one value is greater or less than the other value. The comparison will return a true or false result. These operators are shown in following table:

OPERATOR	DESCRIPTION
=	Equal to
>	Greater than
<	Less than
>=	Equal to or Greater than
<=	Less than or Equal to
<>	Not equal to

13.2 Logical Operators

Sometimes we might need to make more than one comparisons before a decision can be made and an action taken. In this case, using numerical comparison operators alone is not sufficient, we need to use additional operators, and they are the logical operators.

The logical operators are shown in the following table:

OPERATOR	DESCRIPTION
And	Both sides must be true
Or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates true

the above operators can be used to compare numerical data as well as non-numeric data such as text(string). In making strings comparison, there are certain rules to follows: Upper case letters are less than lowercase letters, "A"<"B"<"C"<"D".....<"Z" and number are less than letters.

13.3 Using the If control structure with the Comparison Operators

To effectively control the Visual Basic 2013 program flow, we shall use the If control structure together with the conditional operators and logical operators. There are basically three types of If control structures, namely If….Then statement, If….Then… Else statement and If….Then….ElseIf statement.

13.3(a) If….Then Statement

This is the simplest control structure which instructs the computer to perform a certain action specified by the Visual Basic 2013 expression if the condition is true. However, when the condition is false, no action will be performed. The syntax for the if…then.. statement is

If condition Then

Visual Basic 2013 expressions

End If

Example 13.1

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim myNumber As Integer
myNumber = TextBox1.Text
If myNumber > 100 Then
Label2.Text = " You win a lucky prize"
End If
End Sub
```

* When you run the program and enter a number that is greater than 100, you will see the "You win a lucky prize" statement. On the other hand, if the number entered is less than or equal to 100, you don't see any display.

13.3(b) If….Then…Else Statement

Using just If….Then statement is not very useful in programming and it does not provide choices for the users. In order to provide a choice, we can use the If….Then…Else Statement. This control structure will ask the computer to perform a certain action specified by the Visual Basic 2013 expression if the condition is met. And when the condition is false ,an alternative action will be executed. The syntax for the if…then.. Else statement is

Example 13.2

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim myNumber As Integer
myNumber = TextBox1.Text
If myNumber > 100 Then
Label2.Text = " Congratulation! You win a lucky prize"
Else
Label2.Text = " Sorry, You dif not win any prize"
End If
End Sub
```

* When you run the program and enter a number that is greater than 100, the statement "Congratulation! You win a lucky prize" will be shown. On the other hand, if the number entered is less than or equal to 100, you will see the "Sorry, You dif not win any prize" statement

Example 13.3

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim myNumber, MyAge As Integer
myNumber = TextBox1.Text
MyAge = TextBox2.Text

If myNumber > 100 And myAge > 60 Then
Label2.Text = " Congratulation! You win a lucky prize"
Else
Label2.Text = " Sorry, You did not win any prize"
End If

End Sub
```

* This program use the logical operator And beside the conditional operators. This means that both the conditions must be fulfilled in order for the conditions to be true, otherwise the second block of code will be executed. In this example, the number entered must be more than 100 and the age must be more than 60 in order to win a lucky prize, any one of the above conditions not fulfilled will disqualify the user from winning a prize.

13.3(c) If….Then…ElseIf Statement

If there are more than two alternative choices, using just If.…Then….Else statement will not be enough. In order to provide more choices, we can use the If.…Then…ElseIf Statement.The general structure for the if…then.. Else statement is

```
If condition Then

Visual Basic 2013 expression

ElseIf condition Then

Visual Basic 2013 expression

ElseIf condition Then

Visual Basic 2013 expression

.

.

Else
```

Visual Basic 2013 expression

End If

Example 13.4

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim Mark As Integer

Dim Grade as String

Mark = TextBox1.Text
If myNumber >=80 Then
Grade=”A”

ElseIf Mark>=60 and Mark=40 and Mark

Grade=”C”

Else

Grade=”D”

End If
End Sub
```

Visual Basic 2013 Lesson 14: Making Decisions using Select Case

In the previous lesson, we have learned how to control the program flow using the If…ElseIf control structure. In this lesson, you will learn how to use the **Select Case** control structure in Visual Basic 2013 to control the program flow . The Select Case control structure is slightly different from the If….ElseIf control structure . The difference is that the Select Case control structure basically only make decision on one expression or dimension (for example the examination grade) while the If …ElseIf statement control structure may evaluate only one expression, each If….ElseIf statement may also compute entirely different dimensions. Select Case is preferred when there exist multiple conditions as using If…Then..ElseIf statements will become too messy.

14.1 The Select Case…End Select Structure

The structure of the Select Case control structure in Visual Basic 2013 is as follows:

Select Case test expressionCase expression list 1
Block of one or more Visual Basic 2013 statements
Case expression list 2
Block of one or more Visual Basic 2013 Statements
Case expression list 3
.
.
Case Else
Block of one or more Visual Basic 2013 StatementsEnd Select

14.2 The usage of Select Case is shown in the following examples

Example 14.1: Examination Grades

```
Dim grade As String
Private Sub Compute_Click( )
grade=txtgrade.Text
Select Case grade
Case “A”
Label1.Text=”High Distinction”
Case “A-”
Label1.Text=”Distinction”
Case “B”
Label1.Text=”Credit”
Case “C”
Label1.Text=”Pass”
Case Else
Label1.Text=”Fail”

End Select
```

