Chapter 7

FOR-NEXT LOOPS

Aziz Jafri, Tariq Jadoon

This chapter is based on the content of the lecture delivered by Dr Tariq Jadoon on Thu 18th Oct 2007.

Whenever you want to execute a piece of code repeatedly you use loops. Suppose we want to repeatedly print “Eid Mubarak” we could write the following program:

```vbnet
Private Sub Form_Click()
    Beginning: Debug.Print "Eid Mubarak" ' "Beginning" is a label
    GoTo Beginning ' This statement tells the computer to go back to "Beginning"
End Sub
```

The above code repeatedly prints “Eid Mubarak” in the Immediate Window. We could have also printed on the form but we used the Immediate Window for convenience. This window unlike the form, has an added advantage of being scrollable and plus we can use it to perform mathematical calculations, thus it also acts like a scratch pad. The command “Debug.Print” is used to print in the Immediate Window. The window can be activated by pressing Ctrl+G. Below is a screen shot of the immediate window in which we have performed two mathematical calculations.

In the immediate window you can use the Print statement or the symbol ? as shown above.
Now coming back to the program, when this program is executed we see “Eid Mubarak” being repeatedly printed. The following is a flow chart of the above program.

Here we see that after printing “Eid Mubarak”, the computer is told to go back to the label “Beginning” which consequently causes “Eid Mubarak” to be printed again. This process is repeated in a never-ending fashion, i.e. an infinite loop: A loop which has no end!! The computer hangs up after a while and VB crashes requiring us to restart Visual Basic again. This is because program control gets stuck in an infinite loop.

This can be remedied by making the program loop a finite number of times. Suppose we want to print “Eid Mubarak” ten times. A quick and dirty way of doing this is by repeatedly writing ten print statements, as follows:

But, suppose we want to print “Eid Mubarak” 50 times or let’s say a 100 times. Doing it this way would surely be a tedious task. A better way of doing this would be to use
a counter. Each time we print something we can increment the counter and check to see whether the counter has reached a “terminal” or “ending condition”. This is achieved by checking for a terminal condition and this check can be performed either before or after the actual “body of the loop” i.e. the statements that we intend to repeat. The following flowchart explains how to set up a counter to print “Eid Mubarak” ten times.
Let us now build a program based on the above flowchart.

![Program Code Example]

Notice that depending upon what value we initialize the counter to, we would have a slightly different terminating condition to execute the loop the desired number of times. For example if we initialize the counter to 1, we would have to modify the terminating condition to If i > 10 Then ....

When we click on the *print* button we see that “Eid Mubarak” gets printed ten times.
Although this is perfectly ok, it is not a good enough way of doing this. Using “Go To” statements is bad programming practice. Visual Basic provides a built in structure known as a FOR-NEXT loop, which is a much more elegant way of achieving the same result.

The following code demonstrates how to use a FOR-NEXT loop.
The FOR-NEXT Loop can be understood best with the following flow chart.

Here we see that the statement:

For i=1 to 10

initializes a counter which in this case is the variable i to 1 and then checks to see whether i is greater than 10. If i is greater than 10 the body of the loop will not get executed and program control will move to the line after Next i. Otherwise
the body of the loop, i.e. whatever is between the “For” statement and the “Next” statement is executed. Here:

    Picture1.Print i; “Eid Mubarak”

is the body of the loop.

The statement “Next i” is responsible for incrementing the counter. After incrementing the counter program control loops back to the terminal condition and it is checked to see whether the counter is greater than ten. If not, the body of the loop is executed again otherwise program control continues after Next i. Thus, For-Next loops are a very neat way of executing a piece of code repeatedly. When we click on the “print” button we see that “Eid Mubarak” gets printed ten times as shown below.

For-Next loops are a very powerful programming tool. Suppose you wish to make a program that takes in a number as input from the user and prints the respective table of that number. We could code it as follows:
Private Sub Command1_Click()
    Picture1.Cls  ' Clear the picture box each time you click the command button
    Dim num, i As Integer
    num = Val(Text1.Text)  ' num is equal to whatever is entered in the text box
    For i = 1 To 10
        Picture1.Print i;"x"; num;"=";i*num  ' The respective table of the number entered is printed
    Next i
End Sub

Suppose we enter 8 in the textbox and click on print we see the following output.

![Image of a window showing the multiplication table of 8](Image)

In the above for loops we see that the counter denoted by the variable \( i \) is incremented by 1 at the end of the loop. What if we would like to increment the counter by a value other than 1? This can be done by using the keyword Step. So if we want an increment of 2 we could modify the above program as follows.
Private Sub Command1_Click()

Picture1.Cls
Dim num, i As Integer
num = Val(Text1.Text)
For i = 1 To 10 step 2
    Picture1.Print i; "x"; num; "="; i * num
Next i  
' Next i, now increments i by 2
End Sub

So, now if we enter 8 in the textbox and click on print, we see the following output.

![Screenshot of a form with numbers printed](image)

Notice that on the left side we get an increment of 2 each time unlike the previous case where we were getting an increment of 1. This is because by default the step size is set to one so formerly we were getting an increment of 1, but now we set the step size to two so after each time the loop executes the counter is incremented by 2. The step size can have any value, it can even be negative, for e.g. if you wish to count down you could have a step size of -1. The step size however cannot be a variable. It has to be a constant.

If the step size is a negative number note that the terminal condition checks to see whether the number is less than the terminal number i.e. the To number.
rather than greater than it. Notice you may even use decimal numbers as step sizes.

For-Next loops can be used to make very interesting figures. Suppose we want to make ten concentric circles. This can be easily done by using one for loop as follows:

Another interesting thing you could do is change the drawwidth of each circle.

A good figure to practice using a For-Next loop is shown below.
This figure was made in class. Try making it yourself again. Note that you have to use only one loop.

Now modify your code so that you have something that looks like this. Again, use only one loop.

A few further things to try are Labs 1, 2 and 3 of Week 5 at:

http://suraj.lums.edu.pk/~cs101a06