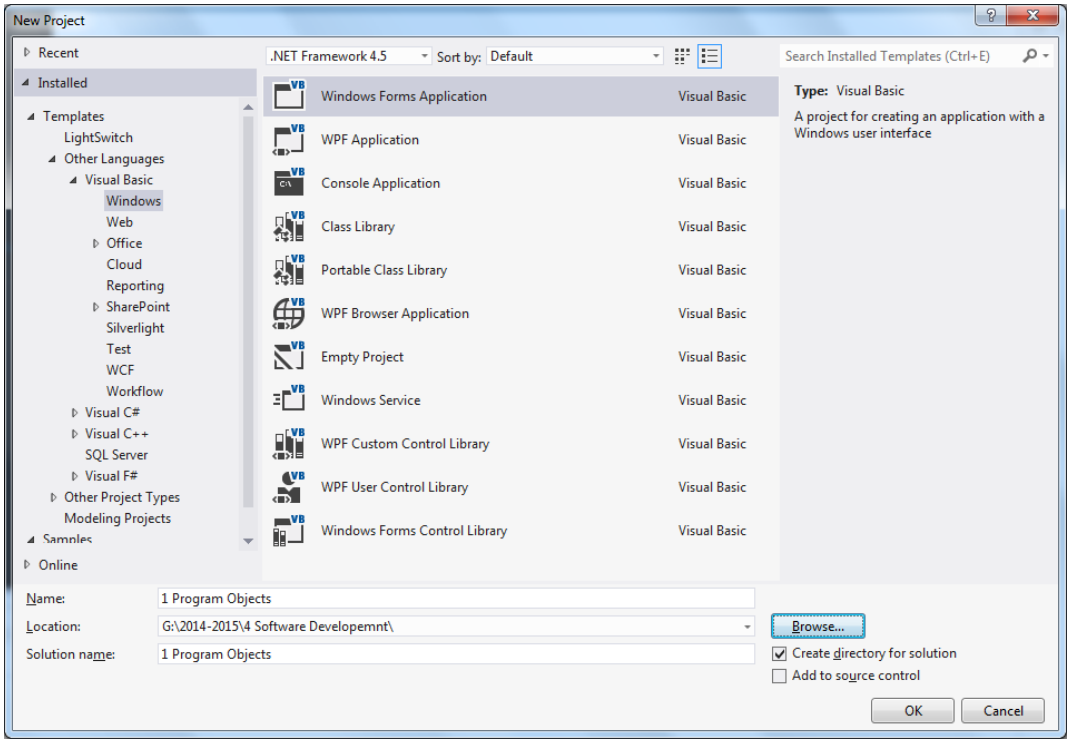
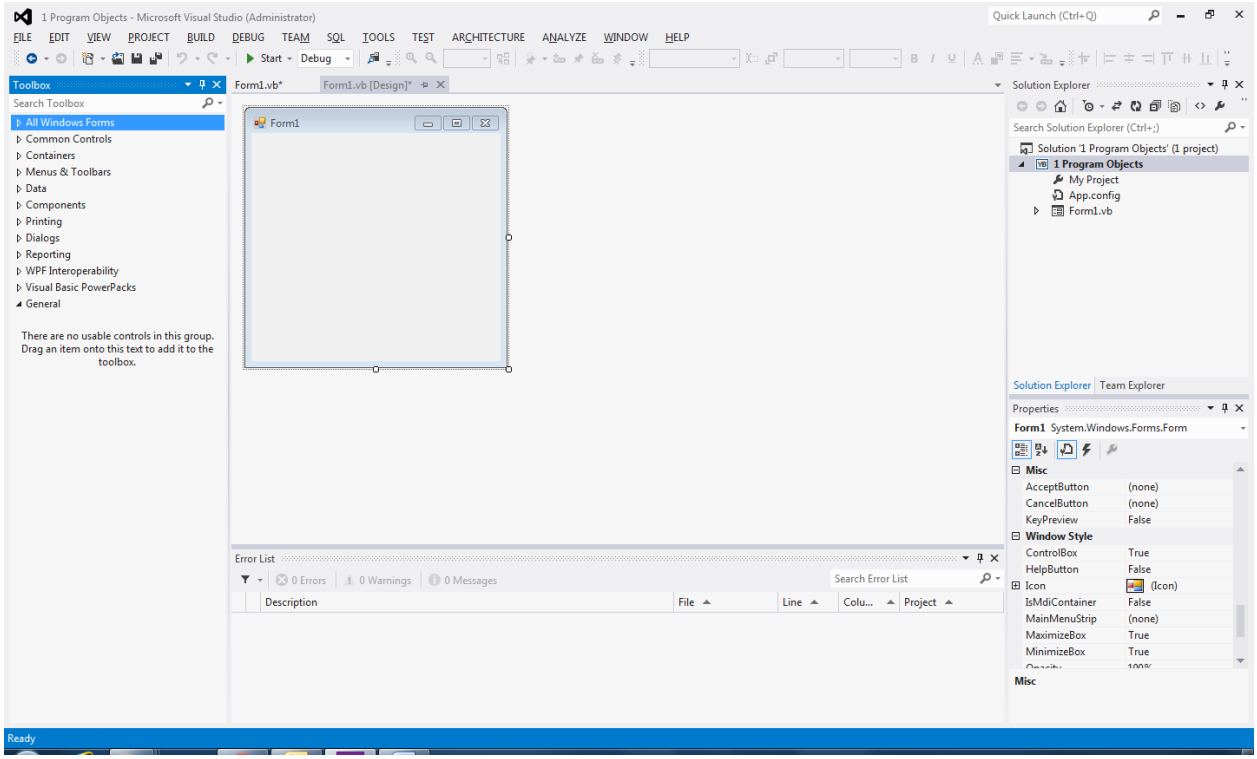
Visual Basic

