# Visual Basic 2017 Made Easy

*By*

*Dr.Liew*

# Disclaimer

Visual Basic 2017 Made Easy is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microsoft Corporation.

# Trademarks

Microsoft, Visual Basic, Excel and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks belong to their respective owners.

# Liability

The purpose of this book is to provide basic guides for people interested in Visual Basic 2017 programming. Although every effort and care has been taken to make The information as accurate as possible, the author shall not be liable for any error, Harm or damage arising from using the instructions given in this book.

## Acknowledgement

I would like to express my sincere gratitude to many people who have made their contributions in one way or another to the successful publication of this book.

My special thanks go to my children Xiang, Yi and Xun who have contributed their ideas and help in editing this book. I would also like to appreciate the support provided by my beloved wife Kim Huang and my youngest daughter Yuan. I would also like to thank the millions of readers who have visited my **Visual Basic Tutorial** website at **vbtutor.net** for their support and encouragement.

## About the Author

Dr. Liew Voon Kiong holds a bachelor's degree in Mathematics, a master's degree in Management and a doctorate in Business Administration. He has been involved in Visual Basic programming for more than 20 years. He created the popular online Visual Basic Tutorial at **www.vbtutor.net**, which has attracted millions of visitors since 1996. It has consistently been one of the highest ranked Visual Basic websites.

To provide more support for Visual Basic students, teachers, and hobbyists, Dr. Liew has written this book to complement the free Visual Basic 2017 tutorial with much more content. He is also the author of the Visual Basic Made Easy series, which includes **Visual Basic 6 Made Easy, Visual Basic 2008 Made Easy, Visual Basic 2010 Made Easy, Visual Basic 2013 Made Easy** and **Excel VBA Made Easy.** Dr. Liew's books have been used in high school and university computer science courses all over the world.

# Contents

Chapter 1 Introduction to Visual Basic 2017 11
1.1 A Brief Description of Visual Basic 2017 11
1.2 The Visual Studio 2017 IDE 13
1.3 Creating a New Project in Visual Studio 2017 14
Chapter 2 Designing the Interface 19
2.1 Customizing the Form 19
2.2 Adding Controls to the Form 24
Chapter 3 Writing the Code 27
3.1 The Concept of Event-Driven Programming 27
3.2 Writing the Code 29
Chapter 4 Working with Controls 31
4.1 TextBox 31
Example 4.1 Adding two numbers in two text boxes 31
4.2 Label 32
Example 4.2 Displaying output on a Label 33
4.3 List Box 34
4.3.1 Adding Items to the List Box 34
a) Adding items using the String Collection Editor 34
b) Adding Items using the Add() Method 35
Example 4.3 Adding an Item to a List Box 35
Example 4.4 Adding items to a List Box via an input box 36
Example 4.5 Creating Geometric Progression 37
4.3.2 Removing Items from a List Box 39
Example 4.6 Removing an item from a list box 39
Example 4.7 Deleting an item from a list box via an input box 40
Example 4.8 Removing a selected item from a list box 41
Example 4.9 Removing multiple selected items in a list box 41
Example 4.10 Removing all items in a list box using the Clear method 41
4.4 ComboBox 42
4.4.1 Adding Items to a combo box 42
4.4.2 Removing Items from a Combo box 46
Chapter 5 Handling Images 47
5.1 Loading an Image in a Picture Box 47

5.1.1 Loading an Image at Design Time 47
5.1.2 Loading an Image at Runtime 50
5.2 Loading an Image using Open File Dialog Control 50
Chapter 6 Working with Data 54
6.1 Data Types 54
6.1.1 Numeric Data Types 54
6.1.2 Non-numeric Data Types 55
6.1.3 Suffixes for Literals 56
6.2 Variables and Constants 56
6.2.1 Variable Names 56
6.2.2 Declaring Variables 57
Example 6.1 Declaring Variables using Dim 57
Example 6.2 Displaying Message using MsgBox 58
6.2.3 Assigning Values to Variables 58
Example 6.3 Incompatible Data Type 59
6.2.4 Scope of Declaration 60
6.2.5 Declaring Constants 61
Example 6.4 Calculating the Area of Triangle 61
Chapter 7 Array 62
7.1 Introduction to Arrays 62
7.2 Dimension of an Array 62
7.3 Declaring Arrays 63
Example 7.1 Find the Length of an Array 63
Example 7.2 Using the Length Property 64
Example 7.3 Find the Length of a Two-Dimensional Array 64
Example 7.4 Populating a List Box Involving an Array 65
Chapter 8 Performing Mathematical Operations 67
8.1 Mathematical Operators 67
8.2 Writing Code that Performs Mathematical Operations 68
Example 8.1 Standard Arithmetic Calculations 68
Example 8.2 Pythagorean Theorem 68
Example 8.3: BMI Calculator 69
Chapter 9 String Manipulation 71
9.1 String Manipulation Using + and & signs 71

Example 9.1 String Concatenation 71
Example 9.2 Data Mismatch 72
9.2 String Manipulation Using Built-in Functions 74
9.2.1 Len Function 74
Example 9.3 Finding the Length of a Phrase 74
9.2.2 Right Function 75
Example 9.4 Extracting the Right Portion of a Phrase 75
9.2.3 Left Function 75
9.2.4 Mid Function 76
Example 9.5 Retrieve Part of a Text Using Mid Function 76
Example 9.6 Extracting Text from a Phrase 76
9.2.5 Trim Function 77
Example 9.7 Trimming Both Side of a Phrase 77
9.2.6 Ltrim Function 77
9.2.7 The Rtrim Function 78
9.2.8 The InStr function 78
9.2.9 Ucase and the Lcase Functions 78
9.2.10 Chr and the Asc functions 78
Chapter 10 Using If…Then…Else 80
10.1 Conditional Operators 80
10.2 Logical Operators 81
10.3 Using If...Then...Else 81
10.3.1 If…Then Statement 81
Example 10.1 Lucky Draw 82
10.3.2 If...Then…Else Statement 82
Example 10.2 Lucky Draw Simulation 82
Example 10.3 Lucky Draw 84
10.3.3 If….Then…ElseIf Statement 85
Example 10.4 Grade Generator 86
Chapter 11 Using Select Case 88
Example 11.1: Examination Grades 88
Example 11.2 Using Case Is 89
Example 11.3 Select Case using a Range of Values 90
Example 11.4 Examination Grade 91

Chapter 12 Looping 93
12.1 For...Next Loop 93
Example 12.1 Creating a Counter 93
Example 12.2 Sum of Numbers 93
Example 12.3 Step-down For Next Loop 94
Example 12.4 Demonstrate Exit For 94
12.2 Do Loop 94
Example 12.5 Do While...Loop 95
Example 12.6 Summation of Numbers 95
12.3 While...End While Loop 96
Example 12.3 Demonstrating While...End While Loop 96
Chapter 13 Sub Procedures 98
13.1 What is a Sub Procedure 98
Example 13.1 A Sub Procedure that Adds Two Numbers 98
Example 13.2: Password Cracker 99
Chapter 14 Creating Functions 102
14.1 Creating User-Defined Functions 102
Example 14.1: BMI Calculator 102
Example 14.2 Future Value Calculator 104
14.2 Passing Arguments by Value and by Reference 105
Example 14.3 ByRef and ByVal 106
Chapter 15 Mathematical Functions 108
15.1 The Abs Function 108
Example 15.1 Compute Absolute Number 108
15.2 The Exp function 109
Example 15.2 Compute Exponential Value 109
15.3 The Fix Function 110
Example 15.3 Truncate Decimals using Fix 110
15.4 The Int Function 111
15.5 The Log Function 111
Example 15.4 Calculate Logarithm of a Number 111
15.6 The Rnd( ) Function 112
15.7 The Round Function 113
Example 15.6 Rounding a Number 113

Chapter 16 The Format Function 115
16.1 Format Function for Numbers 115
16.1.1 Built-in Format function for Numbers 115
Example 16.1 Formatting Numbers 116
16.1.2 User-Defined Format 116
Example 16.2 User-Defined Formats 117
16.2 Formatting Date and Time 118
16.2.1 Formatting Date and time using predefined formats 118
Example 16.3 Formating Date and Time 119
16.2.2 Formatting Date and time using user-defined formats 120
Example 16.4 Formatting Date and Time 120
Chapter 17 Using Checkbox and Radio Button 122
17.1 Check Box 122
Example 17.1: Shopping Cart 122
Example 17.2 Another Shopping Cart 124
Example 17.3 Formatting Text 124
17.2 Radio Button 126
Example 17.4 Shopping Cart 126
Example 17.2 Using Groupbox 128
Chapter 18 Errors Handling 130
18.1 Introduction 130
18.2 Using On Error GoTo Syntax 130
Example 18.1 Division Errors 131
18.3 Errors Handling using Try...Catch...End Try Structure 132
Example 18.2 Data Type Mismatch Error 133
Chapter 19 Object Oriented Programming 135
19.1 Concepts of Object-Oriented Programming 135
19.1.1 Encapsulation 135
19.1.2 Inheritance 135
19.1.3 Polymorphism 135
19.2 Creating Class 136
Example 19.1 BMI Calculator 137
Chapter 20 Creating Graphics 140
20.1 Introduction 140

20.2 Creating the Graphics Object 141
20.3 Creating the Pen Object 141
20.4 Drawing a Line 142
Example 20.1 Drawing a Straight Line 142
20.5 Drawing Lines that Connect Multiple Points 143
Example 20.2 Drawing Lines that Connect Multiple Points 144
20.6 Drawing a curve that Connect Multiple Points 145
Example 20.3 Drawing a Curve that Connect Multiple Points 145
20.7 Drawing Quadratic Curve 146
Example 20.4 Drawing a Quadratic Curve 146
20.8 Drawing Sine Curve 148
Example 20.5 Drawing a Sine Curve 149
20.9 Drawing a Rectangle 150
20.10 Customizing Line Style of the Pen Object 151
Example 20.6 Drawing a Rectangle with DashStyle 152
20.11 Drawing an Ellipse 153
Example 20.7 Drawing an Ellipse 154
Example 20.8 Drawing an Ellipse 156
20.12 Drawing a Circle 156
Example 20.9 Draw a Circle 156
20.13 Drawing Text 157
Example 20.10 Drawing Text 158
Example 20.11 Drawing Text Input by the User 160
20.14 Drawing Polygons 161
Example 20.12 Drawing a Triangle 162
Example 20.13 Drawing a Quadrilateral 163
20.15 Drawing a Pie 164
Example 20.14 Drawing a pie that sweeps clockwise through 60 degree. 164
20.16 Filling Shapes with Color 165
Example 20.15 Drawing and Filling a Rectangle with Color 165
Example 20.16 Drawing and Filling an Ellipse with Color 167
Example 20.17 Drawing and Filling a Polygon with Color 168
Example 20.18 Drawing and Filling a Pie 169
Chapter 21 Using Timer 171

Example 21.1 Creating a Digital Clock 171
Example 21.2 Creating a Stopwatch 172
Example 21.3 Creating a Digital Dice 174
Chapter 22 Creating Animation 176
Example 22.1 Creating Moving Object 176
Example 22.2 Creating an Animated Dice 177
Example 22.3 Creating a Slot Machine 180
Chapter 23 Working with Databases 184
23.1 Introduction to Database 184
23.2 Creating a Database Application 185
23.3 Creating Connection to a Database using ADO.NET 186
23.4 Populating Data in ADO.NET 194
Example 23.1 Creating a Database 196
23.5 Browsing Records 197
23.6 Editing, Saving, Adding and Deleting Records 198
Example 23.2 Browsing Records 199
23.7 Accessing Database using DataGridView 202
Example 23.3 Browsing Data Using DataGridView 203
23.8 Performing Arithmetic Calculations in a Database 204
Example 23.4 Performing Arithmetic Calculation 204
Example 23.5 Calculating Average 206
Example 23.6 Using SQL Count Function 207
Chapter 24 Reading and Writing Text Files 210
24.1 Introduction 210
24.2 Reading a Text File 210
24.3 Writing to a Text File 214
Chapter 25 Building Console Applications 217
Example 25.1: Displaying a Message 219
Example 25.2 Creating a Text File Writer in Console 220
Example 25.3 Creating a Text File Reader in Console 221
Example 25.4 Creating a Console App using If...Then....Else 222
Chapter 26 Creating Menu Bar and Toolbar 225
26.1 Creating Menu Items on the Menu Bar 225
26.2 Creating the Toolbar 233

Chapter 27 Deploying your VB 2017 Applications 239

Appendix 244

Index 245

# Chapter 1 Introduction to Visual Basic 2017

- A brief description of Visual Basic 2017
- Getting to know the Visual Basic 2017 Integrated Development Environment

## 1.1 A Brief Description of Visual Basic 2017

Visual Basic is a third-generation event-driven programming language first released by Microsoft in 1991. The final version of the classic Visual Basic was Visual Basic 6. Visual Basic 6 is a user-friendly programming language designed for beginners. Many developers still favor VB6 over its successor VB.NET.

In 2002, Microsoft released Visual Basic.NET(VB.NET) to replace Visual Basic 6. Thereafter, Microsoft declared VB6 a legacy programming language in 2008. However, Microsoft still provides some form of support for VB6. VB.NET is a fully object-oriented programming language implemented in the .NET Framework. It was created to cater for the development of the web as well as mobile applications.

Subsequently, Microsoft has released many versions of VB.NET. They are Visual Basic 2005, Visual Basic 2008, Visual Basic 2010, Visual Basic 2012, Visual Basic 2013, Visual Basic 2015 and Visual Basic 2017. Although the .NET portion was discarded in 2005, all versions of the Visual Basic programming language released since 2002 are regarded as VB.NET programming language

Visual Basic 2017 was released in 2017. It is bundled as a .NET desktop development component Visual Studio 2017. VB2017 can be used to build windows desktop applications using the .NET framework. Besides that, Visual Studio Community 2017 also comes with other Windows development tools that include Universal Windows Platform Development that creates applications for the Universal Windows Platform with C#, VB , JavaScript and C++. On top of that, it also includes Desktop Development with C++.

In addition, to cater for the increasing needs of web and cloud-based applications, VS2017 also provides the Web and Cloud development tools that include ASP.NET, Python, Azure SDK, Node.js, data storage and processing, data science and analytical applications as well as Office/SharePoint development. Furthermore, VS2017 also cater for the development of mobile applications by including the mobile and gaming tools like mobile development with .NET, game development with Unity, mobile development with JavaScript, mobile development with C++ and game development with C++. With the mobile development and gaming tools, you can build IOS and Android mobile apps and mobile games.

You can download Visual Studio 2017 from the following link:

https://visualstudio.microsoft.com/vs/older-downloads/

After downloading the file, run the VS2017RC community installer file. If you receive a User Account Control notice, click Yes. Next, it will ask you to acknowledge the Microsoft License Terms and the Microsoft Privacy Statement, as shown in Figure 1.1. Click Install to continue.

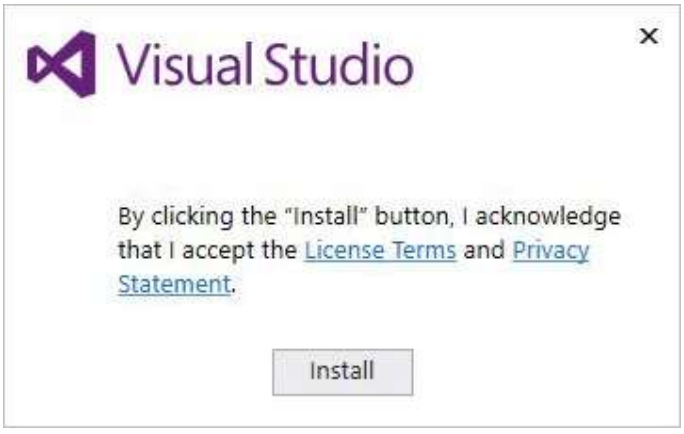


**Figure 1.1**

You will see several status screens that show the progress of the installation. After installation completed, you can select the feature set that you want, as shown in Figure 1.2. Since we are keen on developing Visual Basic 2017 desktop app, we will select the .NET

desktop development component. Besides that, you might want to install a language by clicking the Language packs. After making your selections, click install.

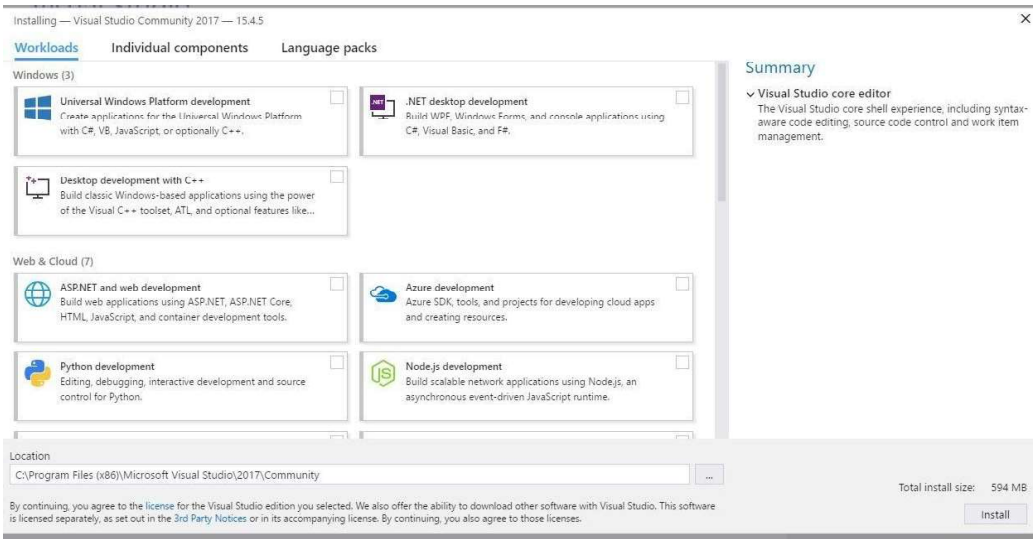


**Figure 1.2**

Upon completion of the installation, you can start programming in Visual Basic 2017.