Example 14.2

In this example, you can use the keyword Is together with the comparison operators.Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click’Examination Marks

```
Dim mark As Single
mark = mrk.Text
Select Case mark
Case Is >= 85
Label1.Text= “Excellence”
Case Is >= 70
```

```
Label2.Text= “Good”
Case Is >= 60
Label3.Text = “Above Average”
Case Is >= 50
Label4.Text= “Average”
Case Else
Label5.Text = “Need to work harder”
End Select

End Sub
```

Example 14.3

Example 14.2 can be rewritten as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

‘Examination Marks

Dim mark As Single
mark = Textbox1.Text
Select Case mark

Case 0 to 49
Label1.Text = “Need to work harder”

Case 50 to 59
Label1.Text = “Average” s

Case 60 to 69
Label1.Text= “Above Average”

Case 70 to 84
Label1.Text = “Good”

Case 85 to 100
Label1.Text= “Excellence”

Case Else
Label1.Text= “Wrong entry, please reenter the mark”

End Select

End Sub
```

Example 14.4

Grades in high school are usually presented with a single capital letter such as A, B, C, D or E. The grades can be computed as follow:

```
 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

‘Examination Marks

Dim mark As Single

mark = TextBox1.Text

Select Case mark

Case 0 To 49
Label1.Text = “E”

Case 50 To 59

Label1.Text = “D”
Case 60 To 69

Label1.Text = “C”
Case 70 To 79

Label1.Text = “B”

Case 80 To 100
Label1.Text = “A”

Case Else
Label1.Text = “Error, please reenter the mark”

End Select

End Sub
```

The output:

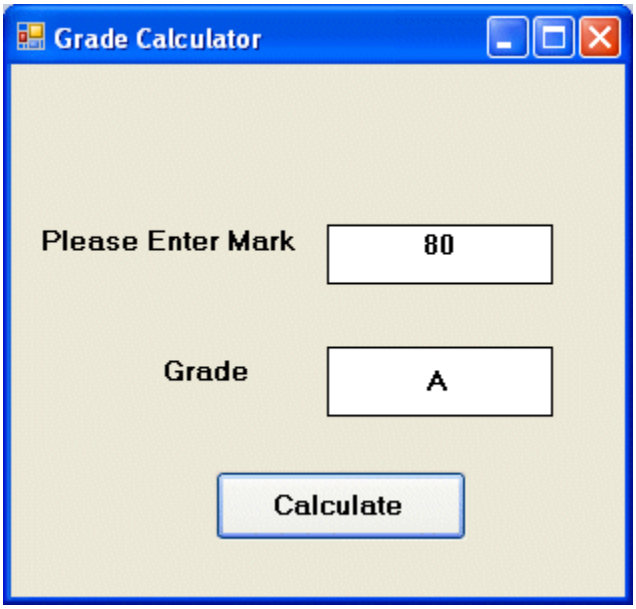


Figure 14.1

Visual Basic 2013 Lesson 15: Looping

We will often need to write code that does a job repeatedly until a certain condition is met, this process is called looping . Visual Basic 2013 allows a procedure to run repetitively as many times as long as the processor and memory could support. For example, we can design a program that adds a series of numbers until the sum exceeds a certain value, or a program that asks the user to enter data repeatedly until he or she enters the word 'Finish'. In Visual Basic 2013, there are three types of Loops, the **For…..Next** loop, the **Do** loop and the **While…..End While** loop

15.1 For….Next Loop

The most common loop in Visual Basic 2013 is the For….Next loop. The structure of a For…Next loop is as shown below:

For counter=startNumber **to** endNumber (**Step increment**)One or more Visual Basic 2013

statements

Next

To exit a For…..Next Loop, you can place the **Exit For** statement within the loop; and it is normally used together with the If….Then statement. For its application, you can refer to example 15.1 d.

Example 15.1 a

```
Dim counter as Integer

For counter=1 to 10
ListBox1.Items.Add (counter)
Next
```

* The program will enter number 1 to 10 into the list box.

Example 15.1b

```
Dim counter , sum As Integer

For counter=1 to 100 step 10
sum+=counter
ListBox1.Items.Add (sum)
Next
```

* The program will calculate the sum of the numbers as follows:

sum=0+10+20+30+40+……

Example 15.1c

```
Dim counter, sum As Integer

sum = 1000
For counter = 100 To 5 Step -5
sum – = counter
ListBox1.Items.Add(sum)
Next
```

*Notice that increment can be negative.

The program will compute the
subtraction as follow:
1000-100-95-90-…….

Example 15.1d

```
Dim n as Integer

For n=1 to 10
If n>6 then
Exit For
End If
Else
ListBox1.Items.Add ( n)
Next
End If
Next
```

The process will stop when n is greater than 6.

15.2 Do Loop

The Do Loop structures are

a) Do While condition
Block of one or more Visual Basic 2013 statements
Loop

b) Do
Block of one or more Visual Basic 2013 statements
Loop While condition

c) Do Until condition
Block of one or more Visual Basic 2012 statements
Loop

d) Do
Block of one or more Visual Basic 2012 statements
Loop Until condition

* Exiting the Loop

Sometime we need exit to exit a loop prematurely because a certain
condition is fulfilled. The syntax to use is **Exit Do**. Lets examine the following examples:

Example 15.2(a)

```
Do while counter <=1000

TextBox1.Text=counter
counter +=1

Loop
```

* The above example will keep on adding until counter >1000.

The above example can be rewritten as

```
Do
TextBox1.Text=counter
counter+=1
Loop until counter>1000
```

Example 15.2(b)

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

Dim sum, n As Integer

ListBox1.Items.Add("n" & vbTab & "Sum")
ListBox1.Items.Add("————————————-")
Do
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)
If n = 100 Then
Exit Do
End If
Loop

End Sub
```

