# AL MUTHANNA UNIVERSITY

# COLLEGE OF ENGINEERING

# DEPARTMENT OF CHEMICAL ENGINEERING

# Computer Programming I

# (Theory & Practical)

# 2nd Year

# Session: 2019-2020

# 1st Semester

# Visual Basic 2012

## 1. Introduction

Microsoft launched Visual Basic 2012 in the year 2012. It is a fully object-oriented programming language implemented on the .NET Framework. Similar to the earlier version of VB.NET programming languages, VB2012 is integrated with other Microsoft Programming languages in an Integrated Development Environment "IDE" called **Visual Studio Express 2012**.

Although Microsoft had launched a few newer versions of Visual Studio until the latest Visual Studio 2017, you can still download the older version Visual Studio 2012 Express Edition from the following link:

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

From the download link as shown in Figure 1.1, select the free Visual Studio Express 2012 for Windows Desktop.

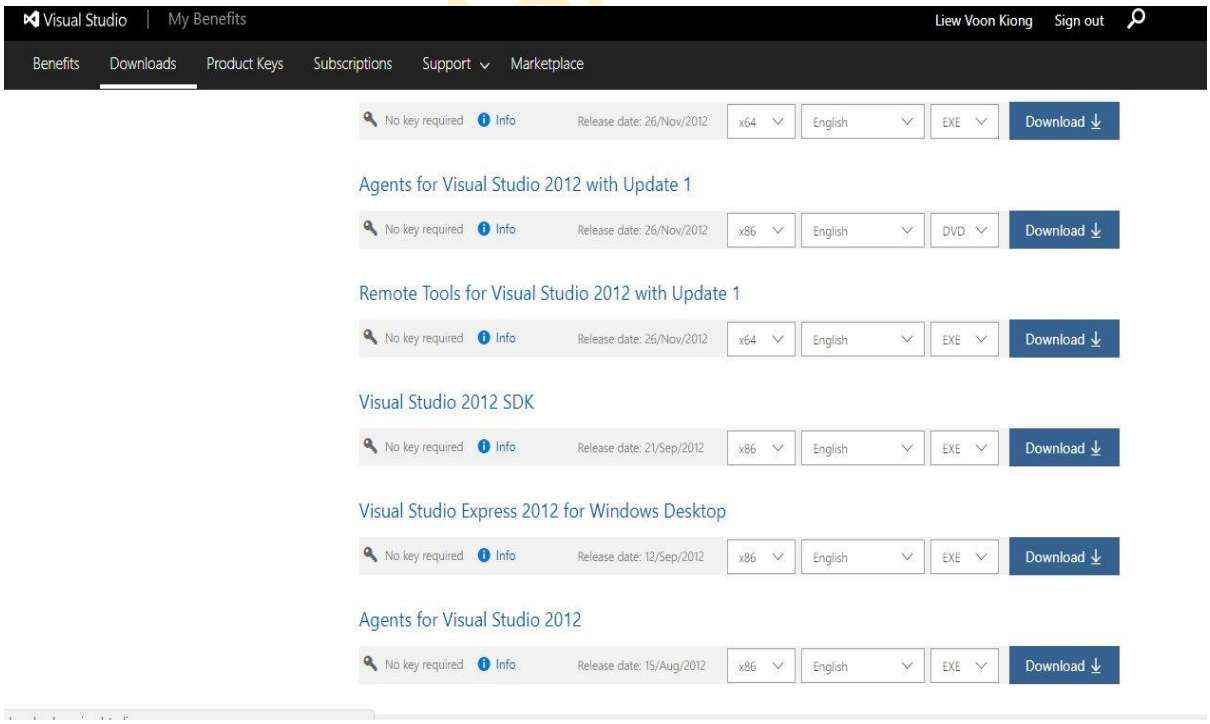



**Figure 1.1 Visual Studio Express 2012 Download Link**

When you launch Visual Studio Express 2012, the start page will appear, as shown in Figure 1.2 below.

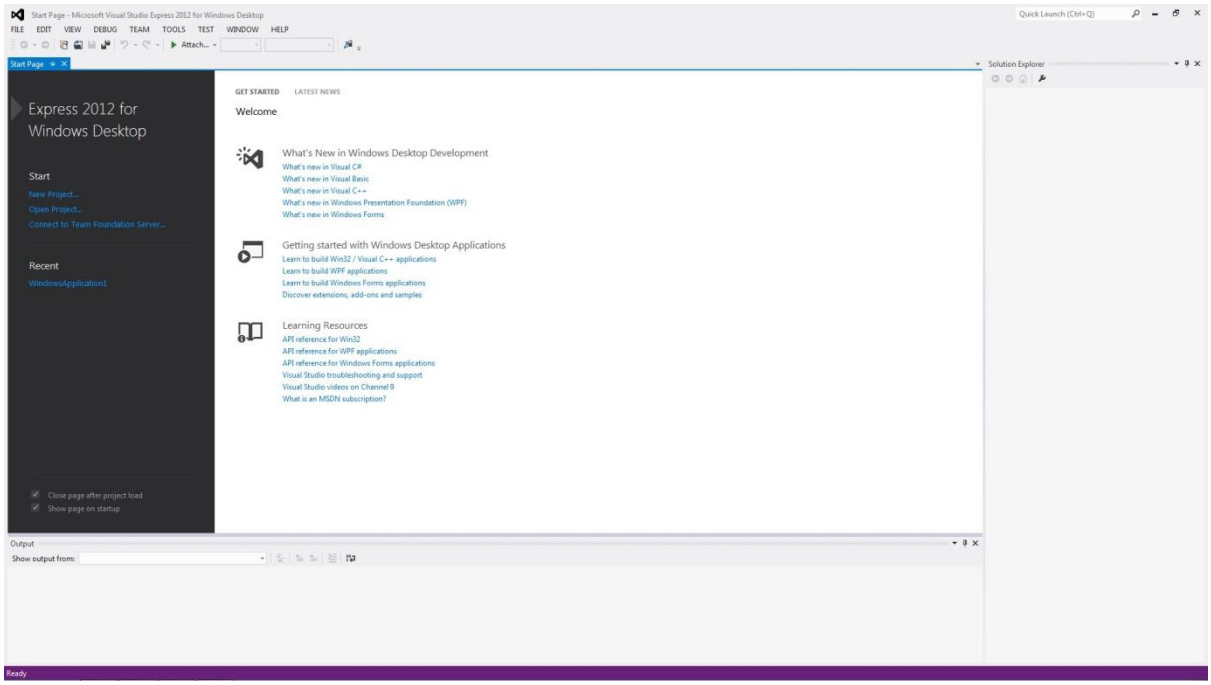



**Figure 1.2: Visual Studio 2012 Start Page**

To start a new Visual Studio Express 2012 project, simply click on New Project to launch the Visual Studio New Project page, as shown in Figure 1.3

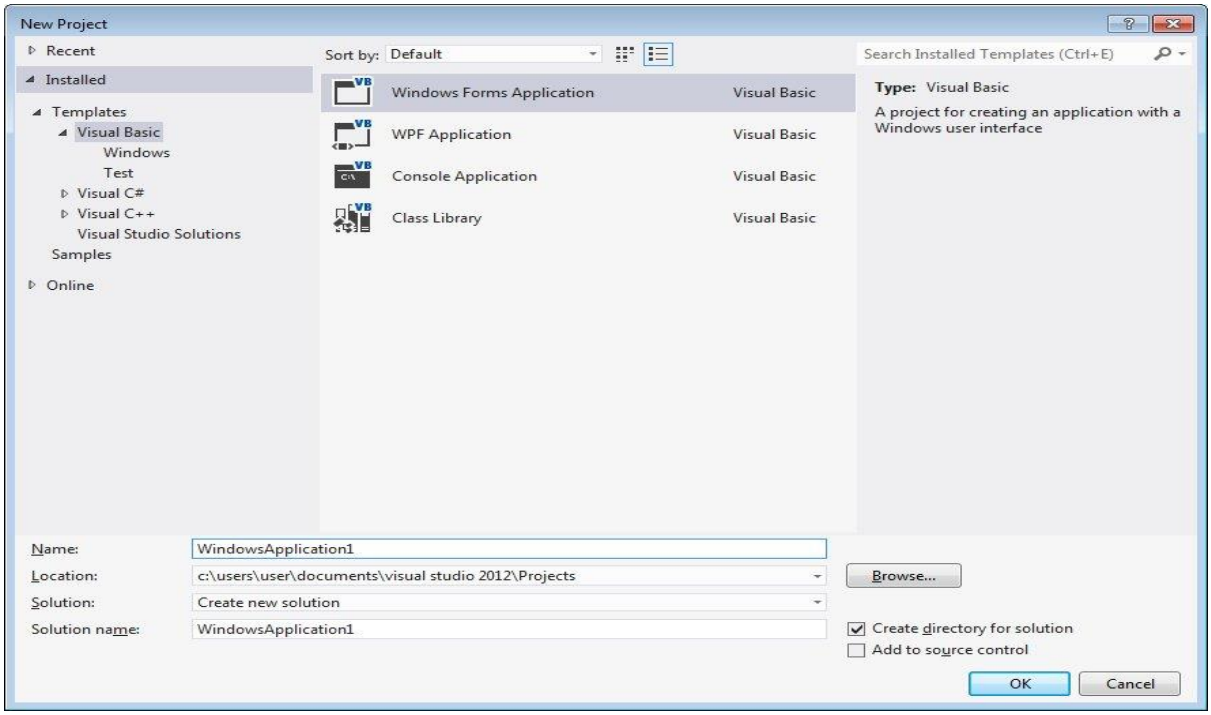



**Figure 1.3: Visual Studio 2012 Project Page**

The New Project Page comprises three templates, Visual Basic, Visual C# and Visual C++. Since we are going to learn Visual Basic 2012, we shall select Visual Basic. Visual Basic 2012 offers you four types of projects that you can create. As we are going to learn to create Windows Applications, we will select Windows Forms Application.

At the bottom of this dialog box, you can change the default project name WindowsApplication1 to some other name you like, for example, MyFirstProgram. After you have renamed the project, click OK to continue. The following IDE Windows will appear; it is similar to Visual Basic 2010. The Toolbox is not shown until you click on the Toolbox tab. When you click on the Toolbox tab, the common controls Toolbox will appear.

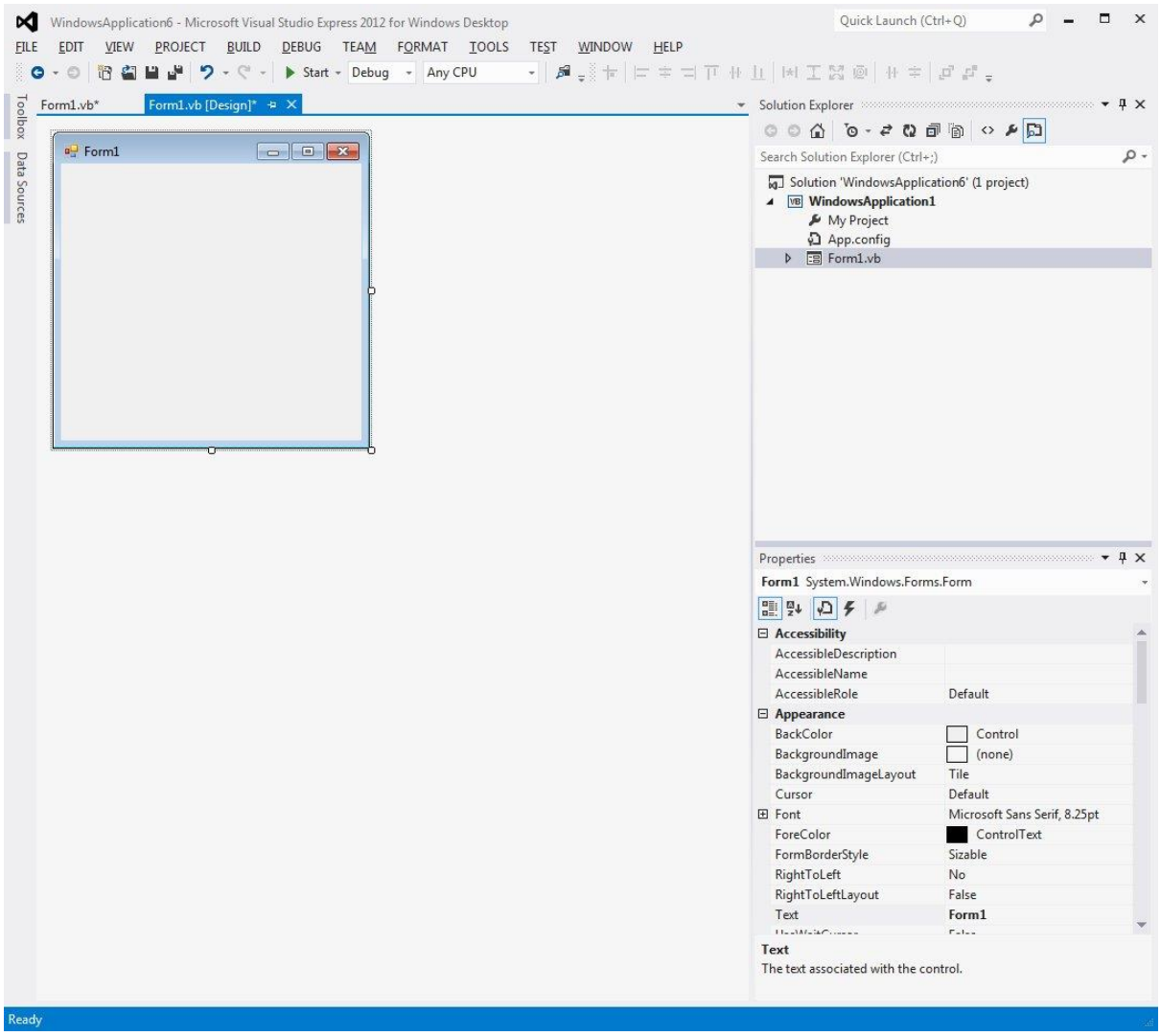



**Figure 1.4 Visual Basic 2012 IDE**

Visual Basic Express 2012 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 allow a programmer to develop his or her VB programs. The toolbox is shown in Figure 1.5.

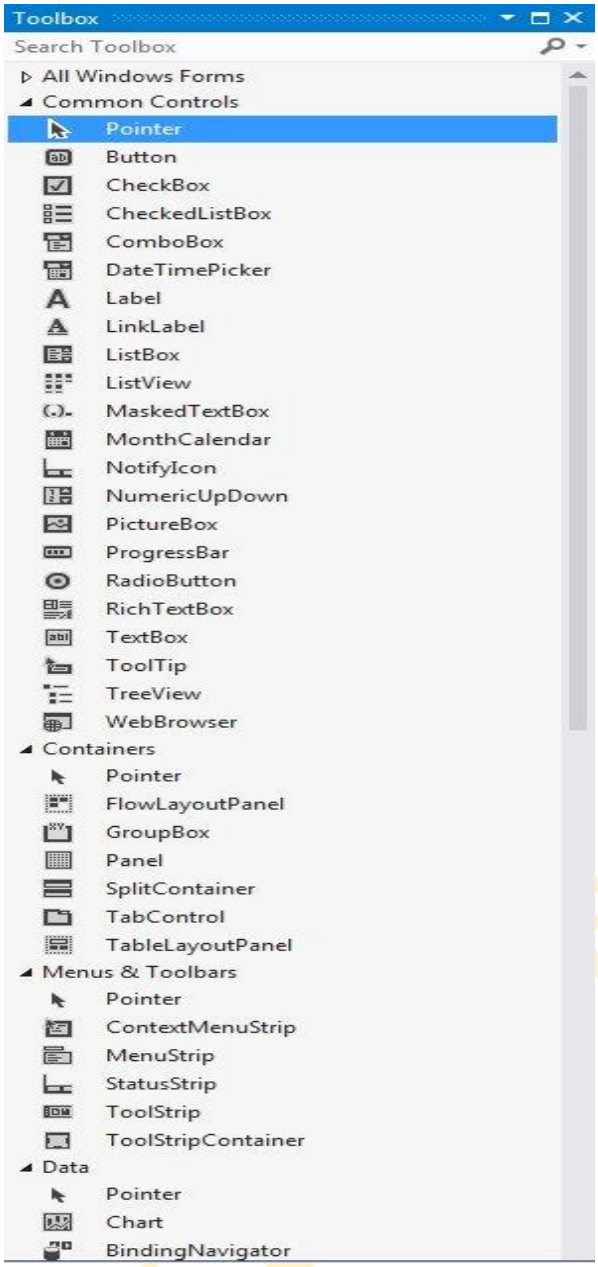



**Figure 1.5 Visual Basic 2012 Toolbox**

Now, we shall proceed to show you how to create your first program. First, change the text of the Form to My First Program in the properties window, it will appear as the title of the program. Next, insert a Button and change its text to OK. The design interface is shown in Figure 1.6

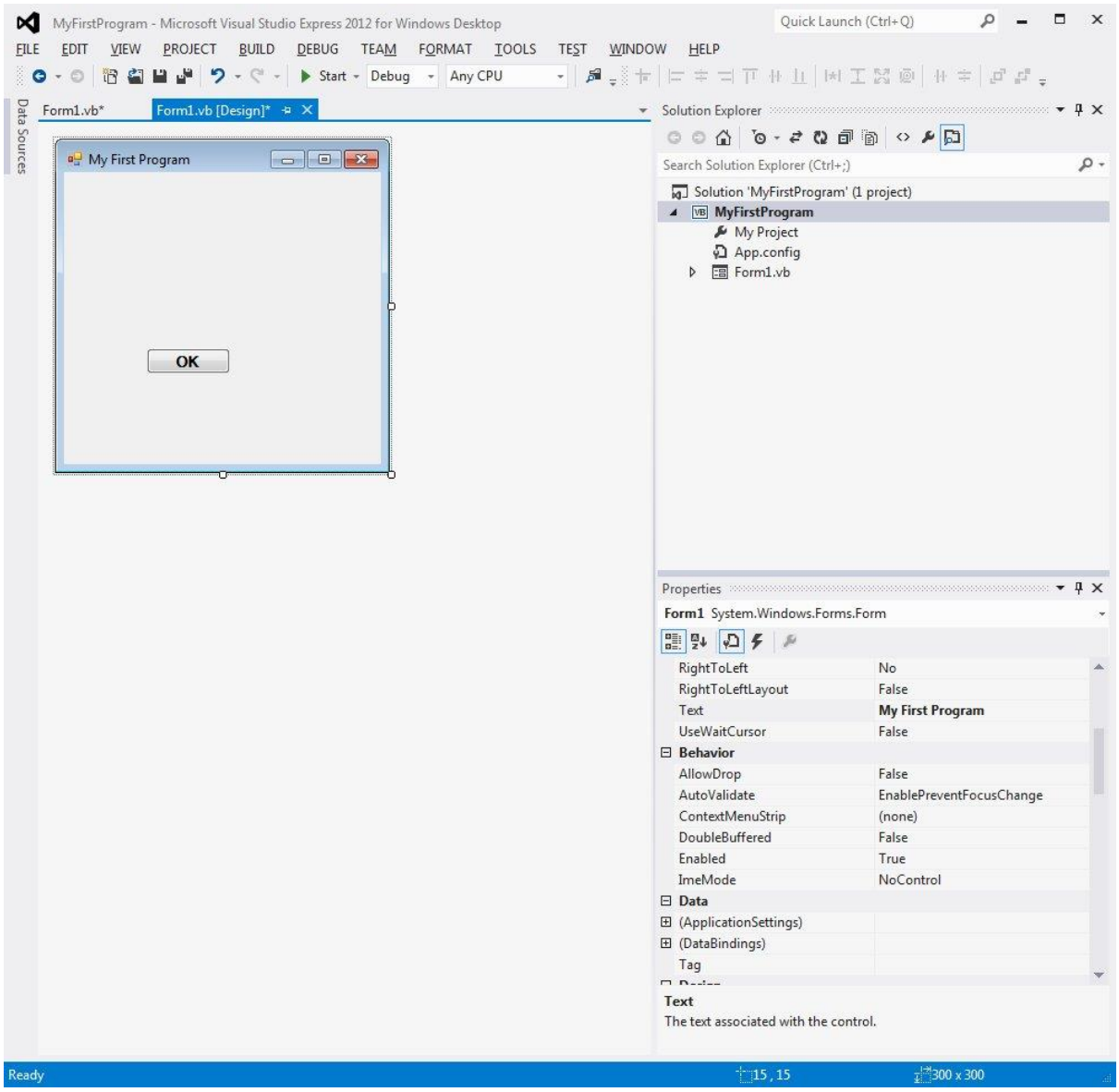



**Figure 1.6 The Design Interface**

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 1.7.

```
MsgBox("My First Visual Basic 2012 Program")
```

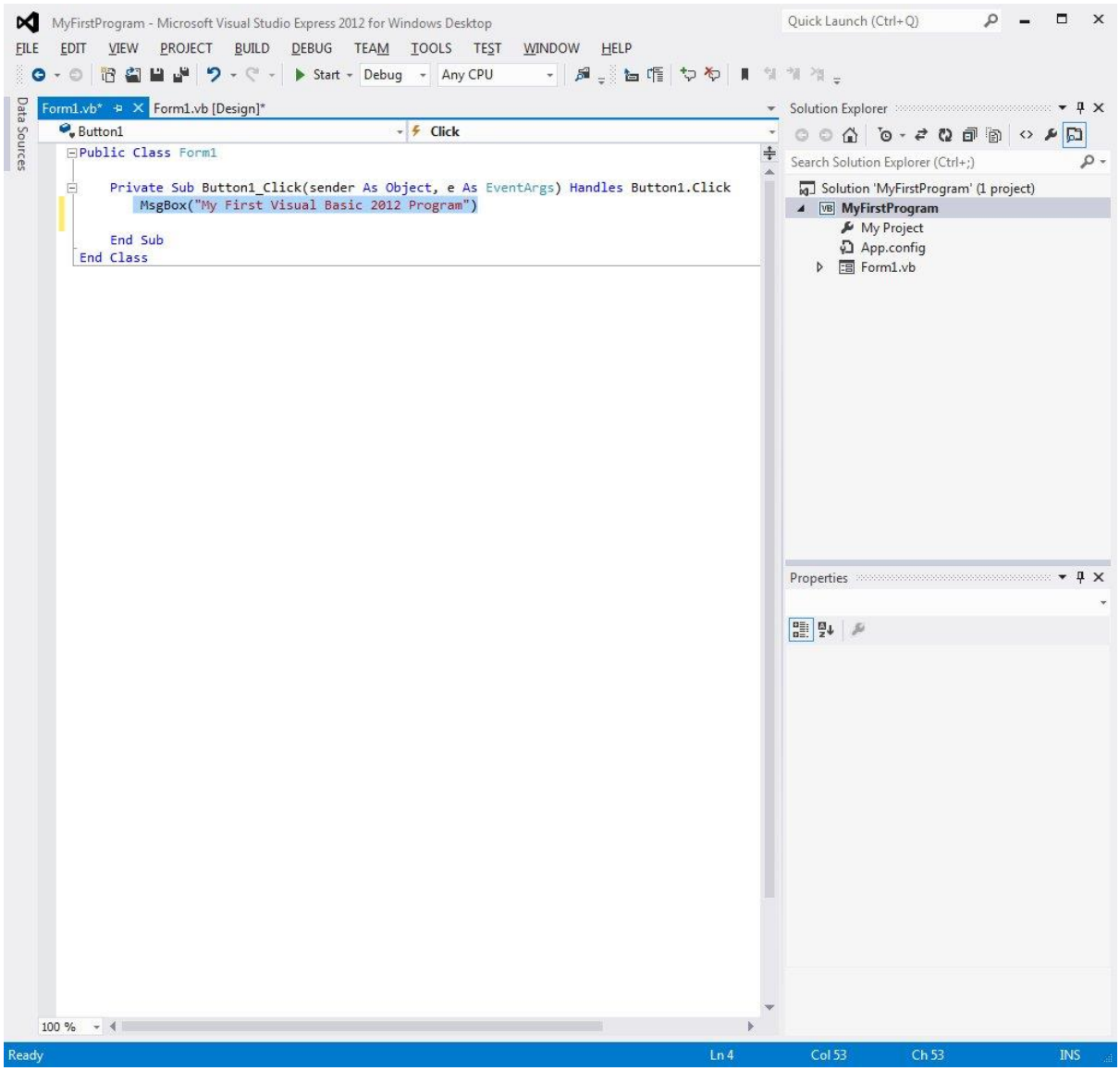



**Figure 1.7 The Code Window**

Now click on the Start on the toolbar to run the program then click on the OK Button, a dialog box that displays the ”My First Visual Basic 2012 Program” message will appear, as shown in Figure 1.8.

