Repetition Structures

Outline

- Loop Controls
- Do While Loops
- Interactive Do While Loops
- For/Next Loops
- Exit-Controlled Loops

What is a loop?

- A loop is a repetition control structure.
- It causes a single statement or block to be executed repeatedly

Loop Controls

- This chapter explores the different methods that programmers use to construct repeating sections of code and how they can be implemented in Visual Basic.
- A repeated procedural section of code is commonly called a loop, because after the last statement in the code is executed, the program branches, or loops back to the first statement and starts another repetition.
- Each repetition is also referred to as iteration or a pass through the loop.
Loop Controls

- Constructing repetitive sections of code requires the following four elements:
  - A repetition statement that both defines the boundaries containing the repeating section of code and controls whether the code will be executed or not.
  - A condition that needs to be evaluated.
  - A statement that initially sets the condition.
  - A statement within the repeating section of code that modifies the condition in such a way that the repetitions, at some point, stop.
- There are three different forms of Visual Basic repetition structures:
  - Do While structures
  - For-Next structures
  - Do-Until structures.

Pretest and Posttest Loops

- In a pretest loop, the condition being tested is evaluated at the beginning of the repeating section of code.
- The condition is tested before any statement(s) within the loop is executed.
- Pretest loops are also referred to as entrance-controlled loops. Both the Do While and For-Next loop structures are examples of such loops.
- In a posttest loop, the condition being tested is evaluated at the end of the repeating section of code.
- Such loops always execute the loop statement(s) at least once before the condition is tested.
- Posttest loops are also referred to as exit-controlled loops.
- The Do-Until construct is an example of a posttest loop.
**Fixed Count Versus Variable Condition Loops**

- In a **fixed-count** loop, the condition is used to keep track of how many repetitions have occurred.
- In a **variable-condition** loop, the tested condition depends on a variable that can change interactively with each pass through the loop.
- All of Visual Basic's repetition statements can be used to produce both fixed-count loops and variable-condition loops.

**Do While Loops**

- The syntax of the *Do While* loop is as follows:
  ```javascript
  Do While BOOLEAN_EXPRESSION
  STATEMENT(S)
  Loop
  ```
  
- A *Do While* loop consists of:
  - A Boolean expression: the loop condition that will control the repetitions of the
    body of the loop
  - A body of the loop is the set of statements that will be repeated
- The body of the loop will be repeated if the Boolean expression is true. Otherwise, the loop will stop.
- The *Do While* loop is an **entrance-controlled** loop since it checks the loop condition before it enters the loop.
- In creating loops, it is the programmer’s responsibility to ensure an exit from the loop. If this is not done, then an **infinite** loop – a loop that never ends – is created.

**Example: Find the Sum of 10 Numbers**

```vbnet
Const Size As Byte = 10
Dim Count As Integer, Sum, Data As Double
Count = 0
Sum = 0
Do While (Count < Size)
    Data = InputBox("Enter a Number.")
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
    Count = Count + 1
Loop
DisplayLabel.Text = "The sum 10 numbers is " & Sum
```

**Example: Find the Sum of Numbers**

```vbnet
Const Sentinel As Double = -1
Dim Count As Integer, Sum, Data As Double
Count = 1
Sum = 0
Data = InputBox("Enter a Number (-1 to stop).")
Do While (Data <> Sentinel)
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
    Count = Count + 1
    Data = InputBox("Enter a Number (-1 to stop).")
Loop
DisplayLabel.Text = "The sum of " & Count & " numbers is " & Sum
```
Interactive *Do While* Loops

*Do While* loops can be used with interactive data entry to produce highly adaptable programs.

---

**Do-Until** Loop Structure

- The syntax of a *Do-Until* loop is as follows:

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

- The *Do-Until* loop executes the statement(s) within the loop as long as the condition is False.
- The *Do-Until* loop is an Exit-controlled loop.