* The loop in the above example can be replaced by the following loop:

```
Do Until n = 10
n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)

Loop
```

The output is as shown in Figure 15.1

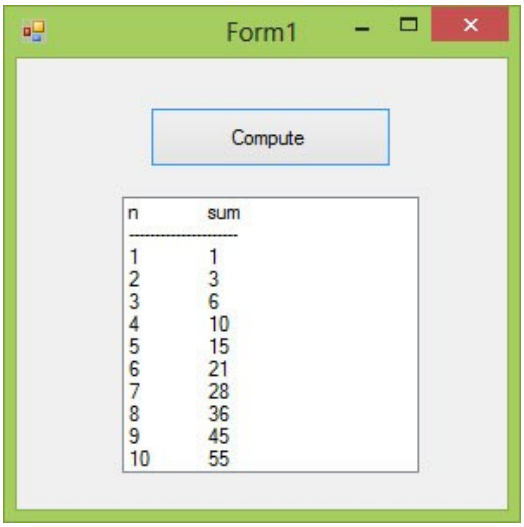


Figure 15.1

15.3 While….End While Loop

The structure of a While….End While Loop is very similar to the Do Loop. it takes the following form:

While conditions

Visual Basic 2013 statements

End While

Example 15.3

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim sum, n As Integer
ListBox1.Items.Add("n" & vbTab & "sum")
ListBox1.Items.Add("————————————-")

While n <> 10

n += 1
sum += n
ListBox1.Items.Add(n & vbTab & sum)

End While

End Sub
```

Visual Basic 2013 Lesson 16: Sub Procedures

In Visual Basic 2013, sub procedure is a procedure that performs a specific task and to return values, but it does not return a value associated with its name. However, it can return a value through a variable name. Sub procedures are usually used to accept input from the user, display information, print information, manipulate properties or perform some other tasks. It is a program code by itself and it is not an event procedure because it is not associated with a runtime procedure or a control such as button. It is called by the main program whenever it is required to perform a certain task.

Sub procedures help to make programs smaller and easier to manage. A sub procedure begins with a Sub keyword and ends with an End Sub keyword. The program structure of a sub procedure is as follows:

Sub ProcedureName (arguments)

Statements

End Sub

Example 16.1

In this example, we create a sub procedure sum to sum up two values that are specified as the arguments. The main program can reference a procedure by using its name together with the arguments in the parentheses.
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
sum(5, 6) End Sub Sub sum(a As Single, b As Single) MsgBox("sum="& a + b)

End Sub

Running the program produces a message box

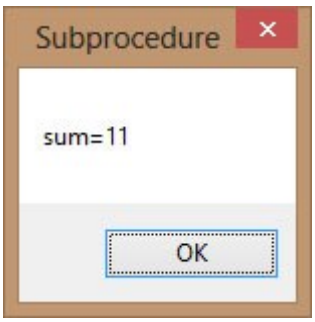


Figure 16.1

Example 16.2: Password Cracker

This is a passwords cracking program where it can generate possible passwords and compare each of them with the actual password; and if the generated password found to be equal to the actual password, login will be successful. In this program, a timer is inserted into the form and it is used to do a repetitive job of generating the passwords.

We create a passwords generating procedure **generate ()** and it is called by the e Timer1_Tick() event so that the procedure is repeated after every interval. The interval of the timer can be set in its properties window where a value of 1 is 1 millisecond, so a value of 1000 is 1 second; the smaller the value, the shorter the interval. However, do not set the timer to zero because if you do that, the timer will not start. We shall set the Timer's interval at 100 which is equivalent to 0.1 second. The Timer1.Enabled property is set to false so that the program will only start generating the passwords after you click on the Generate button. Rnd is a VB function that generates a random number between 0 and 1. Multiplying Rnd by 100 will obtain a number between 0 and 100. Int is a Visual Basic 2013 function that returns an integer by ignoring the decimal part of that number.

Therefore, Int(Rnd*100) will produce a number between 0 and 99, and the value of Int(Rnd*100)+100 will produce a number between 100 and 199.Finally, the program uses If…Then…Else to check whether the generated password is equal the actual password or not; and if they are equal, the passwords generating process will be terminated by setting the Timer1.Enabled property to false.

The Code

```
Public Class Form1
Dim password As Integer Dim crackpass As Integer
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Timer1.Enabled = True
End Sub
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
generate()
If crackpass = password Then
Timer1.Enabled = False
Label1.Text = crackpass
MsgBox("Password Cracked!Login Successful!")
Else Label1.Text = crackpass
Label2.Text = "Please wait…"
End If
End Sub
Sub generate()

crackpass = Int(Rnd() * 100) + 100

End Sub
```

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
password = 123
End Sub
End Class
```

The output

Figure 16.2: Password Generating Phase

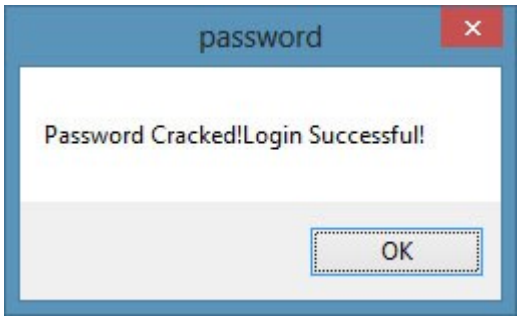


Figure 16.3: Message Showing Successful Login

Visual Basic 2013 Lesson 17: Creating Functions

A function is similar to a sub procedure in the sense that both are called by the main procedure to fulfill certain tasks. However, there is one difference, a function returns a value whilst a sub procedure does not. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers, or simply called user-defined functions.