**Figure 1.8 The Message Box**

The function MsgBox is a built-in function of Visual Basic 2012 and it will display the text enclosed within the brackets.

## 2. Working with Controls

### 2.1 What are Controls

Controls in Visual Basic 2012 are objects that can be placed on the Form to perform various tasks. The toolbox contains the controls, which are categorized into Common Controls, Containers, Menus, Toolbars, Data, Components, and Printings and Dialogs. At the moment, we will focus on the common controls. Some frequently used common controls are Button, Label, ComboBox, ListBox, PictureBox, and TextBox. To insert a control into your Form in Visual Basic 2012 IDE, drag the control from the toolbox and drop it onto the Form, or double click the control. You can reposition and resize it as you like.
When you click on the Toolbox tab, the common controls Toolbox will appear.

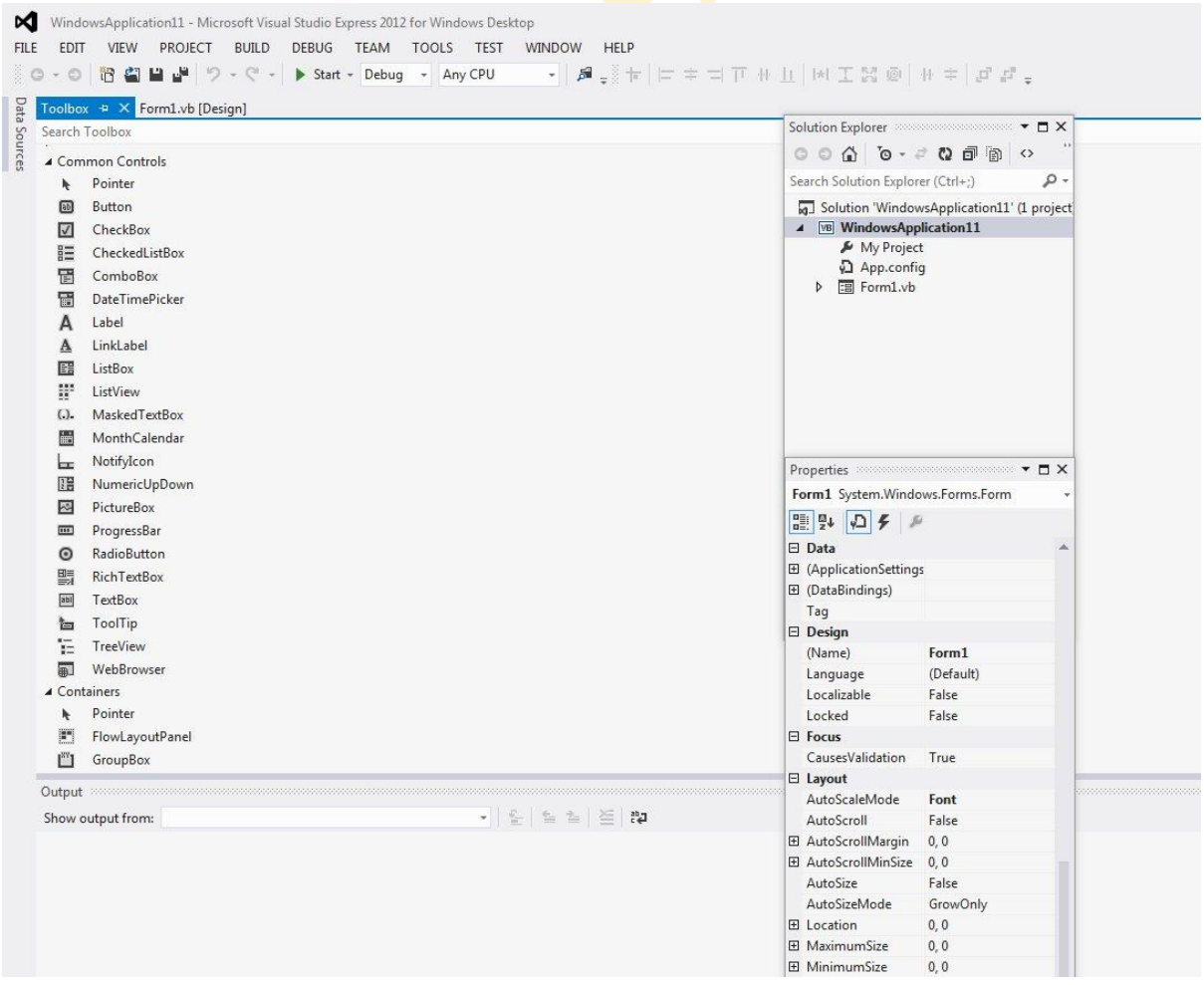


**Figure 2.1 Toolbox, Solution Explorer, and Properties windows**

## 2.2 Creating Your First Application

To create your first application in Visual Basic 2012, drag the Button control into the Form, and change its default Text 'Button1' to OK in the properties window, the word OK will appear on the Button in the Form, as shown in Figure 2.2:

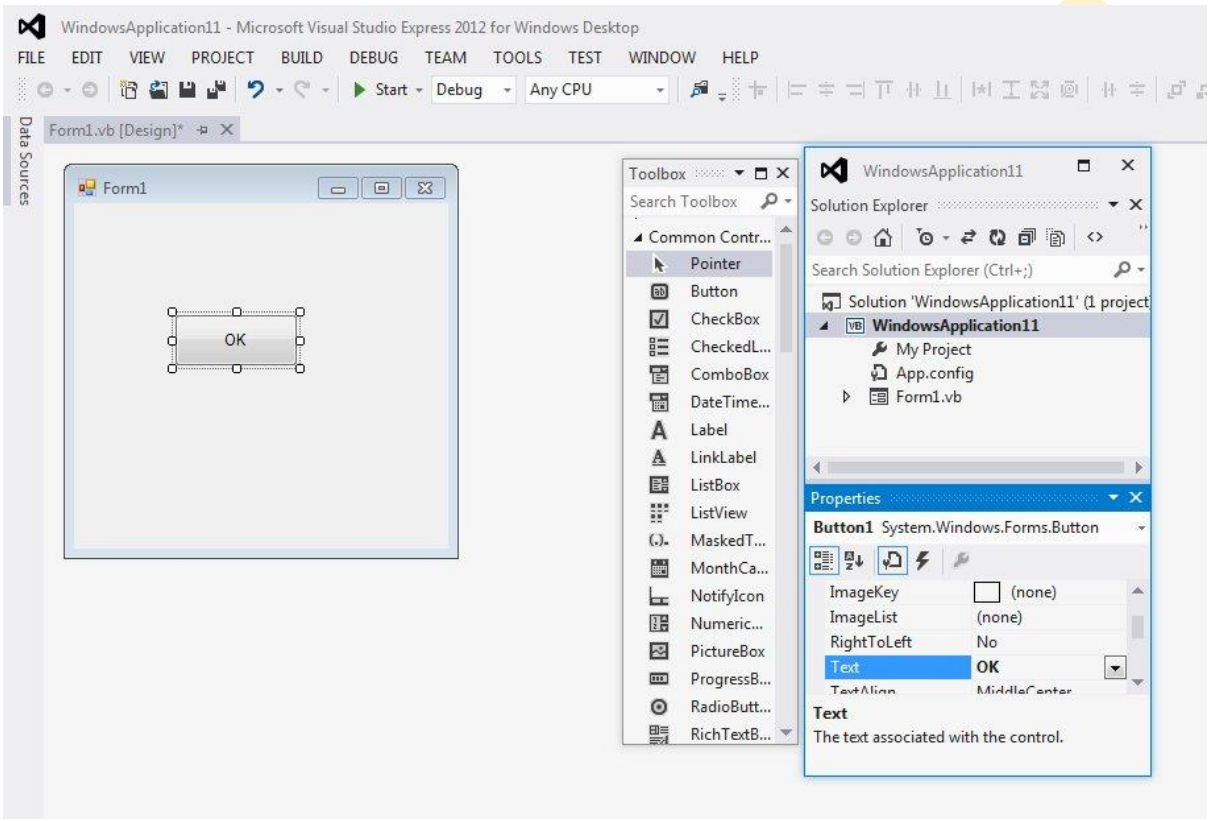



**Figure 2.2 Text Property**

Now click on the OK Button and the code window appears. Enter the code as follows (example1):

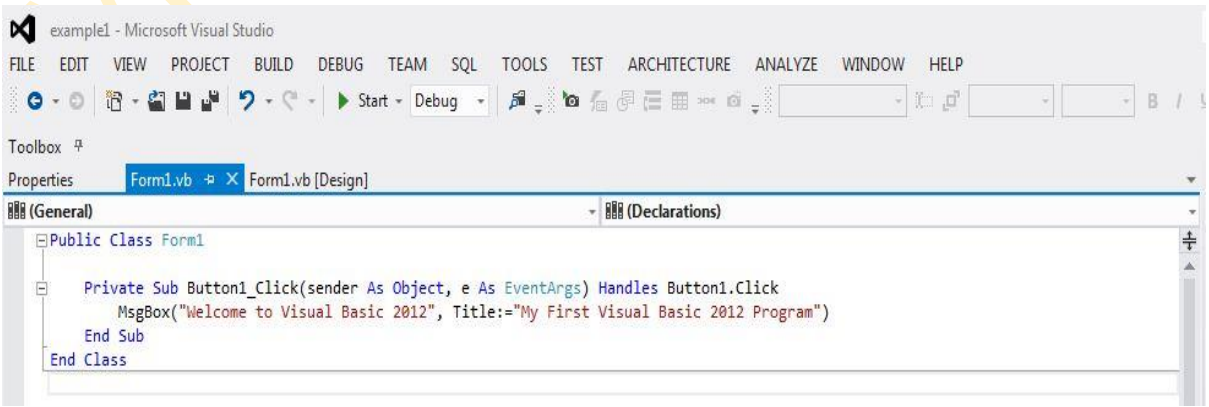


**Figure 2.3 Code window**

When you run the program "click the start Button" then click on the OK Button, a dialog box will appear and displays the "Welcome to Visual Basic 2012" message, as shown in Figure 2.4. The argument Title: is to assign the title of the dialog box.:


**Figure 2.4 Message Box**

## 2.3 Using the Text Box

Next, we will show you how to create a simple calculator that adds two numbers using the TextBox control. In this program, you insert two Text Boxes, three Labels, and one Button. The two Text Boxes are for the user to enter two numbers, Label1 is to display the addition operator "+" and Label2 is to display the equal sign "=", and Label3 is to display the result. Now change the Text property of Button1 to "Calculate", then double click on this "Calculate" Button and enter the following code in the code window (example2):

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1, num2, product As Single
        num1 = CSng(TextBox1.Text)
        num2 = CSng(TextBox2.Text)
        product = num1 + num2
        Label3.Text = CStr(product)
    End Sub
End Class
```

When you run the program (press on "**Start**" button) and enter two numbers for example "1000" in the TextBox1 and "2000" in the TextBox2,

pressing the calculate Button will give you the addition result of the two numbers "3000" as shown in Figure 2.5.

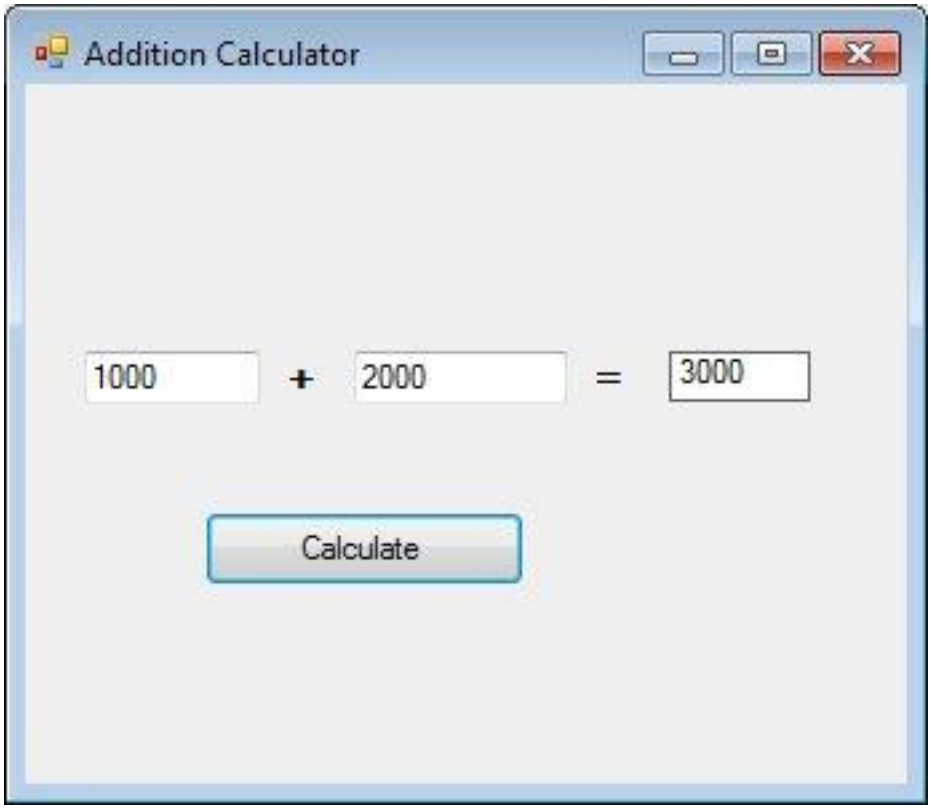



**Figure 2.5 Addition Calculator**

## 3. Working with Control Properties

### 3.1 The Control Properties

All controls in Visual Basic 2012 IDE have properties. By altering the properties of a control, we are able to customize its appearance and how it responds to an event. Basically, you can set the properties of the controls in the properties window of Visual Basic 2012 IDE at design time or at runtime. Figure 3.1 is the typical properties window for a Form. It refers particularly to the interface of the first application you have created in the previous lesson.

In the properties window, the item appears at the top part is the object currently selected (in Figure 3.1, the object selected is Form1). At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available.

You may also alter other properties of a control such as font, location, size, foreground color, background color, MaximizeBox, MinimizeBox and etc. You can also change the properties of the object at runtime to give special effects such as change of color, shape, animation effect and so on.

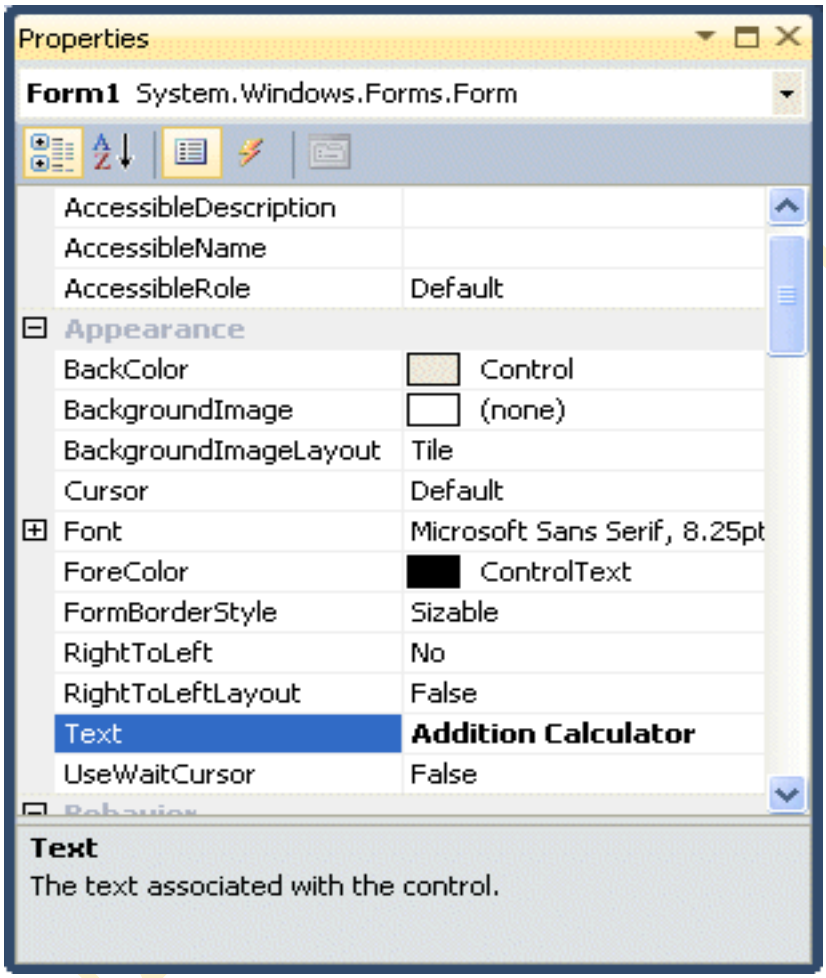



**Figure 3.1 Control Properties**

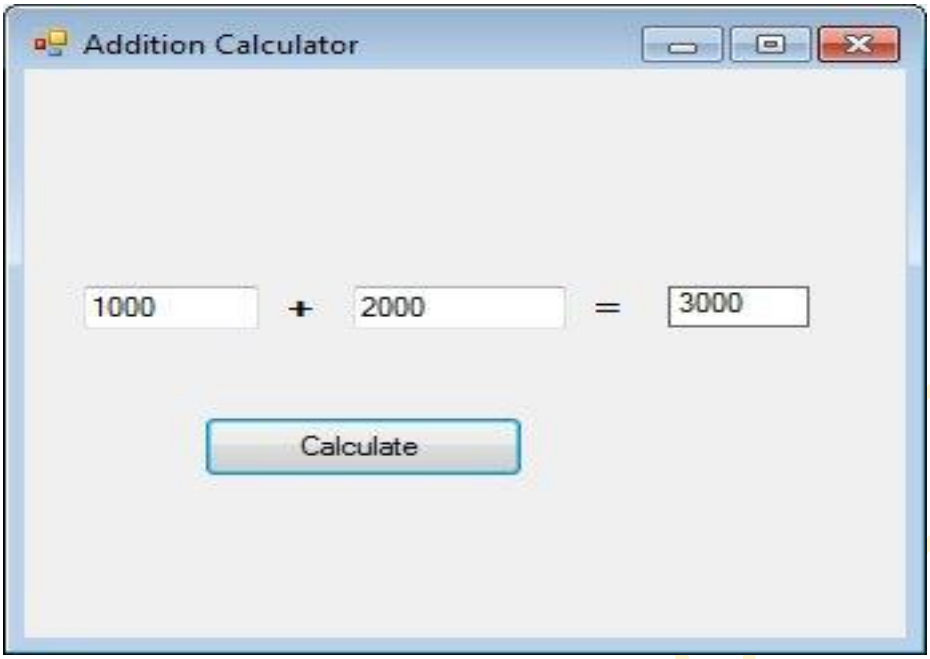



**Figure 3.2 Addition Calculator**

### 3.1.1 Customizing the Form

We can customize a **Form** by changing the values in the properties window. First of all, let's change the title of the Form. The title of the Form is defined by the Text property and its default name is Form1. To change the Form's title to any name that you like, simply click in the box on the right of the Text property and type in the new name, in this example, the title is Addition Calculator. After this, you can notice that the Form's title will appear on top of the window.

In addition, we can customize the Form's color. The following code will change the Form color to yellow every time the Form is loaded. Visual Basic 2012 uses RGB (Red, Green, Blue) to determine the colors. The RGB code for Magenta is (255,0,255) while **Me** in the code refer to the current Form and BackColor is the property of the Form's background color. The Formula to assign the RGB color to the Form is **Color.FormArgb(RGB code)**. The event procedure is as follows (example3):

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.BackColor = Color.FromArgb(255, 0, 255)
    End Sub
End Class
```

You may also use the following procedure to assign the color at runtime.

