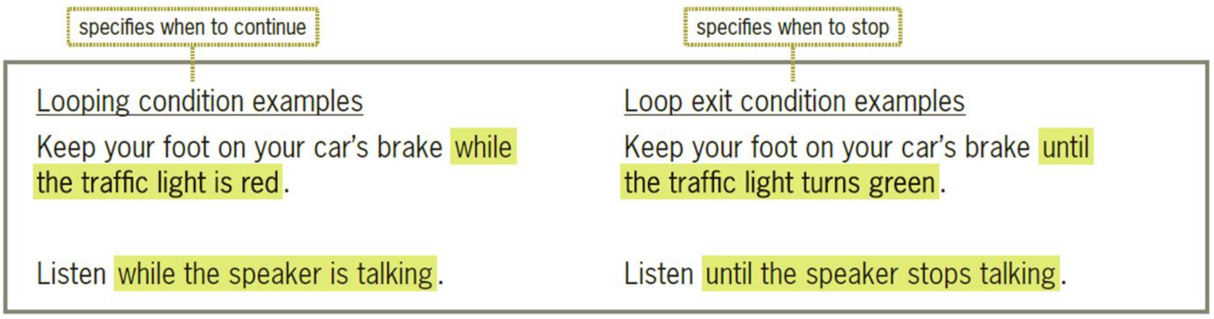
Repetition Structures

Programmers use the repetition structure, referred to more simply as a loop, when they need the computer to repeatedly process one or more program instructions. The loop contains a condition that controls whether the instructions are repeated. Like the condition in a selection structure, the condition in a repetition structure must evaluate to either true or false. In many programming languages, the condition can be phrased in one of two ways: It can either specify the requirement for repeating the instructions or specify the requirement for not repeating them. The requirement for repeating the instructions is referred to as the looping condition because it indicates when the computer should continue "looping" through the instructions. The requirement for not repeating the instructions is referred to as the loop exit condition because it tells the computer when to exit (or stop) the loop. Figure 2-36 shows examples of both types of conditions that you use every day. The conditions are shaded in the figure.


specifies when to continue
specifies when to stop

Looping condition examples

Keep your foot on your car's brake while the traffic light is red.

Listen while the speaker is talking.

Loop exit condition examples

Keep your foot on your car's brake until the traffic light turns green.

Listen until the speaker stops talking.

Figure 2-36 Examples of familiar looping conditions and loop exit conditions

The condition in a loop can appear at either the top or the bottom of the loop. When the condition is at the top of the loop, the loop is referred to as a pretest loop because the condition is evaluated before the instructions within the loop are processed. When the condition is at the bottom of the loop, the loop is referred to as a posttest loop because the condition is evaluated after the instructions within the loop are processed.

The Visual Basic language provides three different statements for coding loops: Do…Loop, For…Next, and For Each…Next. The Do…Loop and For…Next statements are covered in this semester.

Do…Loop Statement (Pretest Loop)

Figure 2-37 shows the syntax for using the Do…Loop statement to code a pretest loop. The examples in this Figure display the numbers 1 through 5 in a label control. The loop's condition, which must follow either the keyword While or the keyword Until, can be phrased as either a looping condition or a loop exit condition. You use the While keyword in a looping condition to specify that the loop body should be processed while (in other words, as long as) the condition is true. You use the Until keyword in a loop exit condition to specify that the loop body should be processed until the condition becomes true, at which time the loop should stop. Like the condition in an If…Then…Else statement, the condition in a Do…Loop statement can contain variables, named constants, literals, properties, methods, keywords, and operators; it also must evaluate to a Boolean value. The condition is evaluated with each repetition of the loop and determines whether the computer processes the loop body. Figure 2-37 includes examples of using the Do…Loop statement to code the pretest loops. In each example, the loop will stop when the value in intNum is 6.

Do...Loop Statement-Pretest Loop

Syntax:

```
   Do {While | Until} condition
    loop body instructions to be processed either while the condition is  true or until
    the condition becomes true
  Loop
```

Example 1: pretest loop using a looping condition

```
Public Class Form1
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num As Integer = 1
        Do While num <= 5
            Label1.Text = Label1.Text & num & "  "
            num = num + 1
        Loop
    End Sub
End Class
```