17.1 Creating User-Defined Functions

To create a user define function in Visual Basic 2013, you can type the function procedure directly into the code window as follows:

Public Function functionName (Argument As dataType,…….) As dataType
or
Private Function functionName (Argument As dataType,…….) As dataType The keyword Public indicates that the function is applicable to the whole project and
the keyword Private indicates that the function is only applicable to a certain module or procedure. Argument is a parameter that can pass a value back to the function. You can include as many arguments as you can.

Example 17.1: BMI Calculator

This BMI calculator is a Visual Basic 2013 program that can calculate the body mass index, or BMI of a person based on the body weight in kilogram and the body height in meter. BMI can be calculated using the formula weight/(height)2, where weight is measured in kg and height in meter. If you only know your weight and height in lb and feet, then you need to convert them to the metric system.

If your BMI is more than 30, you are considered obese. You can refer to the following range of BMI values for your weight status.

- Underweight = <18.5
- Normal weight = 18.5-24.9
- Overweight = 25-29.9
- Obesity = BMI of 30 or greater

The Code

```
Public Class Form1

Private Function BMI(Height As Single, weight As Single) As Double
BMI = weight / Height ^ 2

End Function

Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles BtnCal.Click
```

```
Dim h As Single, w As Single
h = Val(TextBox1.Text)
w = Val(TextBox2.Text)
LblBMI.Text = BMI(h, w)
End Sub
End Class
```

The output

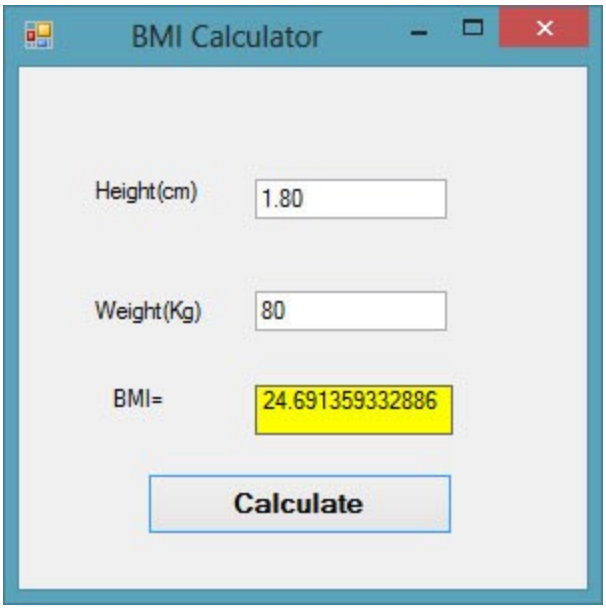


Figure 17.1

Example 17.2: Future Value Calculator

The concept of future value is related to time value of money. For example, if you deposit your money in a bank as a savings account or a fixed deposit account for a certain period of time, you will earn a certain amount of money based on the compound interest computed periodically, and this amount is added to the principal if you continue to keep the money in the bank. Interest for the following period is now computed based on the initial principal plus the interest (the amount which becomes your new principal). Subsequent interests are computed in the same way.

For example, let's say you deposited $1000 in a bank and the bank is paying you 5% compound interest annually. After the first year, you will earn an interest of $1000×0.05=$50 . Your new principal will be $1000+$1000×0.05=$1000(1+0.05)=$1000(1.05)=$1050.

After the second year, your new principal is $1000(1.05)x1.05=$1000(1.05)2 =$1102.50. This new principal is called the future value.

Following the above calculation, the future value after n years will be

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

Where PV represents the present value, FV represents the future value , i is the interest rate and n is the number of periods (Normally months or years).

The Code

```
Public Class Form1
Private Function FV(pv As Single, i As Single, n As Integer) As Double

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

End Function

Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles BtnCal.Click

Dim FutureVal As Single
Dim PresentVal As Single
Dim interest As Single
Dim period As Integer
PresentVal = TxtPV.Text

interest = TxtInt.Text
period = TxtN.Text

FutureVal = FV(PresentVal, interest, period)
LblFV.Text = Format(FutureVal, "$#,##0.00″)
End Sub
End Class
```

The Output

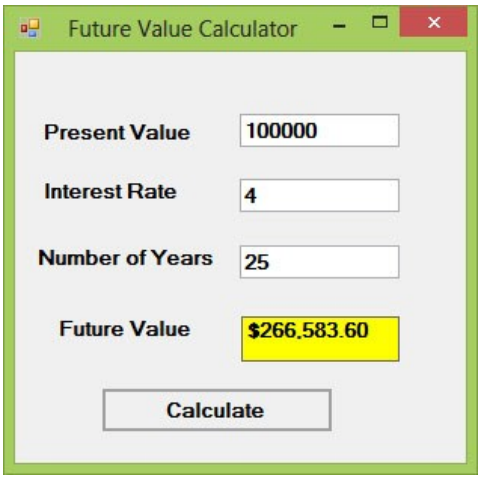


Figure 17.2

17.2 Passing Arguments by Value and by Reference

Functions can be called by value or called by reference. By default, the arguments in the function are passed by reference. If arguments are passed by reference, original data will be modified and no longer preserved. On the one hand, if arguments are passed by value, original data will be preserved. The keyword to pass arguments by reference is **ByRef** and the keyword to pass arguments by value is **ByVal.**

For example,

Private Function FV(ByVal pv As Single, ByRef i As Single, n As Integer) As Double

The function FV receives pv by value, i by reference and n by reference. Notice that although ByRef is not use to pass n, by default it is passed by reference.