IDE – Integrated Development Environment





Objects – Properties – Methods

Objects

The following are examples of objects:

- Form
- Button
- Groupbox
- Combobox
- Checkbox

Properties

The following are examples of objects:

- Height
- Width
- Colour
- Name
- Text

Events

The following are examples of objects:

- Clicking mouse button
- Pressing a key
- Scrolling on mouse

Methods

A method is a procedure created as a member of a class. Methods are used to access or manipulate the characteristics of an object or a variable. There are mainly two categories of methods you will use in your classes:

If you are using a control such as one of those provided by the Toolbox, you can call any of its public methods. The requirements of such a method depend on the class being used.

A method is an action that responds to an event. For example, let's say you have a button. A user clicks this button, which is the event. Your program should act according to the event, which it does by running the code inside a method.

The following are examples of objects:

- Print form
- Show

Variables

What are Variables?

A variable is a storage location in computer memory and is used for storing information while the program is running**.** The information that is stored in a variable may change, hence its called a variable

About Variables

You the programmer make up a name for the variable. Visual Basic associates that name with a location in the computer's RAM. The value currently associated with the variable is stored in that memory location

Declaring Variables

A variable declaration is a statement that creates a variable in memory

Example: *Dim VariableName As DataType*

Dim is an abbreviation for Dimension
VariableName is the programmer designated name
As is a keyword
DataType is one of many possible keywords for the type of value the variable will contain

Example: Dim intLength as Integer

Declaring Multiple Variables

Dim intLength, intWidth, intDepth as Integer Or in 3 separate statements

Dim intLength as Integer
Dim intWidth as Integer
Dim intDepth as Integer

Visual Basic Data Types

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and etc everyday. Similarly in Visual Basic 2010, we have to deal with all sorts of data, some can be mathematically calculated while some are in the form of text or other forms. VB2010 divides data into different types so that it is easier to manage when we need to write the code involving those data.