## 1.2 The Visual Studio 2017 IDE

When you launch Microsoft Visual Studio 2017, you will be presented with the Start Page of Microsoft VS 2017, as shown in Figure 1.3

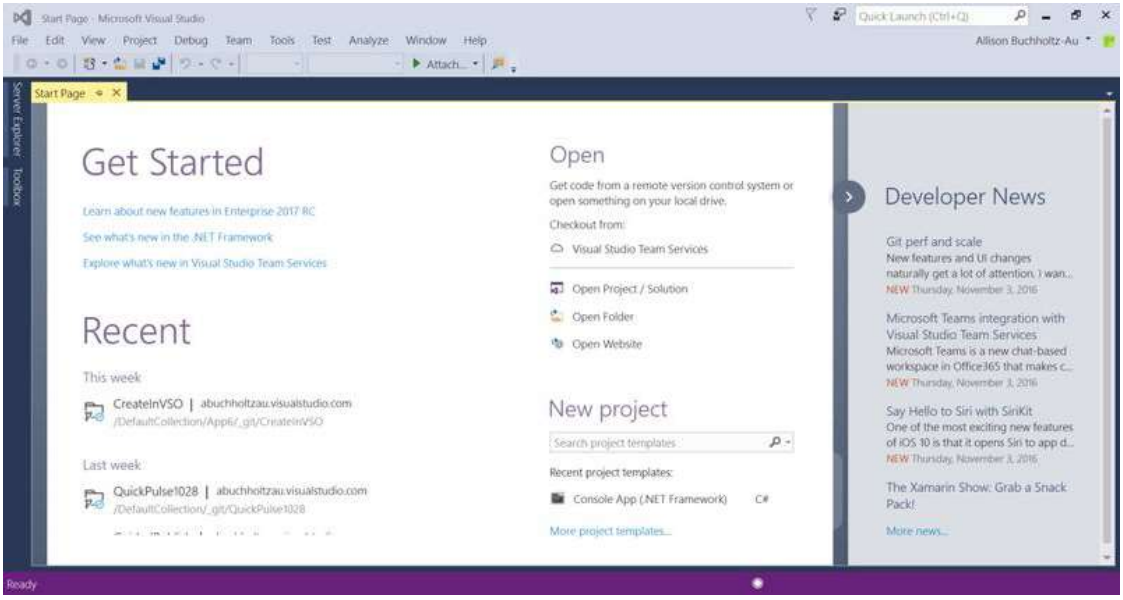


**Figure 1.3 Visual Studio 2017 IDE**

The Visual Studio 2017 start page comprises a few sections, the Get Started section, the Recent section, the Open section, the New project section and the Developers News section. In the start page, you can either start a new project, open a project or open a recent project. Besides that, you can also check for the latest news in Visual Studio 2017 for Windows Desktop. The Start Page also consists of a menu bar and a toolbar where you can perform various tasks by clicking the menu items.

## 1.3 Creating a New Project in Visual Studio 2017

To start a new Visual Studio 2017 project, click on New Project under the Start section to launch the Visual Studio 2017 New Project page as shown in Figure 1.4. You can also choose to open a recent project:

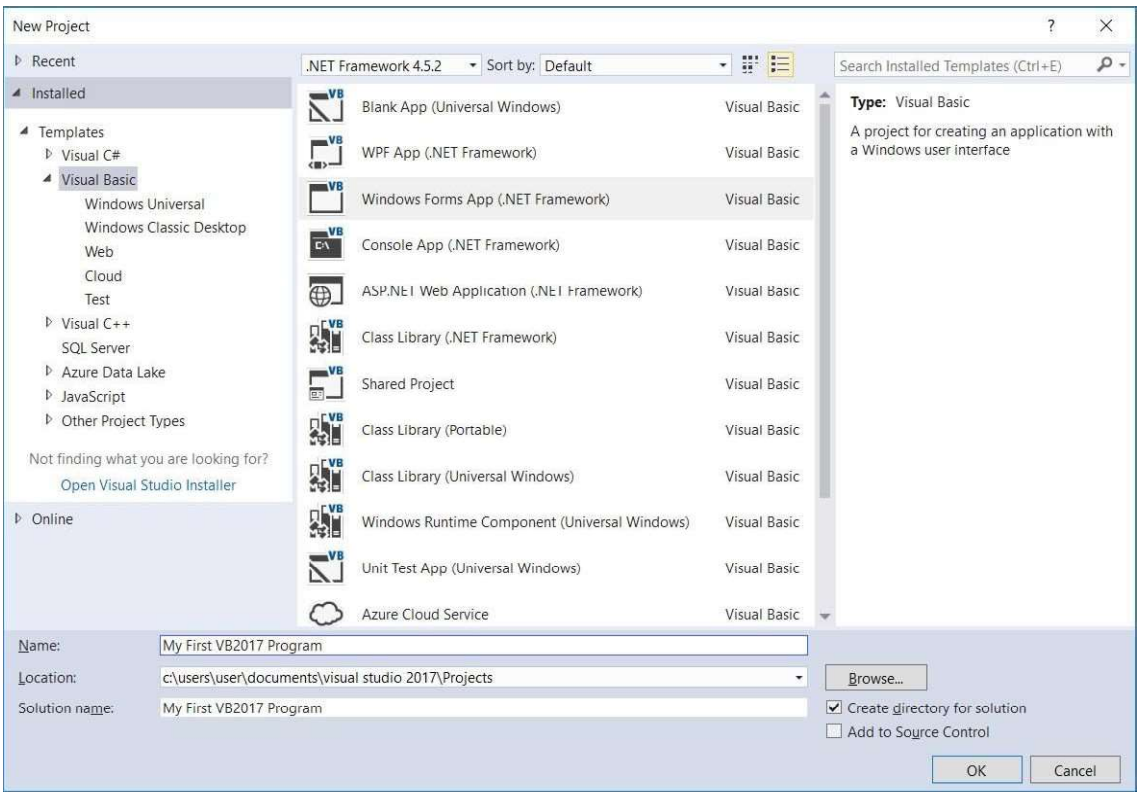


**Figure 1.4 Visual Studio 2017 New Project Page**

The New Project Page comprises a few templates, among them are Visual Basic, Visual C# and Visual C++. Since we are only learning Visual Basic 2017, we shall select Visual Basic. Visual Basic 2017 offers you several types of projects that you can create; they are *Blank Apps*, *Windows Forms APP(.NET Framework), WPF App(.NET Framework), Console App(.NET Framework) ,Class Library(.NET Framework), Shared Project* and more. Since we are only learning how to create windows desktop applications, we shall select Windows Forms App.

At the bottom of this dialog box, you can change the default project name WindowsApplication1 to some other name you like, for example, My First Visual Basic 2017 Program. After you have renamed the project, click OK to continue. The Visual Basic 2017 IDE Windows will appear, as shown in Figure 1.5. Visual Basic 2017 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 2017 programs.

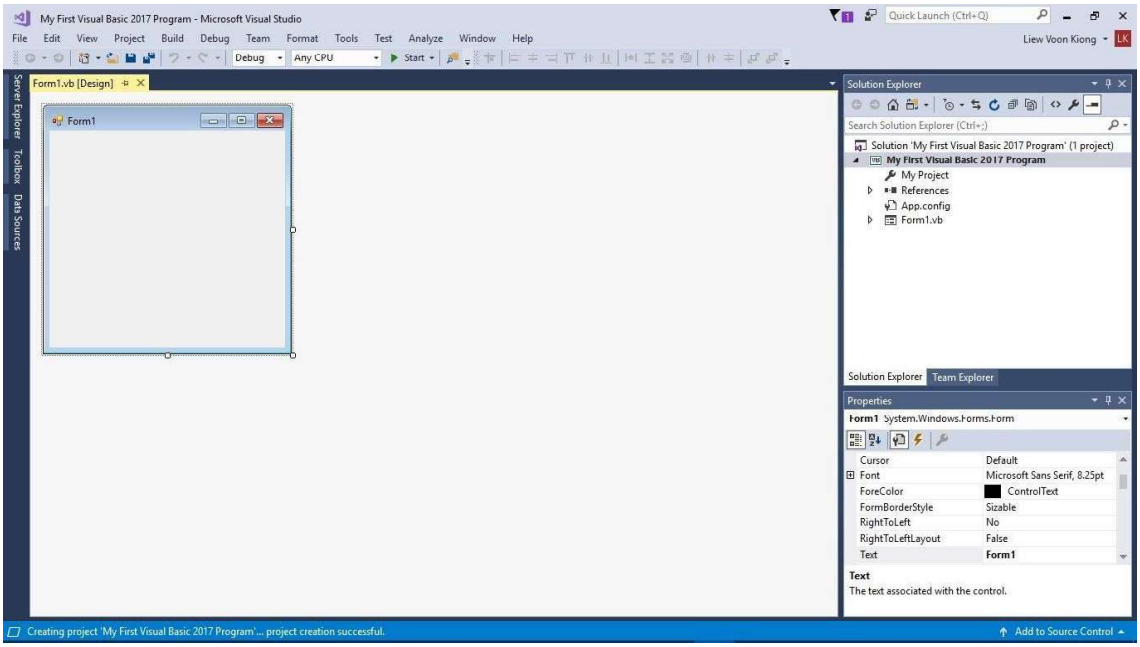


**Figure 1.5 The Visual Basic 2017 Express IDE**

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+Alt+x, the common controls Toolbox will appear, as shown in Figure 1.6. You can drag and move your toolbox around and dock it to the right, left ,top or bottom of the IDE.

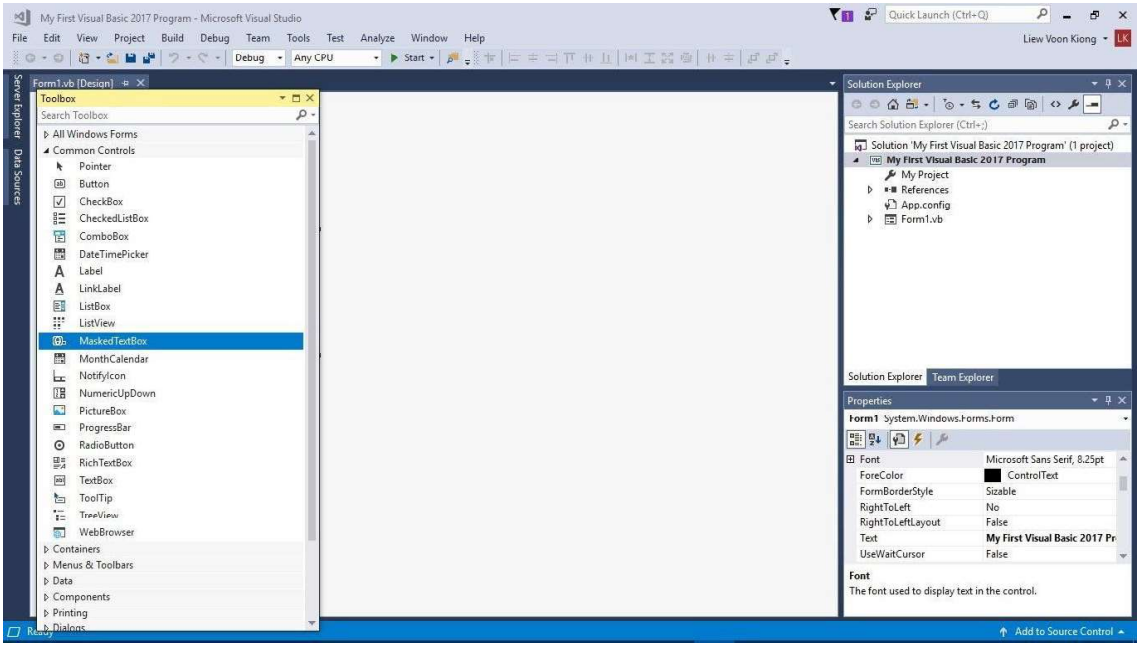

**Figure 1.6 Visual Basic 2017 Express Toolbox**

Next, we shall proceed to show you how to create your first program. First, change the text of the form to 'My First Visual Basic 2017 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 1.7