Example 17.2(a)

In this example, we created two functions that compute the square root of a number , the first uses the keyword ByRef and the second uses the keyword ByVal.

The Code

```
Public Class Form1

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

End Sub
Private Function sqroot(ByRef x As Single) As Double
x = x ^ 0.5
sqroot = x
End Function
Private Function sqroot1(ByVal y As Single) As Double
y = y ^ 0.5
sqroot1 = y
End Function

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim u As Single
u = 9
MsgBox(3 * sqroot(u), , “ByRef”)
MsgBox(“Value of u is ” & u, , “ByRef”)

End Sub

Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
Dim u As Single
```

```
u = 9
MsgBox(3 * sqroot1(u), , “ByVal”)
MsgBox(“Value of u is ” & u, , “ByVal”)
End Sub
End Class
```

The Output

Case 1: Passing argument using ByRef

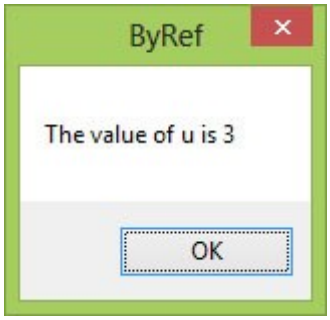


Figure 17.3

Notice that the value of u has been changed to 3

Case 2: Passing argument using ByVal

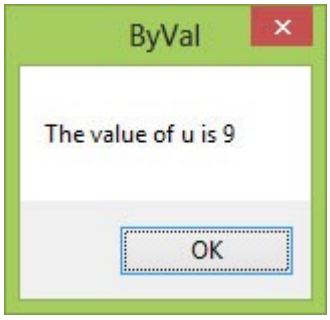


Figure 17.4

Visual Basic 2013 Lesson 18: The Math Functions

In lesson 11, we have learned how to write codes in Visual Basic 2013 that perform mathematical operations using standard mathematical operators. However, for more complex mathematical calculations, we need to use the built-in math functions in Visual Basic 2013. There are numerous built-in mathematical functions in Visual Basic 2013 which we shall introduce them one by one in this lesson.

18.1 The Abs function

The Abs function returns the absolute value of a given number.
The syntax is

Math. Abs (number)

* The Math keyword here indicates that the Abs function belong to the Math class. However, not all mathematical functions belong to the Math class.

Example 18.1

In this example, we shall add a text box control for the user to input his or her number and a label control to display the absolute value of the number. We need to use the Val function to convert text to numeric value. Rename the textbox as TxtNum and the label as LblAbs.

The Code

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles BtnComp.Click
LblAbs.Text = Math.Abs(Val(TxtNum.Text))
End Sub
```

The output

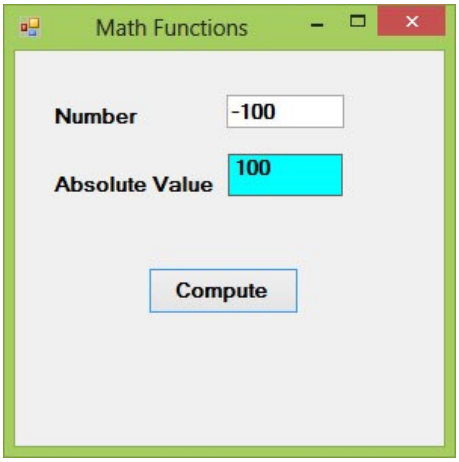


Figure 18.1

18.2 The Exp function

The Exp function returns the exponential value of a given number. For example, Exp(1)=e=2.71828182

The syntax is

Math.Exp (number)

Example 18.2

In this example, we shall add a text box control for the user to input his or her number and a label control to display the exponential value of the number. Rename the textbox as TxtNum and the label as LblAbs.

The Code

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles BtnComp.Click

LblExp.Text = Math.Exp(Val(TxtNum.Text))

End Sub
```

The Output

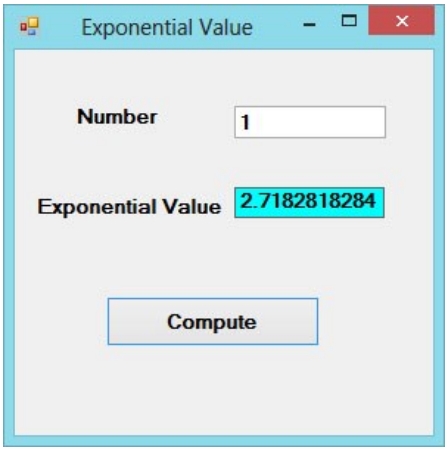


Figure 18.2

18.3 The Fix Function

The Fix function truncates the decimal part of a positive number and returns the largest integer smaller than the number. However, when the number is negative, it returns the smallest integer larger than the number. Fix does not belong to the Math class therefore we do not use the Math keyword.