Numeric Data Types

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

Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .They are summarized in Table 6.2

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

Suffixes for Literals

Literals are values that you assign to a data. In some cases, we need to add a suffix behind a literal so that VB2010 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 below

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

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

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

Managing Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In term of VB 2010, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Visual Basic 2010, you have to follow a set of rules.

Variable Names

The following are the rules when naming the variables in Visual Basic 2010

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

Examples of valid and invalid variable names are displayed below

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

Basic mathematical operations

Addition (+) Subtraction (-) Multiplication (*) Division (/) Exponentiation (^) Integer Division (\)
Finding the remainder (Mod)

However, for other operations, you can use the methods available in the System.Math class.

Some of the members of the System.Math class include the following:

Trigonometric functions (Sin, Cos, Tan, etc)
Logarithmic functions (Log and Log10)
Constants (PI and E)
Power functions (Exp, Pow, and Sqrt)
Boundary functions (Floor, Ceiling)
Comparative functions (Max, Min)
Sign-related functions (Abs)

Example

```
Private Sub PerformMathFunctions()
    Dim i As Integer

    i = Math.Pow(2, 3)
    MessageBox.Show(i)
    i = Math.Sqrt(16)
    MessageBox.Show(i)
    i = Math.Round(5.34444)
    MessageBox.Show(i)
End Sub
```

Constants

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

Declaring a Constant

The format to declare a constant is
Const *Constant Name* As *Data Type = Value*

Example

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Const Pi As Single=3.142
    Const Temp As Single=37
    Const Score As Single=100
End Sub
```

Documenting – Commenting – Indenting

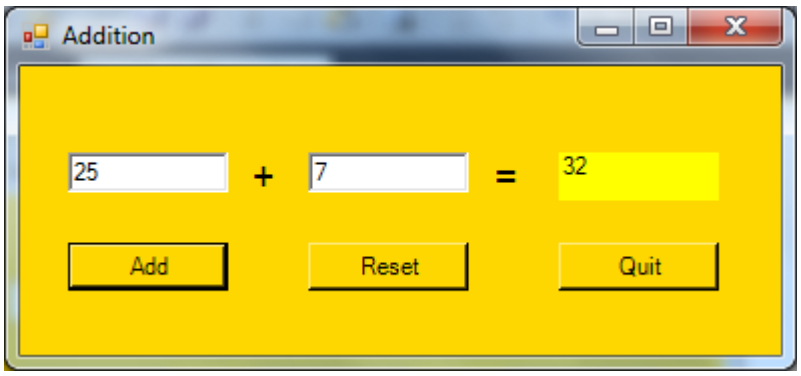
```
' Name:        Ray O'Connor
' College:     Colaiste Stiofain Naofa
' Course:      Software Development

Dim num1 As Single, num2 As Single, answer As Single

Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System. EventArgs) Handles btnAdd.Click
        ' number in first textbox converted to a number and stored in memory
        num1 = Val(txtNum1.Text)

        ' number in second textbox converted to a number and stored in memory
        num2 = Val(txtNum2.Text)
        If (num1 = 0) Or (num2 = 0) Then
                ' a message box pops up
                MsgBox("Error - you must enter a number")
        Else
                ' first number added to second and stored in memory
                answer = num1 + num2
                lblDisplay.Text = answer
        End If
End Sub
```

Addition Application



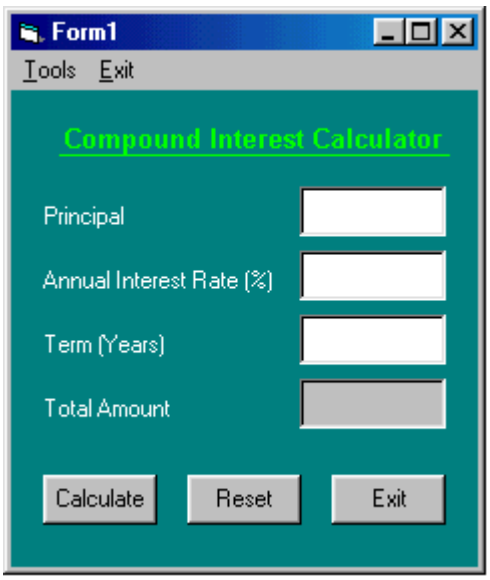