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.BackColor = Color.Magenta
    End Sub
End Class
```

Both procedures above will load the Form with a magenta background as in Figure 3.3:


**Figure 3.3 Form Background Color**

In Figure 3.4 some of the common colors and the corresponding RGB codes. You can always experiment with other combinations, but remember the maximum number for each color is 255 and the minimum number is 0.

| Color | RGB code | Color | RGB code | Color | RGB code |
|---|---|---|---|---|---|
| | 255,0,0 | | 255,255,0 | | 255,166,0 |
| | 0,255,0 | | 0,255,255 | | 0,0,0 |
| | 0,0,255 | | 255,0,255 | | 255,255,255 |

**Figure 3.4 Colors and RGB Code Combinations**

The following is another program that allows the user to enter the RGB codes into three different TextBoxes and when he or she clicks the "Display Color" Button, the background color of the Form will change according to the RGB codes, see Figure 3.5. So, this program allows users to change the color properties of the Form at runtime (example4).

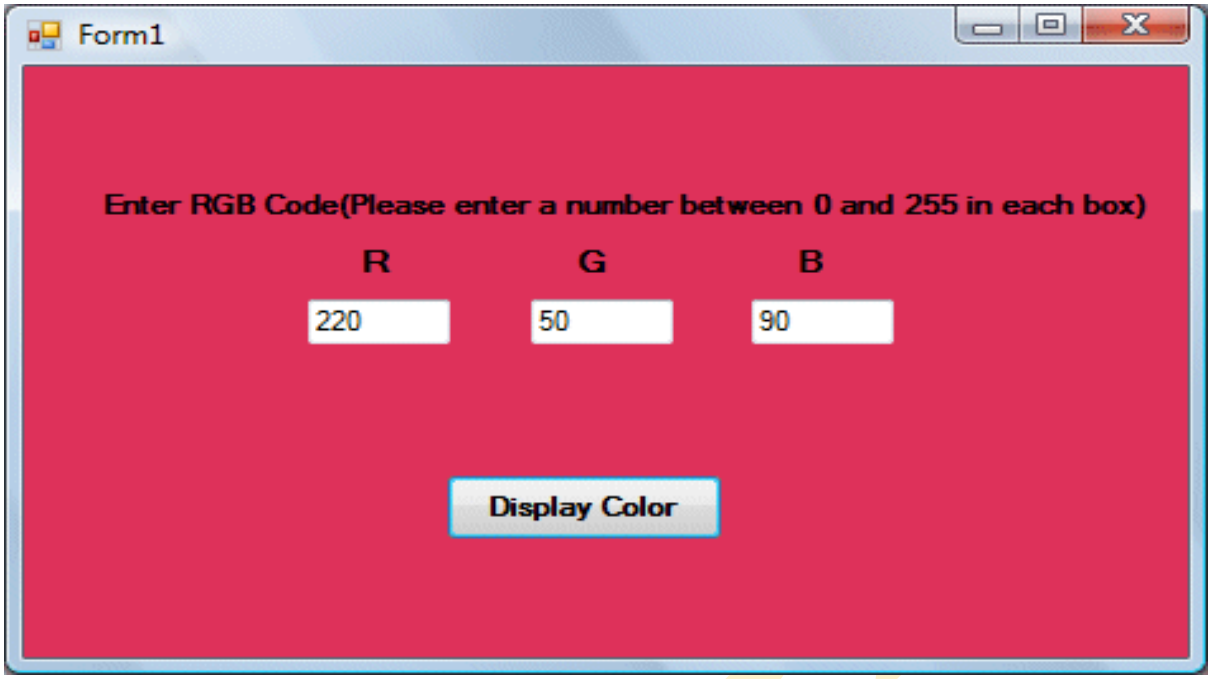



**Figure 3.5 RGB Code Program**

### The code of example 4

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim rgb1, rgb2, rgb3 As Integer
        rgb1 = CInt(TextBox1.Text)
        rgb2 = CInt(TextBox2.Text)
        rgb3 = CInt(TextBox3.Text)
        Me.BackColor = Color.FromArgb(rgb1, rgb2, rgb3)
    End Sub
End Class
```

## 4. Object Oriented Programming

Before learning how to write the program code in Visual Basic 2012, we think it is better for you to grasp the meaning of object-oriented programming.

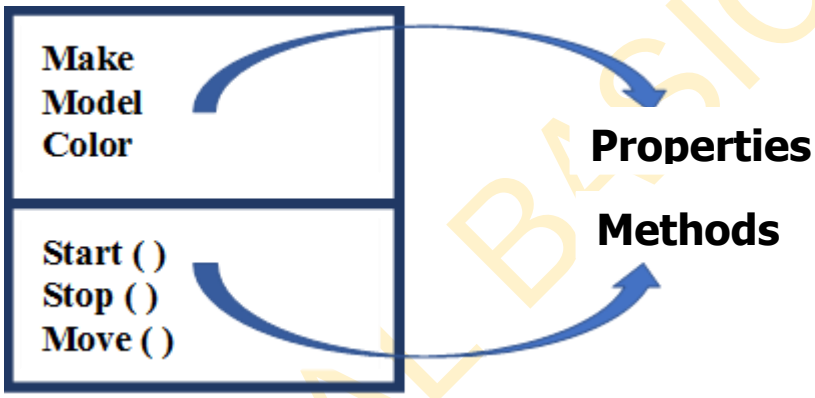
As we have mentioned earlier, Visual Basic 2012 is a full-fledged object-oriented programming language. What does it mean by object-oriented programming (OOP) language? For a programming language to qualify as an OOP language, it must have four core technologies as follows:

- **Encapsulation**
- **Abstraction**
- **Inheritance**

- **Polymorphism**

Before OOP we have procedural programming which divides the program into a several functions, so we have a data which stored in the variables and functions that operate on the data. This type of programming is very simple and straight forward. In this procedural programming as the program grow you will have many functions on all over the place. You will find yourself copying and pasting lines of code over and over, you may make a change to one function and then the second functions brake.

There is too much interdependency between all these functions it becomes a problematic. OOP came to solve these problems, in OOP we combine a group of related variables and functions into a unit, we call that unit an object, we refer to these variables as properties and the functions as methods. Here is an example; think of a car, a car is an object with properties such as make, model, and color and methods like start ( ), stop ( ), and move ( ).




### 4.1.1 Encapsulation

Encapsulation refers to the creation of self-contained modules that bind processing functions to the data. These user-defined data types are called classes. Each class contains data as well as a set of methods that manipulate the data. The data components of a class are called instance variables and one instance of a class is an object. For example, in a library system, a class could be a member, and John and Sharon could be two instances (two objects) of the library class. In OOP we group related variables and functions that operate on them into objects and this what we call encapsulation.

### 4.1.2 Abstraction

We can hide some of the properties and methods from the outside and this gives us a couple of benefits; First, is making the interface of those objects simpler. Using and understanding an object with a few properties and methods is easier than object with several properties and methods. The second benefit is to help us reduce the impact of change, if we change these inner or private methods none of these changes will leak to outside because we don't have any code that touch these methods outside the containing object. If we delete a method or change its parameters but none of these changes will impact the rest of applications code, so with abstraction we reduce the impact of changes.

### 4.1.3 Inheritance

Classes are created according to hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, then only the processing and data associated with that unique step needs to be added. Everything else about that step is inherited. The ability to reuse existing objects is considered a major advantage of object technology. So inheritance is a mechanism that allows you to eliminate a redundant code.

### 4.1.4 Polymorphism

Poly means many and morphism means form, so polymorphism means many forms. In OOP polymorphism is a technique that allows you to get with long switch/case statements. It also allows new shapes to be easily integrated.

VB2012 is a fully functional Object Oriented Programming Language, just like other OOP such as C++ and Java. It focuses more on the data itself while VB6 and earlier versions focus more on the actions. VB6 and its predecessors are known as a procedural or functional programming language. Some other procedural programming languages are C, Pascal, and Fortran.

Visual Basic 2012 allows users to write programs that break down into modules. These modules will represent the real-world objects and are knows as classes or types. An object can be created out of a class and it is known as an instance of the class. A class can also comprise subclass. For example, the apple tree is a subclass of the plant class and the apple in your backyard is an instance of the apple tree class. Another example

is student class is a subclass of the human class while your son John is an instance of the student class.

A class consists of data members as well as methods. In Visual Basic 2012, the program structure to define a Human class can be written as follows:

**Example 5:**

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        'Data Members
        Dim Name, Birthdate, Gender As String
        Dim Age As Integer
        Name = TextBox1.Text
        Birthdate = TextBox2.Text
        Gender = TextBox3.Text
        Age = CInt(TextBox4.Text)
        MsgBox(Name)
        MsgBox(Birthdate)
        MsgBox(Gender)
        MsgBox(Age)
    End Sub

End Class
```

## 5. Writing the Code

In this lesson, you will learn some basic theories about Visual Basic 2012 programming. We shall focus more on learning by doing. We will keep the theories short so that it would not be too difficult for beginners.

Visual Basic 2012 is an object-oriented and event driven programming language. Event-driven means the VB program runs in response to the user's action. The actions include clicking the command Button, entering text in a Text Box, choosing an item, closing the application and more.

### 5.1 The Event Procedure

Each event is related to an object; it is an incident that happens to the object due to the action of the user. A class has events as it creates an instant of a class or an object. When we start a windows application in Visual Basic 2012, 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**, as shown in the Form1 properties windows in Figure 5.1.

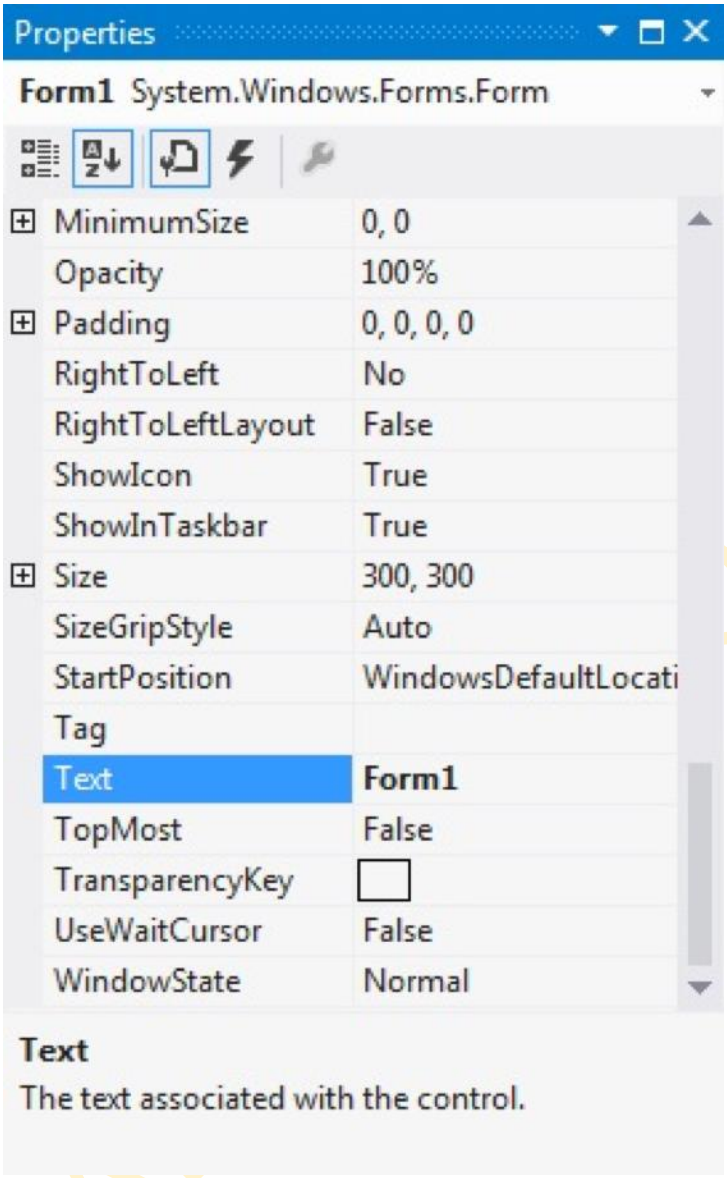



**Figure 5.1 Properties window**

To start writing code in Visual Basic 2012, double click on any part of the Form to go into the code window as shown in Figure 5.2. This is the structure of an event procedure. In this case, the event procedure is to load Form1 and it starts with **Private Sub** and 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. You don't have to worry the rest of the stuff at the moment, they will be explained in later lessons.

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub
End Class
```

There are other events associated with the Form1 class, such as click, cursorChanged, DoubleClick, DragDrop, Enter and more, as shown in the diagram Figure 5.2 (It appears when you click on the upper right pane of the code window)

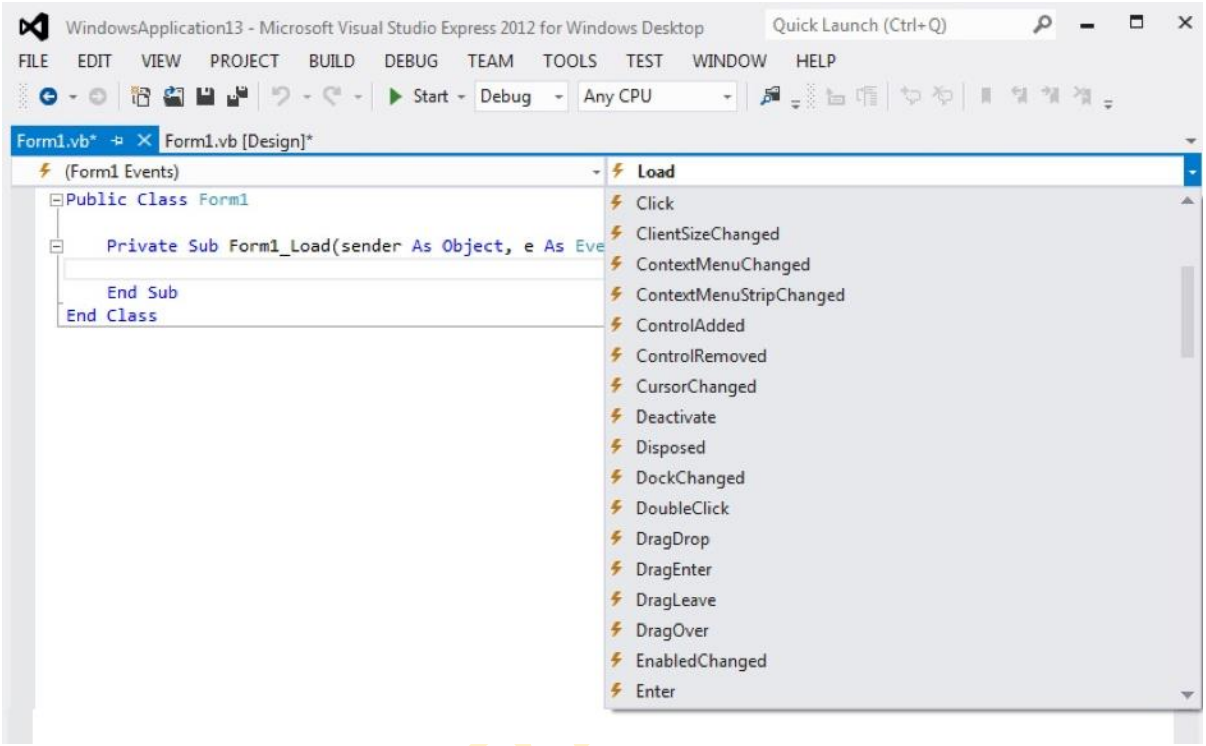



**Figure 5.2 Events associated with the Form1 class**

## 5.2 Writing the code

Now you are ready to write the code for the event procedure so that it will do something more than loading a blank Form. The code must be entered between **Private Sub…….End Sub**. Let's enter the following code :

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.Text = "My First VB2012 Program"
        Me.ForeColor = Color.LightGoldenrodYellow
        Me.BackColor = Color.RoyalBlue
    End Sub
End Class
```

The first line of the code will change the title of the Form to My First Visual Basic 2012 Program, the second line will change the foreground object to LightGoldenrodYellow( in this case, it is a Label that you insert into the

Form and change its text to Foreground) and the last line changes the background to RoyalBlue color. The equal operator = in the code is used to assign something to the object, like assigning yellow color to the foreground of the Form1 object (or an instance of Form1). Me is the name given to the Form1 class. We can also call those lines as Statements. So, the actions of the program will depend on the statements entered by the programmer.
The output is shown Figure 5.3 below:


**Figure 5.3 My First VB2012 Program**

Here is another example. In this project, you insert one Button into the Form and change its Label text to "Display Hidden Names". Click on this Button to enter the code window and key-in the following code:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim name1, name2, name3, names As String
        name1 = "George"
        name2 = "Michael Chan"
        name3 = "Norman"
        names = name1 & "," & name2 & "," & name3
        MsgBox(" The hidden names are " & names, Title:="Hidden Names")
    End Sub
End Class
```

The keyword Dim is to declare variables name1, name2 and name3, names as string, which means they can only handle text. The variable names is to accept the names that are joined together by the "&" signs as a single string. The function MsgBox is to display the string in a

message box, and the last argument "Hidden Names" is the title of the MsgBox function. The output is shown in Figure 5.4 below:

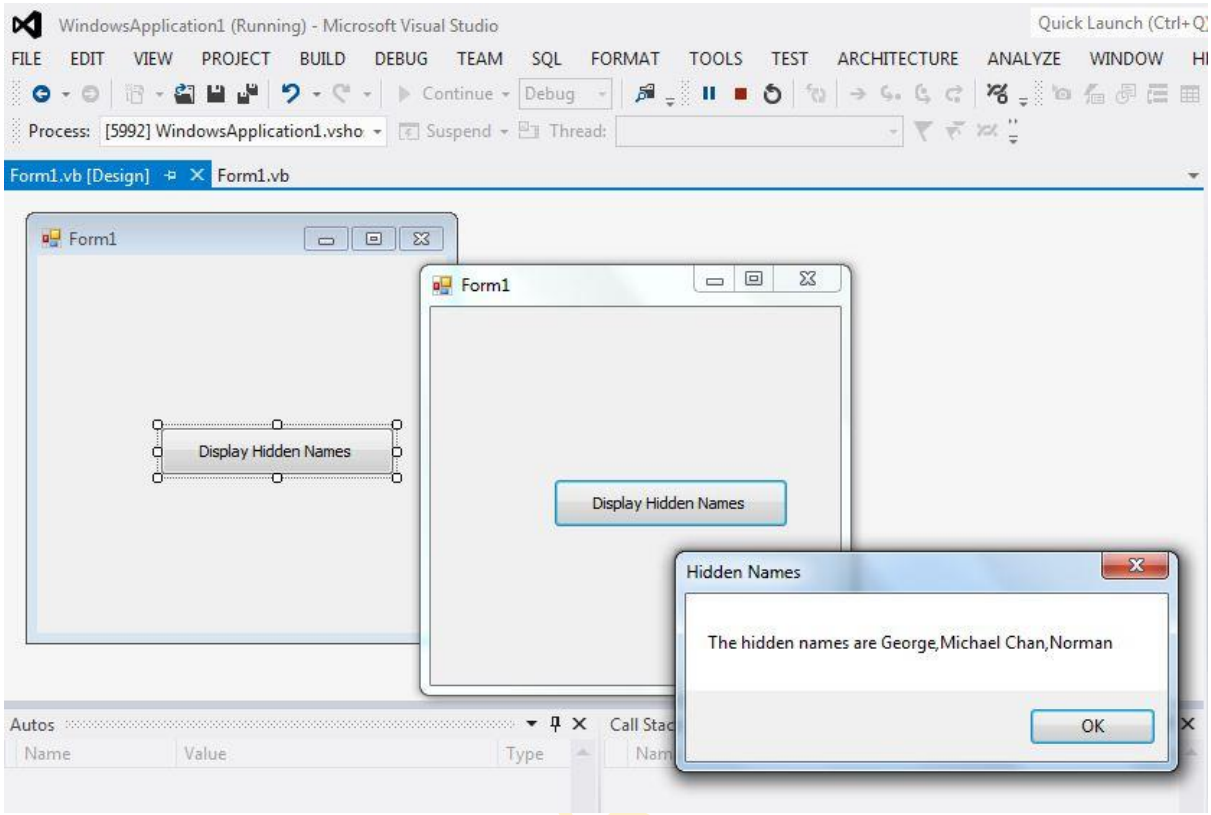



**Figure 5.4 Example of MsgBox**

## 6. Managing Data

Visual Basic programming often involves manipulation of all sorts of data. Among them, some can be calculated while others are in the Form of text, date, time and more. Visual Basic 2012 divides data into different types so that it is easier for the programmers to manage them.

### 6.1 Visual Basic 2012 Data Types

Visual Basic 2012 classifies data into two major data types, the **numeric** data type and the **non-numeric** data type.

#### 6.1.1 Numeric Data Types

Numeric data types are types of data that consist of numbers that can be computed mathematically. Some examples of numeric data types are examination marks, height, weight, the price of goods, monthly bills, fees and etc.

In Visual Basic 2012, numeric data are classified into **seven** types, depending on the range of values they can store. Calculations that do not require precision can use **Integer** or the **Long integer** in the computation. On the other hand, programs that require high precision calculation need to use **Single and Double** precision data types which also called **floating point numbers**. In addition, for currency calculation, you can use the **currency** data types. Lastly, if even more precision is required in performing a calculation, 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 of Values |
|---|---|---|
| Byte | 1 byte | 0 to 255 |
| Integer | 2 bytes | -32,768 to 32,767 |
| Long | 4 bytes | -2,147,483,648 to 2,147,483,648 |
| Single | 4 bytes | -3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values |
| Double | 8 bytes | -1.79769313486232E+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232E+308 for positive values |
| Variant(numeric) | 16 bytes | Any value as large as double |
| 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 +/-7.9228162514264337593543950335 |

### 6.1.2 Non-numeric Data Types

Non-numeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. These types of data comprise text or string data types, the Date data types, the Boolean data types, the Object data type and the Variant data type. We can summarise the data types as shown in Table 6.2.

**Table 6.2 Non-numeric Data**

| Data Type | Storage | Range of Values |
|---|---|---|
| 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 3,9999 |
| Boolean | 2 bytes | True or False |
| Object | 4 bytes | Any embedded object |
| Variant(text) | Length+22 bytes | Same as variable-length string |

### 6.1.3 Suffixes for Literals

Literals are values that you assign to a data. Usually, we add a suffix behind a literal so that Visual Basic 2012 can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data. Some of the suffixes are displayed in Table 6.3.