Example 18.3

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles BtnComp.Click

LblFixNum1.Text = Fix(Val(TxtPosNum.Text))
LblFixNum2.Text = Fix(Val(TxtNegNum.Text))

End Sub
```

The Output

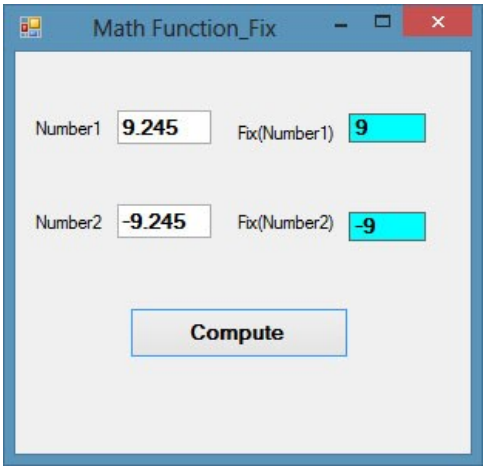


Figure 18.3

18.4 The Int Function

The Int is a function that converts a number into an integer by truncating its decimal part and the resulting integer is the largest integer that is smaller than he number. For example

Int(2.4)=2, Int(6.9)=6 , Int(-5.7)=-6, Int(-99.8)=-100

18.5 The Log Function

The Log function is the function that returns the natural logarithm of a number.

Example 18.4

```
Private Sub BtnComp_Click(sender As Object, e As EventArgs) Handles BtnComp.Click

LblLog.Text = Math.Log(Val(TxtNum.Text))

End Sub
```

The Output

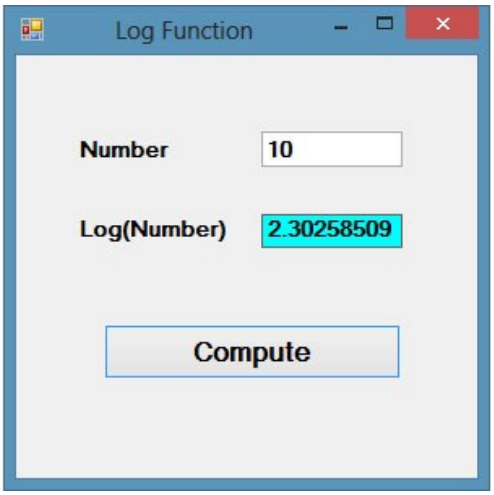


Figure 18.4

18.6 The Rnd( ) Function

Rnd is a very useful function in Visual Basic 2013 . We use the Rnd funciton to write code that involves chance and probability. The Rnd function returns a random value between 0 and 1. Random numbers in their original form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 integers ranging from 1 to 6, which makes the program behave like a virtual dice, we need to convert the random numbers to integers using the formula Int(Rnd*6)+1.

Example 18.5

```
Private Sub BtnGen_Click(sender As Object, e As EventArgs) Handles BtnGen.Click

LblRnd.Text = Int(VBMath.Rnd() * 6) + 1

End Sub
```

Notice that the Rnd() function belongs to the VBMath class in Visual Basic 2013. This is different from Visual Basic 2012, where you can omit the VBMath keyword.

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

The Output

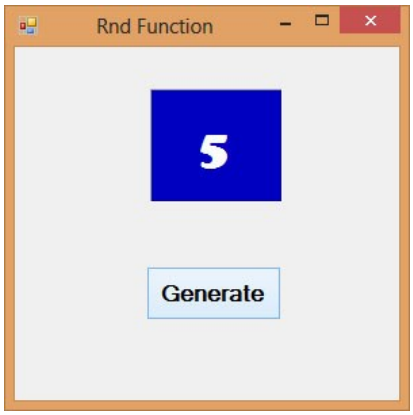


Figure 18.5

*We shall learn how to create an animated dice using a Timer control in later lesson

18.7 The Round Function

The Round function is the function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Math.Round (7.2567, 2) =7.26

Example 18.6

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

Label1.Text = Math.Round(Val(TextBox1.Text), 2)

End Sub
```

The Output

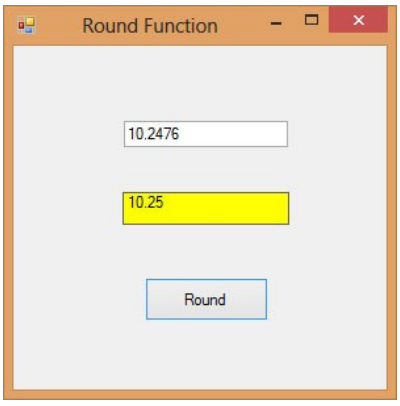


Figure 18.6

Visual Basic 2013 Lesson 19: The Format Function

The Format function in Visual Basic 2013 is a very useful formatting function. It is used to display numbers as well as date and time in various formats.

19.1 Format function for Numbers

There are two types of Format functions for numbers, one of them is the built-in or predefined format while another one can be defined by the user.

19.1(a) Built-in Format function for Numbers

The syntax of the built-in Format function is

Format (n, "style argument")

where n is a number.

The list of style arguments in Visual Basic 2013 is given in Table 19.1.

Table 19.1

STYLE ARGUMENT	EXPLANATION	EXAMPLE
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with	Format(6648972.265,

	separators or separators between thousands and rounds it up to two decimal places.	"Standard")= 6,648,972.27
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

Example 19.1

```
Private Sub BtnFormat_Click(sender As Object, e As EventArgs) Handles BtnFormat.Click

Label1.Text = Format(8972.234, "General Number")
Label2.Text = Format(8972.2, "Fixed")
Label3.Text = Format(6648972.265, "Standard")
Label4.Text = Format(6648972.265, "Currency")
Label5.Text = Format(0.56324, "Percent")

End Sub
```

The Output

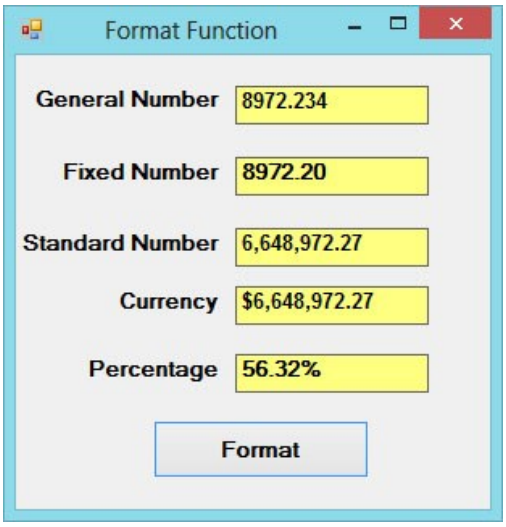


Figure 19.1

19.1(b) User-Defined Format

The syntax of the user-defined Format function is

Format (n, "user's format")