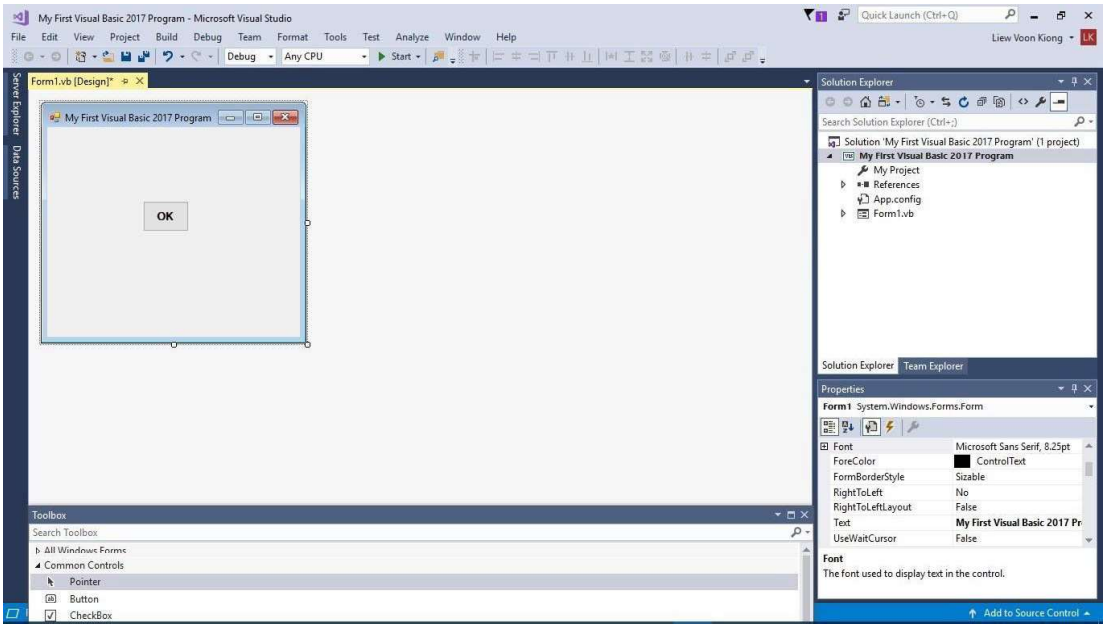


**Figure 1.7 The Design Interface**

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 2017 Program")
```

Now click on the Start button 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 2017 Program" message will appear, as shown in Figure 1.8. The function **`MsgBox`** is a built-in function of Visual Basic 2017 which can display the text enclosed within the brackets

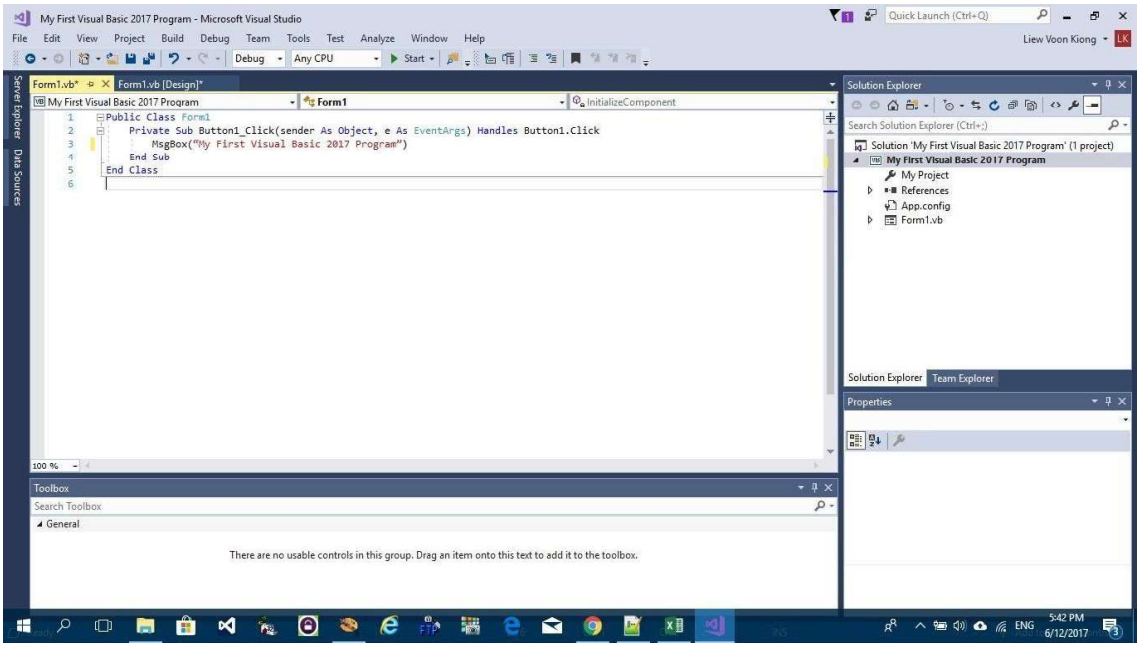


**Figure 1.8 Visual Basic 2017 Code Window**


**Figure 1.9 The Runtime Interface**

**Summary**

- In section 1.1, you have learned about the history of Visual Basic 2017
- In section 1.2, you have learned how to install and launch Visual Basic Studio 2017
- In section 1.3, you have learned how to launch the new project dialog and the Visual Basic 2017 IDE. You have also learned how to write your first program.

# Chapter 2 Designing the Interface

- Customizing
- Adding controls
- Setting Control Properties

Since Visual Basic 2017 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 customize the default form by changing its properties at design phase and at runtime. After customizing the default form, you may proceed to add controls from the toolbox to the form and then customize their properties.

## 2.1 Customizing the Form

When you start a new Visual Basic 2017 project, the VS2017 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. The name of the default form is Form1. The properties window displays all the properties related to Form1 and their corresponding attributes or values. You can change the name of the form, the title of the form using the text property, the background color, the foreground color, the size and more. Try changing the properties in Table 1.

**Table 1**

| ***Property*** | ***Value*** |
|---|---|
| Name | MyForm |
| Text | My First Visual Basic 2017 Program |
| BackColor | Aqua |
| ForeColor | DarkBlue |
| MaximizeBox | False |

In fact, you do not have to type in the color manually, you may select a color from the color drop-down list that comprises three tabs, Custom, Web, and System, as shown in Figure 2.1. Clicking on the drop-down arrow will bring out a color palette or a list of color rectangles where you can select a color.


**Figure 2.1**

Another method of setting the colors is to manually type in the RGB color code or the hex color code. The values of R, G and B range from 0 to 255, by varying the values of the RGB we can obtain different colors. For example, an RGB value of 128, 255, 255 yields the cyan color.

On the other hand, the hex color code system use a six-digit, three-byte hexadecimal number to represent colors. The bytes represent the red, green and blue components of the color. One byte represents a number in the range 00 to ff (in hexadecimal notation), or 0 to 255 in decimal notation. For example, **`#0000ff`** represents the cyan color. However, when you type in the hex color code in the properties window of VS2017, it automatically converts the color to RGB color or the color name. Figure 2.2 shows a list of Hex color codes and the corresponding colors.

| color | code | color | code | color | code | color | code | color | code |
|---|---|---|---|---|---|---|---|---|---|
| | eeeeee | | ffffcc | | ffccff | | ff99ff | | ff66ff |
| | dddddd | | ffff99 | | ffcccc | | ff99cc | | ff66cc |
| | cccccc | | ffff66 | | ffcc99 | | ff9999 | | ff6666 |
| | bbbbbb | | ffff33 | | ffcc66 | | ff9966 | | ff6633 |
| | aaaaaa | | ffff00 | | ffcc33 | | ff9933 | | ff6633 |
| | 999999 | | ccffff | | ffcc00 | | ff9900 | | ff6600 |
| | 888888 | | ccffcc | | ccccff | | cc99ff | | cc66ff |
| | 777777 | | ccff99 | | cccccc | | cc99cc | | cc66cc |
| | 666666 | | ccff66 | | cccc99 | | cc9999 | | cc6699 |
| | 555555 | | ccff33 | | cccc66 | | cc9966 | | cc6666 |
| | 444444 | | ccff00 | | cccc33 | | cc9933 | | cc6633 |
| | 333333 | | 99ffff | | cccc00 | | cc9900 | | cc6600 |
| | 222222 | | 99ffcc | | 99ccff | | 9999ff | | 9966ff |
| | 111111 | | 99ff99 | | 99cccc | | 9999cc | | 9966cc |
| | ff0000 | | 99ff66 | | 99cc99 | | 999966 | | 996699 |
| | ee0000 | | 99ff33 | | 99cc33 | | 999933 | | 996633 |
| | cc0000 | | 66ffff | | 66ccff | | 6699ff | | 6666ff |
| | 33ffff | | 00ffff | | 00ccff | | 3399ff | | 3366ff |
| | ff00ff | | cc00ff | | 00ee00 | | 0000ff | | 6600ff |

**Figure 2.2 Hex Color Codes**

The design interface is shown in Figure 2.2 and the runtime interface is shown in Figure 2.4. In the runtime interface, notice that the title has been changed from Form1 to My First Visual Basic 2017 Program, background color changed to aqua , the text OK color has been changed to dark blue and the window cannot be maximized.

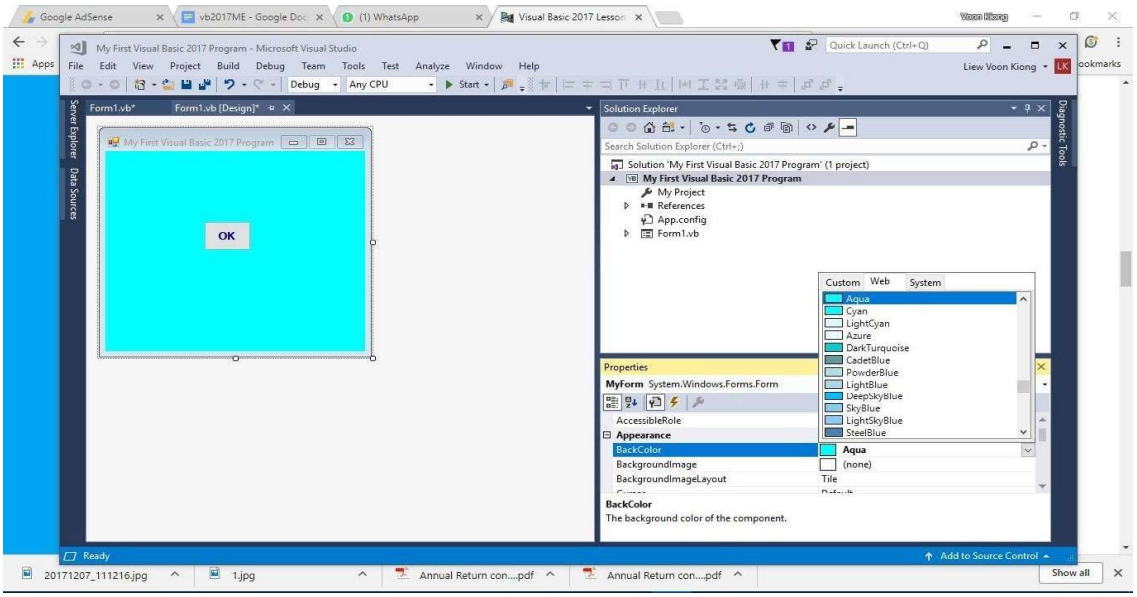

**Figure 2.3**

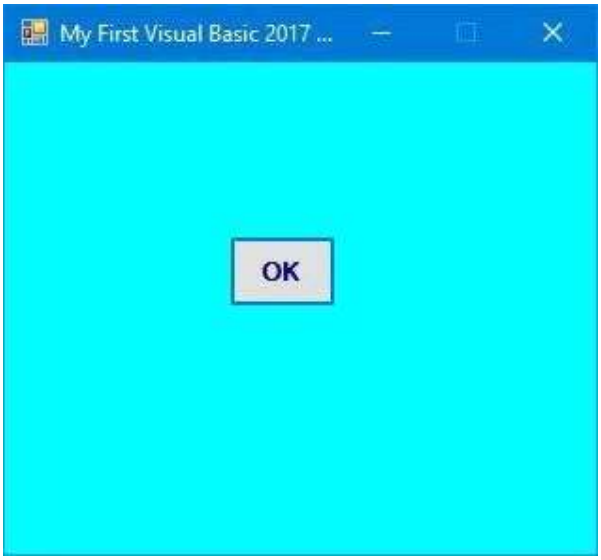


**Figure 2.4**

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

```
ObjectName.property
```

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

```
Me.BackColor=Color.Blue
```

In addition, you can also use the **FromArgb** method to specify the color using the RGB codes, as follows:

```
Me.BackColor = Color.FromArgb(0, 255, 0)
```

To achieve the same interface as shown in Figure 2.4, use following code :

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Me.Text = "My First Visual Basic 2017 Program"
Me.BackColor = Color.Cyan
```

```
Me.MaximizeBox = False
Me.MinimizeBox = True
End Sub
```

Instead of using the color name cyan, you can use RGB code, as follows:

```
Me.BackColor = Color.FromArgb(0,255,255)
```

Press F5 to run the program and you will get the exact interface as that shown in Figure 2.4. In addition, you can also specify the size, the opacity and the position of the default form using the code, as follows:

```
Private Sub Form1_Load(sender As Object, e As EventArgs Handles
MyBase.Load
Me.Text = "My First VB2017 Project"
Me.BackColor =Color.Beige
Me.MaximizeBox = False
Me.MinimizeBox = True
Me.Size = New Size(400, 400)
Me.Opacity = 0.85
Me.CenterToParent()
End Sub
```

The runtime interface is as shown in Figure 2.5

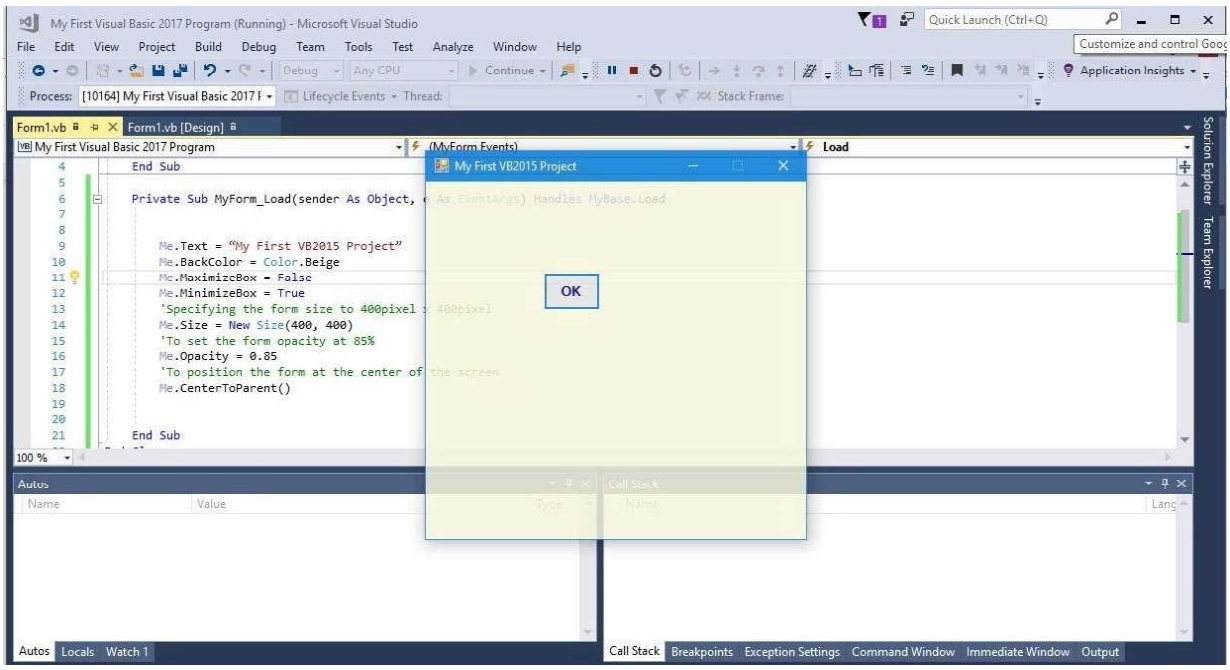


**Figure 2.5**

## 2.2 Adding Controls to the Form

In section 2.1, we have learned how to build an initial interface in Visual Basic 2017 by customizing the default form. Next, we shall continue to build the interface by adding some controls to the form. The controls are objects that consist of three elements, i.e. properties, methods, and events. Controls can be added to the form from the Toolbox. Among the controls, the most common ones are the button, label, textbox, list box, combo box, picture box, checkbox, radio button 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.

The Toolbox is usually hidden when you start Visual Basic 2017 IDE, you need to click View on the menu bar and then select Toolbox to reveal the Toolbox, as shown in Figure 2.6. You can also use shortcut keys Ctrl+w+x to bring out the toolbox.

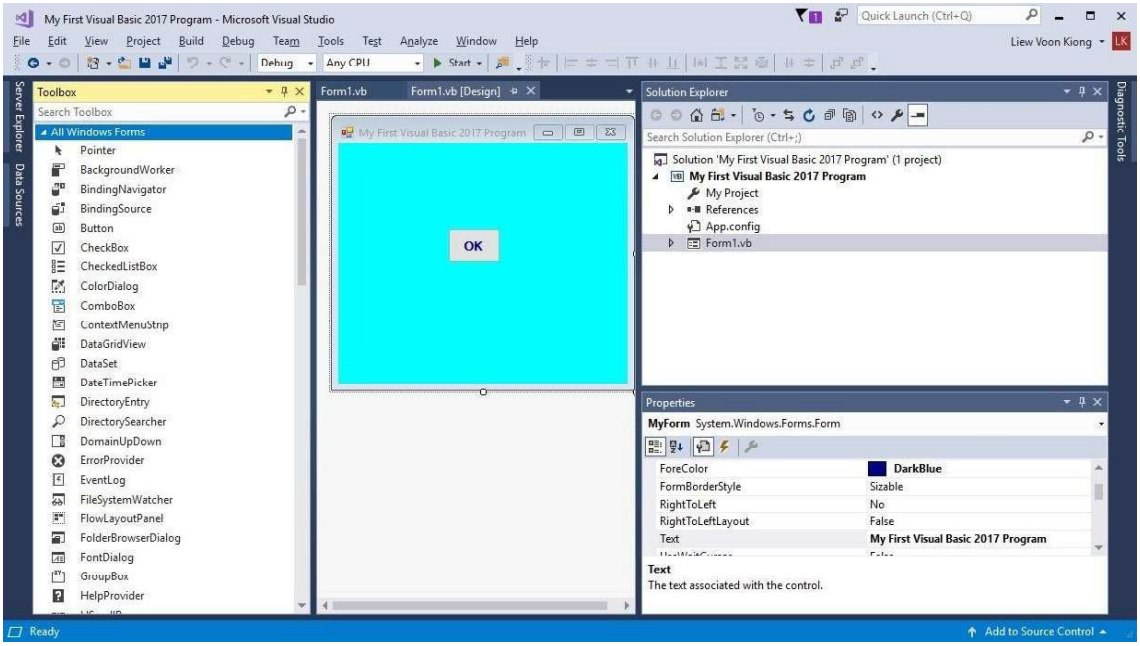


**Figure 2.6: Toolbox**

You can drag the Toolbox to any position you like. You can also dock the Toolbox by right-clicking on the Toolbox and choose dock from the pop-up menu. The docked Toolbox appears side by side with the Solution Explorer, and as one of the tabbed windows together with the Form Design window and the code window, as shown in Figure 2.7.

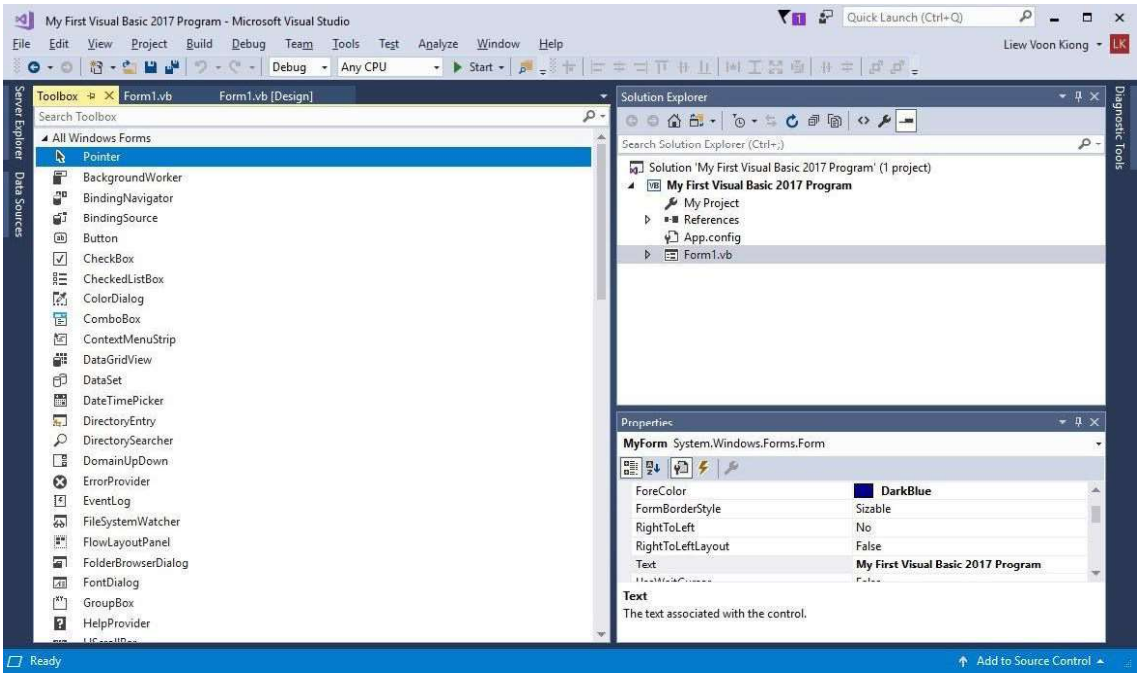

**Figure 2.7**

You can also dock the Toolbox at the bottom, below the default form, as shown in Figure 2.8. Further, you may also pin the Toolbox to the side bar or the bottom bar. We suggest that you place the toolbox alongside or at the bottom of the default form so that it is easy for you to add controls from the toolbox into the form.

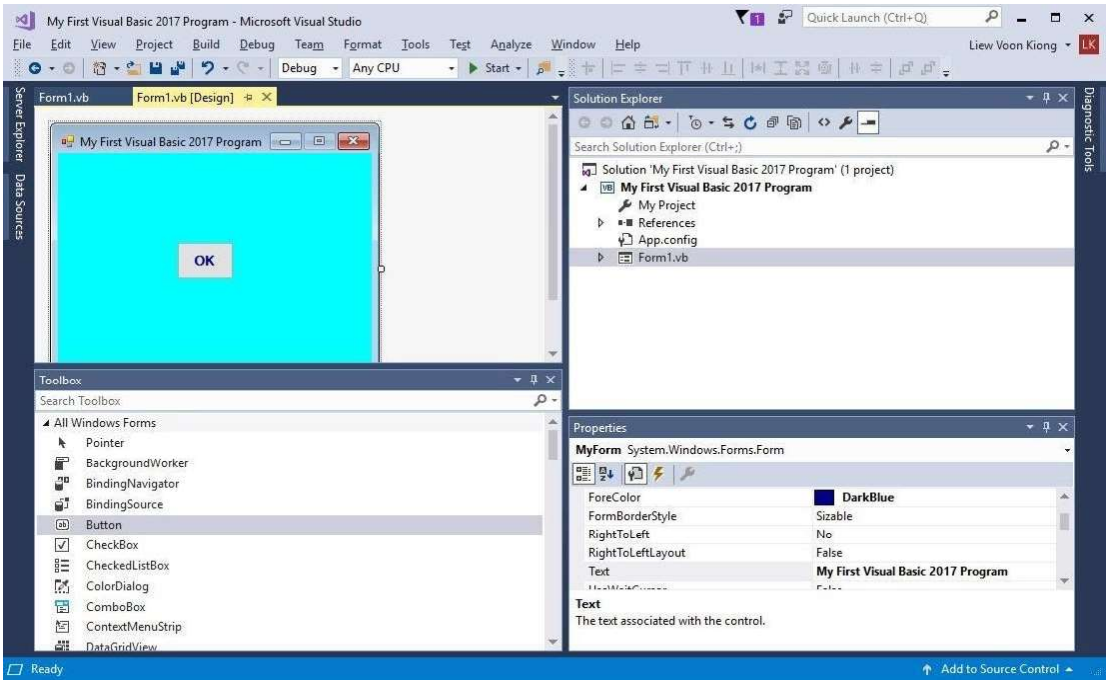


**Figure 2.8**

To add a control to the form, click the control and drag it onto the form. You can drag the control around in the form and you can also resize it.

To demonstrate how to add the 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 in Figure 2.9. Finally, add two buttons to the form and change the text to View and Close in their respective properties' windows. The picture viewer is not yet functional until we write code to it. We shall deal with the programming part in the coming chapters.