**Table 6.3 Suffixes for Literals**

| Suffix | Data Type |
|---|---|
| **&** | **Long** |
| **!** | **Single** |

| # | Double |
|---|---|
| @ | Currency |

In addition, we need to enclose string literals within two quotations and we enclose 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 = #12:00:00 AM#
```

## 6.2 Managing Variables

A Variable is like a mailbox in the post office as the content of the variable changes every now and then, just like the mailbox. In Visual Basic 2012, variables are areas allocated by the computer memory to hold data. In addition, each variable must be given a name. To name a variable in Visual Basic 2012, you have to follow a set of rules.

### 6.2.1 Variable Names

The following are the rules when naming the variables in Visual Basic 2012

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

Some examples of valid and invalid variable names are displayed in Table 6.4

**Table 6.4 Naming the variables**

| Valid Name | Invalid Name |
|---|---|
| My_Car | My.Car |
| ThisYear | 1NewBoy |
| Long_Name_Can_Be_USED | He&HisFather *& Not allowed |

### 6.2.2 Declaring Variables

In Visual Basic 2012, we need to declare a variable by assigning a name and a relevant data type before using it. If you fail to do so, the program may encounter an error. Usually, we declare the variables in the general section of the code window using the Dim statement. The syntax to declare a variable is as follows:

**Dim** VariableName **As DataType**

### Example 6.1

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As 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
End Class
```

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

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

For string declaration, there are **two** possible Forms, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same syntax as example 6.1 above. However, for the fixed-length string, you have to use the syntax as shown below:

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

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

Example 6.2:

Dim yourName As String * 10

yourName can hold 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 (true or false) and etc. The following are some examples:

```
firstNumber = 100
secondNumber = firstNumber - 99
username = "John Lyan"
userpass.Text = password
Label1.Visible = True
Command1.Visible = False
Label4.Caption = TextBox1.Text
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber + ThirdNumber
```

## 6.3 Constants

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

### 6.3.1 Declaring a Constant

The syntax to declare a constant is:

**Const ConstantName As** Data Type **= Value**

## Example 6.3

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Const Pi As Single = 3.142
        Const Temp As Single = 37
        Const Score As Single = 100
    End Sub
End Class
```

## 7. Mathematical Operations

In Visual Basic 2012, we can write code to instruct the computer to perform mathematical operations. To write code for mathematical operations, we need to use arithmetic operators. Visual Basic 2012 arithmetic operators are very similar to the normal arithmetic operators, only with little variations. The plus and minus operators are the same while the multiplication operator uses the * symbol and the division operator uses the / symbol. The list of Visual Basic 2012 arithmetic operators is shown in Table 7.1.

**Table 7.1**

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

Example 7.1

In this program, insert two TextBoxes, four Labels, and a Button. Click the Button and enter the code as shown below. When you run the program, it will perForm the four basic arithmetic operations and displays the results on the four Labels.

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1, num2, sum, difference, product, quotient As Single
        num1 = CSng(TextBox1.Text)
        num2 = CSng(TextBox2.Text)
        sum = num1 + num2
        difference = num1 - num2
        product = num1 * num2
        quotient = num1 / num2
        Label1.Text = CStr(sum)
        Label2.Text = CStr(difference)
        Label3.Text = CStr(product)
        Label4.Text = CStr(quotient)
    End Sub
End Class
```

## Example 7.2

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

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim a, b, c As Single
        a = CSng(TextBox1.Text)
        b = CSng(TextBox2.Text)
        c = CSng((a ^ 2 + b ^ 2) ^ (1 / 2))
        Label1.Text = CStr(c)
    End Sub
End Class
```

## Example 7.3: BMI Calculator

A lot of people are obese now and it could affect their health seriously. Obesity has proven by the medical experts to be one of the main factors that bring 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 can create a VB BMI calculator. The BMI calculator can calculate the body mass index or BMI of a person based on the body weight in kilogram and the body height in meter. BMI can be calculated using the Formula weight/( height )^2, where weight is measured in kg and height in meter. If you only know your weight and height in lb and feet, then you need to convert them to the metric system (you could indeed write a VB program for the conversion).

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim height, weight, bmi As Single
        height = CInt(TextBox1.Text)
        weight = CInt(TextBox2.Text)
        bmi = CSng((weight) / (height ^ 2))
        Label1.Text = CStr(bmi)
    End Sub
End Class
```

The output is shown in the Figure 7.1. In this example, your height is 1.80m( about 5.11foot), your weight is 75 kg (about 168Ib), and your BMI is about 23.14815. The reading suggests that you are healthy. (Note; 1 foot=0.3048, 1 lb= 0.45359237 kilogram)

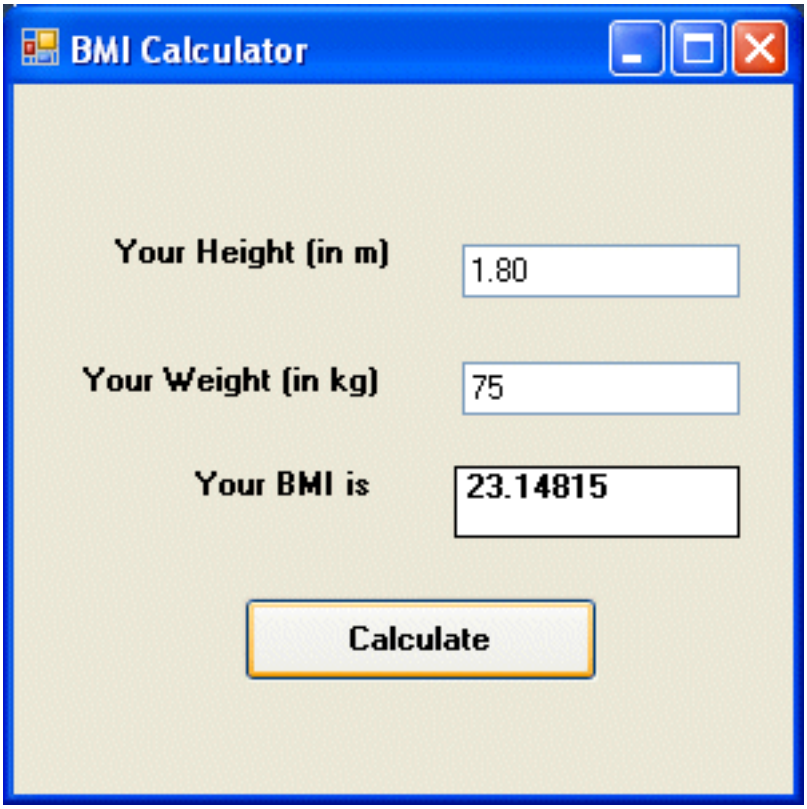



**Figure 7.1**

From the above examples, you can see that writing code that involve arithmetic operations is relatively easy. Here are more arithmetic projects you can try to program:

Area of a triangle
Area of a rectangle
Area of a circle
Volume of a cylinder
Volume of a cone
Volume of a sphere
Compound interest
Future value
Mean
Variance
Sum of angles in polygons
Conversion of lb to kg
Conversion of Fahrenheit to Celsius

# 8. String Manipulation

## 8.1 String Manipulation Using + and & signs.

In Visual Basic 2012, strings can be manipulated using the **&** sign and the **+** sign, both perform the string concatenation which means combining two or more smaller strings into larger strings. For example, we can join "Visual","Basic" and "2012" into "Visual Basic 2012" using "Visual"&"Basic"&"2012" or "Visual " + "Basic" + "2012", as shown in the Examples below

## Example 8.1(a)

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim text1, text2, text3, text4 As String
        text1 = "Visual"
        text2 = " Basic"
        text3 = " 2012"
        text4 = text1 + text2 + text3
        MsgBox("text4")
        MsgBox(text4)
    End Sub
End Class
```

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.

## Example 8.1(b)

```
Public Class Form1
    Private Sub Label1_Click(sender As Object, e As EventArgs) Handles Label1.Click
        Dim text1, text3 As String
        Dim Text2 As Integer
        text1 = "Visual Basic "
        Text2 = 2012
        text3 = text1 + Text2
        Label1.Text = text3
    End Sub
End Class
```

This code will produce an error because of data mismatch. However, using & instead of + will be all right.

```
Public Class Form1
    Private Sub Label1_Click(sender As Object, e As EventArgs) Handles Label1.Click
        Dim text1, text3 As String
        Dim Text2 As Integer
        text1 = "Visual Basic "
        Text2 = 2012
        text3 = text1 & Text2
        Label1.Text = text3
    End Sub
End Class
```

You can combine more than two strings to Form a larger string, like the following example:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim text1, text2, text3, text4, text5, text6 As String
        text1 = "Welcome"
        text2 = " to"
        text3 = " Visual"
        text4 = " Basic"
        text5 = " 2012"
        text6 = text1 + text2 + text3 + text4 + text5
        Label1.Text = text6
    End Sub
End Class
```

Running the above program will produce the following screen shoot.

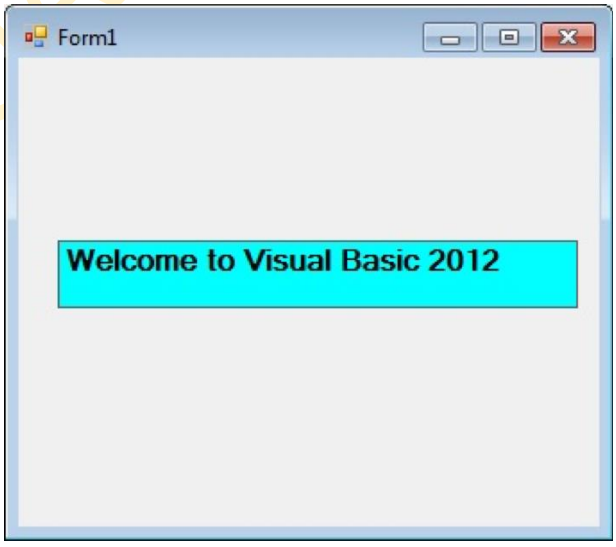


Figure 8.1

## 8.2 String Manipulation Using VB2012 Built-in Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input and return a value, which is passed on to the main program to finish the execution. There are numerous string manipulation functions, which are built into Visual Basic 2012, but I will only discuss a few here and will explain the rest of them in later lessons.

## 8.2 (a) The Len Function

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

### For example,

```
Len (Visual Basic) = 12
```

and

```
Len (welcome to VB tutorial) = 22
```

### Example 8.3

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = CStr(Len(TextBox1.Text))
    End Sub
End Class
```

The output:

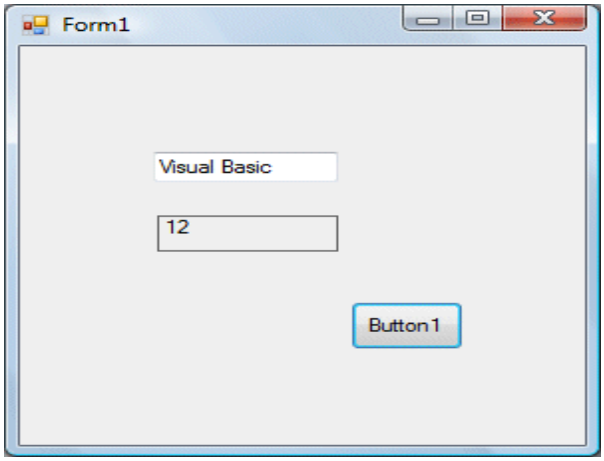



Figure 8.2

## 8.2(b) The Right Function

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

**Right ("Phrase", n)**

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

**Right("Visual Basic", 4) = asic**

However, this syntax is not applicable in VB2012. In VB2012, we need use the following Format

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

### Example 8.4

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim text1 As String
        text1 = TextBox1.Text
        Label1.Text = Microsoft.VisualBasic.Right(text1,4)
    End Sub
End Class
```

The above program returns four rightmost characters of the phrase entered into the TextBox.

The Output:

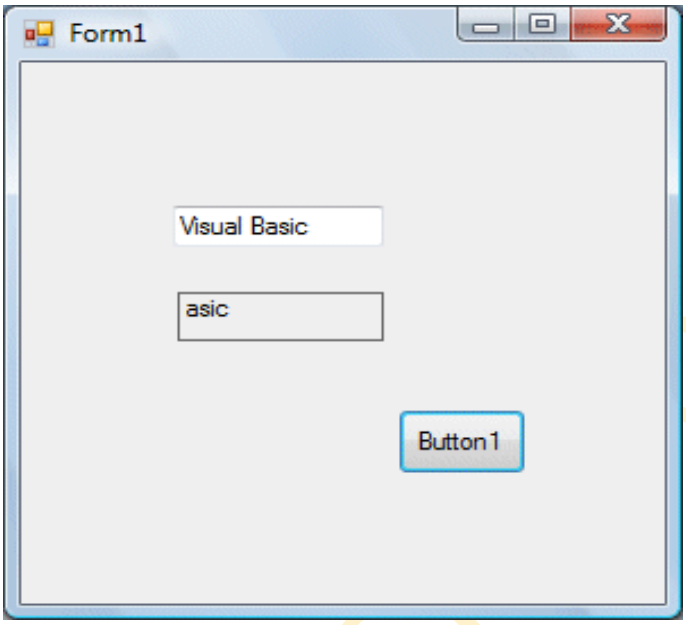



**Figure 8.3**

*The reason of using the full reference is that many objects have the Right properties so using Right on its own will make it ambiguous to Visual Basic 2012.

## 8.2(c)The Left Function

The Left function extracts 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 going to be extracted. For example,

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

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim text1 As String
```

```
        text1 = TextBox1.Text
        Label1.Text = Microsoft.VisualBasic.Left(text1,4)
    End Sub
End Class
```

## 9. Using If…Then…Else

In this lesson, you will learn how to write Visual Basic 2012 code that can make decisions when it processes input from the user controls the program flow.

### 9.1 Decision Making using If…Then…Else

For example, we can write a Visual Basic 2012 program that can ask the computer to perform a certain task until a certain condition is met or a program that will reject non-numeric data. In order to control the program flow and to make decisions, we use the If…Then…Else control structure. The if…Then…Else control structure employs the **conditional** operators and the **logical** operators to make decision.

### 9.2 Conditional Operators

The conditional operators are powerful tools that resemble mathematical operators. These operators allow a Visual Basic 2012 program to compare data values and then decide what actions to take, whether to execute a program or terminate the program and more. They are also known as numerical comparison operators. Normally they are used to compare two values to see whether they are equal or one value is greater or less than the other value. The comparison will return a true or false result. These operators are shown in Table 9.1.

**Table 9.1 Conditional Operators**

| Operator | Description |
|---|---|
| = | **Equal to** |
| > | **Greater than** |
| < | **Less than** |
| >= | **Greater than or equal to** |
| <= | **Less than or Equal to** |
| <> | **Not equal to** |

## 9.2 Logical Operators

Sometimes we might need to make more than one comparison before a decision can be made. In this case, using numerical comparison operators alone is not sufficient; we need to use the logical operators. These logical operators are shown in Table 9.2.

**Table 9.2 Logical Operators**

| Operator | Description |
|---|---|
| **And** | Both input sides must be true to get a true output |
| **Or** | One side or other must be true |
| **Xor** | One side or other must be true but not both |
| **Not** | Negates true |

* In making strings comparison, there are certain rules to follows: Upper case letters are less than lowercase letters, "A" < "a" , "A"<"B"<"C"<"D".....<"Z" and numbers are less than letters 21 < "A".

## 9.3 Using the If control structure with the Comparison Operators

To effectively control the Visual Basic 2012 program flow, we shall use the If control structure together with the conditional operators and logical operators. There are three types of **If** control structures, namely **If....Then** statement, **If....Then... Else** statement and **If....Then....ElseIf** statement.

## 9.3(a) If....Then Statement

This is the simplest control structure, which instructs the computer to perform a certain action specified by the Visual Basic 2012 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 2012  expression
End If
```

### Example 9.1

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myNumber As Integer
          myNumber = CInt(TextBox1.Text)
          If myNumber > 100 Then
              Label1.Text = " You win a lucky prize"
          End If
    End Sub
End Class
```

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

## 9.3(b) If….Then…Else Statement

Using just If….Then statement is not very useful in programming and it does not provide choices for the users. In order to provide a choice, we can use the If….Then…Else Statement. This control structure will ask the computer to perform a certain action specified by the Visual Basic 2012 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

```
If condition Then
     Visual Basic 2012 expression
Else
     Visual Basic 2012  expression
End If
```

### Example 9.2

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myNumber As Integer
        Dim age As Integer
        myNumber = CInt(TextBox1.Text)
        age = CInt(TextBox2.Text)
        If myNumber > 100 And age > 60 Then
            Label1.Text = " Congratulation! You win a lucky prize"
        Else
            Label1.Text = " Sorry, You did not win any prize"
        End If
    End Sub
End Class
```

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

## 9.3(c) If….Then…ElseIf  Statement

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

```
If condition Then
     Visual Basic 2012  expression
ElseIf condition Then
     Visual Basic 2012  expression
ElseIf condition Then
     Visual Basic 2012  expression
Else
     Visual Basic 2012  expression
End If
```

### Example 9.4

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim Mark As Integer
        Dim Grade As String
        Mark = CInt(TextBox1.Text)
        If Mark >= 80 Then
            Grade = "A"
            Label1.Text = "A"
        ElseIf Mark >= 60 And Mark < 80 Then
            Grade = "B"
            Label1.Text = "B"
        ElseIf Mark >= 40 And Mark < 60 Then
            Grade = "C"
            Label1.Text = "C"
        Else
            Grade = "D"
            Label1.Text = "D"
        End If
    End Sub
End Class
```

## 10. Using 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 basically only make a decision on one expression or dimension (for example the examination grade) while the If …ElseIf statement control structure may evaluate only one expression, each If….ElseIf statement may also compute entirely different dimensions. Select Case is preferred when there exist multiple conditions as using If… Then… ElseIf statements will become too messy.

conditions using If…Then..ElseIf statements will become too messy.

## 10.1 The Select Case Structure

The syntax of the Select Case control structure in Visual Basic 2012 is as follows:

```
Select Case test expressionCase expression list 1
      Block of one or more statements
Case expression list 2
      Block of one or more Statements
Case expression list 3
.
.
.
Case Else
      Block of one or more Statements
End Select
```

## 10.2 The usage of Select Case is shown in the following examples

Example 10.1

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        ' Examination Grades
        Dim grade As String
        grade = TextBox1.Text
        Select Case grade
            Case "A"
                Label1.Text = "High Distinction"
            Case "A-"
                Label1.Text = "Distinction"
```

```
            Case "B"
                Label1.Text = "Credit"
            Case "C"
                Label1.Text = "Pass"
            Case Else
                Label1.Text = "Fail"
        End Select
    End Sub
End Class
```

## Example 10.2

In this example, you can use the keyword Is together with the comparison operators

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim mark As Single
        mark = CSng(TextBox1.Text)
        Select Case mark
            Case Is >= 85
                Label1.Text = "Excellence"
            Case Is >= 70
                Label1.Text = "Good"
            Case Is >= 60
                Label1.Text = "Above Average"
            Case Is >= 50
                Label1.Text = "Average"
            Case Else
                Label1.Text = "Need to work harder"
        End Select
    End Sub
End Class
```

## Example 10.3

Example 10.2 can be rewritten as follows:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        'Examination Marks
        Dim mark As Single
        mark = CSng(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
End Class
```

## Example 10.4

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

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        'Examination Marks
        Dim mark As Single
        mark = CSng(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 re-enter the mark"
        End Select
    End Sub
End Class
```

The output of Example 10.4

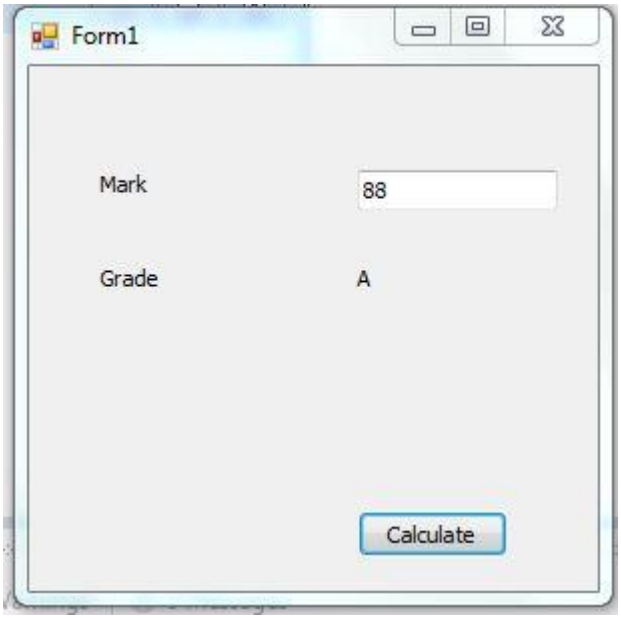



**Figure 10.1**

## 11. Looping

Looping is required when we need to process something repetitively until a certain condition is met. For example, we can design a program that adds a series of numbers until the sum exceeds a certain value. We can also write a program that prompts the user to enter data repeatedly until he or she enters the word 'Finish'. In Visual Basic 2012, there are three types of Loops, they are the For…..Next loop, the Do loop. and the While…..End while loop

### 11.1 Looping using the For….Next Loop

The syntax is:

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

To exit a For…..Next Loop, you can place the Exit For statement within the loop, please refer to example 11.1 d.

Example 11.1 a

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim counter As Integer
        For counter = 1 To 10
            ListBox1.Items.Add(counter)
        Next
    End Sub
End Class
```

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

Example 11.1b

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim counter, sum As Integer
        For counter = 1 To 100 Step 10
            sum += counter
            ListBox1.Items.Add(sum)
      Next
    End Sub
End Class
```

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

sum=1, (1+10+1), (12+20+1), (33+30+1), (64+40+1), .....

**Example 11.1c**

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim counter, sum As Integer
        sum = 1000
        For counter = 100 To 5 Step -5
            sum -= counter
            ListBox1.Items.Add(sum)
```

```
        Next
    End Sub
End Class
```

1000-100, 900-95, 805-90, ……..

*Notice that increment can be negative.