```
Dim num1 As Single, num2 As Single, answer As Single

Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAdd.Click
    num1 = Val(txtNum1.Text)
    num2 = Val(txtNum2.Text)
    If (num1 = 0) Or (num2 = 0) Then
       MsgBox("Error - you must enter a number")
    Else
       answer = num1 + num2
       lblDisplay.Text = answer
    End If
End Sub

Private Sub btnReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnReset.Click
    txtNum1.Text = ""
    txtNum2.Text = ""
    lblDisplay.Text = ""
   num1=num2=answer=0
End Sub
```

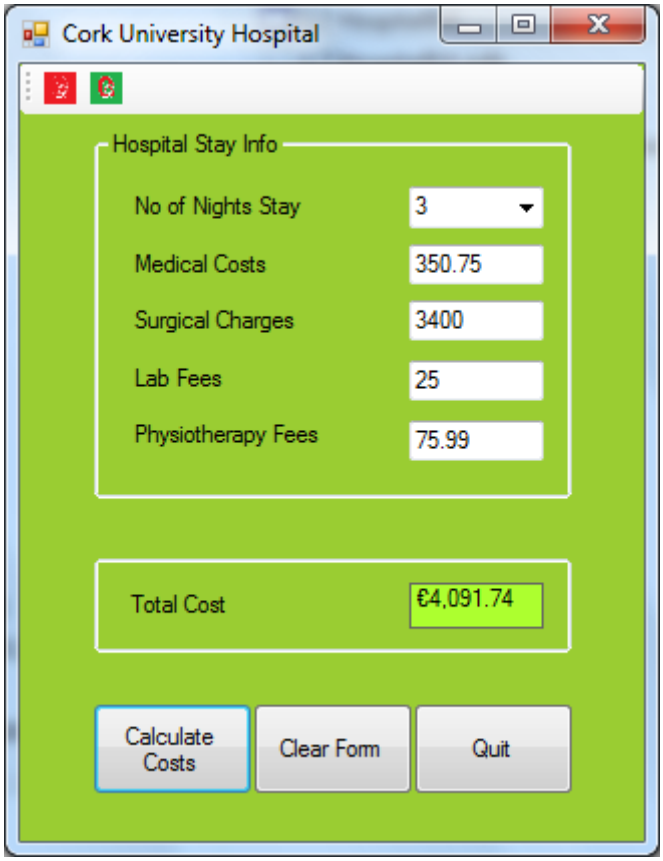