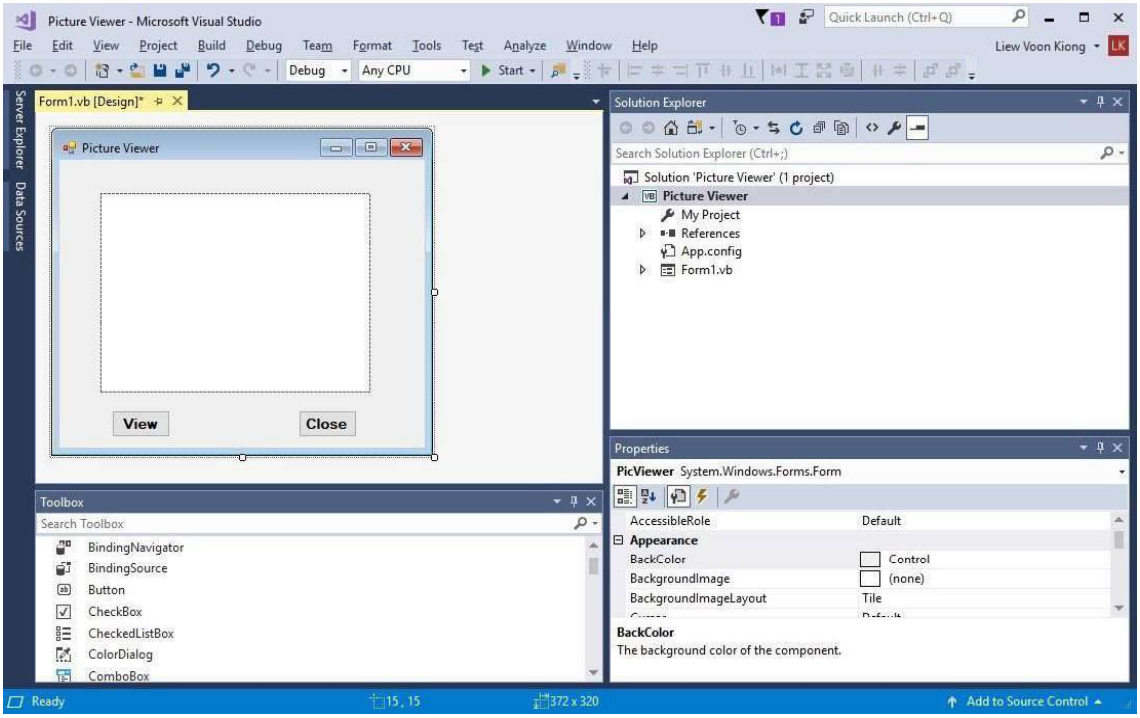


**Figure 2.9**

**Summary**

- In section 2.1, you have learned how to customize the form by changing the values of its properties.
- In section 2.2, you have learned how to add controls to the form and change their properties at design phase and at runtime.

# Chapter 3 Writing the Code

- Learn the basics of writing code

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

## 3.1 The Concept of Event-Driven Programming

Visual Basic 2017 is an event-driven programming language. Event-driven means that the code is executed in response to events triggered by the user actions like clicking the mouse, pressing a key on the keyboard, selecting an item from a drop-down list, typing some words into textbox and more. It may also be an event that runs in response to some other programs. Some of the common events in Visual Basic 2017 are load, click, double-click, drag and drop, pressing the keys and more.

Every form and every control you place on the form has a set of events related to them. 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 3.1 illustrates the event procedure load associated with the Form (its name has been changed to PicViewer therefore you can see the words PicViewer events) and Figure 3.2 shows the events associated with button.

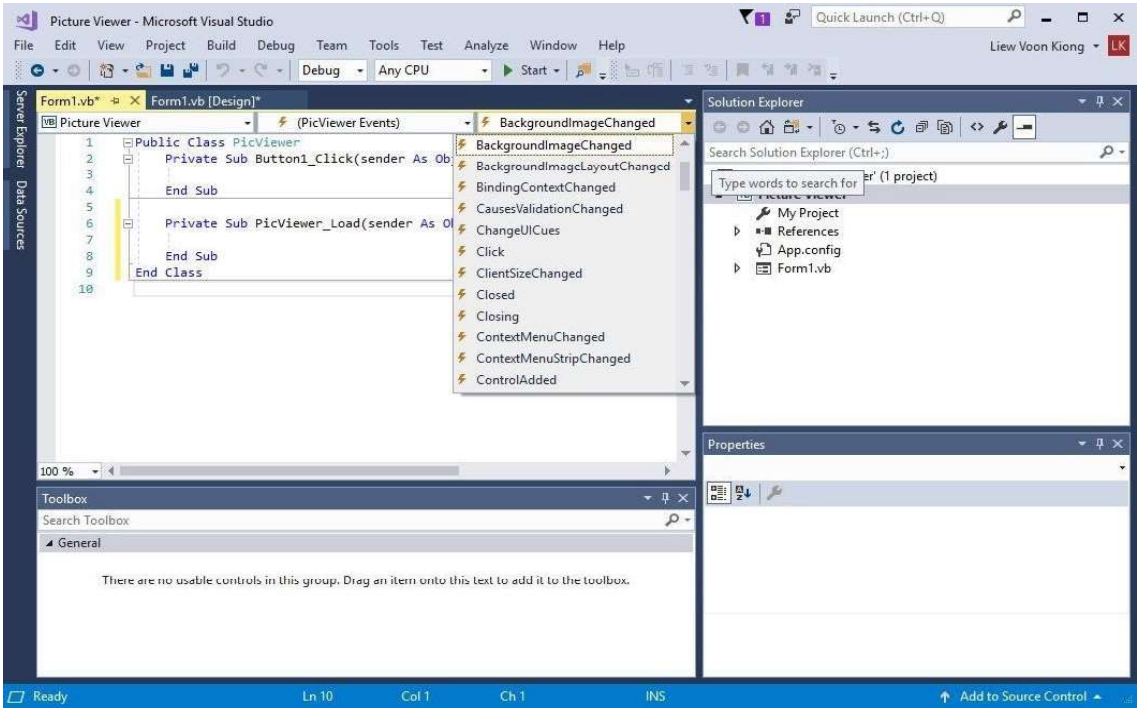


**Figure 3.1: Events associated with Form**

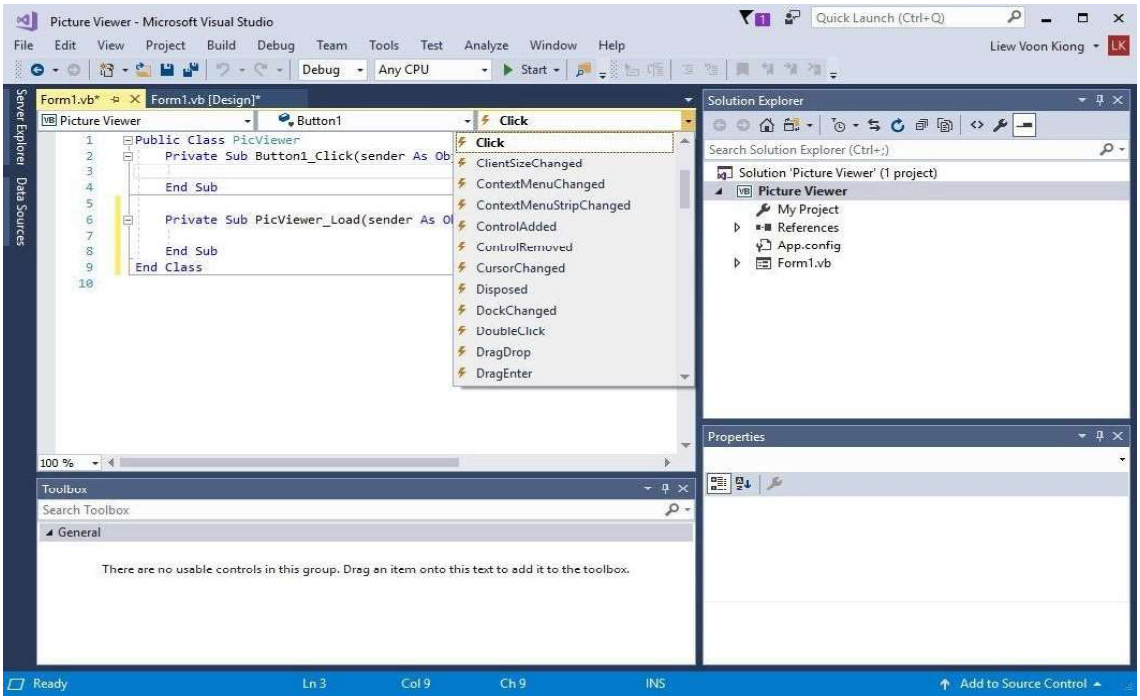


**Figure 3.2: Events associated with the button**

## 3.2 Writing the Code

To start writing the code, click on any part of the form to go into the code window as shown in Figure 3.1. The event procedure is to load Form1 and it starts with the keywords **Private Sub** and ends 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 2017")
```

**The Code**

```
Public Class Form1
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MsgBox ( "My First Visual Basic 2017 Program", ,"My Message")
End Sub
End Class
```

MsgBox is a built-in function in Visual Basic 2017 that displays a message in a message box. The MsgBox function comprises a few arguments, the first is the message and the third one is the title of the message box. When you run the program, a message box displaying the text "My First Visual Basic 2017 Program" will appear, as shown in Figure 3.3.

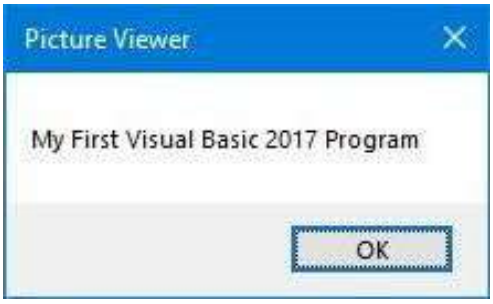


**Figure 3.3**

You will notice that above Private Sub structure there is a preceding keyword **`Public Class Form1`**. This is the model of an object-oriented programming language. When we start a windows application in Visual Basic 2017, 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.

You can write code to perform arithmetic calculations. For example, you can use the arithmetic operator plus to perform addition of two numbers, and display the result in a message box, as shown below:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
MsgBox("2" & "+" & "5" & "=" & 2 + 5)
End Sub
```

*The symbol & (ampersand) is to perform string concatenation. The output is as shown in Figure 3.4


**Figure 3.4**

**Summary**

- In section 3.1, you have learned the concepts of event driven programming
- In section 3.2, you have learned how to write code for the controls

# Chapter 4 Working with Controls

- Learn to work with a text box
- Learn to work with a label control
- Learn to work with a list box
- Learn to work with a combo box

In the preceding chapter, we have learned how to write simple Visual Basic 2017 code. In this chapter, we shall learn how to write codes for some common controls. Some of the commonly used controls are label, textbox, button, list box and combo box. However, in this chapter, we shall only deal with textbox , label, list box and combo box. We shall deal with the other controls later chapters.

## 4.1 TextBox

A Textbox is the standard control for accepting inputs from the user as well as to display the output. It can handle string and numeric data but not images or pictures. String in a textbox 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 4.1 Adding two numbers in two text boxes

In this program, you add two text boxes and a button on the form. The two text boxes are for accepting inputs from the user. Besides that, we can also program a button to calculate the sum of the two numbers using the plus operator. The value in a textbox is stored using the syntax **`TextBox1.Text`** , where Text is one of the properties of textbox.

The following code will add the value in TextBox1 and the value in TextBox2 and displays the sum in a message box. The runtime interface is illustrated in Figure 4.1.

```
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
```

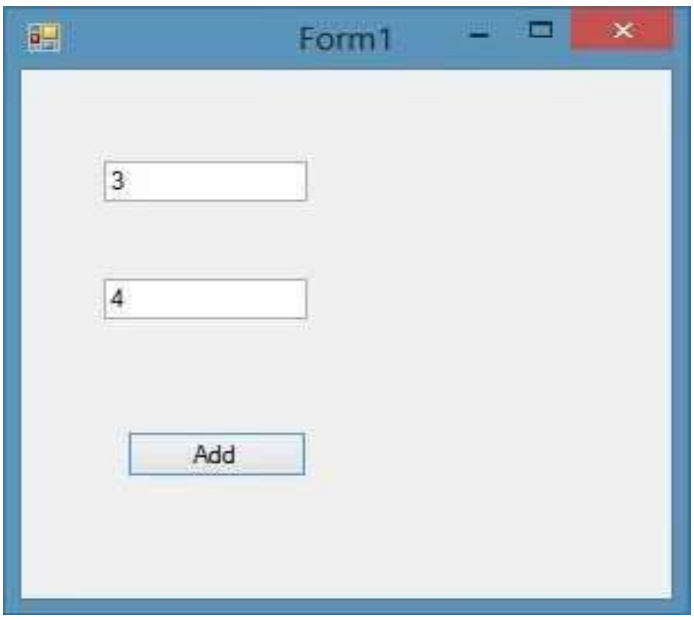


**Figure 4.1**

After clicking the Add button, you will obtain the answer in a message box, as shown in Figure 4.2


:

**Figure 4.2**

## 4.2 Label

The Label control can be used for multiple purposes like providing instructions and guides to the users, displaying outputs and more. It is different from the textbox because it is read only , which means the user cannot change or edit its content at runtime. Using the syntax

**`Label.Text`**, it can display string as well as numeric data . You can change its text property in the properties window or  at runtime by writing an appropriate code.

## Example 4.2 Displaying output on a Label

Based on Example 4.1, we add two labels, one is for displaying 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 the font size to 10. For the second label, delete the default text Label2 and change its font to bold and the 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 display the sum on a 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 instead of adding them.

The output is as shown in Figure 4.3

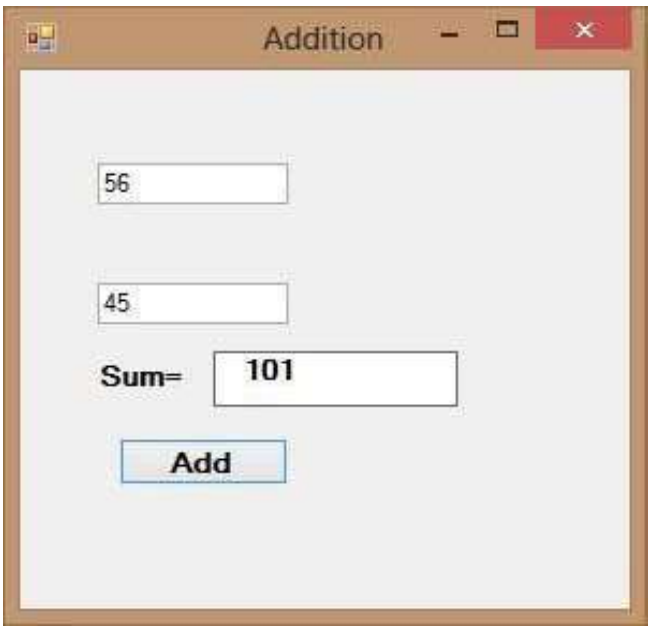


**Figure 4.3**

## 4.3 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 by the programmer at design time or at runtime using a code. We can also write code to allow the user to add items to the list box or remove the items from the list box.

### 4.3.1 Adding Items to the List Box

#### a) Adding items using the String Collection Editor

To demonstrate how to add items at design time, start a new project and insert a list box on the form then right-click on the list box to access the properties window. Next, click on collection of the Item property to launch the *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 4.4. After clicking on the OK button, the items will be displayed in the text box, as shown in Figure 4.5


**Figure 4.4**

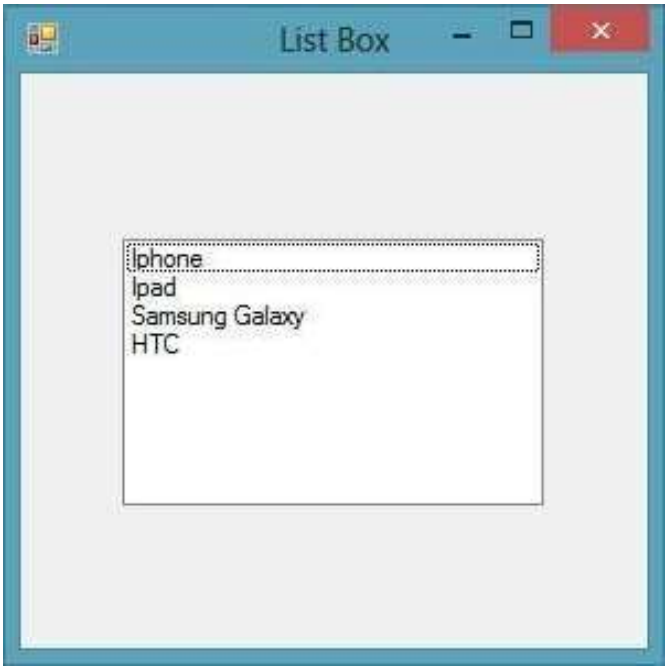


**Figure 4.5**

## b) Adding Items using the Add() Method

Items can also be added at runtime using the **Add()** method. Before we proceed further, we should know that Visual Basic 2017 is an object-oriented programming language. Therefore, visual basic 2017 comprises objects. All objects have methods and properties, and they can be differentiated and connected by hierarchy. For a list box, an Item is an object subordinated to the object ListBox . The Item object 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")
```

## Example 4.3 Adding an Item to a List Box

In this example, the item "Nokia" will be added to the end of the list, as shown in Figure 4.6

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


**Figure 4.6**

## Example 4.4 Adding items to a List Box via an input box

In this example, you can allow the user to add items via a popup input box. First, we create a variable `myitem` and then assign a value to `myitem` via the `InputBox` function that store the input from the user. We then use the Add() method to add the user's item into the list box. The code is as follows:

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

The runtime interface is as shown in Figure 4.7

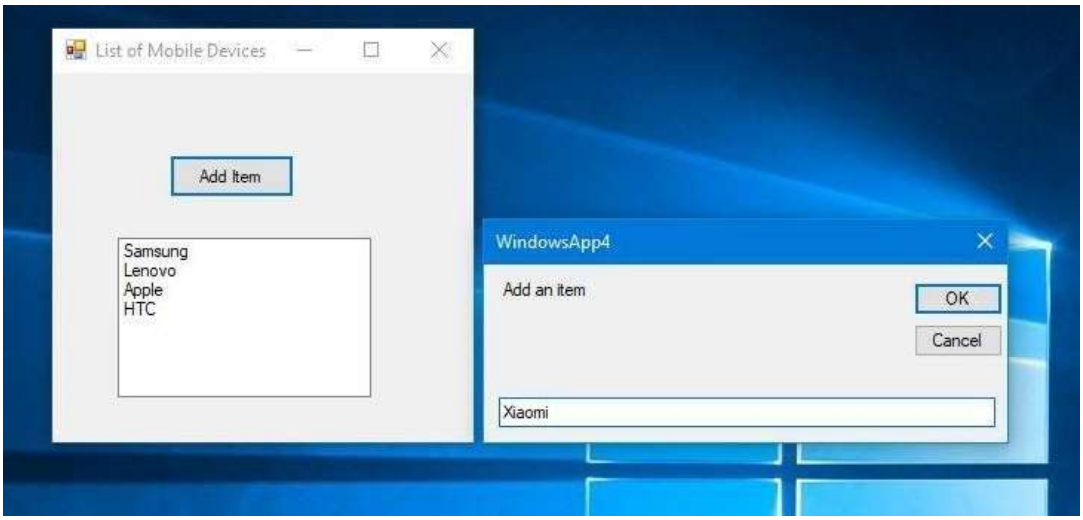


**Figure 4.7**

After typing the item 'Xiaomi" in the input box, the item will be added to the list box, as shown in Figure 4.8.


**Figure 4.8**

## Example 4.5 Creating Geometric Progression

This is a program that generates a geometric progression and displays the results in a list box. Geometric progression is a sequence of numbers where each subsequent number is found by multiplying the previous number with a fixed number. This fixed number is called the common ratio. The common ratio can be a negative number, an integer, a fraction and any number but it must not be a zero or 1.

The formula to find the nth term of the geometric progression is $\mathbf{ar^{n-1}}$ , where a is the first number and r is the common ratio.

In this program, we employ the **`Do.... Loop Until`** statement to generate the numbers in a geometric progression. To design the UI, we need to insert three text boxes for the user to enter the first number, the common ratio and the number of terms. We also need to insert a list box to list the generated numbers. Besides that, a command button is needed for the user to generate the numbers in the geometric progression. In addition, we also add another button for clearing the list of generated numbers.

To add the numbers to the list box, we use the **`Add()`** method. The syntax is **`ListBox1.Items.Add(x)`**, where x can be any variable.

**The code**

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