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

Example 11.1d

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim n As Integer
        For n = 1 To 10
            If n > 6 Then
                Exit For
            Else
                ListBox1.Items.Add(n)
            End If
        Next
    End Sub
End Class
```

The process will stop when n is greater than 6.

## 11.2 The Do Loop

The Do Loop structures are

**a)**

```
Do While condition
     Block of one or more statements
Loop
```

**b)**

```
Do
     Block of one or more statements
Loop While condition
```

**c)**

```
Do Until condition
     Block of one or more statements
Loop
```

**d)**

```
Do
     Block of one or more statements
Loop Until condition
```

* Exiting the Loop

Sometimes we need to exit a loop prematurely because of a certain condition is fulfilled. The syntax to use is known as Exit Do. Let's examine the following examples

**Example 11.2(a)**

**Do while counter**

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

The above example can be rewritten as

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim counter As Integer
        Do
            TextBox1.Text = CStr(counter)
            counter += 1
        Loop Until counter > 1000
    End Sub
End Class
```

## Example 11.2(b)

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
       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
End Class
```

## Example about loops:

```
Public Class Form1
    Dim cnt As Integer
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        While cnt <= 10
            ListBox1.Items.Add(cnt)
            cnt = cnt + 1
        End While
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        Do While cnt <= 10
            ListBox1.Items.Add(cnt)
            cnt = cnt + 1
        Loop
    End Sub
```

```
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
        Do Until cnt = 11
            ListBox1.Items.Add(cnt)
            cnt = cnt + 1
        Loop
    End Sub

    Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
        'We can initialize cnt start value
        'cnt = 3
        Do
            ListBox1.Items.Add(cnt)
            cnt = cnt + 1
        Loop While cnt <= 10
    End Sub

    Private Sub Button5_Click(sender As Object, e As EventArgs) Handles Button5.Click
        Do
            ListBox1.Items.Add(cnt)
            cnt = cnt + 1
        Loop Until cnt > 10
    End Sub
End Class
```

**Example While …. Loop (pre-test):**

**The Structure is:**

```
-----------------------------------------------------------
While <condition>
     Instructions
End While
-----------------------------------------------------------
```

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim i As Integer = 0
        While i <= 100
            ListBox1.Items.Add(i)
            i += 1
        End While
    End Sub
End Class
```

**Example Do …Loop Until (post-test):**

**The Structure is:**

```
----------------------------------------------------------------
Do
      Instructions
Loop Until <condition>
----------------------------------------------------------------
```

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim j As Integer = 0
        Do
            ListBox1.Items.Add("  " & j & "   Hello World")
            j += 1
        Loop Until j = 100
    End Sub
End Class
```

**Example For … Next Loop (iterative/counter):**

```
Public Class Form1
    Dim str As String = "Hello World"
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        For k As Integer = 99 To 0 Step -1
            ListBox1.Items.Add(k & "   " & str)
            'or we can write
            'ListBox1.Items.Add(k & vbTab & str)
        Next
    End Sub
End Class
```

## 12. Functions

A function is a procedure that returns a value, which is passed on to the main procedure to finish the execution. There are two types of functions

in Visual Basic 2012, the built-in functions (or internal functions) and the functions created by the programmers. The syntax of a function is:

```
Function Name (arguments)
```

The arguments are values that are passed on to the function. In this lesson, we are going to learn two very basic but useful internal functions of Visual Basic 2012 , i.e. the MsgBox( ) and InputBox ( ) functions.

## 12.1 MsgBox ( ) Function

The MsgBox function produces a pop-up message box and prompts the user to click on a command Button before he /she can continue. This syntax is as follows:

**yourMsg = MsgBox(Prompt, Style Value, Title)**

The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command Buttons appear on the message box, please refer to Table 12.1 for types of command Button displayed. The Title argument will display the title of the message board.



**Table 12.1**

| Style Value | Named Constant | Buttons Displayed |
|---|---|---|
| 0 | vbOkOnly | Ok Button |
| 1 | vbOkCancel | Ok and Cancel Buttons |
| 2 | vbAbortRetryIgnore | Abort, Retry and Ignore Buttons. |
| 3 | vbYesNoCancel | Yes, No and Cancel Buttons |

| | | |
|---|---|---|
| 4 | vbYesNo | Yes and No Buttons |
| 5 | vbRetryCancel | Retry and Cancel Buttons |

We can use named constants in place of integers for the second argument to make the programs more readable. In fact, Visual Basic 2012 will automatically show up a list of named constants where you can select one of them.
Example:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
         MsgBox("Click OK to Proceed", MsgBoxStyle.OkOnly, "Startup Menu")
    End Sub
End Class
```

**And example MsgBox:**

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim youMessage As Integer
        youMessage = MsgBox("click ok to proceed", CType(1, MsgBoxStyle), "Startup Menu")
    End Sub
End Class
```

are the same.

youMessage is a variable that holds values that are returned by the MsgBox ( ) function. The types of Buttons being clicked by the users determine the values. It has to be declared as Integer data type in the procedure or in the general declaration section. Table 12.2 shows the values, the corresponding named constant and Buttons.

**Table 12.2**

| Value | Named Constant | Button Clicked |
|---|---|---|
| 1 | vbOk | Ok Button |
| 2 | vbCancel | Cancel Button |

| | | |
|---|---|---|
| 3 | vbAbort | Abort Button |
| 4 | vbRetry | Retry Button |
| 5 | vbIgnore | Ignore Button |
| 6 | vbYes | Yes Button |
| 7 | vbNo | No Button |

## Example 12.1 or msgboxexample2

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim testmsg As Integer
        testmsg = MsgBox("Click to test", CType(1, MsgBoxStyle), "Test message")
        If testmsg = 1 Then
            MessageBox.Show("You have clicked the OK Button")
        Else
            MessageBox.Show("You have clicked the Cancel Button")
        End If
    End Sub
End Class
```

To make the message box looks more sophisticated, you can add an icon besides the message. There are four types of icons available in VB2012 as shown in Table 12.3



**Table 12.3**

| Value | Named Constant | Icon |
|---|---|---|
| 16 | vbCritical |  |

| | | |
|---|---|---|
| 3 | vbQuestion |  |
| 48 | vbExclamation |  |
| 64 | vbInFormation |  |

## Example 12.2

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim testMsg As Integer
           testMsg = MsgBox("Click to Test", CType(vbYesNoCancel +
           vbExclamation, MsgBoxStyle), "Test Message")
        If testMsg = 6 Then
            MessageBox.Show("You have clicked the Yes Button")
        ElseIf testMsg = 7 Then
            MessageBox.Show("You have clicked the No Button")
        Else
            MessageBox.Show("You have clicked the Cancel Button")
        End If

    End Sub
End Class
```

The first argument, Prompt, will display the message

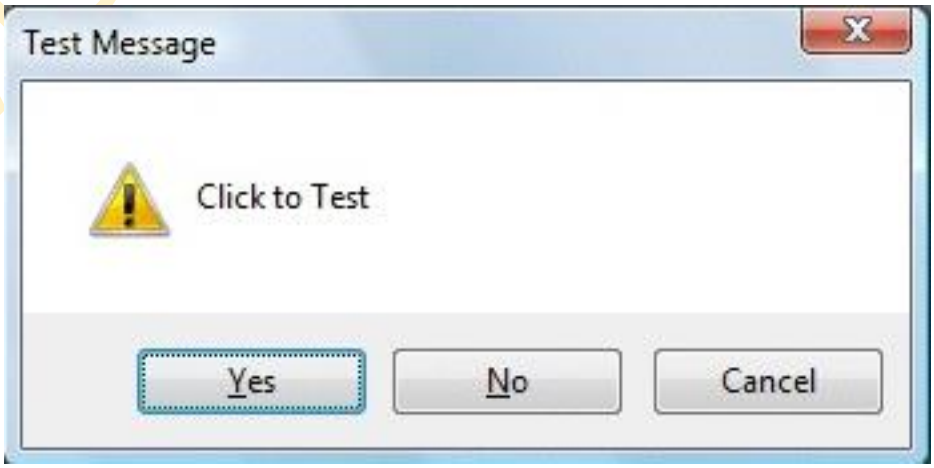


**Figure 12.1**

## 12.2 The InputBox( ) Function

An InputBox( ) function will display a message box where the user can enter a value or a message in the Form of text. The syntax is,

**myMessage= Microsoft.VisualBasic.InputBox(Prompt, Title, default_text, x-position, y-position)**

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

Prompt – The message displayed normally as a question asked.
Title – The title of the Input Box.
default-text – The default text that appears in the input field where the user can use it as his or her intended input or he or she may change to the message he wishes to enter. x-position and y-position – the position or the coordinates of the input box.

However, in Visual Basic 2012 because InputBox is considered a namespace. So, you need to key in the full reference to the Inputbox namespace.



## Example 12.3

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim userMsg As String
            userMsg = Microsoft.VisualBasic.InputBox("What is your
            message?", "Message Entry Form", "Enter your message here
            ", 100, 200)
```

```
        If userMsg <> "" Then
            MessageBox.Show(userMsg)
        Else
            MessageBox.Show("No Message")
        End If
    End Sub
End Class
```

The inputbox will appear as shown in the figure below when you press the command Button

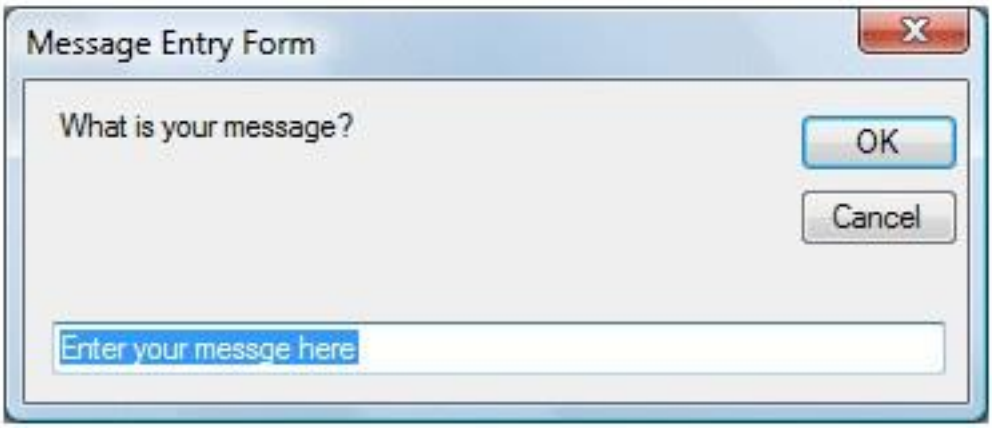



**Figure 12.2**

## 13. The Built-In Functions

There are many built-in functions in Visual Basic 2012. In this lesson, you will learn a couple of built-in functions that deal with string manipulation.

### 13.1 The Mid Function

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

**Mid(phrase, position,n)**

*phrase is the string from which a part of the text is to be retrieved
*position is the starting position of the phrase from which the retrieving process begins.
*n is the number of characters to retrieve.

## Example 13.1

## See String Function example

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myPhrase As String
        myPhrase = Microsoft.VisualBasic.InputBox("Enter your phrase")
        Label1.Text = Mid(myPhrase, 2, 6)
    End Sub
End Class
```

* This program will extract text starting from position 2 of the phrase and the number of characters extracted is 6.

The figures are shown below:

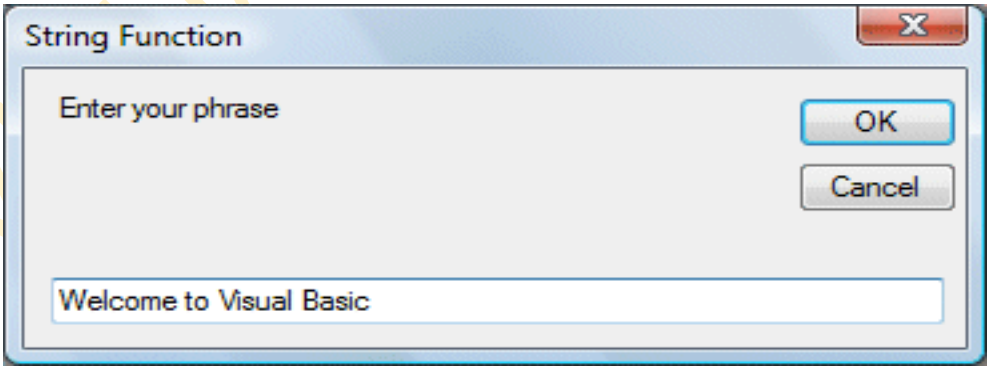



**Figure 13.1**

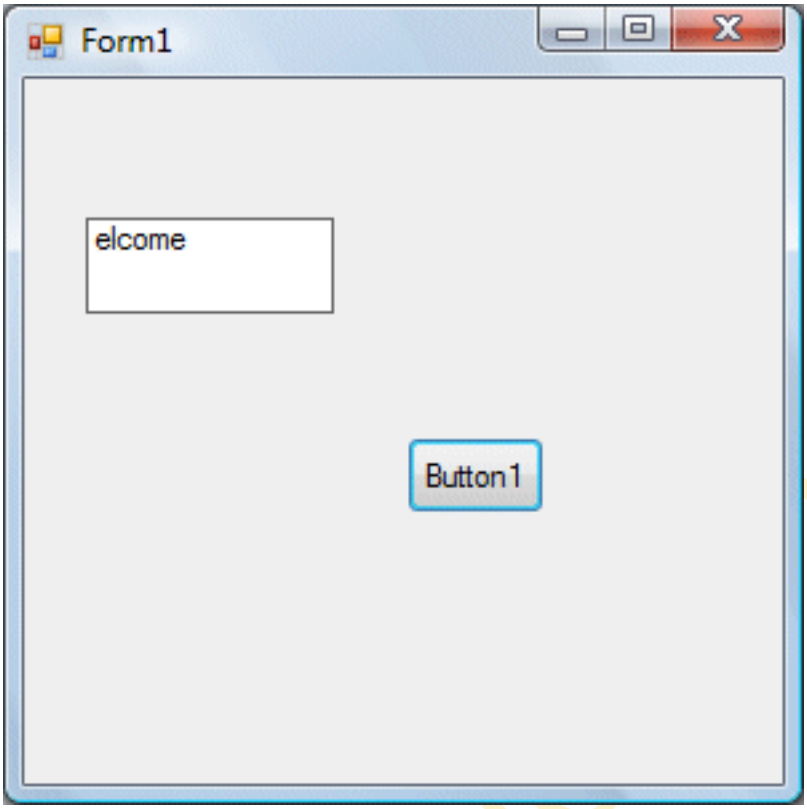



**Figure 13.2**

## 13.2 The Right Function

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

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

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

Microsoft.Visualbasic.Right ("Visual Basic", 4) = asic

## Example 13.2

The following code extracts the right portion any phrase entered by the user.

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myword As String
        myword = TextBox1.Text
        Label1.Text = Microsoft.VisualBasic.Right(myword, 4)
    End Sub
End Class
```

## 13.3 The Left Function

The Left function extracts the left portion of a phrase. The Format 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 going to be extracted. For example:

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

## Example 13

The following code extracts the left portion any phrase entered by the user.

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myword As String
        myword = TextBox1.Text
        Label1.Text = Microsoft.VisualBasic.Left(myword, 4)
    End Sub
End Class
```

## 13.4 The Trim Function

The Trim function trims the empty spaces on both sides of the phrase. The Format is Trim(Phrase). For example,

Trim (  Visual Basic  ) = Visual basic

## Example 13.4

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myPhrase As String
         myPhrase = Microsoft.VisualBasic.InputBox("Enter your phrase")
         Label1.Text = Trim(myPhrase)
    End Sub
End Class
```

## 13.5 The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the phrase. The Format is Ltrim(Phrase). For example,

Ltrim (  Visual Basic 2012  ) = Visual basic 2012

## 13.6 The Rtrim Function

The Rtrim function trims the empty spaces of the right portion of the phrase. The Format is Rtrim(Phrase). For example,

Rtrim (Visual Basic 2012    ) = Visual Basic 2012

## 13.7 The InStr function

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

Instr (n, original phase, embedded phrase)

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

Instr(1, "Visual Basic 2012 ","Basic") = 8

*The function returns a numeric value.

You can write a program code as shown below:

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
            Label1.Text = CStr(InStr(1, "Visual  Basic", "Basic"))
            'first character number of embedded phrase
    End Sub
End Class
```

## 13.8 The UCase and the LCase Functions

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

The syntax is

```
Microsoft.VisualBasic.UCase("Phrase")
Microsoft.VisualBasic.LCase("Phrase")
```

**For example,**

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = Microsoft.VisualBasic.UCase("Visual Basic")
        Label2.Text = Microsoft.VisualBasic.LCase("Visual Basic")
    End Sub
End Class
```

## 13.9 The Chr and the Asc functions

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

Chr(charcode)

and the Format of the Asc function is

Asc(Character)

The following are some examples:

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

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

Example 13. 9

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myChar As String
        Dim charcode As Integer
        'Label1.Text = Chr(101)
        charcode = CInt(Microsoft.VisualBasic.InputBox("Enter your character code"))
        Label1.Text = CStr(Chr(charcode))
        myChar = Microsoft.VisualBasic.InputBox("Enter ASCII character or symbol")
        Label2.Text = CStr(Asc(myChar))
    End Sub
End Class
```

## 14. The Math Functions

In this lesson, you will learn how to use the built-in math functions in Visual Basic 2012. There are numerous built-in math functions in Visual Basic 2012. Let's examine them one by one.

### 14.1 The Abs function

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

The syntax is

Math. Abs (number)

* The Math keyword indicates that the Abs function belong to the Math class.

### 14.2 The Exp function

The Exp of a number x is the exponential value of x, i.e. $e^x$ . For example, Exp(1)=e=2.71828182
The syntax is

## Math.Exp (number)

### Example 14.1

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1 As Single
        num1 = CSng(TextBox1.Text)
        Label1.Text = CStr(Math.Exp(num1))
        Label2.Text = CStr(Math.Abs(num1))
    End Sub
End Class
```

### 14.3 The Fix Function

The Fix function truncates the decimal part of a positive number and returns the largest integer smaller than the number. However, when the number is negative, it will return smallest integer larger than the number. For example, Fix(9.2)=9 but Fix(-9.4) = -9

### 14.4 The Int Function

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

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

### Example 14.2

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1 As Single
        num1 = CSng(TextBox1.Text)
```

```
        Label1.Text = CStr(Fix(num1))
        Label2.Text = CStr(Int(num1))
    End Sub
End Class
```

## 14.5 The Log Function

The Log function is the function that returns the natural logarithm of a number. For example, Log(10) = 2.302585

## Example 14.3

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1, num2 As Single ' declare num1 and num2 as Single
        num1 = CSng(TextBox1.Text) ' convert to single what is entered in TextBox1
        num2 = CSng(Math.Log(num1)) ' convert to single and find the log(num1)
        Label1.Text = CStr(num2)  'display the result on Label1
    End Sub
End Class
```

* The logarithm of num1 will be displayed on Label1

## 14.6 The Rnd( ) Function

Rnd is a very useful function in Visual Basic 2012 . We use the Rnd function to write code that involves chance and probability. The Rnd function returns a random value between 0 and 1. Random numbers in their original Form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 integers ranging from 1 to 6, which makes the program behave like a virtual dice, we need to convert the random numbers to integers using the Formula Int(Rnd*6)+1.

## Example 14.4

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num As Integer  'declare num as intiger
        num = CInt(Int(Rnd() * 6) + 1) 'generate random numbers between "0 and 1" and
        'multiply each by 6 then add 1 to each
        ' after that convert to intiger
        Label1.Text = CStr(num) 'display num on Label1
       End Sub
End Class
```

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

## 14.7 The Round Function

The Round function is a Visual Basic 2012 function that rounds up a number to a certain number of decimal places. The Format is Round (n, m) which means to round a number n to m decimal places. For example, Math.Round (7.2567, 2) =7.26

## Example 14.5

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num1 As Single
        num1 = CSng(TextBox1.Text)
        Label1.Text = CStr(Math.Round(num1, 2))
    End Sub
End Class
```

* The Math keyword here indicates that the Round function belong to the Math class.

# 15. The Format Function

The Format function in Visual Basic 2012 is to display the numbers in different Formats. There are two types of Format functions, one of them is the built-in Format function and the other one is defined by the users.

(i) The syntax of the predefined Format function is

**Format (n, "style argument")**

where n is a number.

The list of style arguments in Visual Basic 2012 is given in Table 15.1.

**Table 15.1 List of style arguments**

| Style argument | Explanation | Example |
|---|---|---|
| General Number | To display the number without having separators between thousands. | Format(8972.234, "General Number") = 8972.234 |
| Fixed | To display the number without having separators between thousands and rounds it up to two decimal places. | Format(8972.234678, "Fixed") = 8972.23 |
| Standard | To display the number with separators or separators between thousands and rounds it up to two decimal places. | Format(6648972.265, "Standard")= 6,648,972.27 |
| Currency | To display the number with the dollar sign in front has separators between thousands as well as rounding it up to two decimal places. | Format(6648972.265, "Currency")= $6,648,972.27 |

| | | |
|---|---|---|
| Percent | Converts the number to the percentage Form and displays a % sign and rounds it up to two decimal places. | Format(0.56324, "Percent")=56.32 % |

## Example 15.1

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = Format(8972.234, "General Number")
        Label2.Text = Format(8972.2, "Fixed")
        Label3.Text = Format(6648972.265, "Standard")
        Label4.Text = Format(6648972.265, "Currency")
        Label5.Text = Format(0.56324, "Percent")
    End Sub
End Class
```

The Output window is shown below:

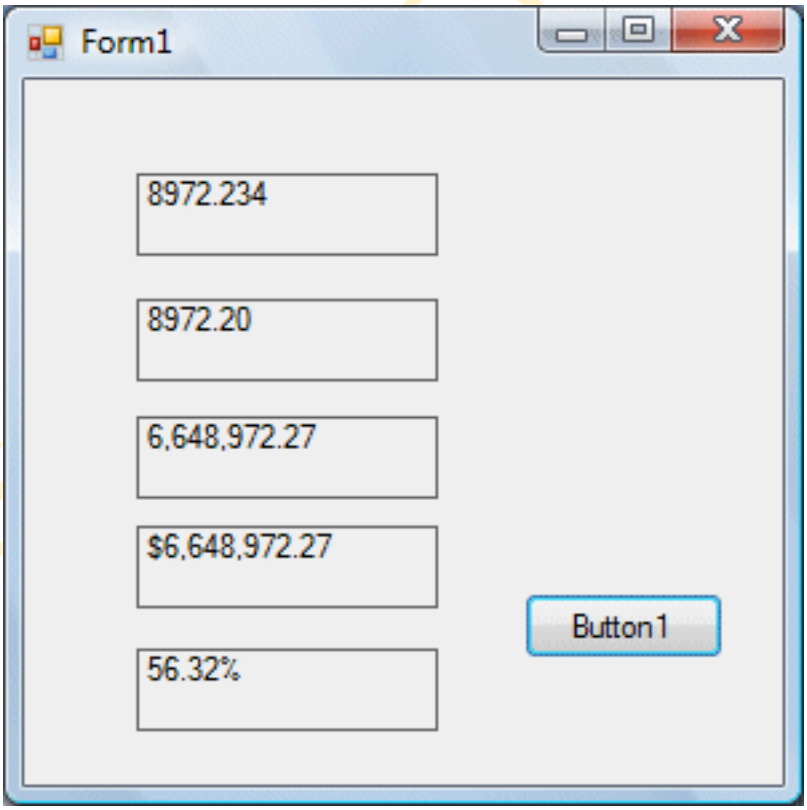



**Figure 15.1**

(ii) The syntax of the user-defined Format function is

## Format (n, "user's Format")

Although it is known as user-defined Format, we still need to follow certain Formatting styles. Examples of user-defined Formatting style are listed in Table 15.2

**Table 15.2 User-Defined Format**

| Example | Explanation | Output |
|---|---|---|
| Format(781234.57,"0") | Rounds to whole number without separators between thousands. | 781235 |
| Format(781234.57,"0.0") | Rounds to 1 decimal place without separators between thousands. | 781234.6 |
| Format(781234.576,"0.00") | Rounds to 2 decimal places without separators between thousands. | 781234.58 |
| Format(781234.576,"#,##0.00") | Rounds to 2 decimal places with separators between thousands. | 781,234.58 |
| Format(781234.576,"$#,##0.00") | Shows dollar sign and rounds to 2 decimal places with separators between thousands. | $781,234.58 |
| Format(0.576,"0%") | Converts to percentage form without decimal places. | 58% |
| Format(0.5768,"0.00%") | Converts to percentage form with 2 decimal places. | 57.68% |

## Example 15.2

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = Format(8972.234, "0.0")
        Label2.Text = Format(8972.2345, "0.00")
        Label3.Text = Format(6648972.265, "#,##0.00")
        Label4.Text = Format(6648972.265, "$#,##0.00")
        Label5.Text = Format(0.56324, "0%")
    End Sub
End Class
```

The Output window is shown below:

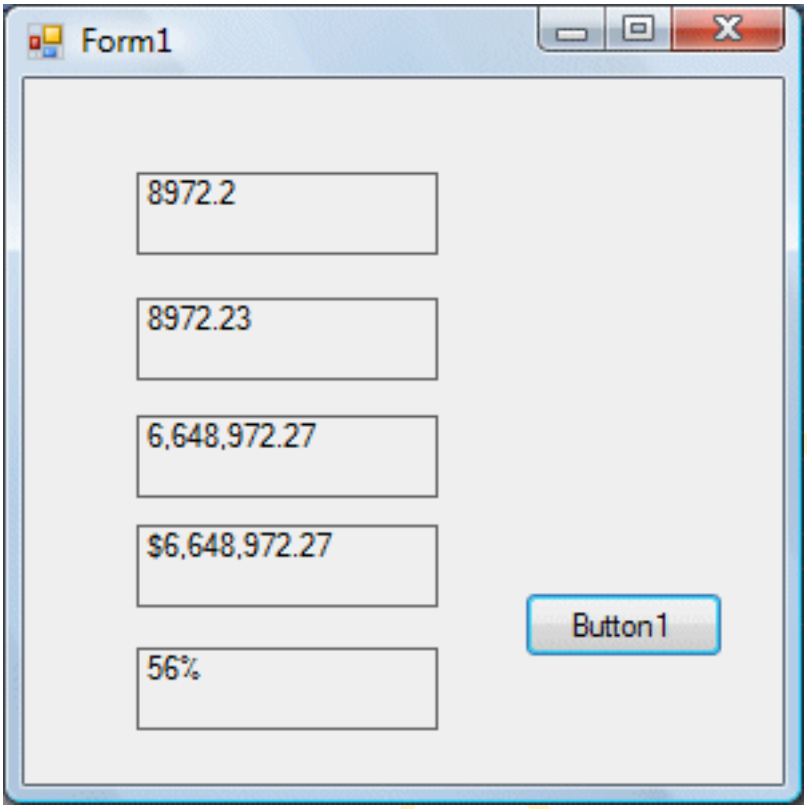



**Figure 15.2**

## 16. Formatting Date and Time

### 16.1 Formatting Date and time using predefined Formats

In Visual Basic 2012, we can Format date and time using predefined Formats or user-defined Formats. The predefined Formats of date and time are shown in Table 16.1.