Exercise – Compound Interest Calculator

You are required to design the form as shown and add the correct VB source code to calculate the compound interest on a principal amount for a number of years at a specified interest rate.

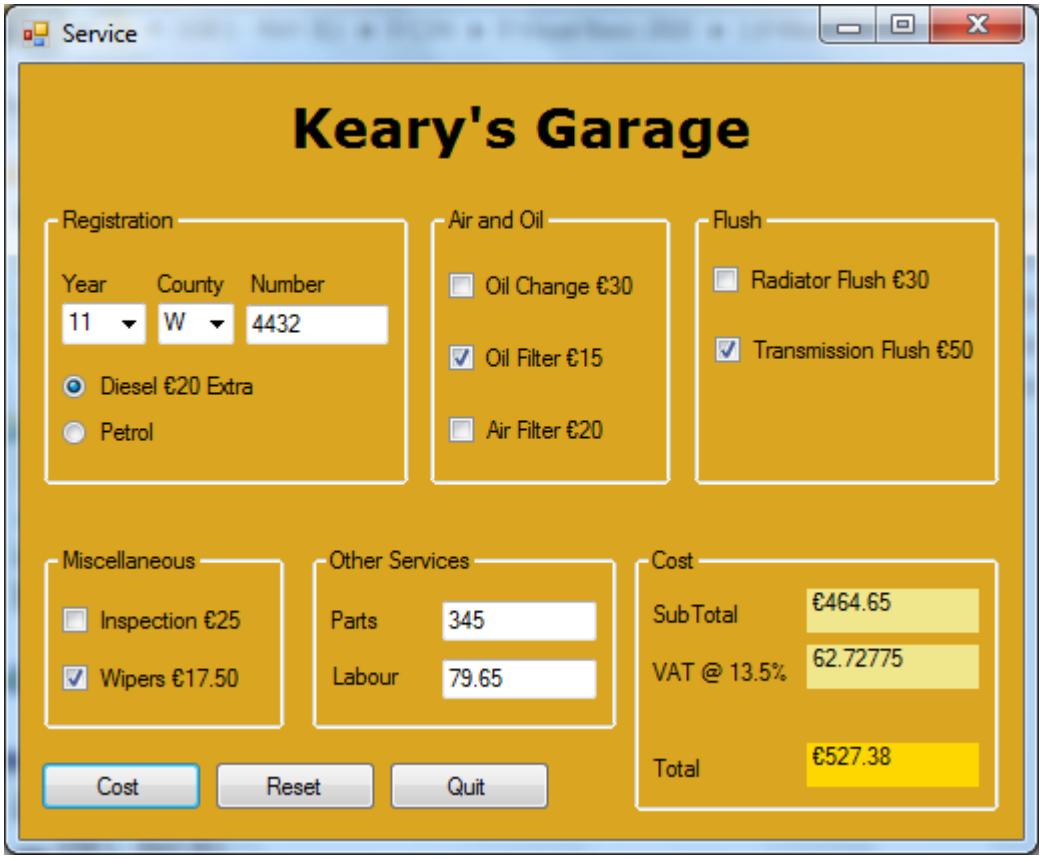


Exercises

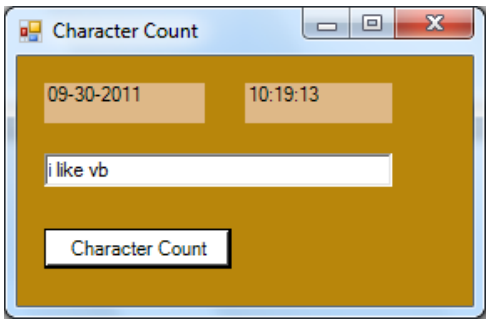
Exercise 1: You are required to design and code the following program.

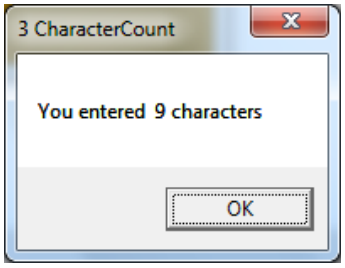


Exercise 2: You are required to design and code the following program.



Character Count Program



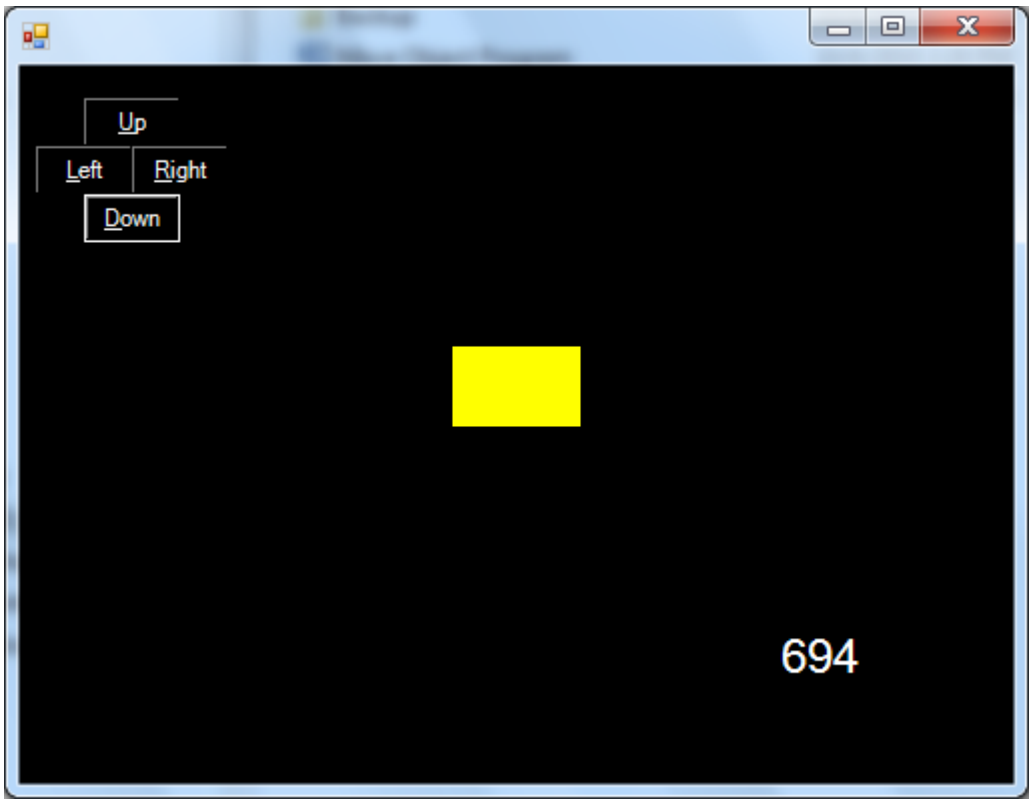