```
BtnComp.Click
        Dim x, n, num As Double
        Dim r As Double
        x = TxtFirst.Text
        r = TxtCR.Text
        num = TxtN.Text
        MyList.Items.Add("n" & vbTab & "x")
        MyList.Items.Add("___________")

        n = 1
        Do
            x = x * r
            MyList.Items.Add(n & vbTab & x)
            n = n + 1
        Loop Until n = num + 1
End Sub

Private Sub BtnClr_Click(sender As Object, e As EventArgs) Handles
BtnClr.Click
        MyList.Items.Clear()
End Sub
```

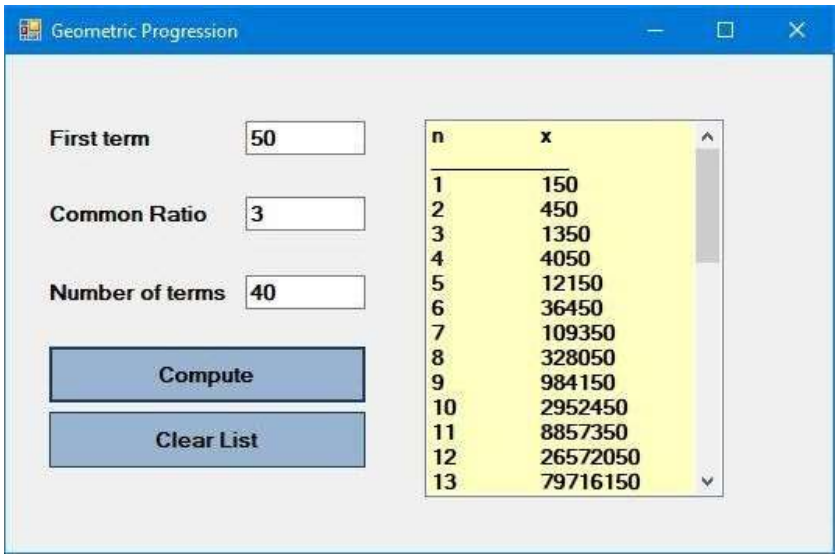


**Figure 4.9**

### 4.3.2 Removing Items from a List Box

To remove items at design time, simply open the String Collection Editor and delete the items 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 4.5.

### Example 4.6 Removing an item from a list box

In this example, add a 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.

### Example 4.7 Deleting an item from a list box via an input box

You can also allow the user to choose an item to delete via an Inputbox. To add this capability, insert a button at design time and change its text to Delete Item. Click on the button and enter the following statements in the code window:

```
Private Sub BtnDelete_Click(sender As Object, e As EventArgs) Handles
BtnDelete.Click
        Dim myitem
        myitem = InputBox("Enter your Item for Deletion")
        MyListBox.Items.Remove(myitem)
End Sub
```

The runtime interface is as shown in Figure 4.10. After entering the item Samsung in the input box and press OK, the item Samsung will be deleted from the list box.

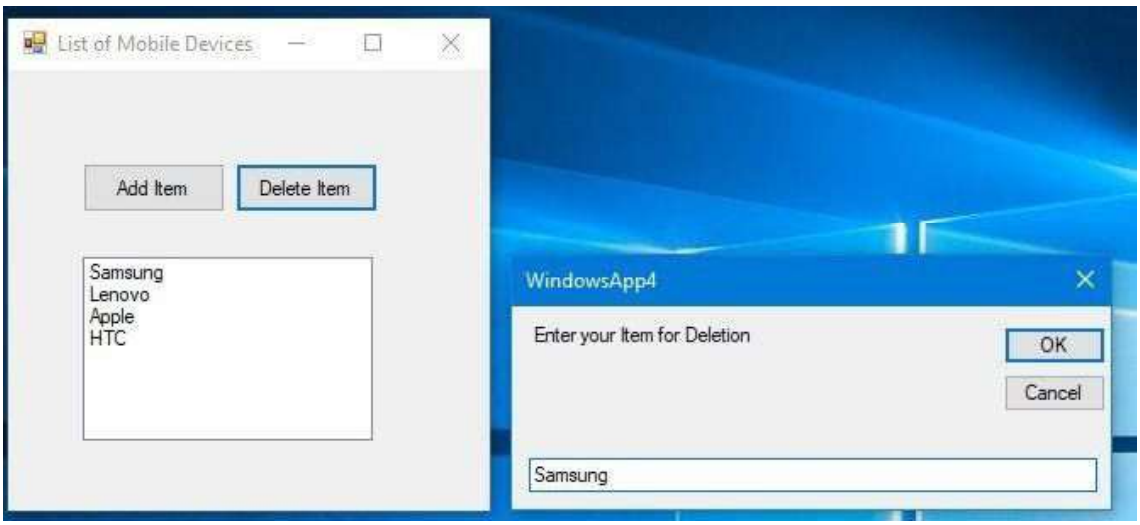


**Figure 4.10**

To remove a selected item from the list box, using the following syntax:

```
Listbox1.Items.Remove(ListBox1.SelectedItem)
```

## Example 4.8 Removing a selected item from a list box

```
Private Sub BtnDelSel_Click(sender As Object, e As EventArgs) Handles
BtnDelSel.Click
        MyListBox.Items.Remove(MyListBox.SelectedItem)
End Sub
```

When the user run the program and select an item to delete, the item will be deleted.

To remove multiple selected items from the list box, you need to use the If...End If structure together with the For…Next loop. Besides that, you also must ensure that the list box allows multiple selection. To enable multiple selection, set the selection mode to `MultipleSimple` in the list box properties windows. The code is as shown in Example 4.7.

## Example 4.9 Removing multiple selected items in a list box

In this example, add an extra button to the previous example and label it as Clear Selected Items. Key in the following code:

```
Private Sub BtnDelSelected_Click(sender As Object, e As EventArgs)
Handles BtnDelSelected.Click
        If MyListBox.SelectedItems.Count > 0 Then
            For n As Integer = MyListBox.SelectedItems.Count - 1 To 0
Step -1
                'remove the current selected item from items
                MyListBox.Items.Remove(MyListBox.SelectedItems(n))
            Next n
        End If
    End Sub
```

To clear all the items at once, use the clear method, as illustrated in Example 4.8.

## Example 4.10 Removing all items in a list box using the Clear method

In this example, add a button and label it “Clear the List”

```
Private Sub BtnClr_Click(sender As Object, e As EventArgs) Handles
BtnClr.Click
        MyListBox.Items.Clear()
End Sub
```

When you run the program and click the “Clear the List” button, all the items will be cleared.

The design interface for remove the items from the list box is as shown in Figure 4.11

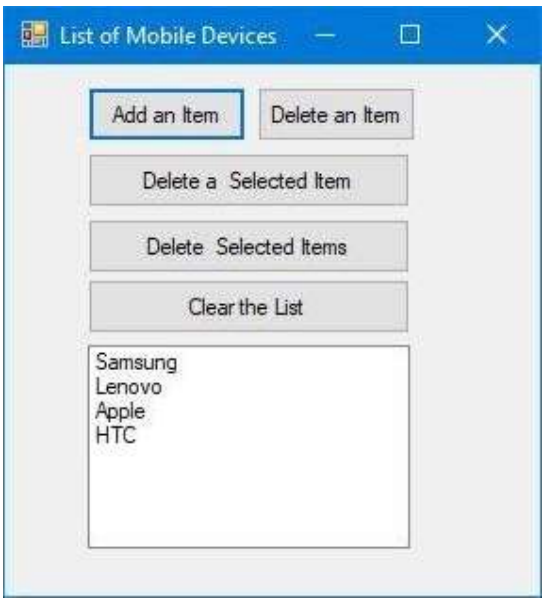


**Figure 4.11**

## 4.4 ComboBox

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 combo box only displays one item at runtime. The user needs to click on the handle (small arrowhead) on the right of the combo box to see all the items in a drop-down list.

### 4.4.1 Adding Items to a combo box

To add items to the combo box at design time, use the String Collection Editor as shown in Figure 4.12. Besides that, if you want to display an item as the default text in the combo box when you run the program, enter the name of the item by replacing the text property of the combo box.


**Figure 4.12**

After clicking the handle of the right side of the combo box, the user will be able to view all the items, as shown in Figure 4.13

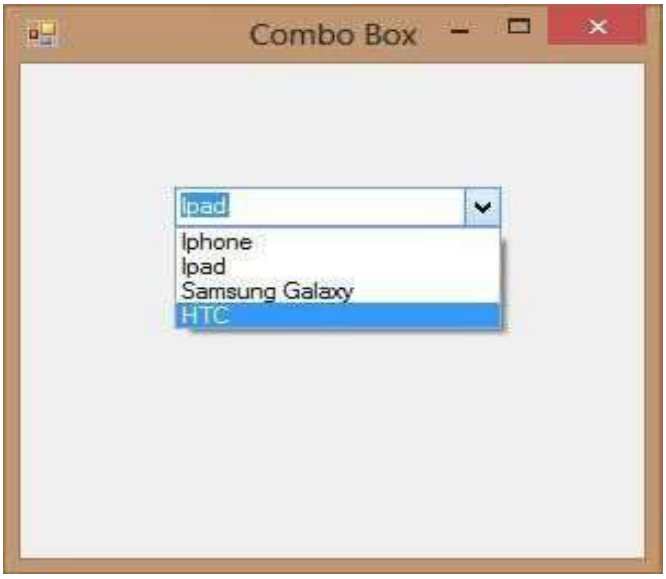


**Figure 4.13**

Besides that, you may add items using the **Add()** method. For example, if you wish to add an item to ComboBox1, key in the following statement. The output is as shown in Figure 4.14

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

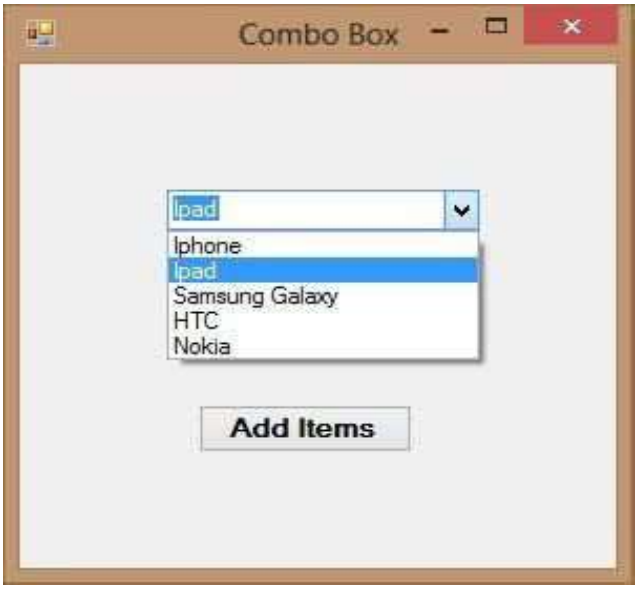


**Figure 4.14**

You can also allow the user to add items via an input box, 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
```

The runtime interface is shown in Figure 4.15

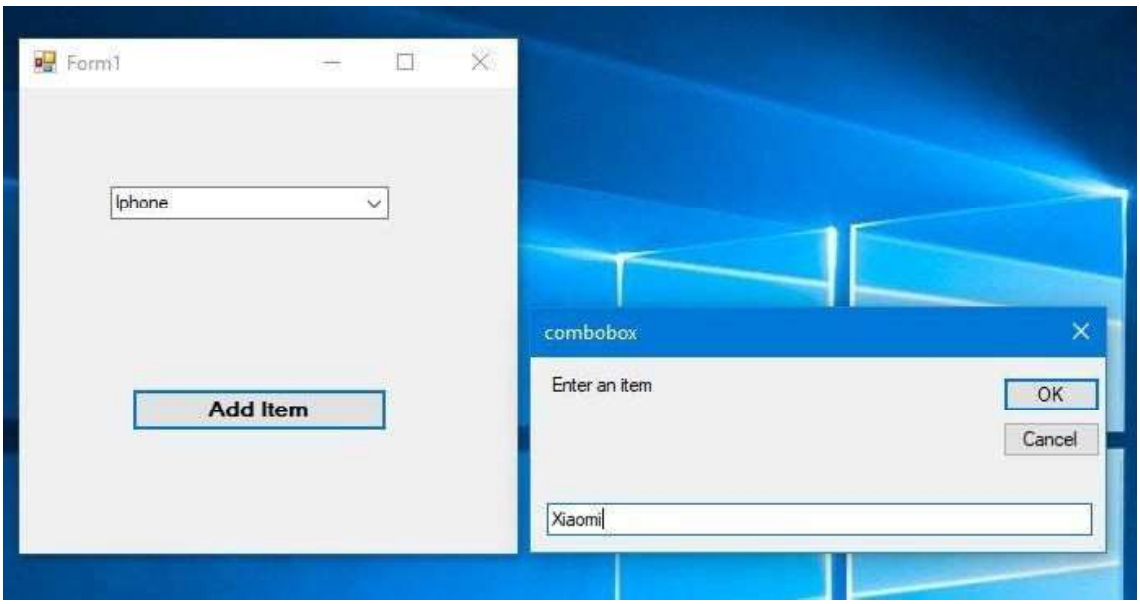


**Figure 4.15**

After you type the item 'Xiaomi' and click Ok, you can see that the item has been added to the combo box, as shown in Figure 4.16.

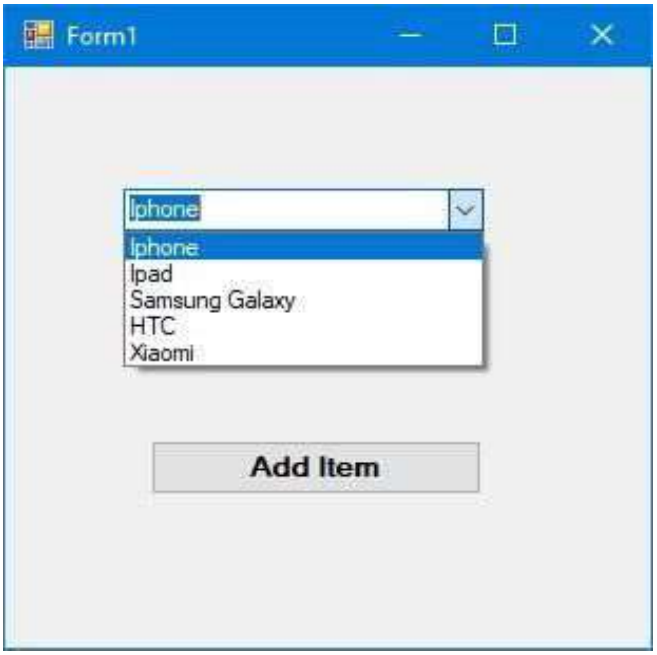


**Figure 4.16**

### 4.4.2 Removing Items from a Combo box

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

To remove the items at runtime, 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 select a specific item to delete, the code is as follows:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
MyCombo.Items.Remove(MyCombo.SelectedItem)
End Sub
```

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
```

**Summary**

- In section 4.1, you have learned how to work with a text box
- In section 4.2, you have learned how to work with a label
- In section 4.3.1, you have learned how to add items to a list box
- In section 4.3.2, you have learned how to remove items from a list box
- In section 4.4.1, you have learned how to add items to a combo box
- In section 4.4.2, you have learned how to remove items from a combo box

# Chapter 5 Handling Images

- Learn to load image in a picture box
- Learn to browse and load images using the common dialog

In this chapter, we shall learn how to load an image into the picture box at design time and at runtime. In addition, we shall also learn how to use a common dialog control to browse for image files in your local drives and then select and load an image into the picture box.

