VB.NET: Control Flow Statements Continue

Do Loop – Post-Test

The `Do Loop - While/Until`, also known as the "Post-test Form of Do Loop," ensures that the code block runs at least once before checking the condition. This means the loop will always execute the statements inside the loop, then evaluate the condition at the end of each iteration.

The syntax is:

```
Do
    ' Statements A
Loop While condition

' OR

Do
    ' Statements A
Loop Until condition
```

Behavior:

- **Statements A** are executed **at least once**, regardless of the condition.
- **After** executing the block, the condition is evaluated.
- While
    - If the condition is `True`, the loop repeats.
    - If the condition is `False`, the loop stops.
- Until
    - If the condition is `True`, the loop stops.
    - If the condition is `False`, the loop repeats.

Example 1: Using **Loop While**

```
Dim attempts As Integer = 1

Do
    Console.WriteLine("Attempt #" & attempts)
    attempts += 1
Loop While attempts <= 5

Console.ReadKey()
```

In this example:

- The loop runs 5 times, printing the message **Attempt #** followed by the current value of `attempts`.
- The loop continues while `attempts` is less than or equal to 5.
- Since the condition is evaluated after each iteration, the loop always runs at least once.

Example 2: Using **Loop Until**

```
Dim oneTimeValue As Integer = 15

Do

    Console.WriteLine("One Time Value is: " & oneTimeValue)
    oneTimeValue += 1

Loop Until oneTimeValue > 10

Console.ReadKey()
```

In this example:

- The variable `oneTimeValue` is initialized to 15.
- The loop runs once, printing `oneTimeValue` even though the condition (`oneTimeValue > 10`) is initially true.
- After the first iteration, the value increases to 16, and the loop terminates.

For Loop

The **`For Loop`** executes a block of code a set number of times based on a loop counter.

The syntax is:

```
For counter As Integer = startValue To endValue Step stepValue
    ' Statements A
Next
```

Behavior:

- **Statements A** are executed for each iteration of the loop, with the counter starting at **`startValue`** and being incremented or decremented by **`stepValue`** in each iteration.
- The loop runs until the counter exceeds the **`endValue`** (for positive **`stepValue`**) or drops below the **`endValue`** (for negative **`stepValue`**).
- The **`stepValue`** is optional. If not provided, it defaults to 1 (increment by 1).
- If the starting value and step direction make it impossible to reach the end value, **the loop will not execute**.

Example: Calculate the sum of ascending numbers from 1 to 10

```
Dim sum As Integer = 0

For i As Integer = 1 To 10
    sum += i
Next

Console.WriteLine("Sum of numbers 1 to 10 is: " & sum)

Console.ReadKey()
```

In this example:

- The **`For Loop`** iterates 10 times, adding each value of **`i`** to **`sum`**.
- After the loop, **`sum`** will hold the total of all numbers from 1 to 10.

Nested Loops

A **Nested Loops** involves placing one loop inside another loop. This technique is useful for processing multi-dimensional structures or generating patterns.

The syntax is:

```
For outerCounter As Integer = startValue To endValue
    For innerCounter As Integer = startValue To endValue
        ' Statements A
    Next
Next
```

Behavior:

- The **outer loop** controls the number of iterations for the **inner loop**.
- For each iteration of the **outer loop**, the **inner loop** completes its full cycle.
- This structure is useful for scenarios like matrix multiplication, pattern printing, or generating complex sequences.

Example: Output a multiplication table

```
For row As Integer = 1 To 5
    For col As Integer = 1 To 5
        Console.Write((row * col).ToString().PadLeft(4) & " ")
    Next
    Console.WriteLine()
Next

Console.ReadKey()
```

In this example:

- The outer loop controls the rows of the multiplication table.
- The inner loop controls the columns, multiplying the current `row` value by the `col` value.
- The output will display a 5x5 multiplication table, where the product of each row and column is printed.

Conclusion

This documentation provides an overview of various control flow mechanisms in VB.NET, including `Select Case`, `While`, `Do While`, and `Do Until` loops. These structures are vital for controlling the execution of your programs based on conditions. To make learning fun, we also included a humorous console art example!