The For – Next Statement

Loops are used to repeat a section of code again and again. The **For – Next** statement is one of the principal looping statements of Basic.

A **For – Next** loop is called a *definite* loop. In a definite loop, the number of times the code contained in the loop will repeat is generally known when the loop begins.

For *CounterVariable* = *StartValue* to *EndValue* [Step *Increment*]
    Statement
    (more statements may follow)
Next [*CounterVariable*]

CounterVariable – is the variable to be used as the counter. It must be a number.
StartValue is the value the counter variable will be initially set to.
EndValue is the value the counter variable is tested against just prior to each iteration of the loop.
Step Increment - is the amount added to the counter variable at the end of each iteration. The default is 1.
Next[CounterVariable] statement marks the end of the loop and causes the counter variable to be incremented.

1.
```
Dim Count as Integer
For Count = 1 to 10
  lstOutput.Items.Add Count
Next Count
```

2.
```
Dim Count as Integer
For Count = 0 to 10 Step 2
lstOutput.Items.Add Count
Next Count
```

3.
```
Dim x as Integer
    For x = 100 to 0 Step -5
  lstOutput.Items.Add x
    Next x
```

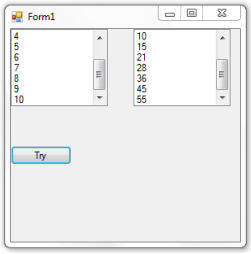
4.
```
Dim x as Integer
    For x = 1 to 5
    lstOutput.Items.Add “Hello”
    Next x
```

5.
```
Dim x as Integer, FirstName as String, LastName as String, Address as String
FirstName = “Bart”
LastName = “Simpson”
Address = “Miramichi”
For x = 1 to 5
      lstOutput.Items.Add(FirstName)
      lstOutput.Items.Add(LastName)
      lstOutput.Items.Add(Address)
Next x
```

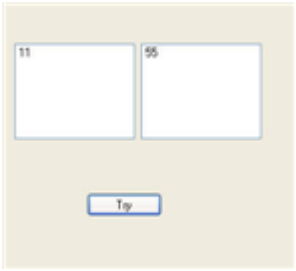
Running Totals

An assignment statement can be placed inside a loop that will keep a running total of certain values. Think of all the times information must be totaled in computer programs.
Use two different list boxes.
eg.

6.
```
Dim x As Integer, Total As Integer
  For x = 1 To 10
      Total = Total + x
      lstx.Items.Add(x)
      lstOutput.Items.Add(Total)
        Next x
```



6b.
```
Dim x As Integer, Total As Integer
  For x = 1 To 10
      Total = Total + x
         Next x
       lstx.Items.Add(x)
      lstOutput.Items.Add(Total)
```



Conversion Tables

For – Next Loops provide a handy means of converting values, such as Imperial and Metric measurements, and printing out a conversion table. eg.

11.
```
Dim Inches As Single, Centimeters As Single
  For Centimeters = 1 To 10
      Inches = Centimeters / 2.54
      lstOutput.Items.Add(Centimeters)
      lstOutput1.Items.Add(Inches)
  Next Centimeters
End Sub
```

PROGRAMMING PROBLEMS

1. Create a program with a listbox in which to display information. Add a command button which would do each of the following (Make the required changes, one by one.):
   (a) count by 1's from 0 to 5
   (b) count by 1's from 1 to 12
   (c) count by 2's to show the even numbers from 0 to 20
   (d) count by 2's to show the odd numbers from 1 to 15
   (e) count backwards from 10 to 1
   (f) display your first name 10 times and have the lines numbered

2. Create a program to display a table under the headings KMH and MPH. Kilometers should be listed under the appropriate heading from 40 to 120 with the equivalent conversions under MPH. Do this in steps of 10. 1km = .6213 miles

3. Create a program to **add** all the numbers from 1 to 20 and display only the answer. Then modify the program to add all the even numbers from 0 to 50.

4. Create a program that will convert Fahrenheit degrees to Celsius. The range of Fahrenheit degrees is to go from 32 to 212 in steps of 10. Print out a table showing corresponding values under the headings "Fahrenheit" and "Celsius." To calculate the number of degrees Celsius, first subtract 32 from the number of F degrees and then take 5/9 of the remainder.

5. Create a program to display:
   (a) the five times table
   (b) all the times tables, one after another

Code Modules

The code module contains no form definitions at all. It contains procedures and declarations that solve particular problems. These modules are created and added to the list of modules in the Project window by clicking on **Project** on the menu bar and then **Add Module**. This module contains code used as a common resource for forms. It contains code shared by more than one form and ensures that the variables declared in the code module are available for use by all other modules including form modules.

The Scope of Variables

A variable's scope is determined by its: visibility and lifetime. Visibility is a question of the number of procedures, forms, and modules that can use the variable.

Variable Types

Local Declared by the **Dim** statement inside event procedures.
Can be used only by that one event.
Value is lost when program again enters same event procedure.