**Table 16.1 Predefined formats of date and time**

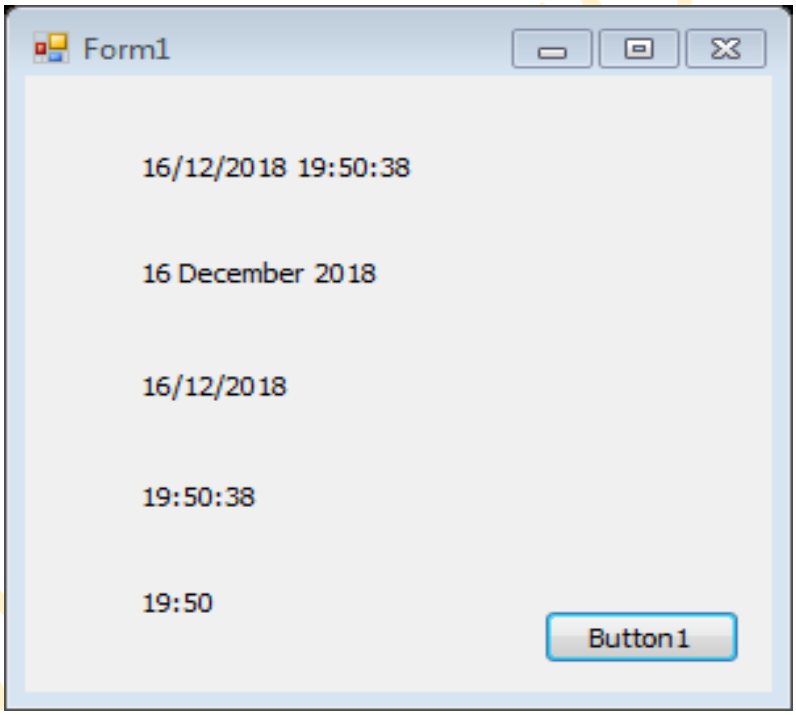
| Format | Explanation |
|---|---|
| Format (Now, "General date") | Formats the current date and time. |
| Format (Now, "Long Date") | Displays the current date in long format. |
| Format (Now, "Short date") | Displays current date in short format |
| Format (Now, "Long Time") | Display the current time in long format. |
| Format (Now, "Short Time") | Display the current time in short format. |

* Instead of "General date", you can also use the abbreviated Format "G", i.e. Format (Now, "G"). For "Long Time", you can use the abbreviated Format "T" and for "Short Time", you may use the abbreviated Format "t"

Example 16.1

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = Format(Now, "General Date")
        Label2.Text = Format(Now, "Long Date")
        Label3.Text = Format(Now, "short Date")
        Label4.Text = Format(Now, "Long Time")
        Label5.Text = Format(Now, "Short Time")
    End Sub
End Class
```

The output is shown in the figure below:



## 16.2 Formatting Date and time using user-defined Formats

Besides using the predefined Formats, you can also use the user-defined Formatting functions. The general syntax of a user-defined for date/time is

Format (expression,style)

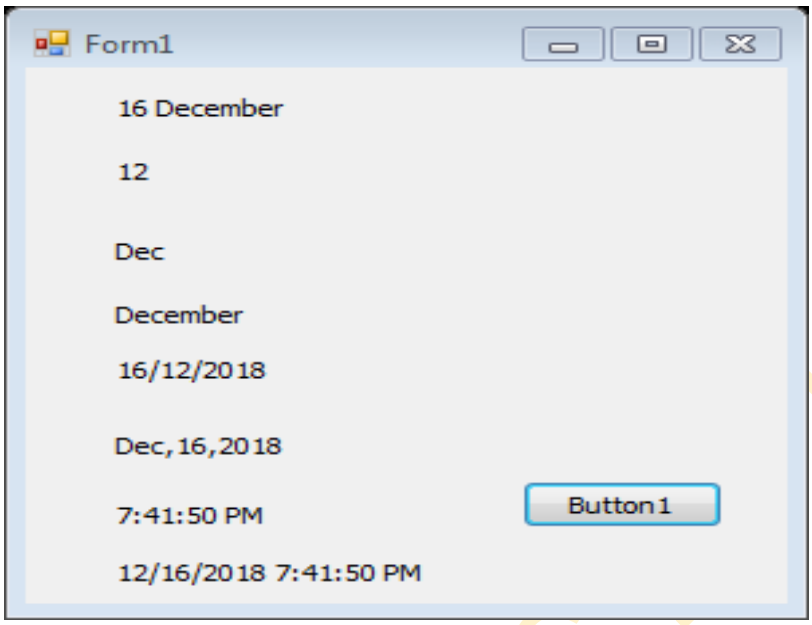
Table 16.2 Some of the user-defined format functions for date and time

| Format | Explanation |
|---|---|
| Format (Now, "M") | Displays current month and date |
| Format (Now, "MM") | Displays current month in double digits. |
| Format (Now, "MMM") | Displays abbreviated name of the current month |
| Format (Now, "MMMM") | Displays full name of the current month. |
| Format (Now, "dd/MM/yyyy") | Displays current date in the day/month/year format. |
| Format (Now, "MMM,d,yyyy") | Displays current date in the Month, Day, Year Format |
| Format (Now, "h:mm:ss tt") | Dispalys current time in hour:minute:second format and show am/pm |
| Format (Now, "MM/dd/yyyy h:mm:ss") | Dispalys current date and time in hour:minute:second format |

Example 16.2

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Label1.Text = Format(Now, "M")
          Label2.Text = Format(Now, "MM")
          Label3.Text = Format(Now, "MMM")
          Label4.Text = Format(Now, "MMMM")
          Label5.Text = Format(Now, "dd/MM/yyyy")
          Label6.Text = Format(Now, "MMM,d,yyyy")
          Label7.Text = Format(Now, "h:mm:ss tt")
          Label8.Text = Format(Now, "MM/dd/yyyy h:mm:ss tt")
    End Sub
End Class
```

The output is shown in the figure below:




## 17. The Checkbox

The checkbox is a control that allows the user to select multiple items. For example, in the Font dialog box of Microsoft Words, there are many checkboxes under the Effects section, such as that shown in the Figure 17.1 below. The user can choose to Format the text with an underline, subscript, small caps, superscript, blink and more.

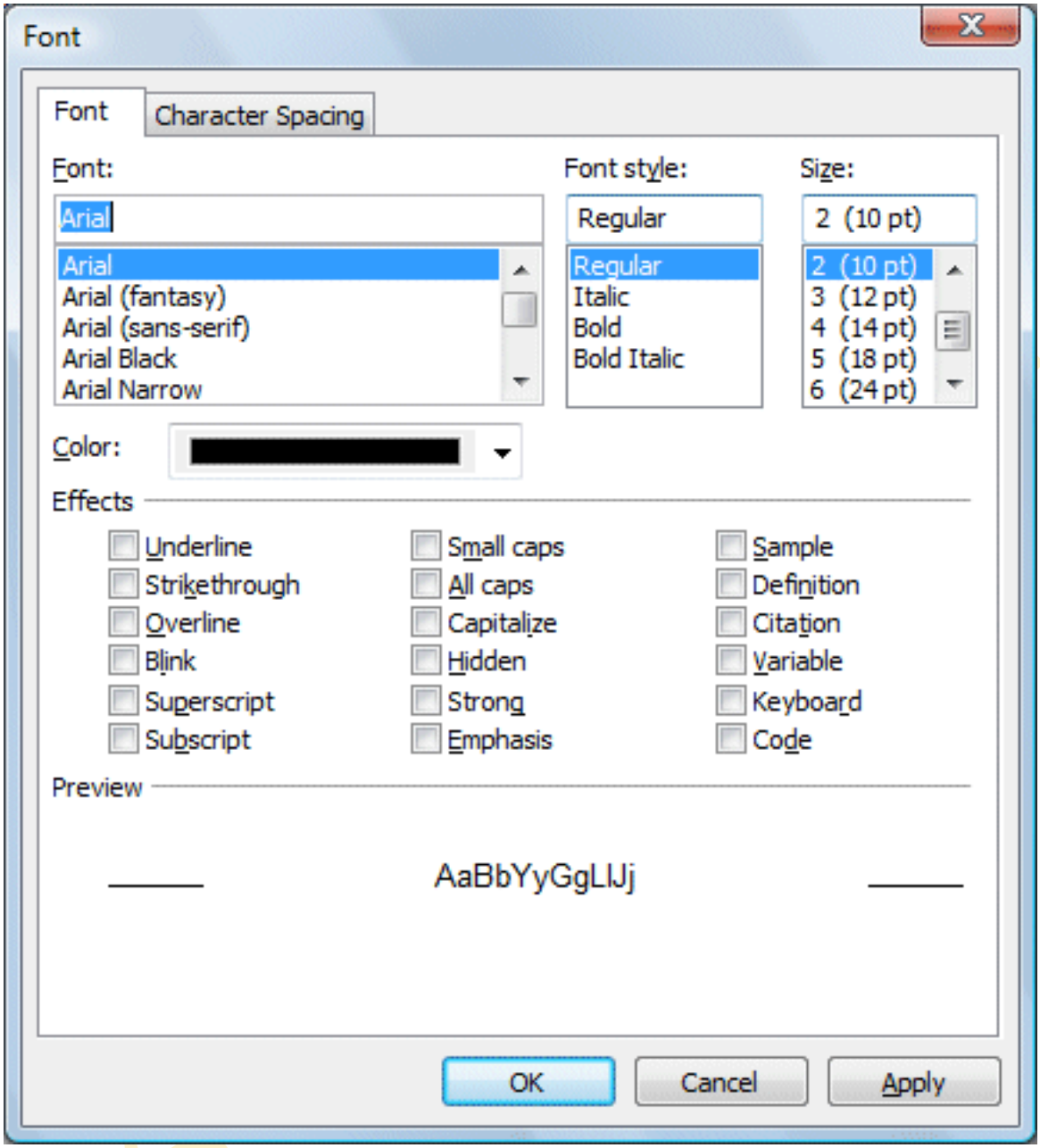



**Figure 17.1**

## Example 17.1 Shopping Cart

In Visual Basic 2012, you may create a shopping cart where the user can click on checkboxes that correspond to the items they intend to purchase, and the total payment can be calculated simultaneously, as shown in Figure 17.1.

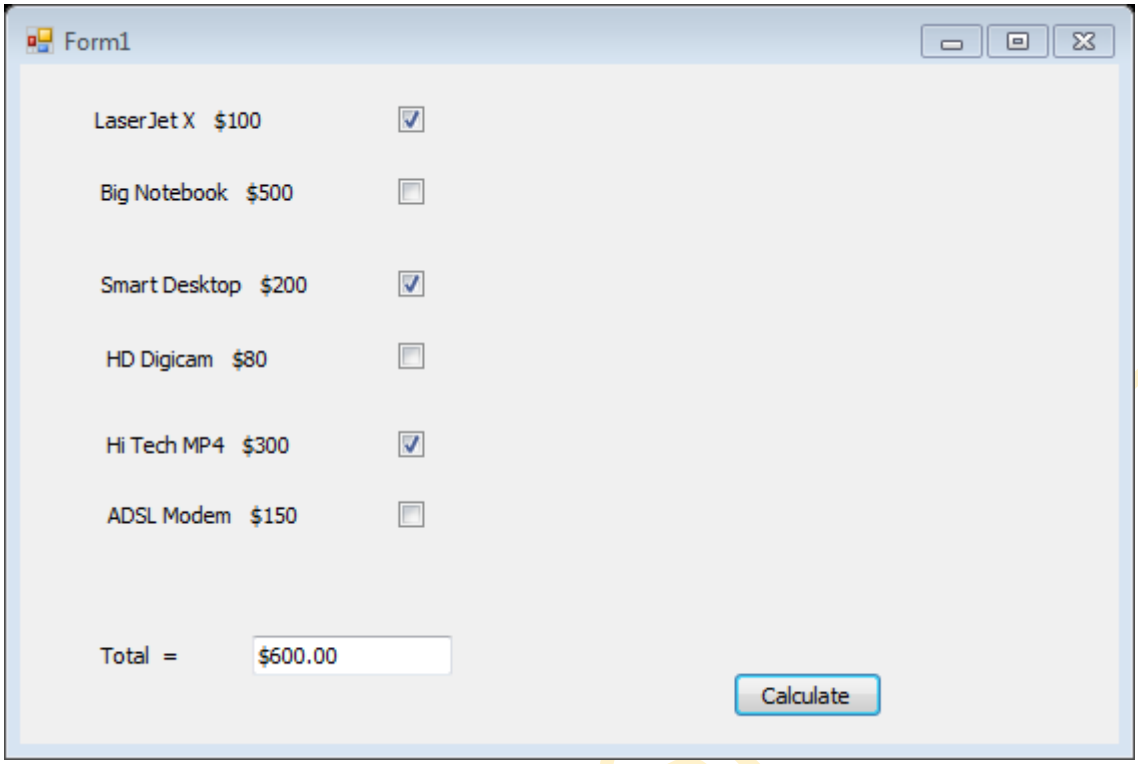



**Figure 17.2**

The program code for the shopping cart is as follows:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Const LX As Integer = 100
        Const BN As Integer = 500
        Const SD As Integer = 200
        Const HD As Integer = 80
        Const HM As Integer = 300
        Const AM As Integer = 150
        Dim sum As Integer

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

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

        If CheckBox3.Checked = True Then
            sum += SD
```

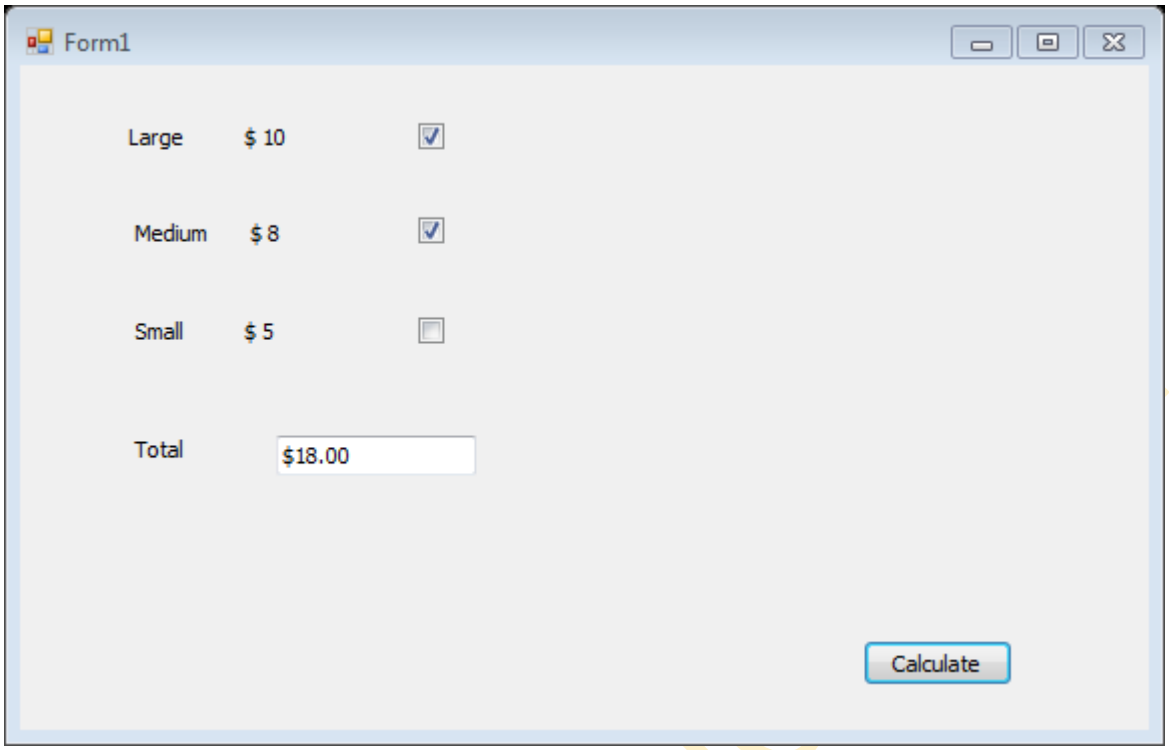
```
        End If

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

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

        If CheckBox6.Checked = True Then
            sum += AM
        End If
        TextBox1.Text = sum.ToString("c")
    End Sub
End Class
```

Here is another example

## Example 17.2

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Const large As Integer = 10
        Const medium As Integer = 8
        Const small As Integer = 5
        Dim sum As Integer

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

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

        If CheckBox3.Checked = True Then
            sum += small
        End If
        TextBox1.Text = sum.ToString("c")

    End Sub
End Class
```

The output as in following Figure:




Example 17.3

In this example, the user can enter text into a Text Box and Format the font using the three checkboxes that represent bold, italic and underline.

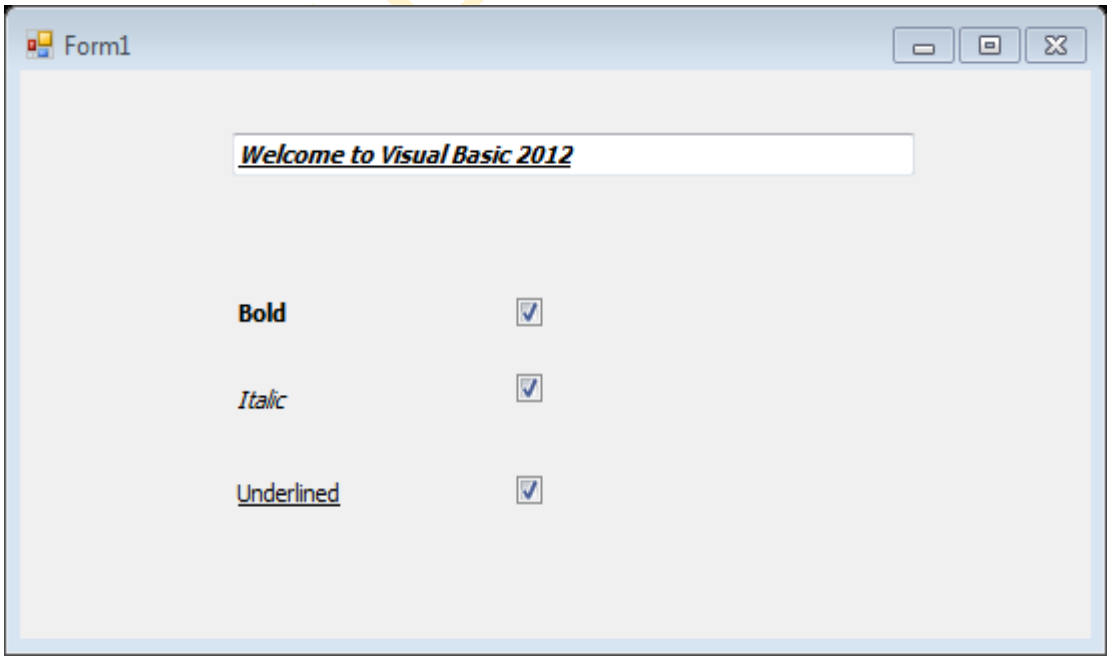


Figure 17.3

The code

```
Public Class Form1

    Private Sub CheckBox1_CheckedChanged(sender As Object, e As EventArgs) Handles
CheckBox1.CheckedChanged
        If CheckBox1.Checked Then
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Bold)
        Else
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Bold)

        End If
    End Sub

    Private Sub CheckBox2_CheckedChanged(sender As Object, e As EventArgs) Handles
CheckBox2.CheckedChanged
        If CheckBox2.Checked Then
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Italic)
        Else
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Italic)

        End If
    End Sub

    Private Sub CheckBox3_CheckedChanged(sender As Object, e As EventArgs) Handles
CheckBox3.CheckedChanged
        If CheckBox2.Checked Then
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or
FontStyle.Underline)
        Else
            TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not
FontStyle.Underline)

        End If
    End Sub
End Class
```

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

```
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style Or FontStyle.Italic)
```

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

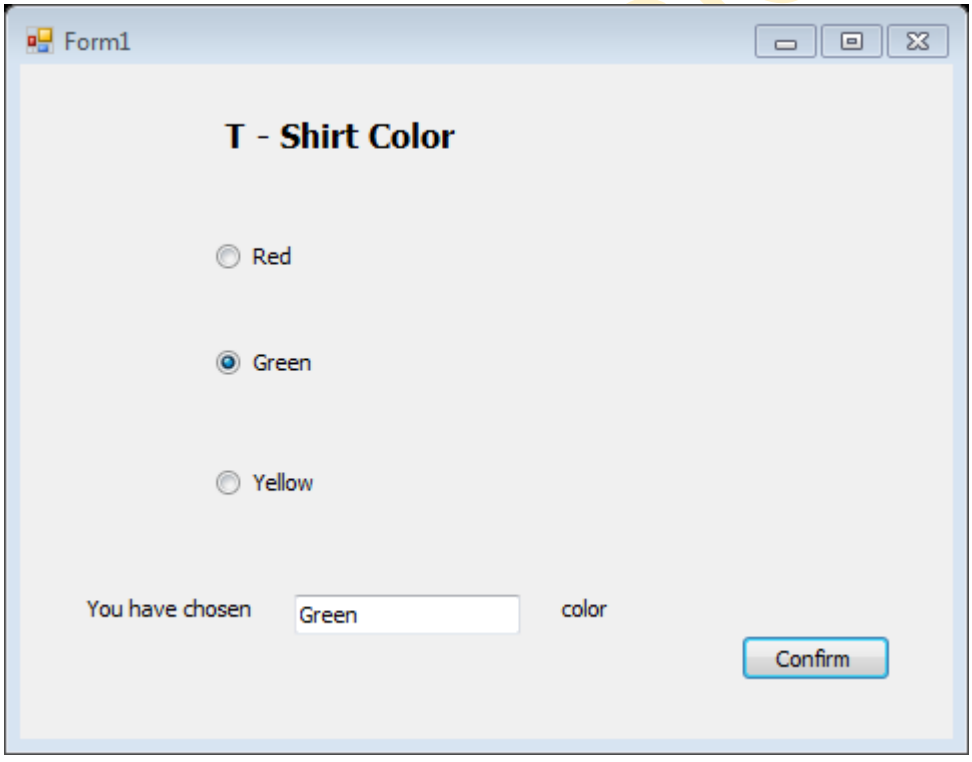
```
TextBox1.Font = New Font(TextBox1.Font, TextBox1.Font.Style And Not FontStyle.Italic)
```

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

## 18. Using Radio Button

The radio Button is another control in Visual Basic 2010 that allows selection of choices. However, it operates differently from the check box. While the check boxes allow the user to select one or more items, radio Buttons are mutually exclusive, which means the user can only choose one item only out of a number of choices. Here is an example which allows the user to select one color only.