---

Example: Find the Sum of 10 Numbers

```vbnet
Const Size As Byte = 10
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
Do
  Data = InputBox("Enter a Number.")
  DisplayListBox.Items.Add(Data)
  Sum = Sum + Data
  Count = Count + 1
Loop Until (Count >= Size)
DisplayLabel.Text = "The sum is " & Sum
```

---

Example: Find the Sum of Numbers

```vbnet
Const Sentinel As Double = -1
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
Do
  Data = InputBox("Enter a Number (-1 to stop).")
  DisplayListBox.Items.Add(Data)
  Sum = Sum + Data
  Count = Count + 1
Loop Until (Data = Sentinel)
DisplayLabel.Text = "The sum of " & Count & " numbers is " & Sum
```
Example: Find the Sum of Numbers

```
Const Sentinel As Double = -1
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
Data = InputBox("Enter a Number (-1 to stop).")
Do
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
    Count = Count + 1
    Data = InputBox("Enter a Number (-1 to stop).")
Loop Until (Data = Sentinel)
DisplayLabel.Text = "The sum of " & Count & " numbers is " & Sum
```

Sentinels

- Data values used to signal either the start or end of a data series are called *sentinels*.
- Sentinels can be used in constructing variable-condition loops.
- A sentinel value for a data series must not conflict with legitimate data values.

Count-controlled loop

- an initialization of the loop control variable (starting point)
- an expression to test for continuing the loop (the ending point)
- an update of the loop control variable to be executed with each iteration of the body (increment the loop control variable)

```
Dim Count As Byte
Count = 4 'initialize loop variable
Do While (Count > 0) 'test expression
    DisplayLabel.Text = Count 'repeated action
    Count = Count - 1 'update loop variable
Loop
DisplayLabel.text = "Done"
```
Count-controlled Loop

```vbnet
Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"
```

Count-controlled Loop

```vbnet
Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"
```

Count-controlled Loop

```vbnet
Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"
```

Count-controlled Loop

```vbnet
Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"
```
Count-controlled Loop

Dim Count As Byte
Count = 4
Do While (Count > 0)
  DisplayLabel.Text = Count
  Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
  DisplayLabel.Text = Count
  Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
  DisplayLabel.Text = Count
  Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
  DisplayLabel.Text = Count
  Count = Count - 1
Loop
DisplayLabel.text = "Done"
Count-controlled Loop

Dim Count As Byte
Count = 4
Do While (Count > 0) 
DisplayLabel.Text = Count
Count = Count - 1
Loop
DisplayLabel.Text = "Done"

Count = Count - 1
DisplayLabel.Text = Count
Do While (Count > 0)
DisplayLabel.Text = Count
Count = Count - 1
Loop
DisplayLabel.Text = "Done"
Count-controlled Loop

Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"

Dim Count As Byte
Count = 4
Do While (Count > 0)
    DisplayLabel.Text = Count
    Count = Count - 1
Loop
DisplayLabel.text = "Done"
Breaking Out of a Loop

- Sometimes it may be necessary to prematurely break out of a loop when an unusual error condition is detected.
- The Exit Do statement can be used to do this.

Example: Find the Sum of 10 Numbers

```vbnet
Const Size As Byte = 10
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
While (Count < Size)
    Data = InputBox("Enter a Number.")
    If (Data < 0) Then
        DisplayLabel.Text = Data & " is invalid number."
        Data = InputBox("Enter a Number.")
        Exit Do
    End If
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
    Count = Count + 1
Wend
DisplayLabel.Text = "The sum of 10 numbers is " & Sum
```

For-Next Loops

- The creation of fixed-count loops always requires initializing, testing, and modifying the counter variable.
- The For-Next loop groups all three of these operations on a single line.
- The general form of the For-Next loop is as follows:

```
For variable = startingValue To endingValue [Step increment]
statement(s)
Next [variable]
```