Static A kind of local variable.
Can be used only by one event.
Value is not lost when program again enters same event procedure.
Declared by **Static** statement inside event procedure.
eg. *Static ClickCount as Integer*

Global Used to share values among procedures.
2 Types
Global within a form module
Can be used by all events on the same form.
Declared by the **Dim** statement placed in the **General Declarations** section of the form.

Global within all forms and all procedures in project
Can be used by all events on all forms.
Declared by the **Public** keyword in the **General Declarations** section of the **Code Module**.
eg. *Public ClickCount as Integer*

Exercise to Show how Variable Types Work

1. Place the following on a form in a new project:
   - a label named **lblDisplay**. Delete the text
   - a button named **btnCount**. Make **Count** the text.
   - a button named **btnExit**. Make **Exit** the text.

2. Code for the **btnCount** button is:

```
Dim x as integer
    x = x + 1
    lblDisplay.text = x
```

3. Run the program and click on **Count** several times. Why does the value not increase?

4. Change the code in the **btnCount** button so the first line is:

```
Static x as integer
```

5. Run the program again and click **Count** several times. What has changed?

6. Add another button to your form named **Count 2**. Change the Text to **Count 2**. Put exactly the same code in this event procedure as in **Count**. What happens if you click several times on one button and then several times on the other?

7. Add another command button named **Clear**, text **Clear**. Code for this button is:
   lblDisplay.Caption = “”

8. Now delete the **Static** lines from both the command buttons. Add this line for the **under this** section of the form:
   Public class form1
   Dim x as Integer

9. Run the program and hit each of the **Count** buttons several times. Watch the result.

10 Add a new form to your project. Put a **Back** button on this form with code to take you back to Form 1. (Me.Hide() form1.show()) Put a **Continue** button on Form 1 with code to take you to Form 2. (Me.Hide() form2.show) Put a **Count 3** button on Form 2. The code for this button should match the code for the **Count** Button on Form 1. Then run the program and try all buttons. Is the value of **X** passed to Form 2?

10. Now delete the **Dim** statement from the **General Declarations** area of the form.
11. Add a **Code Module**. Put this line of code in the **General Declarations** section of the **Code Module.** *Public x as Integer*

Random Numbers

Randomize—Randomize reseeds the random number generator. In other words, it sets up a different sequence of random numbers. This command must be placed before you attempt to come up with a random number.

Rnd—Rnd causes the computer to choose a random number between 0 and 1. This then has to be changed slightly to get a random number in the range you wish.

Set up a form with the following command buttons—**List Numbers** and **Exit**.

Code for the List Numbers button

```
Dim x as Single, r as Single
Randomize
For x = 1 to 10
        r = rnd ()
        lstRandom.Items.Add(r)
Next x
```

Delete the Randomize statement, run the program a few times, and see what happens.

Put the Randomize statement back in the program.

Then change the r = rnd statement to each of the following in turn, run the program, and note the result.

```
r = r * 10
r = Int(rnd * 10)
r = Int(rnd * 10) +1
r = Int(rnd * 6) +1
r = Int(rnd * 3) +5
```

Programming Problems

Now try the following :

1. Similar to the example above, create a program to list a series of 15 random numbers to satisfy each of the following:
   - (a) from 0 to 5 inclusive
   - (b) from 1 to 5 inclusive
   - (c) from 1 to 6 inclusive
   - (d) from 0 to 255 inclusive
   - (e) a 1 or a 2
   - (f) from 25 to 50 inclusive

2. (a) Create a program to generate a series of 100 random numbers between 1 and 4 inclusive. Put these in a listbox.

(b) Revise the above program to keep track of how many times each of the 1, 2, 3, or 4 is generated and print the results in the listbox.

Inserting pictures through code

Pictures should be placed in an imagebox. They should be stored in the same folder as your program. You can direct the program to look in this folder by adding these lines of code to the FORM…LOAD procedure of your first form.
ChDrive "H:\"

Then place this line of code to add the picture at the point in the program when you wish to do so.

```
picCoin.Load("Head.jpg")
```

To erase a picture, use this line of code
picCoin.Load("")

PROGRAMMING PROBLEMS

3. Create a Game of "Guess My Number – 1 to 100." The computer will come up with a random number in this range. The user will guess until he gets the answer. After each guess, the computer will prompt him with "Higher" or "Lower." It will also display the number of guesses.

4. Create a program to simulate the flipping of a coin. Make as realistic a game as possible of this by letting the player choose which we wants and telling him the results. Provide a "Play again" button that would clear the first result and allow him to choose again. Keep track of correct and incorrect responses and display the results in the label.

5. Create a program to simulate the rolling of a pair of dice.

6. Suppose you have a younger brother or sister who is just learning the times tables. Make a game that will test him by asking him questions involving any two numbers from 1 to 12 being multiplied together. Your program should tell him whether his answer is right or wrong. It should also keep track of the number correct and the number incorrect and display that information in labels.