Although it is known as user-defined format, we still need to follows certain formatting styles. Examples of user-defined formatting style are listed in Table 19.2

Table 19.2

FORMAT	DESCRIPTION	OUTPUT
Format(781234.576,"0″)	Rounds to whole number without separators between thousands	781235
Format(781234.576,"0.0″)	Rounds to 1 decimal place without separators between thousands	781234.6

Format(781234.576,”0.00″)	Rounds to 2 decimal place without separators between thousands	781234.58
Format(781234.576,”#,##0.00″)	Rounds to 2 decimal place with separators between thousands	781,234.58
Format(781234.576,”$#,##0.00″)	Displays dollar sign and Rounds to 2 decimal place with separators between thousands	$781,234.58
Format(0.576,”0%”)	Converts to percentage form without decimal place	58%
Format(0.5768,”0%”)	Converts to percentage form with two decimal places	57.68%

Example 19.2

```
Private Sub BtnFormat_Click(sender As Object, e As EventArgs) Handles BtnFormat.Click

Label1.Text = Format(8972.234, "0.0")
Label2.Text = Format(8972.2345, "0.00")
Label3.Text = Format(6648972.265, "#,##0.00″)
Label4.Text = Format(6648972.265, "$#,##0.00″)
Label5.Text = Format(0.56324, "0%")

End Sub
```

The Output

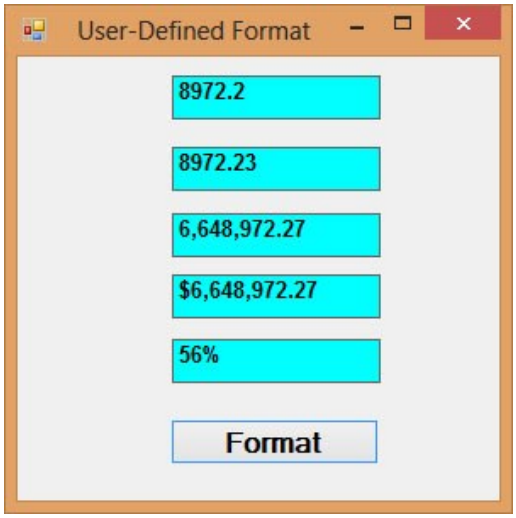


Figure 19.2

19.2 Formatting Date and Time

There are two types of Format functions for Date and time one of them is the built-in or predefined format while another one can be defined by the user.

19.2(a) Formatting Date and time using predefined formats

In Visual Basic 2013 , we can format date and time using predefined formats or user-defined formats. The predefined formats of date and time are shown in Table 19.3

Table 19.3

FORMAT	DESCRIPTION
Format(Now, “General Date”)	Displays current date and time
Format(Now, “Long Date”)	Displays current date in long format
Format (Now, “Short date”)	Displays current date in short format
Format (Now, “Long Time”)	Displays current time in long format.
Format (Now, “Short Time”)	Displays current time in short format.

Example 19.3

```
Private Sub BtnDisplay_Click(sender As Object, e As EventArgs) Handles BtnDisplay.Click

Label1.Text = Format(Now, “General Date”)
Label2.Text = Format(Now, “Long Date”)
Label3.Text = Format(Now, “short Date”)
Label4.Text = Format(Now, “Long Time”) Label5.Text = Format(Now, “Short Time”)

End Sub
```

The output

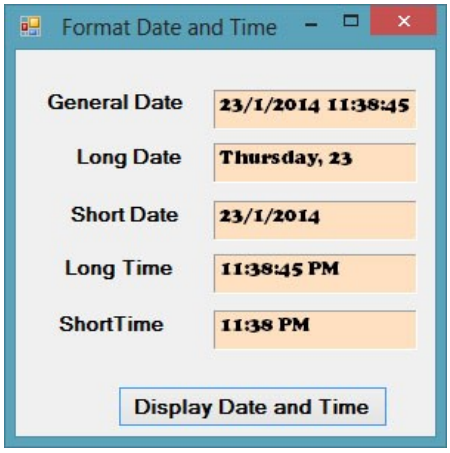


Figure 19.3

You can display dates and time in real-time using a timer and set its property Enabled to true and interval 100. The code is as follows:

Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick

Label1.Text = Format(Now, "General Date")
Label2.Text = Format(Now, "Long Date")
Label3.Text = Format(Now, "short Date")
Label4.Text = Format(Now, "Long Time")
Label5.Text = Format(Now, "Short Time")

End Sub

19.2(b) Formatting Date and time using user-defined formats

Beside using the predefined formats, you can also use the user-defined formatting functions. The syntax of a user-defined format for date and time is

Format (expression,style)

Table 19.4

FORMAT	DESCRIPTION
Format (Now, "m")	Displays current month and date
Format (Now, "mm")	Displays current month in double digits.