```
Private Sub cmdCharCount_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdCharCount.Click
        MsgBox("You entered " & Str$(Len(txtCharacters.Text)) & " characters")
End Sub

Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        lblDate.Text = DateString
        lblTime.Text = TimeString
End Sub

Private Sub tmrTime_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
tmrTime.Tick
        lblTime.Text = TimeString
End Sub
```

Move Label or Image around form




```
Dim time As Single

Private Sub btnDown_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnDown.Click
        lblTarget.Location = New Point(lblTarget.Location.X, lblTarget.Location.Y + 10)
End Sub

Private Sub btnLeft_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnLeft.Click
        lblTarget.Location = New Point(lblTarget.Location.X - 10, lblTarget.Location.Y)
End Sub

Private Sub btnRight_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnRight.Click
        lblTarget.Location = New Point(lblTarget.Location.X + 10, lblTarget.Location.Y)
End Sub

Private Sub btnUp_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnUp.Click
        lblTarget.Location = New Point(lblTarget.Location.X, lblTarget.Location.Y - 10)
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles tmr1.Tick
        time = time + 1
        lblTime.Text = time
End Sub

Private Sub btnDown_MouseHover(sender As Object, e As System.EventArgs) Handles btnDown.MouseHover
        lblTarget.Location = New Point(lblTarget.Location.X, lblTarget.Location.Y + 10)
End Sub
```

Conditions

Sometimes you have to make some choices, and conditional expressions will help you do just that. Visual Basic includes support for conditions, which use data tests to determine which code should be processed next.

If Statements

The most common conditional statement is the If statement. It is equivalent to English questions in the form "If such-and-such is true, then do so-and-so." For instance, it can handle "If you have €20, then you can buy me dinner," but not "If a train departs Chicago at 45 miles per hour, when will it run out of coal?"

If statements example:

```
If (num1 = 0) Or (num2 = 0) Then
    MsgBox("Error - you must enter 2 numbers")
Else
    answer = num1 + num2
    lblDisplay.Text = answer
End If
```

Select Case Statements


```
Private Sub btnEuroNote_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnEuroNote.Click
        Dim NoteColour As String
        Dim EuroNote As Integer

