Introduction to Looping Structures

Applications often have cycles within them. These cycles perform the same tasks over and over again. In fact, much of the usefulness of modern software comes from its ability to do repetitive tasks. Computers are not like us humans - they never get bored doing the same thing over and over!

In this module, you will learn about looping structures. Loops control program flow and allow a set of statements to be executed a number of times.

You will learn how to write the following looping structures:

- For…Next Loop
  - The For…Next loop executes a statement or group of statements for a specific number of times.
  - The loop executes a fixed number of times until the counter reaches an ending value.
- Do…Loop
  - The Do…Loop executes a statement or group of statements over and over.
  - Execution of the loop continues indefinitely as long as the loop condition is true.

The Do...Loop Statement

Repeating one or more statements is referred to as **looping**, or **iteration**. The **Do…Loop** statement is a looping structure that iterates a set of code statements for as long as a condition is **True**. In other words, **do** this **while** this is true.

There are several variations of the Do…Loop based on **where** and **how** Visual Basic evaluates the loop condition. The most common syntax for the Do…Loop statement is as shown on the right:

An example of this format is shown:

```
Dim intNumber As Integer
intNumber = 0
Do While intNumber <= 10
intNumber = intNumber + 1 'increment by 1
Loop
```

- The condition in this example is the Boolean expression `intNumber <= 10`.
- The condition is used to determine if the loop will be repeated.
- If the condition is **True**, the statement `intNumber = intNumber + 1` is executed.
- The condition is then re-evaluated.
- If the condition still evaluates to True, the loop is repeated.
- Iteration continues until the condition evaluates to **False**.

An example is written this way:

```
Dim intNumber As Integer
intNumber = 0
Do
    intNumber = intNumber + 1 'increment by 1
Loop While intNumber < 10
```

- `intNumber = intNumber +1` is the statement body of the loop.
- The statement body **will execute at least once**.
- The condition, the Boolean expression `intNumber < 10`, is evaluated after the first execution of the loop.
- If the condition evaluates to **True**, the statement is executed again.
- The condition is re-evaluated.
- The loop repeats until the condition evaluates to **False**.

The Do...Loop Statement (Continued)

In the code example on the previous screen, the body of the Do…Loop will execute ten times. On the tenth time, `intNumber` is equal to `10`, which makes the loop condition `intNumber < 10` evaluate to **False**. The loop ends at this point. This form of Do…Loop executes the body of the loop at least once.

The first form of Do...Loop that we looked at back on screen two of this module may execute zero times or multiple times. In other words, it may not execute at all if the Boolean condition is False at the very beginning.

If you want the loop to always run at least once in a program, put the condition test at the bottom of the loop.

Infinite Loops

The loop's condition determines when the loop will stop executing. A Do...Loop will keep looping, or iterating, until the condition is **False**.

If the condition is not set up correctly, it will never become **False** and the loop will continue forever! This is known as an **infinite loop**.

Logic errors can result in infinite loops. The following example creates an infinite loop:

```
Dim intNumber As Integer
intNumber = -1
Do While intNumber < 0
    intNumber = intNumber -1
Loop
```

The variable `intNumber` is initialized to `-1`. The statement in the loop body keeps subtracting one from the number each time the loop iterates so the value for `intNumber` is always a negative number. Therefore, the condition of the loop is always **True**, so the loop will never end.

Sentinels and Accumulators

A Do...Loop can be used to get more than one value from the user. When the user is entering more than one value, a **sentinel** or **flag** can be used. In this case, the loop condition compares the entered values to the sentinel value.

The value that the sentinel or flag is given will determine the end of the loop. When the user enters the sentinel value, the loop ends. Using a sentinel or flag provides a way to end a loop that can easily be changed, if necessary.

The Do...Loop can also be used to count and add values entered by the user as the values are entered. A counter in the loop is updated each time the loop iterates. An **accumulator** is a numeric variable that is used to store a value that accumulates, or gets added to, as the program runs.

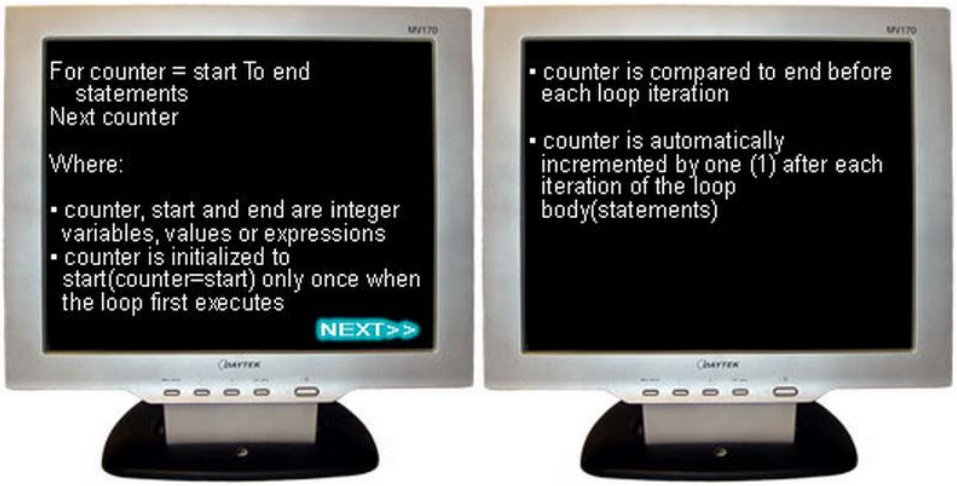
The syntax for updating an accumulator is as follows:

The For...Next Statement

The **For…Next** statement is another type of looping structure that loops through a set of statements for a fixed number of times. We can use the For…Next type of looping structure when we know exactly how many times a loop should be executed.

The For…Next statement is different from the Do…Loop. The Do...Loop executes while a condition is **True**. The For…Next statement executes until a counter reaches an ending value.

The syntax of the For…Next statement is:



The For...Next Statement (Continued)

Let's look at an example:

```
Dim intNumber As Integer
For intNumber = 15 To 25
  MsgBox intNumber 'Display message
box
with_ intNumber value
Next intNumber
```

In this example, the loop will iterate ten times. Each time through the loop, a message box will be displayed with the current value for intNumber. For example, the first message box will display the value 15, the next message box will display 16 and so on until the counter reaches 25. The last message box will display the value 25.

Generally a loop counts forward by 1's. To count forward by a higher amount a *step increment* is added:

For intNumber = 15 to 25 Step 5 (moves 15, 20, 25)

It is also possible to move backwards. Consider the following:

For intNumber = 25 to 15 Step -5 (moves 25, 20, 15)