For-Next Loop Rules

For-Next loops must adhere to the following rules:
- The first statement in a For-Next loop must be a For statement.
- The last statement in a For-Next loop must be a Next statement.
- The For-Next loop counter variable may be either a real or integer variable.
- The initial, final, and increment values may all be replaced by variables or expressions, as long as each variable has a value previously assigned to it and the expressions can be evaluated to yield a number.
- The initial, final, and increment values may be positive or negative, but the loop will not be executed if any one of the following is true:
  - The initial value is greater than the final value and the increment is positive.
  - The initial value is less than the final value and the increment is negative.
- An infinite loop is created if the increment is zero.
- An Exit For statement may be embedded within a For-Next loop to cause a transfer out of the loop.
Example: Find the Sum of 10 Numbers

```
Const Size As Byte = 10
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
For Count = 0 To Size - 1
    Data = InputBox("Enter a Number.")
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
Next Count
DisplayLabel.Text = "The sum of 10 numbers is " & Sum
```

Example: Find the Average for Each Set of Numbers

```
Const Sentinel As String = "End"
Const NoTest As Byte = 3
Dim InnerCount, OuterCount As Integer
Dim Sum, TestScore, Average As Double
Dim Name As String
OuterCount = 1
Name = InputBox("Enter a name.")
Do While (Name <> Sentinel)
    DisplayListBox.Items.Add(OuterCount & ". " & Name)
    Sum = 0
    InnerCount = 1
    Do While (InnerCount <= NoTest)
        TestScore = InputBox("Enter a test score.")
        DisplayListBox.Items.Add("Test score " & InnerCount & " is " & TestScore)
        Sum = Sum + TestScore
        InnerCount = InnerCount + 1
    Loop
    Average = Sum / NoTest
    DisplayListBox.Items.Add("Your test average is " & FormatNumber(Average))
    OuterCount = OuterCount + 1
    Name = InputBox("Enter a name.")
Loop
```

Nested Loops

- A **nested loop** is a loop contained within another loop.
- The loop that contains another loop is called the **outer loop**.
- The loop that is contained within another loop is called the **inner loop**.
- For each single trip through the outer loop, the inner loop runs through its entire sequence.
Exit-Controlled Loops

- Exit-controlled loops test a condition at the bottom of the loop.
- This ensures that the statements within the loop are executed at least one time.
- The exit-controlled loops provided in Visual Basic are the Do-Until Loop and Do-While Loop structures.
- The syntax of the Do-Until Loop structure is as follows:

  ```vbnet
do statement(s) loop until loop condition
  ```

- The statements within a Do-Until Loop are executed until the condition becomes True. If the condition is True to begin with, the statements within the loop will execute only once.
- The syntax of a Do-While Loop is as follows:

  ```vbnet
do statement(s) loop while loop condition
  ```

- The statements within a Do-While Loop are executed as long as the condition is True. If the condition is False to begin with, the statements within the loop will execute only once.

Validity Checks

- Exit-controlled loops are particularly useful in ensuring that the user enters valid data.
- One way of performing validity checks is to repeatedly request valid data until the user enters data that is valid.

Example: Find the Sum of Numbers between 0 and 100

```vbnet
Const Response As String
Dim Count As Integer
Dim Sum, Data As Double
Count = 0
Sum = 0
Do
    Data = InputBox("Enter a Number between 0 to 100.")
    loop until (Data > 0) and (Data < 100)
    DisplayListBox.Items.Add(Data)
    Sum = Sum + Data
    Count = Count + 1
    Response = InputBox("Do you have more data to enter?"")
    loop until (Response = "no") or (Response = "No")
DisplayLabel.Text = "The sum of " & Count & " numbers is " & Sum
```
What Loop to Use?

- **Do While loop**
  - `Do While` Loop condition
  - `Loop`
  - Fixed or variable number of repetitions
  - Repeat if the loop condition is true
  - Not execute the loop at all

- **Do Until loop**
  - `Do` assignment(s)
  - `Loop Until` loop condition
  - Fixed or variable number of repetitions
  - Repeat until the loop condition is true
  - Execute the loop once

- **For-Next loop**
  - Fixed number of repetitions