        EuroNote = Val(cboEuroNotes.Text)
        Select Case Euronote
            Case 5
                NoteColour = "Grey"
            Case 10
                NoteColour = "Red"
            Case 20
                NoteColour = "Blue"
            Case 50
                NoteColour = "Orange"
            Case 100
                NoteColour = "Green"
            Case 200
                NoteColour = "Yellow"
                ' 500 euro is next
            Case Else
                NoteColour = "Purple"

                '        Case 10, 100
                '      presidentName = "!! Non-president"
                '  Case Is > 100
                '      presidentName = "!! Value too large"
                '  Case Else
```

```
            '    presidentName = "!! Invalid value"
        End Select
        lblColour.Text = NoteColour
    End Sub
```

Events (Click, SelectedIndexChanged)

We have mostly looked at the Click event but there are many other events for various controls. We will look at the SelectedIndexChanged for a comboBox as follows:

With the following code you click on a number in the comboBox and once you click the number the code below is executed.

```
Private Sub cboEuroNotes_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cboEuroNotes.SelectedIndexChanged
        Dim NoteColour As String
        Dim EuroNote As Integer
        EuroNote = Val(cboEuroNotes.Text)
        Select Case EuroNote
            Case 5
                NoteColour = "Grey"
            Case 10
                NoteColour = "Red"
            Case 20
                NoteColour = "Blue"
            Case 50
                NoteColour = "Orange"
            Case 100
                NoteColour = "Green"
            Case 200
                NoteColour = "Yellow"
                ' 500 euro is next
            Case Else
                NoteColour = "Purple"
        End Select
        lblColour.Text = NoteColour
End Sub
```

Another example Select Case

Select Case billValue

Case 1

presidentName = "Washington"

Case 2

presidentName = "Jefferson"

Case 5

presidentName = "Lincoln"

Case 20

presidentName = "Jackson"

Case 50

presidentName = "Grant"

Case 10, 100

presidentName = "!! Non-president"

Case > 100

```
        presidentName = "!! Value too large"
Case Else
        presidentName = "!! Invalid value"
End Select
```

Loops

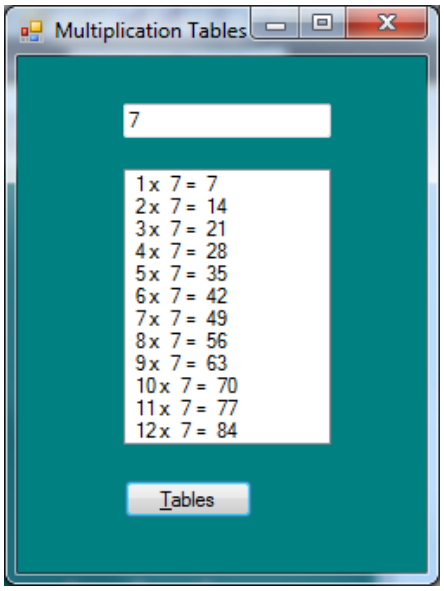
Visual Basic includes three major types of loops: For...Next, For Each...Next, and Do...Loop. Just as conditions allow you to break up the sequential monotony of your code through branches, loops add to the usefulness of your code by letting you repeat a specific block of logic a fixed or variable number of times.

For. . .Next Loops

The For...Next loop uses a numeric counter that increments from a starting value to an ending value, processing the code within the loop once for each incremented value.

```
Dim whichMonth As Integer
For whichMonth = 1 To 12
        ProcessMonthlyData(whichMonth)
Next whichMonth
```

This sample loops 12 times (1 To 12), once for each month.