## 5.1 Loading an Image in a Picture Box

### 5.1.1 Loading an Image at Design Time

In this section, let us develop an image viewer . To create an image viewer, we insert a picture box in the form. Next, change its border property to `FixedSingle` and its background color to white. You might also want to change the `SizeMode` property of the image to `StretchImage` so that the image can fit in the picture box. In the properties window, scroll to the Image property, as shown in Figure 5.1. In the properties window, click on the grey button on the right of the Image item to bring out the “Select Source” dialog box , as shown in Figure 5.2

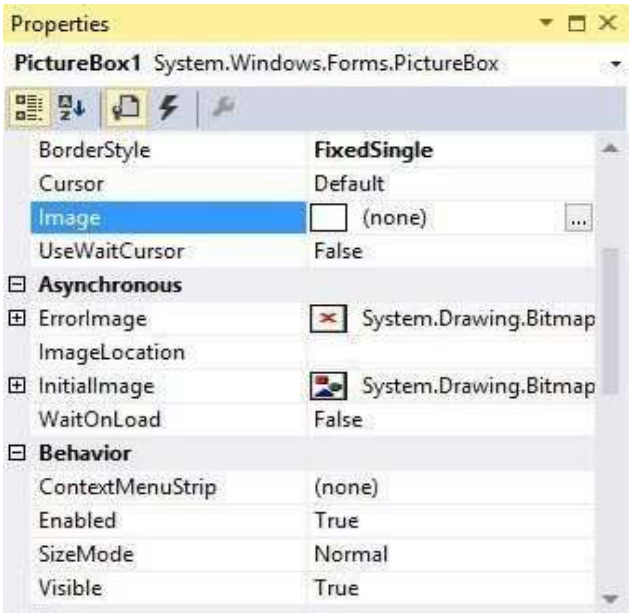

**Figure 5.1**

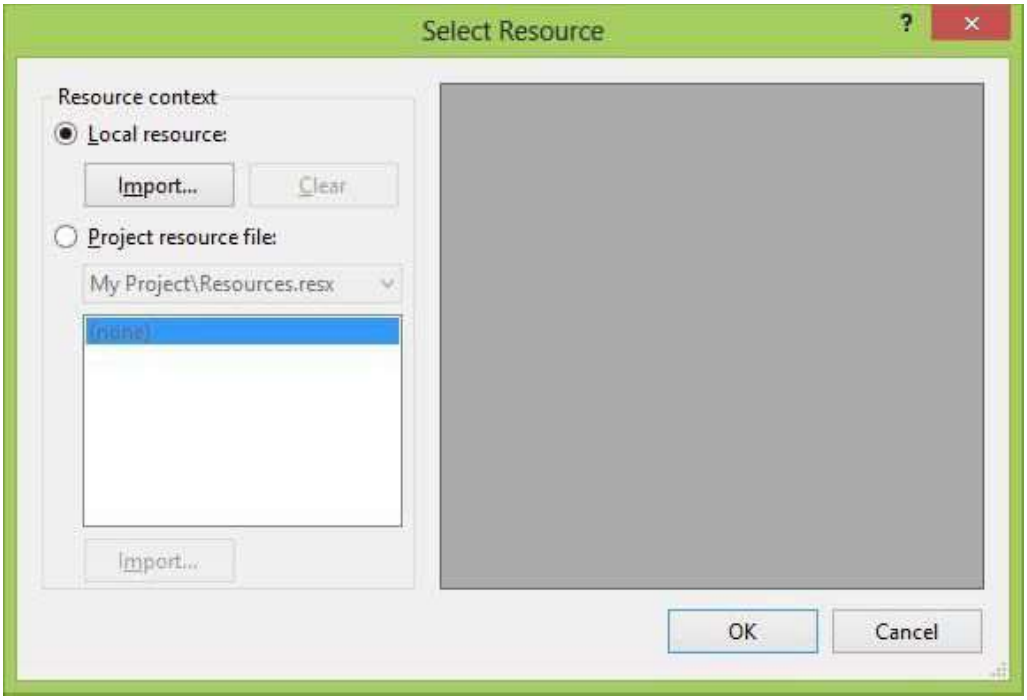


**Figure 5.2**

The next step is to select local resource and click on the Import button to view the available image files in your local drives, as shown in Figure 5.3. 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 5.4

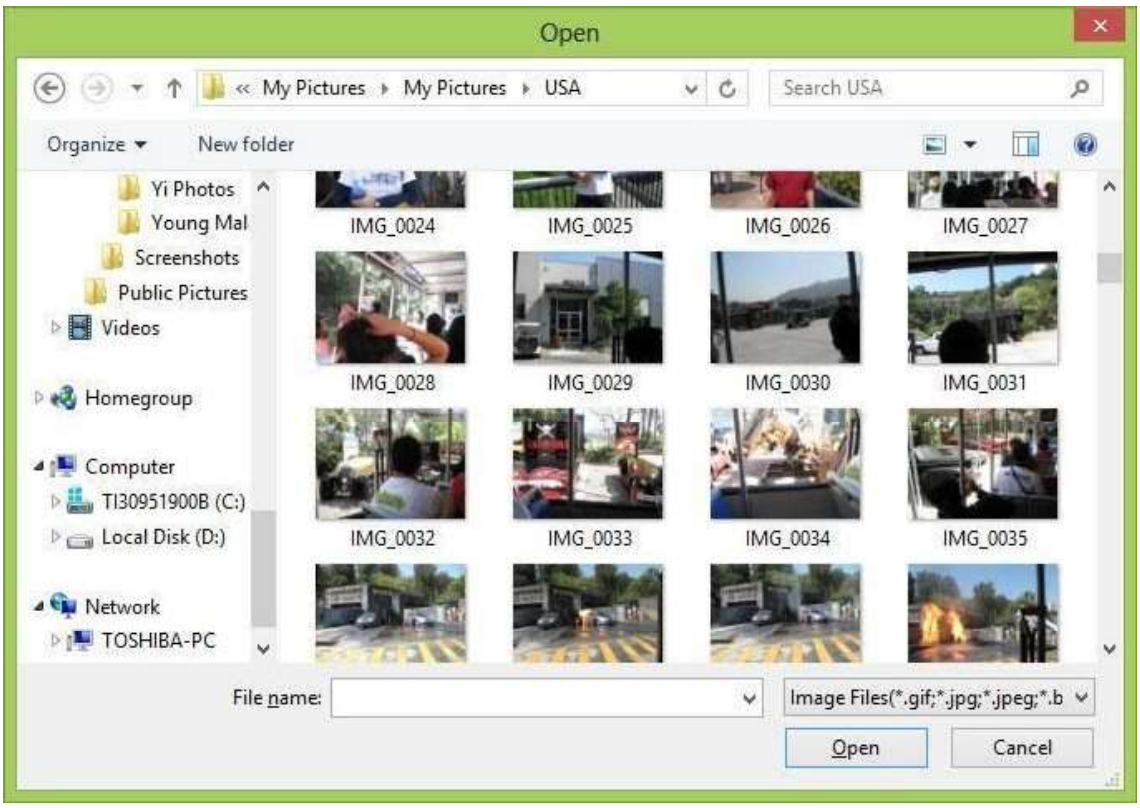


**Figure 5.3**


**Figure 5.4**

### 5.1.2 Loading an Image at Runtime

We can also load an image 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 5.4

## 5.2 Loading an Image using Open File Dialog Control

To load an image in a picture box using the `OpenFileDialog` control, we must 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. For the `OpenFileDialog` to display all types of image files, we must 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 5.5. These are the common image file formats. Besides that, you need to delete the default Filename.

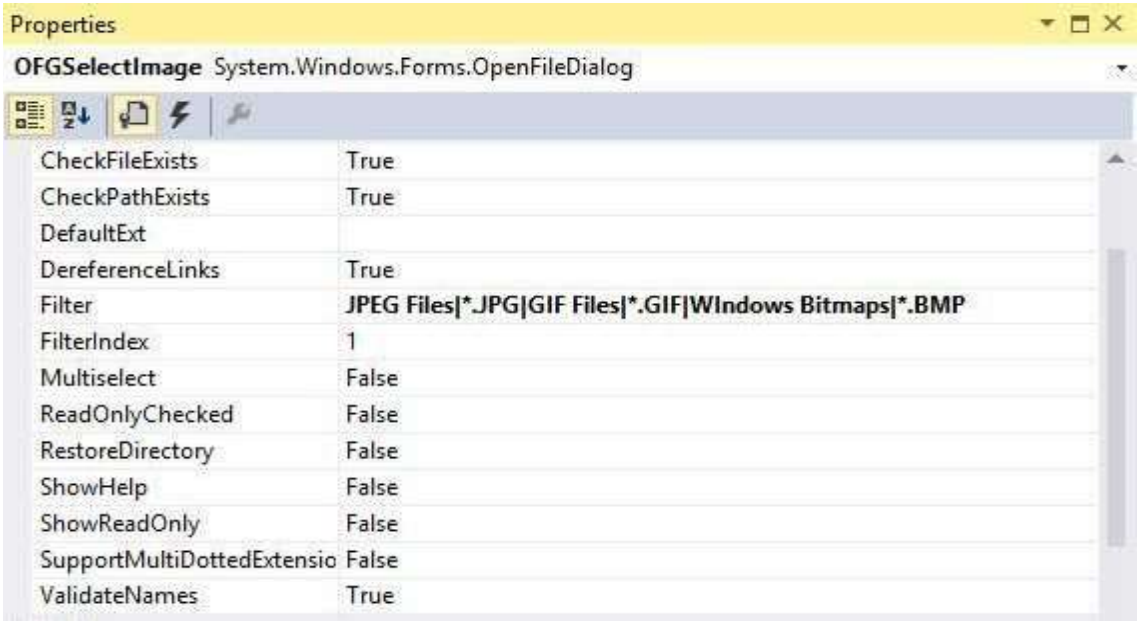


**Figure 5.5**

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 5.6.

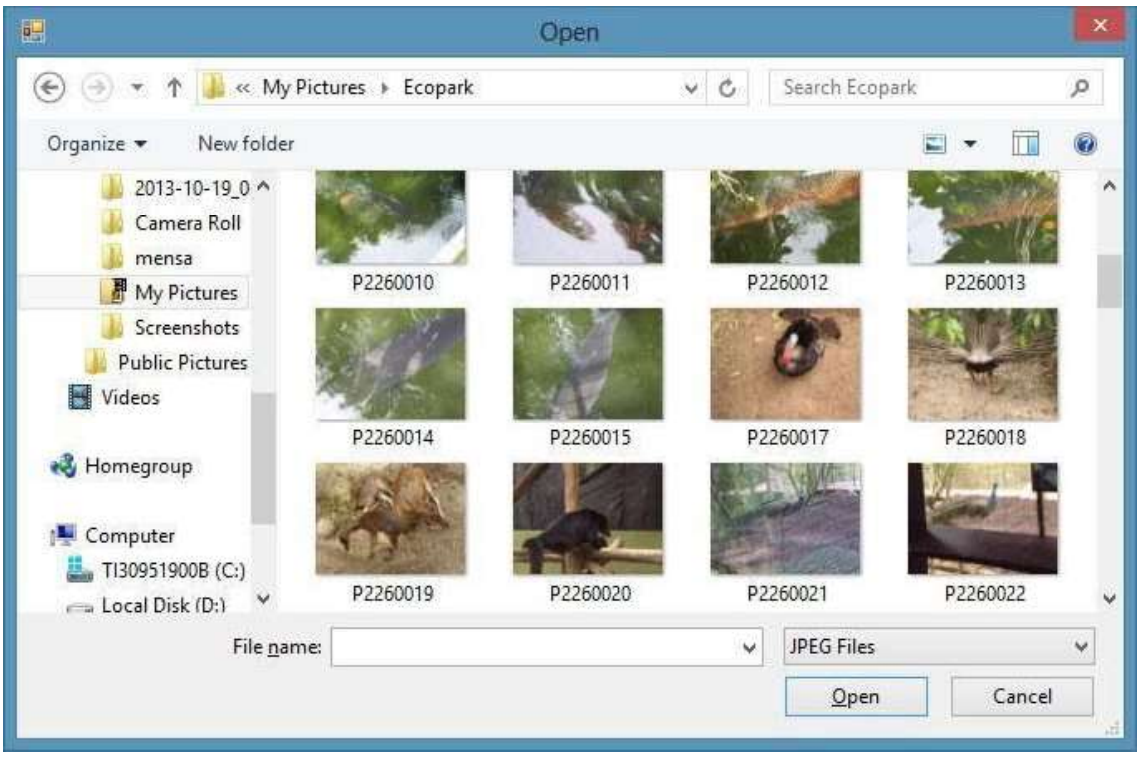


**Figure 5.6**

Notice that that the default image file is JPEG since 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 5.7.

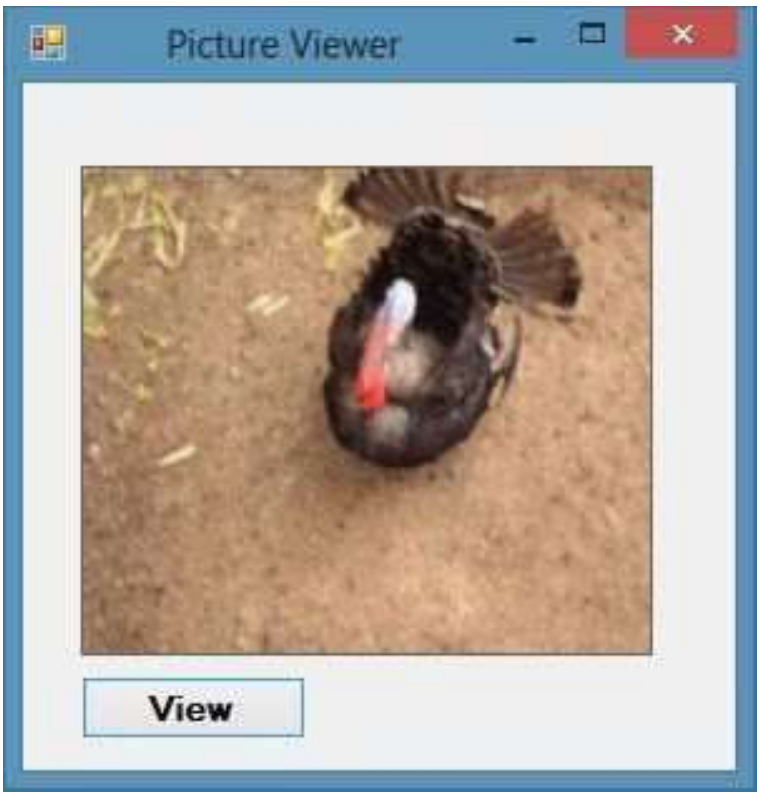


**Figure 5.7**

**Summary**

- In section 5.11, you have learned how to load an image at design time using the properties window
- In section 5.1.2, you have learned how to load an image at runtime
- In section 5.2, you have learned how to load an image using the OpenFileDialog control

# Chapter 6 Working with Data

- Understand various data types
- Learn how to declare variables
- Learn how to declare constants

We deal with many kinds of data in our daily life like names, phone number, addresses, money, date, stock quotes, statistics and more. Similarly, in Visual Basic 2017, we must deal with all sorts 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 2017, data can be stored as variables, constants or arrays. The values of data stored as variables always change, just like the contents of a mailbox or the storage bin while the value of a constant remains the same throughout.

## 6.1 Data Types

Visual Basic 2017 classifies information into two major data types, the numeric data types and the non-numeric data type

### 6.1.1 Numeric Data Types

Numeric data types are types of data comprises numbers. Numeric data are divided into seven types based on the range of values they can store.

Calculations that only involve round figures can use Integer or Long integer. Computations that require high precision must 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 6.1

**Table 6.1 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). |

## 6.1.2 Non-numeric Data Types

Non-numeric data comprises string data types, Date data types, Boolean data types, Object data type and Variant data type .

**Table 6.1 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 |

### 6.1.3 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 6.3.

**Table 6.3 Suffixes**

| 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#
```

## 6.2 Variables and Constants

Data can be stored as a variable or as a constant. Variables are like mailboxes in the post office. The content of the variables changes every now and then, just like the mailboxes.

### 6.2.1 Variable Names

Like the mailboxes, each variable must be given a name. To name a variable, you must 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 6.4

**Table 6.4**

| Valid Names | Invalid Names |
|---|---|
| My_Computer | My.Computer |
| Smartphone123 | 123Smartphone |
| Long_Name_Can_beUSE | LongName&Canbe&Use<br>*& is not acceptable |

### 6.2.2 Declaring Variables

We must 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 DataType
```

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 6.1 Declaring Variables using Dim

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

```
Dim 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 statements in one line, separating each variable with a comma.

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

For the string declaration, there are two possible forms, the variable-length string and the fixed-length string.

## Example 6.2 Displaying Message using MsgBox

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

When you run the program, a message box that shows the text "Happy Birthday!" will appear, as shown in Figure 6.1


**Figure 6.1**

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

```
Dim VariableName As String * n
```

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

```
Dim yourName As String * 10
```

yourName can holds no more than 10 Characters.

### 6.2.3 Assigning Values to Variables

After declaring various variables using the **Dim** statements, we can assign values to those variables. The syntax 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, 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 6.4.

## Example 6.3 Incompatible Data Type

```
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 6.2.


**Figure 6.2**

### 6.2.4 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 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 completed, **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.

### 6.2.5 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 ConstantName As Single=K
```

### Example 6.4 Calculating the Area of Triangle

```
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
```

Running the program and clicking the OK button will produce the following message.

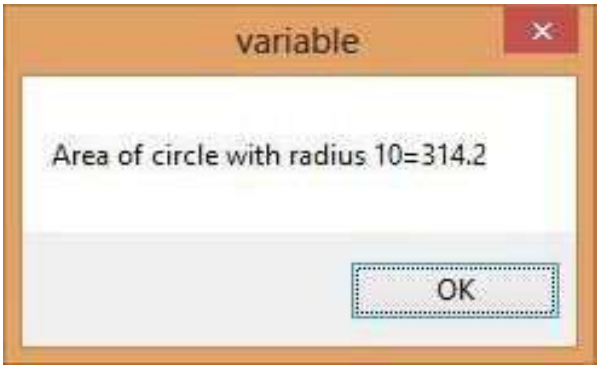


**Figure 6.3**

**Summary**

- In section 6.11, you have understood numeric data types
- In section 6.1.2, you have understood non-numeric data types
- In section 6.1.3, you have learned how to use suffixes for literals
- In section 6.2.1, you have understood rules that govern variable names
- In section 6.2.2, you have learned how to declare variables
- In section 6.2.3, you have understood the scope of declaration of variables
- In section 6.2.3, you have learned how to declare a constant

# Chapter 7 Array

- ❖ Understand the concept of array
- ❖ Learn how to declare a one-dimensional array
- ❖ Learn how to declare a two-dimensional array

## 7.1 Introduction to Arrays

An array is a group of variables with the same data type. When we work with a single item, we only need to use one variable. However, if we deal with a list of items of similar type , it is better to declare an array of variables instead of using a variable for each item

Imagine if we need to enter one thousand names, it would be very tedious to declare one hundred different names. Therefore, instead of declaring one thousand different variables, we just need to declare only one array. We differentiate each item in the array by using a

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.

## 7.2 Dimension of an Array

An array can be one dimensional or multidimensional. One dimensional array is a list of items 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 of the element. The way to reference an element in a two-dimensional array is `ArrayName(x,y)` , where (x,y) is the index of the element. Usually it is sufficient to use one dimensional and two-dimensional arrays, we only need to use higher dimensional arrays if we need to deal with more complex problems. Let me illustrate the arrays with tables.

**Table 7.1 One-Dimensional Array**

| Student Name | SName(0) | SName(1) | SName(2) | SName(3) | SName(4) | SName(5) |
|---|---|---|---|---|---|---|

**Table 7.1 Two-Dimensional Array**

| | | | |
|---|---|---|---|
| SName(0,0) | SName(0,1) | SName(0,2) | SName(0,3) |
| SName(1,0) | SName(1,1) | SName(1,2) | SName(1,3) |
| SName(2,0) | SName(2,1) | SName(2,2) | SName(2,3) |
| SName(3,0) | SName(3,1) | SName(3,2) | SName(3,3) |

## 7.3 Declaring Arrays

We 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 declares 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
```