Example 2: pretest loop using a loop exit condition

```
Public Class Form1
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num As Integer = 1
        Do Until num > 5
            Label1.Text = Label1.Text & num & "  "
            num = num + 1
        Loop
    End Sub
End Class
```

result of using either of the above pretest loop examples

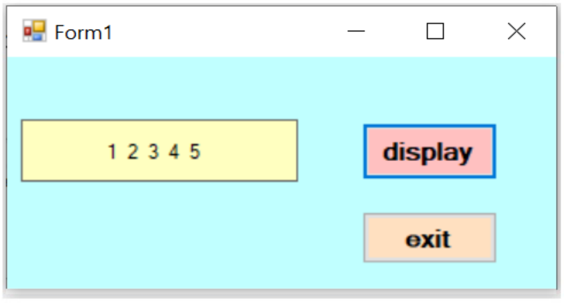
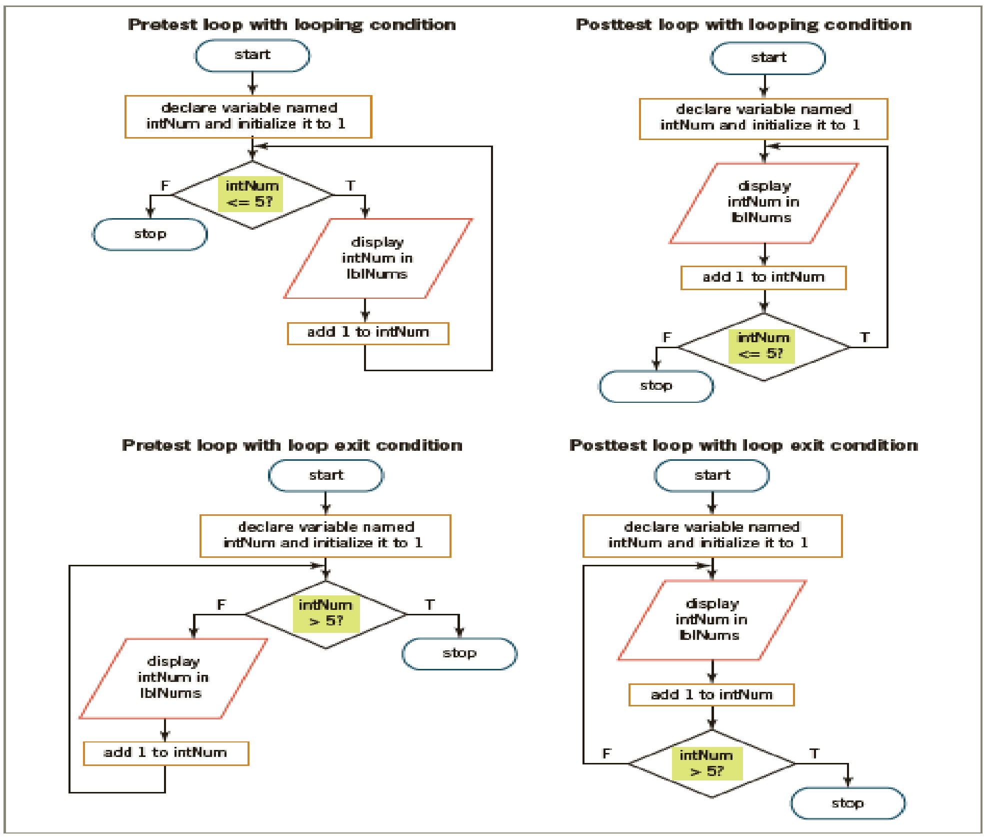


Figure 2-37 Syntax and examples of using the Do…Loop statement to code a pretest loop

It is often easier to understand loops by viewing them in flowchart form. Figure 2-38 shows the flowcharts associated with the examples from Figure 2-37. Like the condition in a selection structure, the condition in a flowchart is represented by a diamond. The diamond is called the decision symbol. The condition is shaded in each decision symbol shown in Figure 2-38. Like the diamond in a selection structure, the diamond in a repetition structure has two flowlines leading out of the symbol. The flowlines are usually marked with a T (for True) and an F (for False); however, they can be marked with a Y and an N (for Yes and No).



Pretest loop with looping condition
start
declare variable named intNum and initialize it to 1
intNum <= 5?
F
T
stop
display intNum in lblNums
add 1 to intNum
Posttest loop with looping condition
start
declare variable named intNum and initialize it to 1
display intNum in lblNums
add 1 to intNum
intNum <= 5?
F
T
stop
Pretest loop with loop exit condition
start
declare variable named intNum and initialize it to 1
intNum > 5?
F
T
stop
display intNum in lblNums
add 1 to intNum
Posttest loop with loop exit condition
start
declare variable named intNum and initialize it to 1
display intNum in lblNums
add 1 to intNum
intNum > 5?
F
T
stop

Figure 2-38 Flowcharts for the loop examples from Figure 2-37

Do…Loop Statement (Posttest Loop)