```
Private Sub btnTables_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnTables.Click
        Dim counter As Integer, num As Integer
        lstTables.Items.Clear()
        num = Val(txtNumber.Text)
        For counter = 1 To 12
            lstTables.Items.Add(Str(counter) + " x " +
        Str(num) + " = " + Str(counter * num))
        Next counter
End Sub
```

For Each. . .Next Loops

A variation of the For loop, the For Each...Next loop scans through a set of ordered and related items, from the first item until the last. Arrays and collection objects also work, as does any object that supports the IEnumerable interface (all these topics are covered in Chapter 6). The syntax is quite similar to the standard For statement:

```
For Each oneRecord In setOfRecords
        ProcessRecord(oneRecord)
Next oneRecord
```

Do. . .Loop Loops

The Multiplication Tables program we designed earlier can be written using the Do While loop instead of the For .. Next loop

```
Private Sub btnTables_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnTables.Click
        Dim counter As Integer, num As Integer
        lstTables.Items.Clear()
        num = Val(txtNum.Text)
        counter = 1
        Do While (counter < 13)
            lstTables.Items.Add(Str(counter) & " x " & Str(num) & " = " & Str(counter * num))
            counter = counter + 1
        Loop
End Sub
```

Other Examples of Do loops

Sometimes you want to repeat a block of code as long as a certain condition is true, or only until a condition is true. The Do...Loop structure performs both of these tasks. The statement includes a While or Until clause that specifies the conditions for continued loop processing. For instance, the following statement does some processing for a set of dates, from a starting date to an ending date:

```
Dim processDate As Date = #1/1/2000#
Do While (processDate < #2/1/2000#)' ----- Perform processing for the current date.
        ProcessContent(processDate)
        ' ----- Move ahead to the next date.
        processDate = processDate.AddDays(1)
Loop
```

Processing in this sample will continue until the processDate variable meets or exceeds 2/1/2000, which indicates the end of processing. The Until clause version is somewhat similar, although with a reversed condition result:

```
Do Until (processDate >= #2/1/2000#)
...
Loop
```

Make the included condition as simple or as complex as you need. Putting the Until or While clause at the bottom of the loop guarantees that the statements inside the loop will always be processed at least once:

```
Do
...
Loop Until (processDate >= #2/1/2000#)
```

If the loop condition is never met, the loop will continue forever. So, if you want your loop to exit at some point (and usually you do), make sure the condition can eventually be met.

There is another loop that is similar to Do...Loop, called the While...End While loop. However, it exists for backward compatibility only. Use the Do...Loop statement instead.

Sequential and Binary Searches

A linear search or sequential search looks down a list, one item at a time, without jumping. In complexity terms this is an `O(n)` search - the time taken to search the list gets bigger at the same rate as the list does.
A binary search is when you start with the middle of a sorted list, and see whether that's greater than or less than the value you're looking for, which determines whether the value is in the first or second half of the list. Jump to the half way through the sublist, and compare again etc. This is pretty much how humans typically look up a word in a dictionary (although we use better heuristics, obviously - if you're looking for "cat" you don't start off at "M"). In complexity terms this is an `O(log n)` search - the number of search operations grows more slowly than the list does, because you're halving the "search space" with each operation.
As an example, suppose you were looking for U in an A-Z list of letters (index 0-25; we're looking for the value at index 20).

A linear search would ask:

`list[0] == 'U'`? No.
`list[1] == 'U'`? No.
`list[2] == 'U'`? No.
`list[3] == 'U'`? No.
`list[4] == 'U'`? No.
`list[5] == 'U'`? No.
... `list[20] == 'U'`? Yes. Finished.
The binary search would ask:

Compare `list[12]` ('M') with 'U': Smaller, look further on. (Range=13-25)
Compare `list[19]` ('T') with 'U': Smaller, look further on. (Range=20-25)
Compare `list[22]` ('W') with 'U': Bigger, look earlier. (Range=20-21)
Compare `list[20]` ('U') with 'U': Found it! Finished.
Comparing the two:

- Binary search requires the input data to be sorted; linear search doesn't
- Binary search requires an ***ordering*** comparison; linear search only requires equality comparisons
- Binary search has complexity O(log n); linear search has complexity O(n) as discussed earlier
- Binary search requires random access to the data; linear search only requires sequential access (this can be very important - it means a linear search can ***stream*** data of arbitrary size)

Sort Algorithms

Bubble Sort, Selection Sort are examples of two sort algorithm.

Sub General Subroutines

Functions