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 forth. 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, the following statement declares an array that consists of 11 elements starting from CusName(0) through to CusName(10)

```
Dim CusName(10) As String
```

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

## Example 7.1 Find the Length of an Array

```
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
```

Running the program displays the length of the array i.e. 11, as shown in Figure 7.1


**Figure 7.1**

You might also declare an array with a non-zero starting index by initializing an index value other than zero, as follows:

```
Dim arrayname As DataType()
arrayName = New String(){1,2,3,....,n)
```

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

## Example 7.2 Using the Length Property

```
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.

he statement to declare a two-dimensional array is as follow, 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)

```
Dim ArrayName(m,n) As dataType
```

## Example 7.3 Find the Length of a Two-Dimensional Array

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
Dim CusName(5,6) As String
MsgBox(CusName.Length)
End Sub
```

The program produces a message box will display the length of the array, i.e.42, as shown in Figure 7.2


**Figure 7.2**

## Example 7.4 Populating a List Box Involving an Array

```
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 7.3 and Figure 7.4

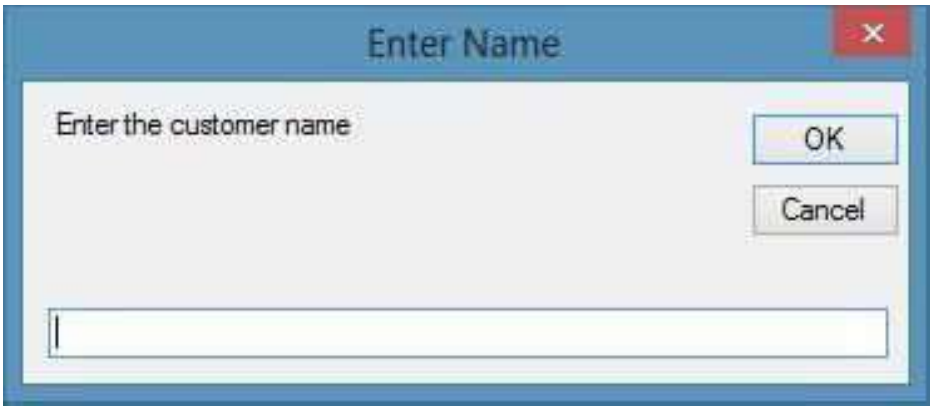


**Figure 7.3**

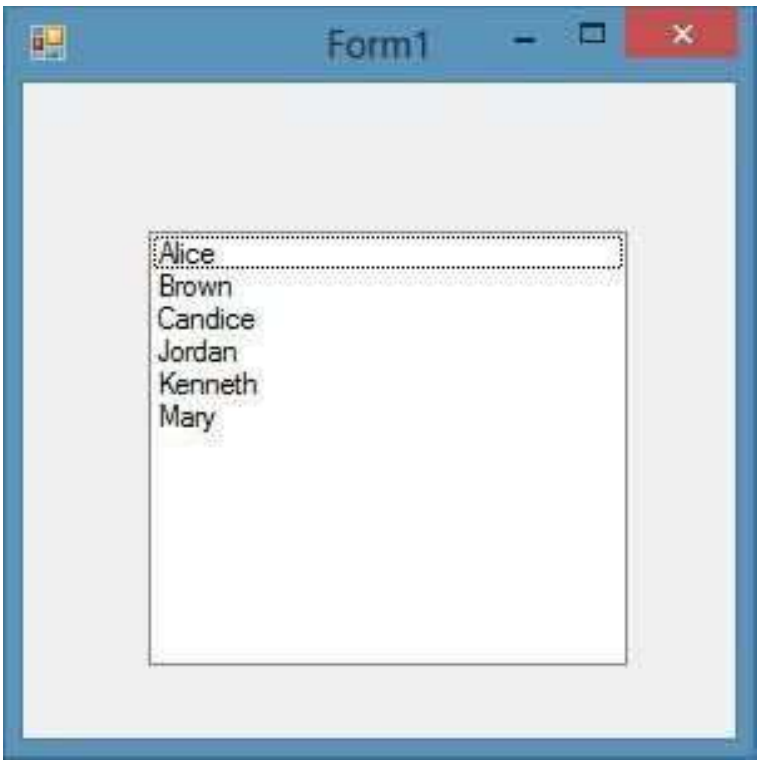


**Figure 7.4**

**Summary**

- In section 7.1, you have understood the concept of arrays
- In section 7.2, you have understood dimension in arrays
- In section 7.3, you have learned how to declare an array

# Chapter 8 Performing Mathematical Operations

- Recognize various mathematical operators
- Learn to write code that perform mathematical operations

Computer execute mathematical calculations much faster than human beings do. However, computer itself cannot perform any mathematical calculations without receiving instructions from the user. In Visual Basic 2017, we can write code to instruct the computer to perform mathematical calculations such as addition, subtraction, multiplication, division and many other kinds of mathematical operations.

## 8.1 Mathematical Operators

For Visual Basic 2017 to perform mathematical calculations, we need to write code that involves the use of various mathematical operators. The mathematical operators are remarkably like the normal arithmetic operators, only with some 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 2017 mathematical operators is shown in table 8.1.

**Table 8.1 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 |

## 8.2 Writing Code that Performs Mathematical Operations

Once you can recognize all the mathematical operators , it is quite easy to write code that can perform mathematical operations. First you need to think of a mathematical problem and the equations as well as formulas that are required to for solving it then write the code using those formulas and equations.

### Example 8.1 Standard Arithmetic Calculations

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. This program performs standard arithmetic operations involving addition, subtraction, multiplication and division. The Code is as follows:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As 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
LblSum.Text=sum
LblDiff.Text=difference
LblPro.Text = product
LblQt.Text = quotient
End Sub
```

### Example 8.2 Pythagorean Theorem

This program uses Pythagorean 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 Pythagorean Theorem, it is written as:

c^2=a^2+b^2

The code is as follows:

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As 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 8.3: BMI Calculator

A lot of people are obese now and that 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 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

To calculate your BMI, you just need use a calculator or create a homemade computer program, and this is exactly what I am showing you here. The BMI calculator is a Visual Basic program that can calculate the 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 Object, ByVal e As EventArgs)
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 Figure 8.1. In this example, height is 1.80m (about 5 foot 11), your weight is 75 kg (about 168lb), and the BMI is 23.14815. The reading suggests that you are healthy. (Note: 1 foot=0.3048, 1 lb. =0.45359237 kilogram)

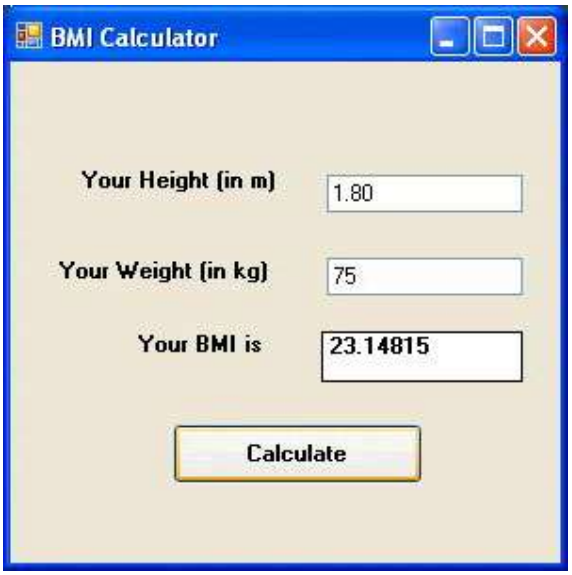


**Figure 8.1**

**Summary**

- In section 8.1, you have recognized all mathematical operators
- In section 8.2, you have learned to write code that perform arithmetic operations

# Chapter 9 String Manipulation

- Learn how to manipulate string using + and & signs
- Learn how to manipulate string using built-in functions

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. A string is a single unit of data that made up of a series of characters that includes letters, digits, alphanumeric symbols etc. 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 consist of numbers.

## 9.1 String Manipulation Using + and & signs

Strings can be manipulated using the & sign and the + sign, both perform the string concatenation which means combining two or more smaller strings into a larger string. For example, we can join “Visual” ,”Basic” and “2017" into “Visual Basic 2017" using “Visual” & ”Basic” or “Visual “+”Basic”, as shown in the Examples below:

### Example 9.1 String Concatenation

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim text1, text2, text3, text4 As String
text1 = "Visual"
text2 = "Basic"
text3 = "2017"
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 9.1

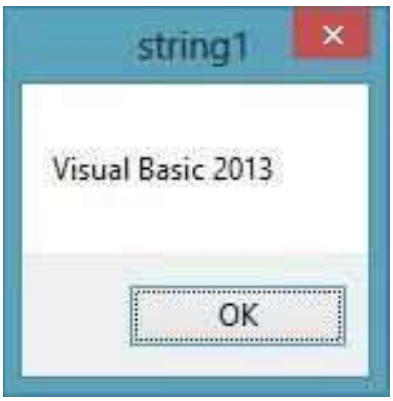


**Figure 9.1**

## Example 9.2 Data Mismatch

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As 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 shown in Figure 9.2.

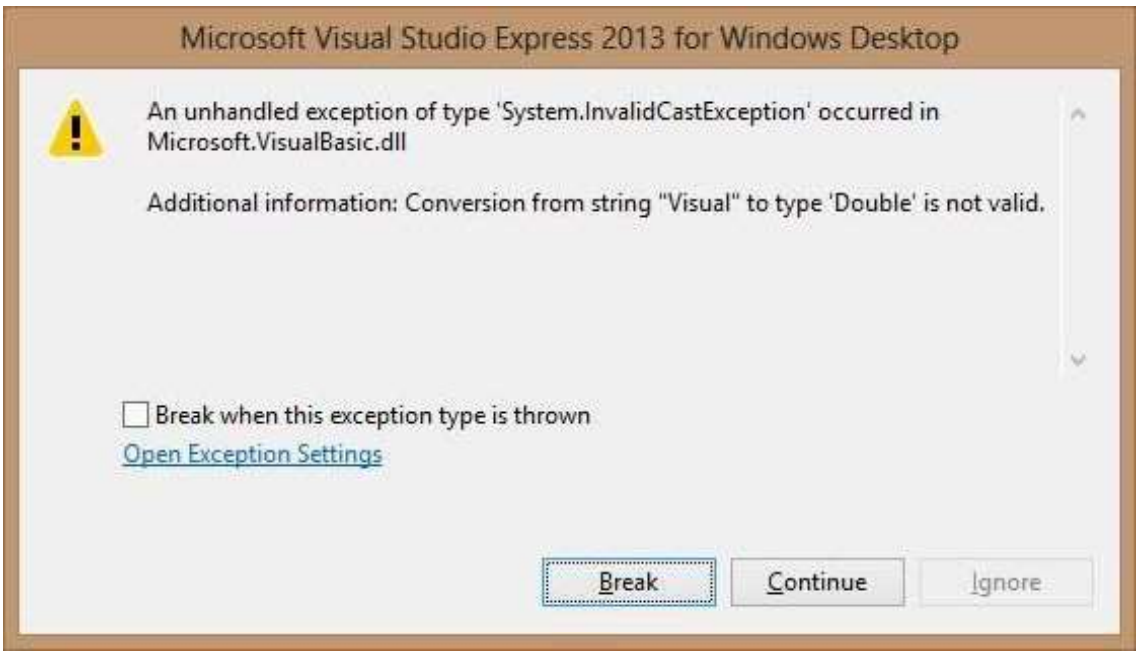


**Figure 9.2**

However, using & instead of + will be alright.

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

The output is shown in Figure 9.3

**Figure 9.3**

## 9.2 String Manipulation Using Built-in Functions

A function is like a normal procedure, but the main purpose of the function is to accept a specific 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 2017.

### 9.2.1 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")
```

### Example 9.3 Finding the Length of a Phrase

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim MyText As String
MyText="Visual Basic 2017"
MsgBox(Len(MyText))
End Sub
```

The output is shown in Figure 9.4

**Figure 9.4**

### 9.2.2 Right Function

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

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

### Example 9.4 Extracting the Right Portion of a Phrase

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim MyText As String
MyText = "Visual Basic"
MsgBox(Microsoft.VisualBasic.Right(MyText, 4))
End Sub
```

The program returns four right most characters of the phrase , as shown in Figure 9.5


**Figure 9.5**

### 9.2.3 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
```

### 9.2.4 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)
```

* position is the starting position of the phrase from which the retrieving process begins, and n is the number of characters to retrieve.

### Example 9.5 Retrieve Part of a Text Using Mid Function

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

This program extracts six characters starting from position 2 of the phrase. For example, if you enter the phrase “Visual Basic 2017”, the extracted text is `isual`.

### Example 9.6 Extracting Text from a Phrase

You can let the user decide the starting position of the text to be extracted as well as the number of characters to be extracted, as shown in the following code:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
BtnExtract.Click
        Dim myPhrase As String
        Dim pos, n As Integer
        myPhrase = TxtPhrase.Text
        pos = TxtPos.Text
        n = TxtNumber.Text
```

```
        LblExtract.Text = Mid(myPhrase, pos, n)
End Sub
```

The runtime interface is shown in Figure 9.6

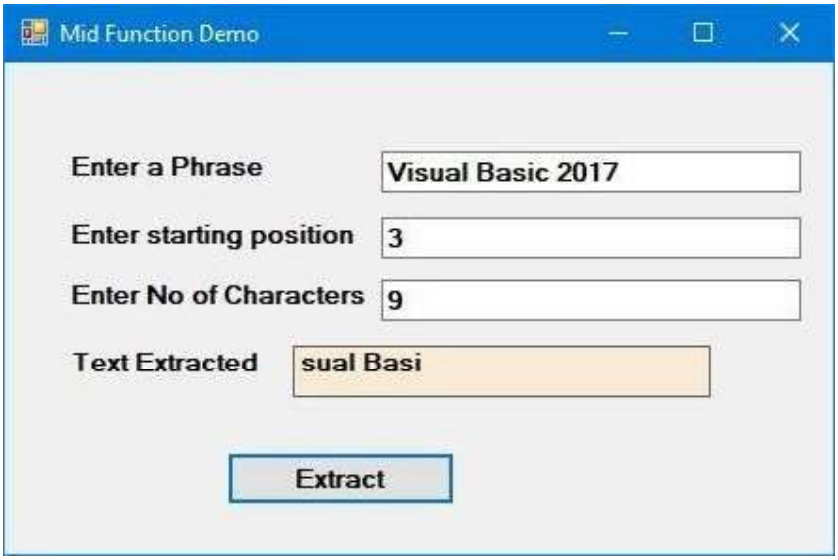


**Figure 9.6**

### 9.2.5 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 9.7 Trimming Both Side of a Phrase

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

### 9.2.6 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 2017") = Visual basic 2017
```

### 9.2.7 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 2017") = Visual Basic 2017
```

### 9.2.8 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 2017 ","Basic")=8
```

### 9.2.9 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
```

### 9.2.10 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. Please refer to the ASCII table in the Appendix for a complete list of the codes. 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 syntax of the Asc function is

```
Asc(Character)
```

The followings are some examples:

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

**Summary**

- In section 9.1, you have learned how to manipulate string using the & and + signs
- In section 9.2, you have learned how to manipulate string using various built-in function

# Chapter 10 Using If…Then…Else

- Learn about the conditional operators
- Learn about the logical operators
- Learn how to use If…Then…Else

In this chapter, we shall learn how to write code that can make decisions and control the program flow in the process. The decision-making process is an important part of programming because it can solve problems in a smart way and provide useful output or feedback to the user.

For example, we can write program that instruct the computer to perform some tasks until some conditions are met. To control the program flow, we must use the conditional operators and the logical operators together with the **If...Then...Else** control structure.

## 10.1 Conditional Operators

Conditional operators allow a program to compare values and then decide what actions to take, whether to execute a program or terminate the program and more. They compare two values to see whether they are equal, one value is greater or less than the other value. The comparison will return a true or a false result. These operators are shown in Table 10.1.

**Table 10.1 Conditional Operators**

| Operators | Description |
|---|---|
| = | Equal to |
| > | Greater than |
| < | Less than |
| >= | More than and equal to |
| <= | Less than and equal to |

| | |
|---|---|
| <> | Not equal to |

## 10.2 Logical Operators

Sometimes we might need to make more than one comparison before a decision can be made and an action taken. In this case, using numerical comparison operators alone is not sufficient, we need to the logical operators. The logical operators are shown in Table 10.2

**Table 10.2 Logical Operators**

| Operators | 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 logical operators can be used to compare numerical data as well as non-numeric data . In making strings comparison, there are specific rules to follows: Upper case letters are less than lowercase letters, “A”<”B”<”C”<”D”…….<”Z” and number are less than letters.

## 10.3 Using If...Then...Else

To effectively control 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.

### 10.3.1 If…Then Statement

This is the simplest control structure which instructs the computer to perform a specific action specified by the 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 2017 expressions
End If
```

### Example 10.1 Lucky Draw

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As 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" message.

## 10.3.2 If...Then…Else Statement

Using only `If...Then` statement does not provide choices for the users. To provide a choice, we can use the `If...Then...Else` Statement. This control structure will ask the computer to perform a specific action specified by the 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 10.2 Lucky Draw Simulation

This is a lucky draw simulation program. We use the **`Rnd()`** function to generate a random number between 0 and 1. In addition, we use the **`Int()`** function in the formula **`Int(Rnd() * 200) + 1`** to generate a random integer between 0 and 200. Next, we use the **`If...Then...Else`** statement to determine the condition for striking a lucky draw.