Figure 2-39 shows the syntax for using the Do…Loop statement to code a posttest loop. Except for the location of the keyword (While or Until) and the condition, the syntax is the same as the one shown earlier in Figure 2-37. The figure also shows how you can modify the Do…Loop statements from Figure 2-37 to use posttest loops rather than pretest ones. (The for the posttest loops are shown earlier in Figure 2-38.) In each example, the loop will stop when the value in num is 6. The difference between a pretest loop and a posttest loop is that the instructions in a posttest loop will always be processed at least once, whereas the instructions in a pretest loop may never be processed. For example, if the num variable was initialized to 6 (rather than to 1), the conditions in both pretest loops would prevent the instructions in those loops from being processed. The instructions in both posttest loops, on the other hand, would be processed once before the conditions were evaluated the first time.

```
Do...Loop Statement-Posttest Loop

Syntax:

   Do
    loop body instructions to be processed either while the condition is  true or until
    the condition becomes true
  Loop {While | Until} condition

Example 1: posttest loop using a looping condition
Public Class Form1
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num As Integer = 1
        Do
            Label1.Text = Label1.Text & num & "  "
            num = num + 1
        Loop While num <= 5
    End Sub
End Class

Example 2: posttest loop using a loop exit condition
Public Class Form1
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles Button1.Click
        Dim num As Integer = 1
        Do
            Label1.Text = Label1.Text & num & "  "
            num = num + 1
        Loop Until num > 5
    End Sub
End Class
```

result of using either of the above posttest loop examples

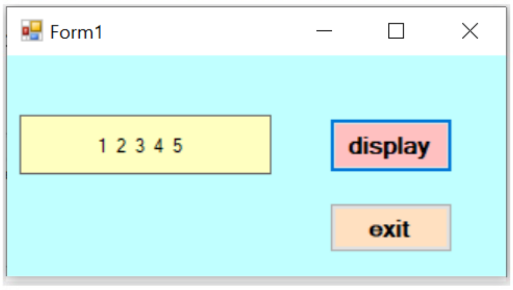


Figure 2-39 Syntax and examples of using the Do…Loop statement to code a posttest loop

To code and then test the Do…Loop Posttest application:

1. Open the Code Editor window and enter the Do…Loop statement shown in either of the examples from Figure 2-39.

2. Save the solution and then start the application. Click the **Display** button to display the numbers from 1 to 5 in the label control.

3. Click the **Exit** button. Now change the 1 in the Dim statement to **6**. Save the solution and then start the application. Click the **Display** button. Because the posttest loop's condition is not evaluated until after the instructions in the loop are processed the first time, the number 6 appears in the label control, as shown in Figure 2-40.

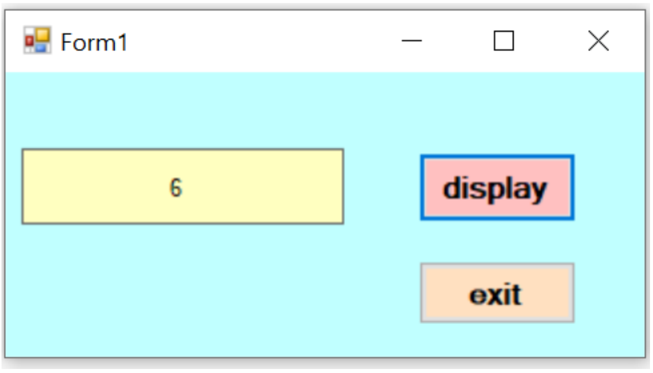


Figure 2-40 Result of the loop instructions being processed before the condition is evaluate

Infinite Loops

A loop that has no way to end is called an infinite loop or an endless loop. An infinite loop is many times created when the loop body does not contain an instruction that will make either the looping condition evaluate to False or the loop exit condition evaluate to True. You can stop an infinite loop by clicking Debug on the menu bar and then clicking Stop Debugging. Or, you can click the Stop Debugging button (the red square) on the Standard toolbar.

To create and then stop an infinite loop:

1. Open the Code Editor window of any example from the examples above (in case pretest or posttest loop) and turn the (num =num + 1) assignment statement into a comment by inserting an apostrophe (') before it. Notice that the loop body no longer provides an instruction that can change the value stored in the num variable from its initial value of 1 to a value that can stop the loop—in this case, 6.
2. Save the solution and then start the application. Click the **Display** button. Wait a few seconds and then click the **Exit** button. Notice that the Exit button does not respond to its Click event. Click the **Display** button again. The Display button also does not respond to its Click event. Neither button responds because the Do…Loop statement is in an infinite loop. The loop has no way to stop because without the (num = num + 1) statement, the num variable’s value remains at 1.
3. Click the **Stop Debugging** button (the red square) on the Standard toolbar to stop the loop.