Format (Now, “mmm”)	Displays abbreviated name of the current month
Format (Now, “mmmm”)	Displays full name of the current month.
Format (Now, “dd/mm/yyyy”)	Displays current date in the day/month/year format.
Format (Now, “mmm,d,yyyy”)	Displays current date in the Month, Day, Year Format
Format (Now, “h:mm:ss tt”)	Dispalys current time in hour:minute:second format and show am/pm
Format (Now, “MM/dd/yyyy h:mm:ss)	Dispalys current date and time in hour:minute:second format

Example 19.4

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick

Label1.Text = Format(Now, “m”)
Label2.Text = Format(Now, “mm”)
Label3.Text = Format(Now, “mmm”)
Label4.Text = Format(Now, “mmmm”)
Label5.Text = Format(Now, “dd/mm/yyyy”)
Label6.Text = Format(Now, “mmm,d,yyyy”)
Label7.Text = Format(Now, “h:mm:ss tt”)
Label8.Text = Format(Now, “MM/dd/yyyy h:mm:ss tt”)

End Sub
```

The output

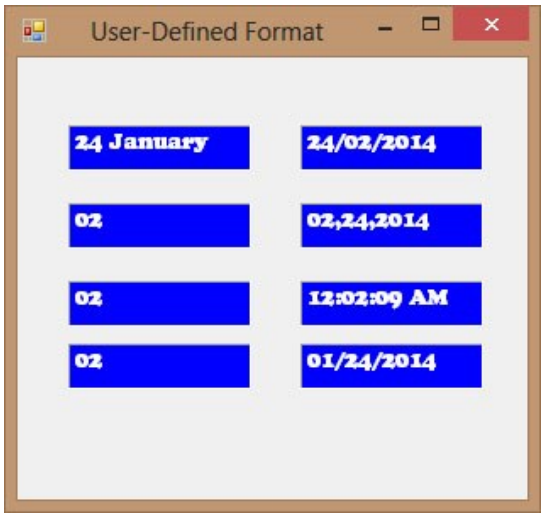


Figure 19.4

Visual Basic 2013 Lesson 20: Using Check Box

We have learned how to use some of the controls in Visual Basic 2013 in lesson 5, lesson 6 and lesson 7. In this lesson, we shall learn a very useful control in Visual Basic 2013, the check box. The Check box allows the user to select one or more items by checking the check box or check boxes concerned. For example, in the Font dialog box of any Microsoft Text editor like FrontPage, there are many check boxes under the Effects section such as that shown in the diagram below. The user can choose underline, subscript, small caps, superscript, blink and etc. In Visual Basic 2013, you may create a shopping cart where the user can click on check boxes that correspond to the items they intend to buy, and the total payment can be computed at the same time.

Example 20.1: Shopping Cart

In this example, we add a few labels, two buttons and six check boxes. We declare the price of each item using the Const keyword. If a check box is being ticked, its state is True else its state is False. To calculate the total amount of purchase, we use the mathematical operator +=. For example, sum+=BN is actually sum=sum+BN. Finally, we use the ToString method to display the the amount in currency.

The Code

```
Public Class Form1

Private Sub BtnCal_Click(sender As Object, e As EventArgs) Handles BtnCal.Click
Const LX As Integer = 100
Const BN As Integer = 500
Const SD As Integer = 200
Const HD As Integer = 80
Const HM As Integer = 300
Const AM As Integer = 150
Dim sum As Integer
If CheckBox1.Checked = True Then
sum += LX
End If

If CheckBox2.Checked = True Then
sum += BN
End If

If CheckBox3.Checked = True Then
sum += SD
End If
If CheckBox4.Checked = True Then
sum += HD
End If

If CheckBox5.Checked = True Then
sum += HM
End If

If CheckBox6.Checked = True Then
sum += AM
End If
LblTotal.Text = sum.ToString("c")
End Sub

Private Sub BtnReset_Click(sender As Object, e As EventArgs) Handles BtnReset.Click
CheckBox1.Checked = False
CheckBox2.Checked = False
CheckBox3.Checked = False
CheckBox4.Checked = False
CheckBox5.Checked = False
CheckBox6.Checked = False
```

End Sub
End Class

The Runtime Interface

Figure 20.1: Shopping Cart

Here is another example

Example 20.2

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

Const large As Integer = 10.0
Const medium As Integer = 8
Const small As Integer = 5
Dim sum As Integer

If CheckBox1.Checked = True Then
sum += large
End If

If CheckBox2.Checked = True Then
sum += medium
End If

If CheckBox3.Checked = True Then
sum += small
End If
Label5.Text = sum.ToString("c")

End Sub

Example 20.3

In this example, the text on the label can be formatting using the three check boxes that represent bold, italic and underline.

The Code

```
Public Class Form1

Private Sub ChkBold_CheckedChanged(sender As Object, e As EventArgs) Handles
ChkBold.CheckedChanged
If ChkBold.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or FontStyle.Bold)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not FontStyle.Bold)

End If
End Sub

Private Sub ChkItalic_CheckedChanged(sender As Object, e As EventArgs) Handles
ChkItalic.CheckedChanged
If ChkItalic.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or FontStyle.Italic)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not FontStyle.Italic)

End If
End Sub

Private Sub ChkUnder_CheckedChanged(sender As Object, e As EventArgs) Handles
ChkUnder.CheckedChanged
If ChkUnder.Checked Then
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or FontStyle.Underline)
Else
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not
FontStyle.Underline)

End If
End Sub
End Class
```

* The above program uses the CheckedChanged event to respond to the user selection by checking a particular check box, it is similar to the click event. The statement

```
LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style Or FontStyle.Italic)
```

will retain the original font type but change it to italic font style.

LblDisplay.Font = New Font(LblDisplay.Font, LblDisplay.Font.Style And Not FontStyle.Italic)

will also retain the original font type but change it to regular font style. (The other statements employ the same logic)

The Output

Figure 20.2