INTRODUCTION TO LOOPS

A loop is a command that executes a set of instructions over and over again. Visual Basic features three types of looping structures that are used to perform repetitive tasks.

THE DO WHILE LOOP

A **Do While** loop is a looping structure that includes an expression that is tested for a TRUE or FALSE value and a statement(s) that is repeated as long as the expression is true.

A **Do While** loop takes the following form:

```
Do While expression
   statement(s)
Loop
```

When the code runs, the expression in the **Do While** statement is tested. If it is true, the statement(s) in the body of the loop are executed. This cycle repeats until the expression returns false.

The **Do While** is like an **If** statement that executes over and over again. Each repetition of the loop is called an **iteration**.

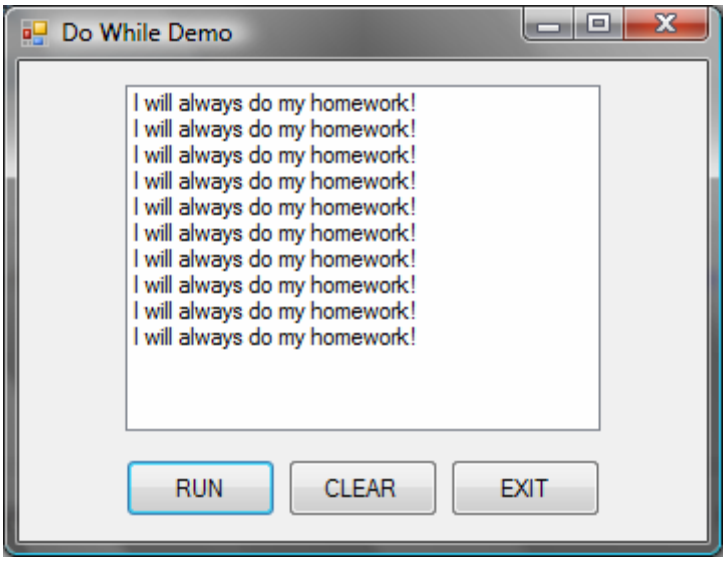
In the following example, the sentence "I will always do my homework!" will appear in a list box ten times:

```
Private Sub btnRun_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles btnRun.Click

   Dim counter As Integer = 0

   Do While counter < 10
      lstMessage.Items.Add("I will always do my homework!")
      counter += 1
   Loop

End Sub
```



COUNTERS

A **counter** is a variable that is regularly incremented or decremented each time a loop iterates. To increment a variable means to add 1 to its value. To decrement a variable means to subtract from its value.

The following statements increment a variable called **counter**:

```
counter = counter + 1
counter += 1
```

The following statements decrement a variable called **counter**:

```
counter = counter - 1
counter -= 1
```

You will often need to keep track of the number of iterations a loop performs. In the above example, a counter variable was used to keep track of the number of iterations so that it only executed 10 times. The counter variable started at 0, and after each iteration, the sentence was added to the list box and 1 was added to the counter. This kept going while the counter was less than 10. Once the counter reached 10, the expression in the **Do While** loop (i.e. counter < 10) was no longer true, and as a result, the loop stopped.

THE DO…LOOP WHILE LOOP

Do...Loop While loops are similar to **Do While** loops, the difference being that in a **Do...Loop While** loop the loop statements are executed first, and then the Boolean expression is checked. If the statement returns TRUE, then the loop statements are executed again. The evaluation and execution of the loop statement continues until the Boolean expression returns false.

The general format of a **Do...Loop While** is as follows:

```
Do
   statement(s)
Loop While expression
```

If we were to use a **Do...Loop While** loop for the same example that we used above, what would be the result?

```
Dim counter As Integer = 0

Do
   lstMessage.Items.Add("I will always do my homework!")
   counter += 1
Loop While counter < 10
```

If you answered that the result would be the same – i.e. it outputs the sentence 10 times in the list box – you're correct! Again, the only difference between a **Do While** and a **Do...Loop While** is that the latter performs at least one iteration before checking if it should do it again. So even though the test expression may be false right from the start, a **Do...Loop While** will still execute at least one iteration before it stops. So, when would you use a **Do...Loop While**? When you need the program to execute an iteration at least once.

THE DO UNTIL LOOP

The **Do Until** loop repeats until its test expression is true. The following is the general format of a **Do Until** loop:

```
Do Until expression
   statement(s)
Loop
```

If we were to use a **Do Until** loop for the above program, it would look something like this:

```
Dim counter As Integer = 0

Do Until counter = 10
   lstMessage.Items.Add("I will always do my homework!")
   counter += 1
Loop
```

You can also use a **Do...Loop Until** loop, which like the **Do...Loop While**, performs at least one iteration before it checks whether it should do it again. The general format of a **Do...Loop Until** loop is as follows:

```
Do
   statement(s)
Loop Until expression
```

THE FOR…NEXT LOOP

A **For…Next** loop is a loop that repeatedly executes a set of statements a specified number of times. In order to keep count of how many times a loop has executed, you will need to use a counter variable, assign it an initial value, set up a conditional statement that will determine how long the loop will execute, and determine the increment value of the counter variable.

```
For CounterVariable = StartValue To EndValue
   statement(s)
Next
```

The **CounterVariable** is the variable that is used as the counter. The **StartValue** is the value the counter variable will initially be set to. The **EndValue** is the value the counter variable is tested against just prior to each iteration of the loop. The **Next CounterVariable** statement marks the end of the loop and causes the counter variable to be incremented.

If we were to use a **For…Next** loop for the above program, it would look something like this:

```
For counter As Integer = 1 To 10
   lstMessage.Items.Add("I will always do my homework!")
Next
```

There is an additional parameter that you can add to a **For…Next** loop which sets the amount added to the counter variable at the end of each iteration. For example, let's say we wanted our program to increment by 2 instead of 1; our loop would look something like this:

```
For counter As Integer = 1 To 10 Step 2
   lstMessage.Items.Add("I will always do my homework!")
Next
```

Now, based on the above code, how many times would the statement be outputted to the list box? If you said 5 times, you are correct! The reason for this is simple: the loop starts with the counter variable set at 1, outputs the sentence to the list box, then the counter variable becomes 3, outputs again, becomes 5, outputs again, becomes 7, outputs again, becomes 9, outputs again, becomes 11, at which point it does not execute again because 11 is greater than 10.

If a Step value is not specified in a **For…Next** loop, the counter variable will increment by 1 at the end of each iteration.