Example 18.1

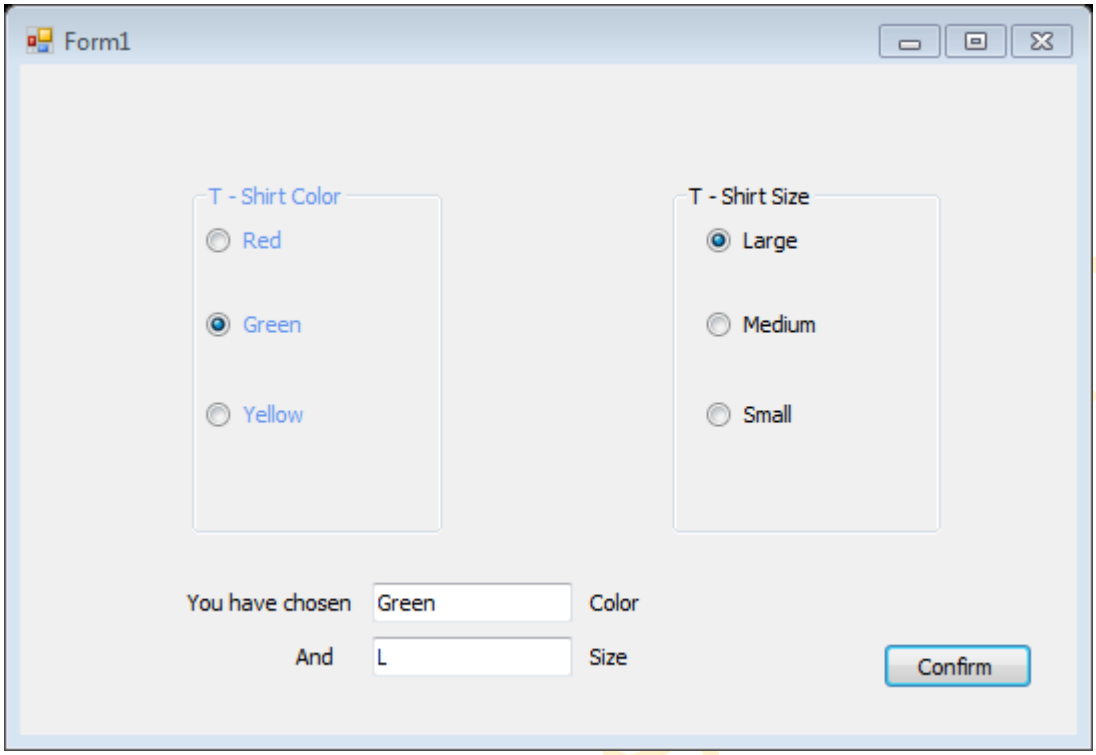


## The Code for Example 18.1:

```
Public Class Form1
    Dim strColor As String
     Private Sub RadioButton1_CheckedChanged(sender As Object, e As
     EventArgs) Handles RadioButton1.CheckedChanged
        strColor = "Red"
    End Sub

    Private Sub RadioButton2_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton2.CheckedChanged
        strColor = "Green"
    End Sub

    Private Sub RadioButton3_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton3.CheckedChanged
        strColor = "Yellow"
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        TextBox1.Text = strColor
    End Sub
End Class
```

Although the user may only select one item at a time, he may make more than one selection if those items belong to different categories. For example, the user wishes to choose T-shirt size and color, he needs to select one color and one size, which means one selection in each category. This is easily achieved in Visual Basic 2010 by using the Groupbox control under the containers categories. After inserting the Groupbox into the Form, you can proceed to insert the radio Buttons into the Groupbox. Only the radio Buttons inside the Groupbox are mutually exclusive, they are not mutually exclusive with the radio Buttons outside the Groupbox. In Example 18.2, the user can select one color and one size of the T-shirt.

## Example 18.2




## The code for Example 18.2

```
Public Class Form1
    Dim strColor As String
    Dim strSize As String

    Private Sub RadioButton1_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton1.CheckedChanged
        strColor = "Red"
    End Sub

    Private Sub RadioButton2_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton2.CheckedChanged
        strColor = "Green"
    End Sub

    Private Sub RadioButton3_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton3.CheckedChanged
        strColor = "Yellow"
```

```
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        TextBox1.Text = strColor
        TextBox2.Text = strSize

    End Sub

    Private Sub RadioButton4_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton4.CheckedChanged
        strSize = "L"
    End Sub

    Private Sub RadioButton5_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton5.CheckedChanged
        strSize = "M"
    End Sub

    Private Sub RadioButton6_CheckedChanged(sender As Object, e As
    EventArgs) Handles RadioButton6.CheckedChanged
        strSize = "S"
    End Sub
End Class
```

## 19. Arrays

An array is a set of values, which are termed *elements*, that are logically related to each other. For example, an array may consist of the number of students in each grade in a grammar school; each element of the array is the number of students in a single grade. Similarly, an array may consist of a student's grades for a class; each element of the array is a single grade.

By using an array, you can refer to these related values by the same name, and use a number that's called an index or subscript to identify an individual element based on its position in the array. The indices of an array range from 0 to one less than the total number of elements in the array. When you use Visual Basic syntax to define the size of an array, you specify its highest index, not the total number of elements in the array. You can work with the array as a unit, and the ability to iterate its elements frees you from needing to know exactly how many elements it contains at design time.

Some quick examples before explanation:

```
' Declare a single-dimension array of 5 numbers. Example(arrays11)
Dim numbers(4) As Integer

'Declare a single-dimension array and set its 4 values. Example(arrays22)
Dim numbers = New Integer() {1, 2, 4, 8}

' Change the size of an existing array to 16 elements and retain the current
values.    Example(arrays33)
ReDim Preserve numbers(15)

' Redefine the size of an existing array and reset the values.
ReDim numbers(15)

' Declare a 6 x 6 multidimensional array.
Dim matrix(5, 5) As Double

' Declare a 4 x 3 multidimensional array and set array element values.
Dim matrix = New Integer(3, 2) {{1, 2, 3}, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}}

' Declare a jagged array

Dim sales()() As Double = New Double(11)() {}
```

## Creating an array:

You can define the size of an array in several ways:

- You can specify the size when the array is declared:

```
' Declare an array with 10 elements.
Dim cargoWeights(9) As Double
' Declare a 24 x 2 array.
Dim hourlyTemperatures(23, 1) As Integer

' Declare a jagged array with 31 elements.

Dim januaryInquiries(30)() As String
```

- You can use a New clause to supply the size of an array when it's created:

```
    ' Declare an array with 10 elements.
    Dim cargoWeights() As Double = New Double(9) {}

    ' Declare a 24 x 2 array.
    Dim hourlyTemperatures(,) As Integer = New Integer(23, 1) {}
```

If you have an existing array, you can redefine its size by using the *ReDim* statement. You can specify that the *ReDim* statement keep the values that are in the array, or you can specify that it create an empty array. The following example shows different uses of the *ReDim* statement to modify the size of an existing array.

```
' Assign a new array size and retain the current values.
ReDim Preserve cargoWeights(20)
' Assign a new array size and retain only the first five values.
ReDim Preserve cargoWeights(4)
' Assign a new array size and discard all current element values.

ReDim cargoWeights(15)
```

- Storing values in an array

You can access each location in an array by using an index of type Integer. You can store and retrieve values in an array by referencing each array location by using its index enclosed in parentheses. Indices for multidimensional arrays are separated by commas (,). You need one index for each array dimension.

The following example shows some statements that store and retrieve values in arrays.

```
Module Example
   Public Sub Main()
      ' Create a 10-element integer array.
      Dim numbers(9) As Integer
      Dim value As Integer = 2

      ' Write values to it.
      For ctr As Integer = 0 To 9
         numbers(ctr) = value
         value *= 2
      Next

      ' Read and sum the array values.
      Dim sum As Integer
      For ctr As Integer = 0 To 9
         sum += numbers(ctr)
```

```
      Next
      Console.WriteLine($"The sum of the values is {sum:N0}")
    End Sub
End Module
' The example displays the following output:

'     The sum of the values is 2,046
```

- Populating an array with array literals

By using an array literal, you can populate an array with an initial set of values at the same time that you create it. An array literal consists of a list of comma-separated values that are enclosed in braces ({}).

When you create an array by using an array literal, you can either supply the array type or use type inference to determine the array type. The following example shows both options.

```
' Array literals with explicit type definition.
Dim numbers = New Integer() {1, 2, 4, 8}

' Array literals with type inference.
Dim doubles = {1.5, 2, 9.9, 18}

' Array literals with explicit type definition.
Dim articles() As String = { "the", "a", "an" }

' Array literals with explicit widening type definition.

Dim values() As Double = { 1, 2, 3, 4, 5 }
```

When you use type inference, the type of the array is determined by the *dominant type* in the list of literal values. The dominant type is the type to which all other types in the array can widen. If this unique type can't be determined, the dominant type is the unique type to which all other types in the array can narrow. If neither of these unique types can be determined, the dominant type is Object. For example, if the list of values that's supplied to the array literal contains values of type Integer, Long, and Double, the resulting array is of type Double. Because Integer and Long widen only to Double, Double is the dominant type.

**Note:**

You can use type inference only for arrays that are defined as local variables in a type member. If an explicit type definition is absent, arrays defined with array literals at the class level are of type Object[].

Note that the previous example defines values as an array of type Double even though all the array literals are of type Integer. You can create this array because the values in the array literal can widen to Double values.

You can also create and populate a multidimensional array by using nested array literals. Nested array literals must have a number of dimensions that's consistent with the resulting array. The following example creates a two-dimensional array of integers by using nested array literals.

```
' Create and populate a 2 x 2 array.
Dim grid1 = {{1, 2}, {3, 4}}
' Create and populate a 2 x 2 array with 3 elements.

Dim grid2(,) = {{1, 2}, {3, 4}, {5, 6}}
```

When using nested array literals to create and populate an array, an error occurs if the number of elements in the nested array literals don't match. An error also occurs if you explicitly declare the array variable to have a different number of dimensions than the array literals.

Just as you can for one-dimensional arrays, you can rely on type inference when creating a multidimensional array with nested array literals. The inferred type is the dominant type for all the values in all the array literals for all nesting level. The following example creates a two-dimensional array of type Double[ , ] from values that are of type Integer and Double.

```
Dim arr = {{1, 2.0}, {3, 4}, {5, 6}, {7, 8}}
```

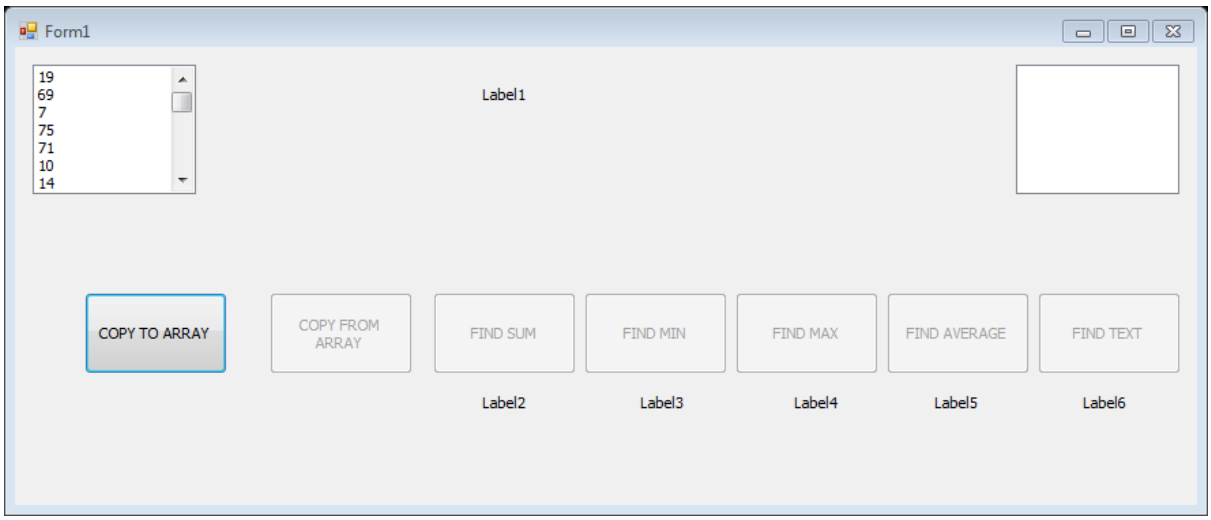
### Example arrayExample2

```
Public Class Form1
    Dim strArray(49) As String

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim rand As New Random
        For counter = 0 To 49
            ListBox1.Items.Add(rand.Next(0, 101)
        Next
        Button1.Enabled = True
        Button2.Enabled = False
        Button3.Enabled = False
        Button4.Enabled = False
        Button5.Enabled = False
        Button6.Enabled = False
        Button7.Enabled = False

    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        For counter = 0 To ListBox1.Items.Count - 1
            strArray(counter) = CStr(ListBox1.Items(counter))
        Next
```

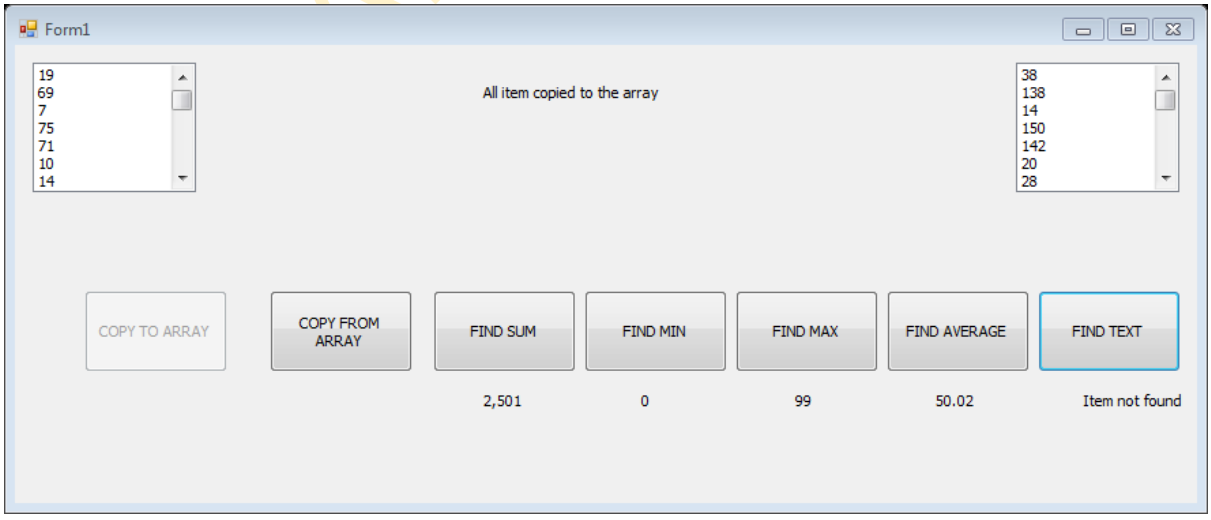
```
    Label1.Text = "All item copied to the array"
    Button1.Enabled = False
    Button2.Enabled = True
    Button3.Enabled = True
    Button4.Enabled = True
    Button5.Enabled = True
    Button6.Enabled = True
    Button7.Enabled = True
End Sub

Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    For counter = 0 To strArray.Length - 1
        ListBox2.Items.Add(CInt(strArray(counter)) * 2)
    Next


End Sub

Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    Label2.Text = sum().ToString("n0")
End Sub
Private Function sum() As Integer
    Dim total As Integer
    For counter = 0 To strArray.Length - 1
        total = CInt(total + CDbl(strArray(counter)))
    Next
    Return total
End Function

Private Sub Button6_Click(sender As Object, e As EventArgs) Handles Button6.Click
    Label5.Text = CStr(sum() / CDbl(strArray.Length.ToString("n2")))
End Sub

Private Sub Button4_Click(sender As Object, e As EventArgs) Handles Button4.Click
    Dim min As Integer
    min = CInt(strArray(0))
    For counter = 0 To strArray.Length - 1
        If CDbl(strArray(counter)) < min Then
            min = CInt(strArray(counter))
        End If
    Next
    Label3.Text = CStr(min)

End Sub

Private Sub Button5_Click(sender As Object, e As EventArgs) Handles Button5.Click
    Dim max As Integer
    max = CInt(strArray(0))
    For counter = 0 To strArray.Length - 1
        If CDbl(strArray(counter)) > max Then
            max = CInt(strArray(counter))
        End If
    Next
    Label4.Text = CStr(max)
End Sub

Private Sub Button7_Click(sender As Object, e As EventArgs) Handles Button7.Click
    Dim strFind As String
    Dim indexOfItem As Integer
    strFind = InputBox("What do you want to search?")
    For counter = 0 To strArray.Length - 1
        If strFind = strArray(counter) Then
```

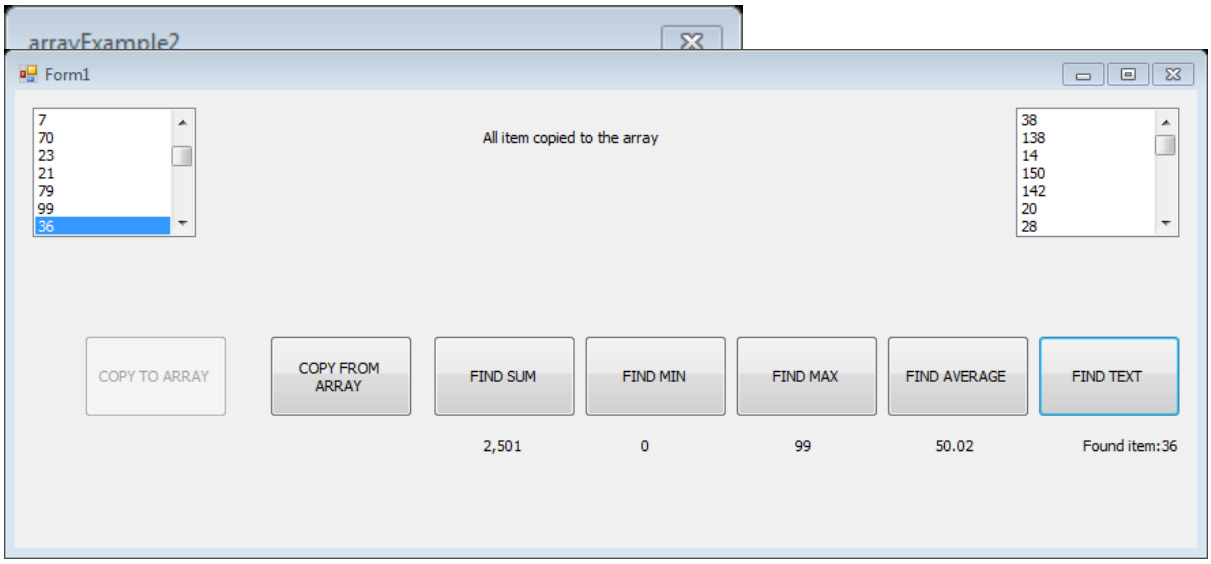
```
                Label6.Text = "Found item:" & strFind
                indexOfItem = counter
                Exit For
            Else
                Label6.Text = "Item not found"
                indexOfItem = -1

            End If
        Next
        ListBox1.SelectedIndex = indexOfItem
    End Sub
End Class
```

The output for this example after start the program as in following picture;




After click the Button (COPY TO ARRAY), other Buttons will be activated and the final output will be as in the following picture






## 20. Creating the Graphics Object

Before you can draw any graphic on a Form, you need to create the Graphics object in VB2012. A graphics object is created using a CreateGraphics() method. You can create a graphics object that draws to the Form itself or a control. For example, if you wish to draw to the Form, you can use the following statement:

```
Dim myGraphics As Graphics =me.CreateGraphics
```

*Always use Dim to define the object. Using me instead of Form1 because it is not allowed in Visual Basic 2012.

Or if you want the graphics object to draw to a picturebox, you can write the following statement:

```
Dim myGraphics As Graphics = PictureBox1.CreateGraphics
```

You can also use the Text Box as a drawing surface, the statement is:

```
Dim myGraphics As Graphics = TextBox1.CreateGraphics
```

The Graphics object that is created does not draw anything on the screen until you call the methods of the Graphics object. In addition, you need to create the Pen object as the drawing tool. We will examine the code that can create a pen in the following section.

## 20.1 Creating a Pen

A Pen can be created using the following code:

```
myPen = New Pen(Brushes.DarkMagenta, 10)
```

where myPen is a Pen variable. You can use any variable name instead of myPen. The first argument of the pen object defines the color of the drawing line and the second argument defines the width of the drawing line.

You can also create a Pen using the following statement:

```
Dim myPen As Pen
myPen = New Pen(Drawing.Color.Blue, 5)
```

Where the first argument define the color (here is blue, you can change that to red or whatever color you want) and the second argument is the width of the drawing line.