```
Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
        Dim myNumber As Integer
        myNumber = Int(Rnd() * 200) + 1
        LblNum.Text = myNumber
        If myNumber > 120 Then
            LblMsg.Text = " Congratulation! You won a lucky prize"
        Else
```

```
            LblMsg.Text = " Sorry, You did not win any prize"
        End If

    End Sub
```

* When you run the program and click the 'Draw" button, if the generated number is greater than 120, the message "Congratulation! You won a lucky prize" will be shown on the label . Otherwise, the label will show the "Sorry, you did not win any prize" message. The outcomes are shown in Figure 10.1 and 10.2


**Figure 10.1**


**Figure 10.2**

## Example 10.3 Lucky Draw

Now we modified Example 10.2 and add in an additional constraint , age. In the program, we use the logical operator **And** beside the conditional operators. This means that both the conditions must be fulfilled for the conditions to be true, otherwise the second block of code will be executed. In this example, the lucky number must be more than 120 and the age must be more than 50 in order to win a lucky prize, any one of the above conditions not fulfilled will disqualify the user from winning a lucky prize. In addition, we make the program more interactive by adding name in the message. The code is as follows:

```
Dim myAge As Integer
Dim myName As String

Private Sub BtnDraw_Click(sender As Object, e As EventArgs) Handles
BtnDraw.Click
 Dim myNumber As Integer
 myAge = TxtAge.Text
 myName = TxtName.Text
 myNumber = Int(Rnd() * 200) + 1
 LblNum.Text = myNumber

  If myNumber > 120 And myAge > 50 Then
LblMsg.Text = " Congratulation " & myName & ",You won a lucky prize!"
 Else
LblMsg.Text = myName & " Sorry" & myName & ", you did not win any prize"
 End If

End Sub
```

The outcomes are shown in Figure 10.3 and Figure 10.4


**Figure 10.3**


**Figure 10.4**

### 10.3.3 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 structure of **`If…Then...ElseIf`** statement is

```
If condition Then
Visual Basic 2017 expression
ElseIf condition 1 Then
Visual Basic 2017 expression
ElseIf condition 2 Then
Visual Basic 2017 expression
.
.
Else
Visual Basic 2017 expression
End If
```

## Example 10.4 Grade Generator

This program uses the **`If... ElseIf`** structure and the **`And`** logical operator to compute the grade for a specific mark.

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles Button1.Click
Dim Mark As Integer
Dim Grade as String
Mark = Val(TextBox1.Text)
If Mark>=80 Then
Grade="A"
ElseIf Mark>=60 And Mark<80 Then
Grade="B"
ElseIf Mark>=50 And Mark<60 Then
Grade="C"
Else
Grade="D"
End If
MsgBox("You grade is " & Grade)
End Sub
```

Running the program and will produce a form when the user can enter the mark. After entering the mark and clicking the ‘Compute Grade’ button, the grade will be displayed in a message box, as shown in Figure 10.5

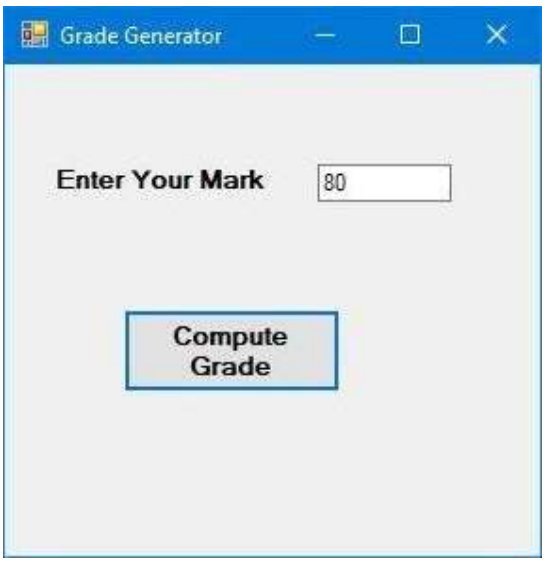


**Figure 10.5**

Clicking the OK produces a message box that shows the grade, as shown in Figure 10.6


**Figure 10.6**

**Summary**

- In section 10.1, you have learned about the conditional operators
- In section 10.2, you have learned about the logical operators
- In section 10.3, you have learned how to write code involving If….Then…Else

# Chapter 11 Using Select Case

- Learn how to write code for Select Case

The `Select Case` control structure is slightly different from the `If...ElseIf` control structure. The difference is that the `Select Case` control structure only make decision on one expression or dimension while the `If…ElseIf` statement control structure may evaluate only one expression, each `If….ElseIf` statement may also compute entirely different dimensions. `Select Case` is preferred when there exist multiple conditions.

The structure of the Select Case control structure in Visual Basic 2017 is as follows:

```
Select Case test expression
Case expression list 1
Block of one or more Visual Basic 2017 statements
Case expression list 2
Block of one or more Visual Basic 2017 Statements
.
.
Case Else
Block of one or more Visual Basic 2017 Statements
End Select
```

## Example 11.1: Examination Grades

This program displays the examination results based on the grade obtained. The test expression here is grade. In this program, we insert a textbox for entering the grade, rename it as txtgrade. Next, insert a label for display the result, rename it as LblResult. Lastly, we insert a button, rename it as BtnCompute then enter the following code:

```
Private Sub BtnCompute_Click( )
Dim grade As String
grade=txtgrade.Text
Select Case grade
Case "A"
  LblResult.Text="High Distinction"
Case "A-"
```

```
  LblResult.Text="Distinction"
Case "B"
  LblResult.Text="Credit"
Case "C"
  LblResult.Text="Pass"
Case Else
  LblResult.Text="Fail"
End Select
End Sub
```

When the user runs the program, enters grade and clicks the 'Compute' button, the output is as shown in Figure 11.1

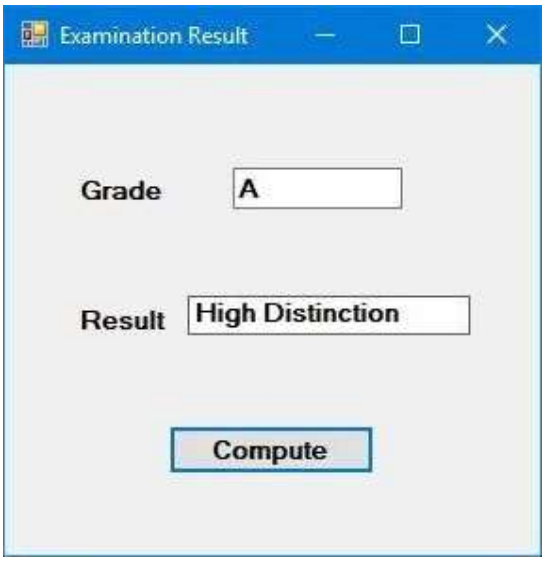


**Figure 11.1**

## Example 11.2 Using Case Is

This example is like the previous example, but we use the **`Case Is`** keyword and the conditional operator >= to compute the results.

```
Private Sub BtnCompute_Click( )
Dim mark As integer
mark = TxtMark.Text
Select Case mark
  Case Is >= 85
```

```
LblComment.Text= "Excellence"
  Case Is >= 70
LblComment.Text= "Good"
  Case Is >= 60
LblComment.Text = "Above Average"
  Case Is >= 50
LblComment.Text= "Average"
  Case Else
LblComment.Text = "Need to work harder"
End Select
End Sub
```

The output is shown in Figure 11.2

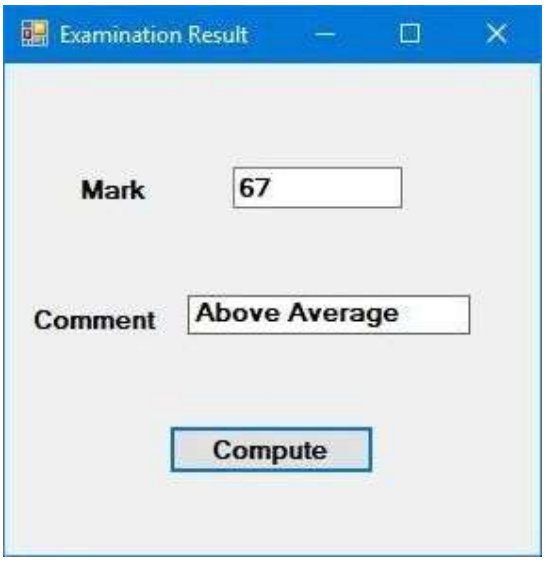


**Figure 11.2**

## Example 11.3 Select Case using a Range of Values

Example 11.2 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"
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 11.4 Examination Grade

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 is as shown in Figure 11.3

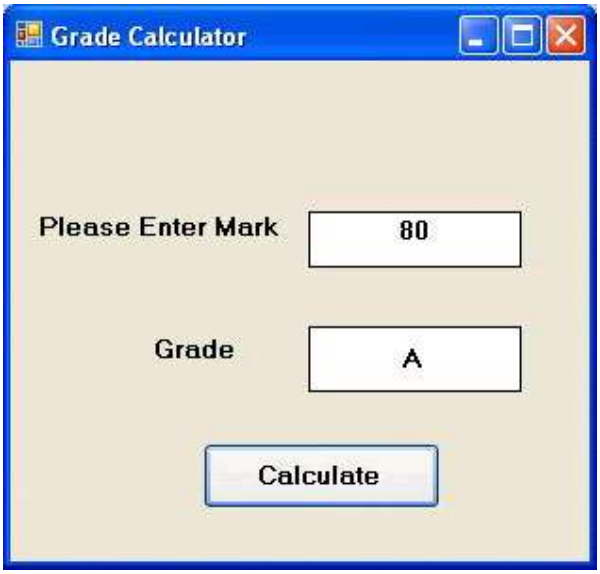


**Figure 11.3**

**Summary**

- In section 11.1, you have learned about the Select Case structure
- In section 11.2, you have learned how to write code using Select Case structure together with the conditional operators.

# Chapter 12 Looping

- Learn how to write code using For Next Loop
- Learn how to write code using Do Loop
- Learn how to write code using While End…While Loop

In programming, we often need to write code that does a job repetitively until a specific condition is met, this process is called looping. For example, we can design a program that adds a series of numbers until the sum exceeds a specific value, or a program that asks the user to enter data repeatedly until he or she enters the word 'Finish'. There are three types of Loops, the **`For...Next`** loop, the **`Do...loop`** and the **`While...End While`** loop

## 12.1 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 2017 statements
Next
```

To exit a **`For...Next`** Loop, you can place the **`Exit For`** statement within the loop; and it is usually used together with the **`If...Then`** statement.

### Example 12.1 Creating a Counter

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

* This loop will enter number 1 to 10 into the list box.

### Example 12.2 Sum of Numbers

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

* This loop will calculate the sum of the numbers as follows:

sum=0+10+20+30+40+……

### Example 12.3 Step-down For Next Loop

```
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 12.4 Demonstrate Exit For

```
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.

## 12.2 Do Loop

The **`Do Loop`** structures are

a)

```
Do While condition
Block of one or more Visual Basic 2017 statements
Loop
```

b)

```
Do
Block of one or more Visual Basic 2012 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
```

Sometimes we need exit to exit a loop prematurely because a specific

condition is fulfilled. The syntax to use is `Exit Do`. Let us examine the following examples:

## Example 12.5 Do While…Loop

```
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 12.6 Summation of Numbers

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As 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 12.1

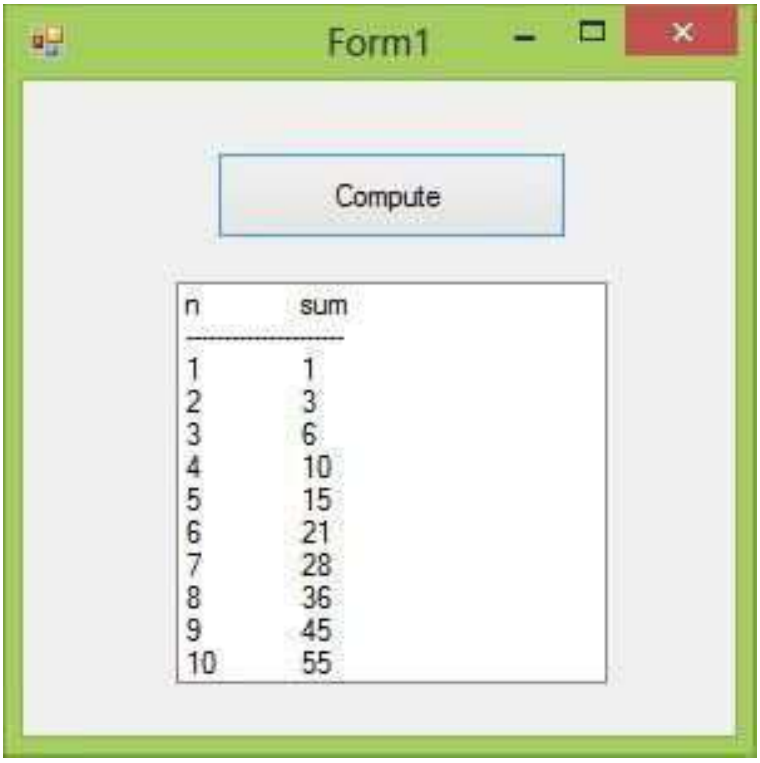


**Figure 12.1**

## 12.3 While…End While Loop

The structure of a `While....End While` Loop is like the `Do` Loop.

```
While conditions
Visual Basic 2017 statements
End While
```

The loop is illustrated in Example 12.3

### Example 12.3 Demonstrating While…End While Loop

```
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
```

**Summary**

- In section 12.1, you have learned how to write code using the For…Next Loop
- In section 12.2, you have learned how to write code using the Do Loop
- In section 12.3. you have learned how to write code using the While...End While Loop

# Chapter 13 Sub Procedures

- Understand the concept of sub procedure
- Learn how to write code for a sub procedure

## 13.1 What is a Sub Procedure

A 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. Sub procedures is a program code by itself and but it is not an event procedure because it is not associated with a runtime procedure or a control. It is called by the main procedure to perform a specific task.

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 13.1 A Sub Procedure that Adds Two Numbers

This sub procedure adds 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 as shown in Figure 13.1


**Figure 13.1**

## Example 13.2: Password Cracker

This password cracking program that can generate possible passwords and then compares each of them with the actual password. 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 password generating procedure **`generate ()`** and it is called by the **`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 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 the user clicks on the Generate button. `Rnd` is a 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 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 is as follows:

```
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
```

The output is shown in Figure 13.2


**Figure 13.2: Passwords generating phase**


**Figure 13.3: Login Phase**

**Summary**

- In section 13.1, you have learned about the concept of a sub procedure
- In section 13.2, you have learned how to write code for a sub procedure

# Chapter 14 Creating Functions

- Understand the concept of Function
- Learn how to create user-defined functions

A function is like a sub procedure in the sense that both are called by the main procedure to fulfil specific tasks. However, there is one difference; a function returns a value whereas 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.

## 14.1 Creating User-Defined Functions

To create a user defined function, 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 whereas the keyword Private indicates that the function is only applicable to a specific module or procedure. Argument is a parameter that can pass a value back to the function. You can include as many arguments as you like.

### Example 14.1: BMI Calculator

This BMI calculator is a program that calculates the body mass index, or BMI of a person based on the body weight in kilogram and the body height in meter. BMI is 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

**The Code**

```
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
```

The output is shown in Figure 14.1

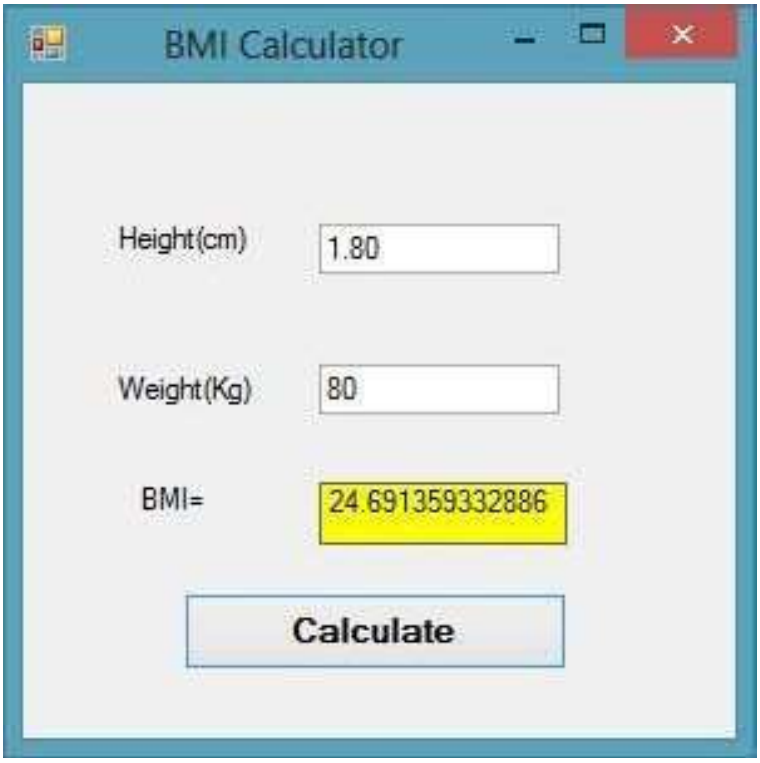


**Figure 14.1**

## Example 14.2 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 account for a specific period, you will earn an interest based on the compound interest rate computed periodically. This profit 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 us 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**

```
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
```

The Output is shown in Figure 14.2

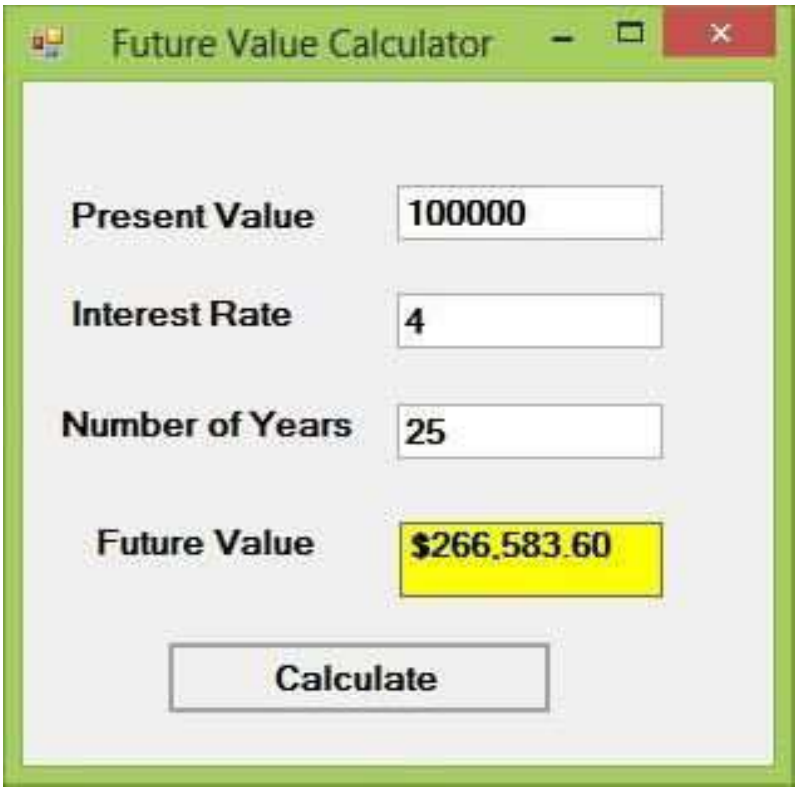


**Figure 14.2**

## 14.2 Passing Arguments by Value and by Reference

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

## Example 14.3 ByRef and ByVal

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**

```
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")
```