Having created the Graphics and the Pen objects, you are now ready to draw graphics on the screen which we will show you in the following section.

In this section, we will show you how to draw a straight line on the Form. First of all, launch Visual basic 2012 Express. In the startup page, drag a Button into the Form. Double click on the Button and key in the following code.

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim myGraphics As Graphics = Me.CreateGraphics
        Dim myPen As Pen
        myPen = New Pen(Brushes.DarkMagenta, 10)
        myGraphics.DrawLine(myPen, 10, 10, 100, 100)
    End Sub
End Class
```

The second line created the Graphics object and the third and fourth line create the Pen object. The fifth draw a line on the Form using the DrawLine method. The first argument use the Pen object created by you, the second argument and the third arguments define the coordinate the starting point of the line, the fourth and the last arguments define the ending coordinate of the line. The general syntax of the Drawline argument is

```
Object.DrawLine(Pen, x1, y1, x2, y2)
```

The output of the program is shown below:

# 21. The DrawRectangle Method

## 21.1 Creating a Rectangle with DrawRectangle Method

There are two methods to draw a rectangle on the screen in VB2012:

### Method 1

Use the **DrawRectangle** method by specifying its upper-left corner's coordinate and it width and height. You also need to create a Graphics and a Pen object to handle the actual drawing. The syntax is:

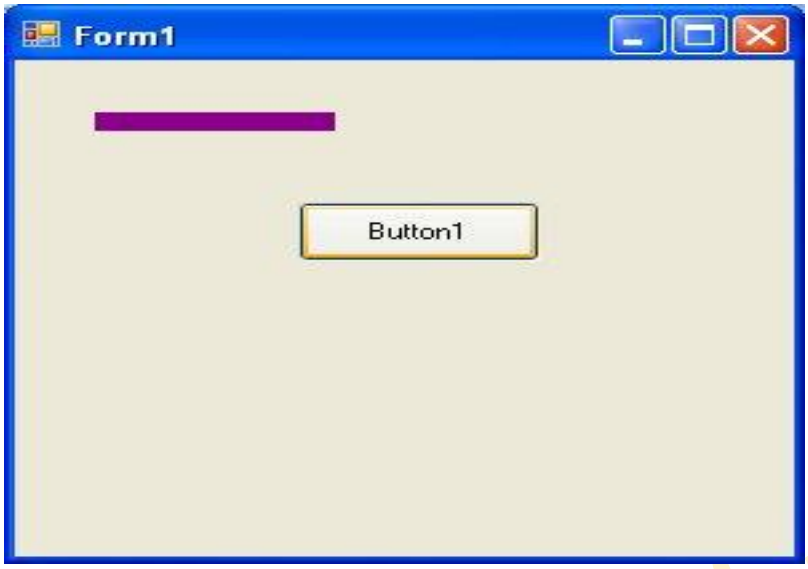
```
myGrapphics.DrawRectangle(myPen, X, Y, Width, Height)
```

*myGraphics is the variable name of the Graphics object and myPen is the variable name of the Pen object created by you. X, Y is the coordinate of the upper left corner of the rectangle.

The code

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Blue, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        myGraphics.DrawRectangle(myPen, 10, 10, 100, 50)
```

```
    End Sub
End Class
```




## Method 2

Create a rectangle object first and then draw this rectangle using the DrawRectangle method. The syntax is as shown below:

```
myGraphics.DrawRectangle(myPen, myRectangle)
```

where myRectangle is the rectangle object created by you, the user.

The code is:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Magenta, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        Dim myRectangle As New Rectangle
        myRectangle.X = 10
        myRectangle.Y = 10
        myRectangle.Width = 100
        myRectangle.Height = 50
        myGraphics.DrawRectangle(myPen, myRectangle)
```

```
    End Sub
End Class
```

You can also create a rectangle object using a one-line code as follows:

```
Dim myRectangle As New Rectangle(X,Y,width, height)
```

and the code to draw the above rectangle is

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Magenta, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        Dim myRectangle As New Rectangle(10, 10, 100, 50)
        myGraphics.DrawRectangle(myPen, myRectangle)
    End Sub
End Class
```

## 21.2 Customizing Line Style of the Pen Object

The shape we draw so far are drawn with a solid line, we can actually customize the line style of the Pen object so that we have dotted line, a line consisting of dashes and more. For example, the syntax to draw a dotted line is shown below:

myPen.DashStyle=Drawing.Drawing2D.DashStyle.Dot

Where the last argument Dot specifies a particular line DashStyle value, a line that makes up of dots here. The following code draws a rectangle with the red dotted line.

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Red, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        myPen.DashStyle = Drawing.Drawing2D.DashStyle.Dot
        myGraphics.DrawRectangle(myPen, 10, 10, 100, 50)
    End Sub
```

```
End Class
```

The output image is shown below:

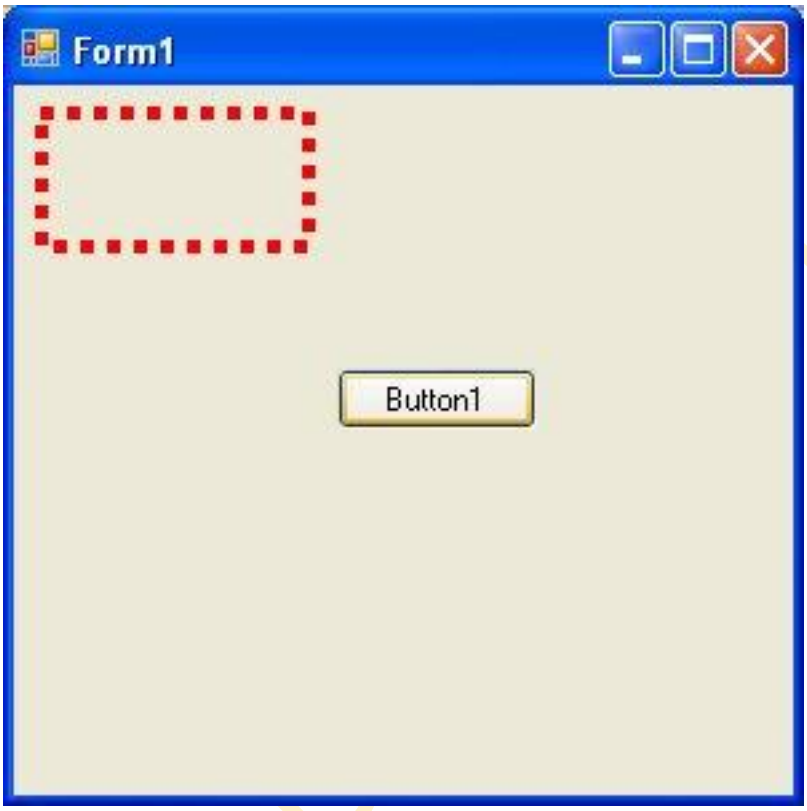



**FIGURE 22.1**

If you change the DashStyle value to DashDotDot, you can draw rectangles with different border, as shown in Figure 22.2.

The possible values of the line DashStyle of the Pen are listed in the table below:

| **DashStyle Value** | **Line Style** |
|---|---|
| Dot | Line consists of dots |
| Dash | Line consists of dashes |

| | |
|---|---|
| DashDot | Line consists of alternating dashes and dots |
| DashDotDot | Line consists of alternating dashes and double dots |
| Solid | Solid line |
| Custom | Custom line style |

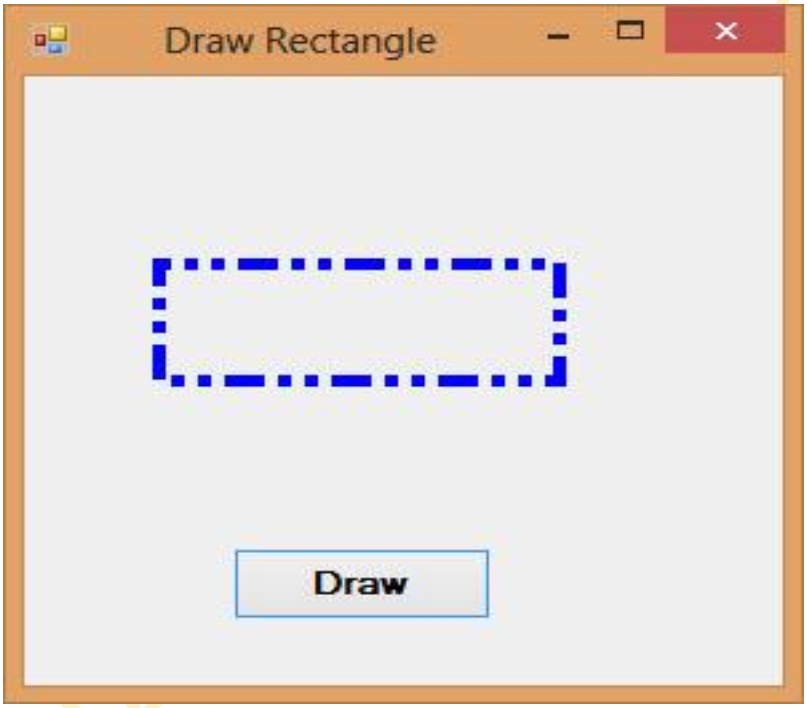


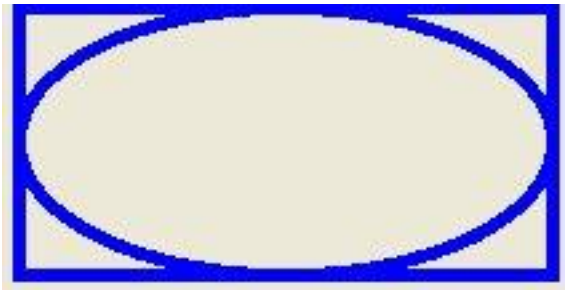

**FIGURE 22.2**

## 22. The DrawEllipse Method

In this lesson, we will learn how to draw ellipse and circle using the DrawEllipse method.

## 22.1 Drawing Ellipse with the DrawEllipse Method

The basic structure of most shapes is a rectangle, an ellipse is no exception. Ellipse is an oval shape that is bounded by a rectangle, as shown below:



Therefore, we need to create a Rectangle object before we can draw an ellipse. This rectangle serves as a bounding rectangle for the ellipse. We still need to use the DrawEllipse method to complete the job. On the other hand, we can also draw an ellipse with the DrawEllipse method without first creating a rectangle. We shall illustrate both methods.

In the first method, let's say you have created a rectangle object known as myRectangle and a pen object as myPen, then you can draw an ellipse using the following statement:

myGraphics.DrawEllipse(myPen, myRectangle)

* Assume you have also already created the Graphics object myGraphics.

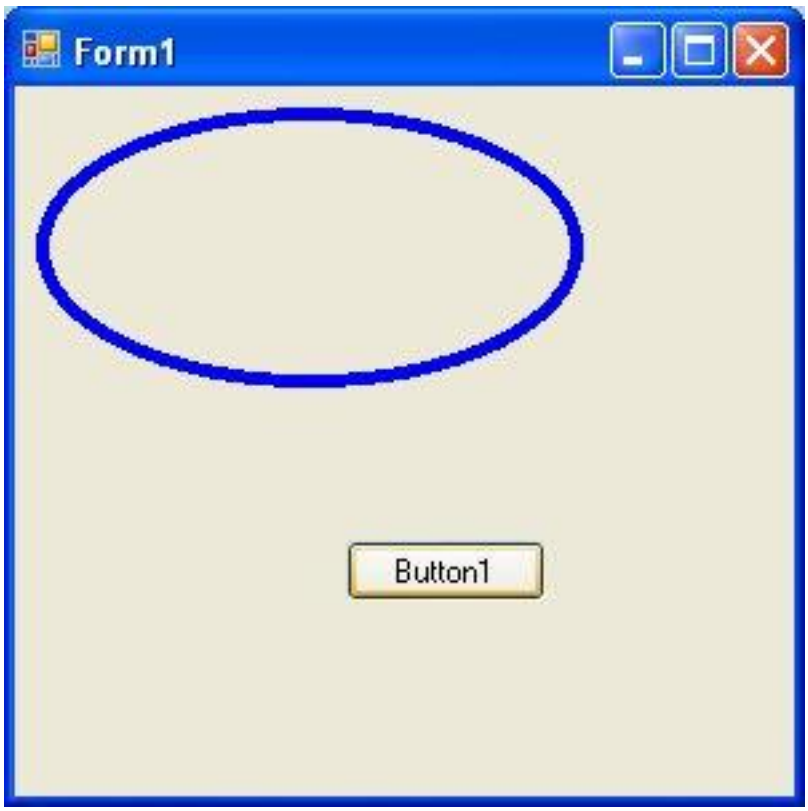
The following is an example of the full code:

Example 23.1(a)

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Blue, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        Dim myRectangle As New Rectangle
```

```
        myRectangle.X = 10
        myRectangle.Y = 10
        myRectangle.Width = 200
        myRectangle.Height = 100
        myGraphics.DrawEllipse(myPen, myRectangle)
    End Sub
End Class
```

The output image is shown in the following diagram:




The second method is using the DrawEllipse method without creating a rectangle object. Of course, you still have to create the Graphics and the Pen objects. The syntax is:

```
myGraphics.DrawEllipse(myPen, X,Y,Width, Height)
```

Where (X, Y) are the coordinates of the upper-left corner of the bounding rectangle, width is the width of the ellipse and height is the height of the ellipse.

The following is an example of the full code:

**Example 22.1(b)**

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Blue, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        myGraphics.DrawEllipse(myPen, 10, 10, 200, 100)

    End Sub
End Class
```

## 22.2 Drawing a Circle

After you have learned how to draw an ellipse, drawing a circle becomes very simple. We use exactly the same methods used in the preceding section but modify the width and height so that they are of the same values.

The following examples draw the same circle.

Example 22.2(a)

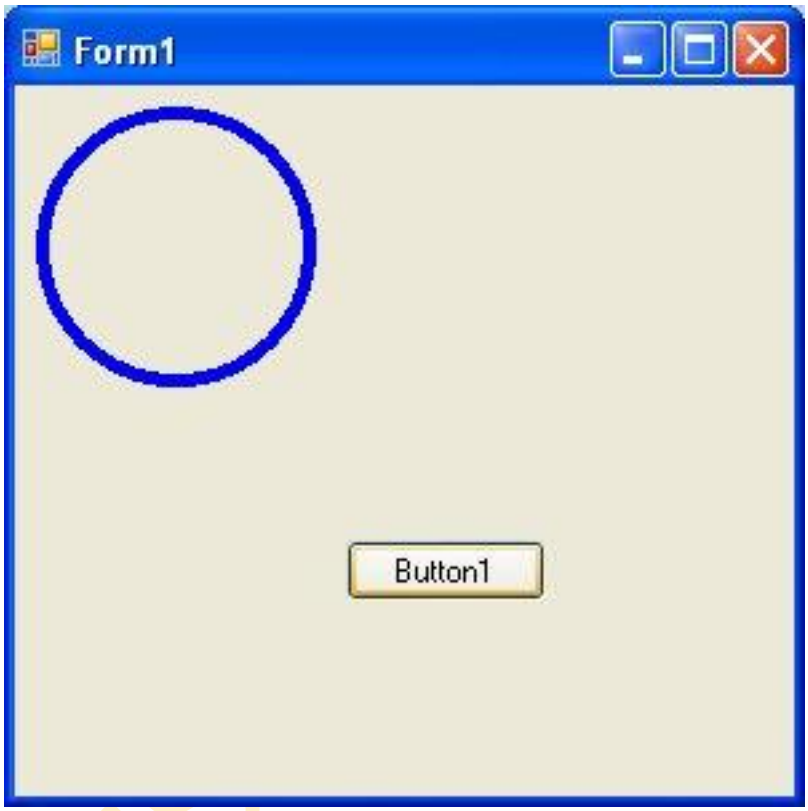
```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Blue, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        Dim myRectangle As New Rectangle
        myRectangle.X = 10
        myRectangle.Y = 10
        myRectangle.Width = 100
        myRectangle.Height = 100
        myGraphics.DrawEllipse(myPen, myRectangle)
    End Sub
End Class
```

Example 22.2(b)

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs)
    Handles Button1.Click
        Dim myPen As Pen
        myPen = New Pen(Drawing.Color.Blue, 5)
        Dim myGraphics As Graphics = Me.CreateGraphics
        myGraphics.DrawEllipse(myPen, 10, 10, 100, 100)
    End Sub
End Class
```

The output image is show below:




## 23. Using Timer

In this lesson, we shall show you how to use the timer in Visual Basic 2012. The timer is used to manage events that are time-related. For example, you can use the timer to create a clock, a stopwatch, a dice, animation and more.

## 27.1 Creating a Clock

To create a clock, you need to use the Timer control that comes with Visual Basic 2012. The Timer control is a control object that is only used by the developer, it is invisible during runtime and it does not allow the user to interact with it.

First of all, start a new project in Visual Basic 2010 and select a new Windows Application. You can give the project any name you wish, we named it MyClock. Change the caption of the Form1 to MyClock in the properties window. Now add the Timer control to the Form by dragging it from the control tool Box. Next, insert a Label control into the Form. Change the Font size of the Label to 14 or any size you wish, and set the Font alignment to be the middle center. Lastly, you shall also set the Interval property of the Timer control to 1000, which reflects a one-second interval(1 unit is 1 millisecond). Besides, set the timer Enabled property to True so that it will start ticking when the program is started.

The statement to create a clock is only a one-line code, as follows:
Label1.Text = TimeOfDay

To create the clock, click on the Timer control and insert the code above, as shown below:

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs)
    Handles Timer1.Tick
        Label1.Text = CStr(TimeOfDay)
    End Sub
End Class
```

The resulting Clock is shown below:


## 27.2 Creating a Stopwatch

We can create a simple stopwatch using the Timer control. Start a new project and name it stopwatch. Change the Form1 caption to Stopwatch. Insert the Timer control into the Form and set its interval to 1000 which is equal to one second. Besides, set the timer Enabled property to False so that it will not start ticking when the program is started. Insert three command Buttons and change their names to StartBtn, StopBtn and ResetBtn respectively. Change their text to "Start", "Stop" and "Reset" accordingly. Now, key in the code as follows:

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs)
    Handles Timer1.Tick
        'To increase one unit per second
        Label1.Text = CStr(Val(Label1.Text) + 1)
    End Sub

    Private Sub StartBtn_Click(sender As Object, e As EventArgs)
    Handles StartBtn.Click
        'To start the Timer
        Timer1.Enabled = True
    End Sub

    Private Sub StopBtn_Click(sender As Object, e As EventArgs)
    Handles StopBtn.Click
        'To stop the Timer
        Timer1.Enabled = False
    End Sub

    Private Sub ResetBtn_Click(sender As Object, e As EventArgs)
    Handles ResetBtn.Click
        'To reset the Timer to 0
```

```
        Label1.Text = CStr(0)
    End Sub
End Class
```

The Interface of the Stopwatch is as shown below:


## 24. Creating Animation

Although Visual Basic 2012 is a programming language designed for creating business applications, it can be used to create animation. In this lesson, we shall illustrate to create animation with VB2012 through a few examples.

### 28.1 Moving an object

In VB2012, you can use the **Top** and **Left** properties of an object to create animation. The Top property defines the distance of the object

from the topmost border of the screen while the Left property defines the distance of the object from the leftmost border of the screen.
By adding (+) or subtracting (-) the distance of the object we can create the animated effect of moving an object. Start a new project and name it as any name you wish. Now insert a PictureBox and in its Image property import a picture from your hard drive or other sources. Next, insert four command Buttons, change their captions to Up, Down, Left and Right. Name them as MoveUpBtn, MoveDowbBtn, MoveLeftBtn and MoveRightBtn.

```
Public Class Form1
    Private Sub MoveUpBtn_Click(sender As Object, e As EventArgs) Handles MoveUpBtn.Click
        PictureBox1.Top = PictureBox1.Top - 10
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
        PictureBox1.Left = PictureBox1.Left - 10
    End Sub

    Private Sub MoveRightBtn_Click(sender As Object, e As EventArgs) Handles MoveRightBtn.Click
        PictureBox1.Left = PictureBox1.Left + 10
    End Sub

    Private Sub MoveDowbBtn_Click(sender As Object, e As EventArgs) Handles MoveDowbBtn.Click
        PictureBox1.Top = PictureBox1.Top + 10
    End Sub
End Class
```

Explanation:

Each time the user clicks on the Down Button, the distance of the PictureBox increases by 10 pixels from the top border, creating a downward motion. On the other hand, each time the user clicks on the Up Button, the distance of the PictureBox decreases by 10 pixels from the top borders, thus creating an upward motion. In addition, each time the user clicks on the Left Button, the distance of the PictureBox decreases by 10 pixels from the left border, thus creating a leftward motion. Lastly, each time the user clicks on the Right Button, the distance of the PictureBox increases by 10 pixels from the left border, thus creating a rightward motion. The interface is shown below:


## 28.2 Creating Animation using Timer

We can create auto animation using timer without the need to manually clicking a command Button. we shall show you how to write the code. First, insert a PictureBox into the Form. In the PictureBox properties window, select the image property and click to import an image file from your external sources such as your hard drive, your Pen drive or DVD. We have inserted an image of a bunch of grapes. Next, insert a Timer control into the Form set its interval property to 100, which is equivalent to 0.1 seconds. Finally, add two command Buttons to the Form, name one of them as AnimateBtn and the other one as StopBtn, and change to caption to Animate and Stop respectively.

We make use of the Left property of the PictureBox to create the motion. PictureBox.Left means the distance of the PictureBox from the left border of the Form. Now click on the Timer control and type in the following code:

```
Public Class Form1
    Private Sub Timer1_Tick(sender As Object, e As EventArgs)
    Handles Timer1.Tick
       If PictureBox1.Left < Me.Width Then
            PictureBox1.Left = PictureBox1.Left + 10
        Else
            PictureBox1.Left = 0
        End If
    End Sub
     Private Sub Button1_Click(sender As Object, e As EventArgs)
     Handles Button1.Click
        Timer1.Enabled = True
    End Sub
     Private Sub Button2_Click(sender As Object, e As EventArgs)
     Handles Button2.Click
        Timer1.Enabled = False
    End Sub
End Class
```

In aforementioned code, Me.Width represents the width of the Form. If the distance of the PictureBox from the left is less than the width of the Form, a value of 10 is added to the distance of the PictureBox from the left border each time the Timer ticks, or every 0.1 seconds in this example. When the distance of the PictureBox from the left border is equal to the width of the Form, the distance from the left border is set to 0, which move the PictureBox object to the left border and then move left again, thus creates an oscillating motion from left to right. We need to insert a Button to stop motion. The code is:

```
Timer1.Enabled = False
```

To animate the PictureBox object, we insert a command Button and key in the following code:

```
Timer1.Enabled = True
```

The Image of the Animation program is shown below: