Loops (Repetition) Structures

Visual Basic allows a procedure to be repeated as many times as long as the processor and memory could support. This is generally called looping. Looping is required when we need to process something repetitively until a certain condition is met. In Visual Basic, we have three types of Loops, they are

- For.....Next loop,
- Do loop
- Existing Loop

1 For....Next Loop

The format is:

For counter = Start **To** End **Step** [Increment]
  One or more VB statements
Next [counter]

The arguments counter, start, end, and increment are all numeric. The increment argument can be either positive or negative. If increment is positive, start must be less than or equal to end or the statements in the loop will not execute. If increment is negative, start must be greater than or equal to end for the body of the loop to execute. If steps isn’t set, then increment defaults to 1.

In executing the For loop, visual basic:

1. Sets counter equal to start.
2. Tests to see if counter is greater than end. If so, visual basic exits the loop (if increment is negative, visual basic tests to see if counter is less than end).
3. Executes the statements.
4. Increments counter by 1 or by increment, if it’s specified.
5. Repeats steps 2 through 4.

For Example:

```
1- For I=0 To 10 step 5
          Statements
   Next I

2- For counter = 100 To 0 Step -5
          Statements
   Next counter
```

Example 1: Design a form and write code to find the summation of numbers (from 0 to 100).

Solution:

```
Private Sub form_load()
Form1.show
Dim I As Integer,Total As Integer
 For I = 0 To 100
      Total= Total +I
Next I
Print "Total=";Total
End Sub
```

Example 2: Design a form and write code to find the summation of even numbers (from 0 to 100).

Solution:

```
Private Sub form_load()
Form1.show
Dim I As Integer,Total As Integer
 For I = 0 To 100 step 2
        Total= Total +I
Next I
Print "Total=" Total
End Sub
```

Example 3: Design a form and write code to find the summation of odd numbers (from 0 to 100).

Solution:

```
Private Sub form_load()
Form1.show
Dim I As Integer,Total As Integer
 For I = 0 To 100
If I mod 2 =1 then Total= Total +I
Next I
Print "Total=";Total
End Sub
```

2 -Do –Loop:

Use a Do loop to execute a block of statements and indefinite number of times. There are several variations of Do…Loop statement, but each evaluates a numeric condition to determine whether to continue execution. In the following Do..Loop, the statements execute as long as the condition is True.

2-1 Do While ..Loop

The formats are

```
Do While condition
Block of one or more VB Statement
Loop
```

When Visual Basic executes this Do..Loop, it first tests condition. If condition is False, it skips past all the statements. If it's True, Visual Basic executes the statements and then goes back to the Do while statement and tests the condition again. Consequently, the loop can execute any number of times, as long as condition is True. The statements never execute if initially False.

For Example: Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do While num <= 100
Total=Total +num
num = num + 1
Loop
Print Total
```

2-2 Do…Loop While:

Another variation of the Do..Loop statement executes the statements first and then tests condition after each execution. This variation guarantees at least one execution of statements.

The formats are

```
Do
Block of one or more VB Statement
Loop condition
```

For Example: Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do
Total=Total +num
num = num + 1
Loop While num <= 100
Print Total
```

2-3 Do Until ….Loop

Unlike the **Do While...Loop** repetition structures, the **Do Until... Loop** structure tests a condition for falsity. Statements in the body of a **Do Until...Loop** are executed repeatedly as long as the loop-continuation test evaluates to False.

The formats are

```
Do Until condition
Block of one or more VB Statement
Loop
```

For Example: Loop counts from 0 to 100.

```
Dim num As Integer, Total
num = 0
Do until num >100
Total=Total +num
num = num + 1
Loop
Print Total
```

2-4 Do… Loop Until

The formats are

Do

Block of one or more VB Statement

Loop Until *condition*

For Example: Loop counts from 0 to 100. Dim num As Integer, Total

```
Num=0
Do
Total=Total +num num =
num + 1
 Loop until num >100
 Print Total
```

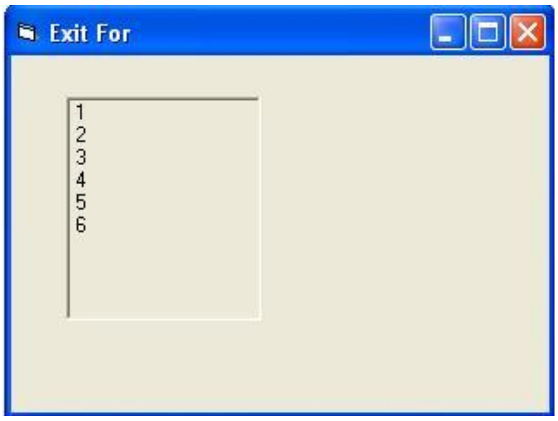
3 Existing Loop:

The exit statement allows you to exit directly from For Loop and Do Loop, Exit For can appear as many times as needed inside a For loop, and Exit Do can appear as many times as needed inside a Do loop (the Exit Do statement works with all version of the Do Loop syntax). Sometimes the user might want to get out from the loop before the whole repetitive process is executed; the command to use is **Exit For** To exit a For.....Next Loop **or Exit Do** To exit a Do… Loop, and you can place the Exit For or Exit Do statement within the loop; and it is normally used together with the If....Then.....statement.

• **Exit For**	• **Exit Do**
The formats are:	The formats are
For *counter*= start **To** end step (increment)	**Do While *condition***
Statements	Statements
Exit for	**Exit Do**
Statement	Statements
Next ***counter***	**Loop**

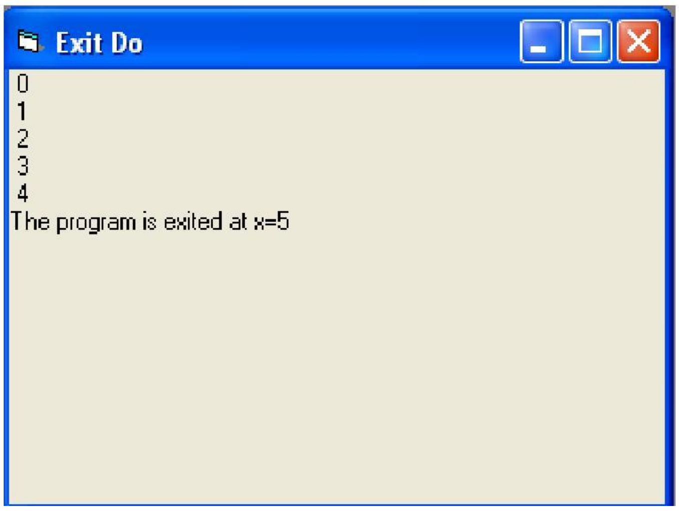
For its application, you can refer to example:

```
     1- Private sub Form Load_( )
Form1.show
Dim n as Integer
For n=1 to 10
  If n>6 then Exit For
Picture1.Print n
 Next
End Sub
```



```
1
2
3
4
5
6
```

```
1- Private sub Form Load_( )
      Form1.show Dim
      x As Integer X=0
      Do While x < 10
      Print x
      x = x + 1
      If x = 5 Then
      Print "The program is exited at x=5"
      Exit Do
      End If
      Loop
      End Sub
```



```
0
1
2
3
4
The program is exited at x=5
```

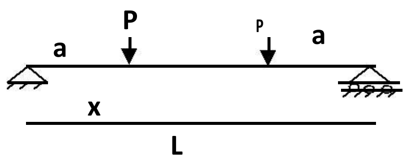
4- Nested Loop: The nested loops are the loops that are placed inside each other. The most inner loop will be executed first, then the outer ones. These are examples of the nested loops.

Possible	**Error (Not Possible)**
For J=1 to 5	**For K=1 to 5**
Statement	Statement
For I=1 to 5	For I=1 to 5
Statement	Statement
Next I	**Next K**
Statement	Statement
Next J	Next I

Example 4: For a simply supported beam shown in Fig below. By using the input box statement, enter the value of length of the beam (L), concentrated load (P), distance (a) from support, modulus of elasticity (E) and moment of inertia (IG). Write a code program to find the value of deflection at distance (X) from support, where X increased by (0.01L) from the following equation. Print the deflection value in separate text box. Designs a form and select all control object are used.

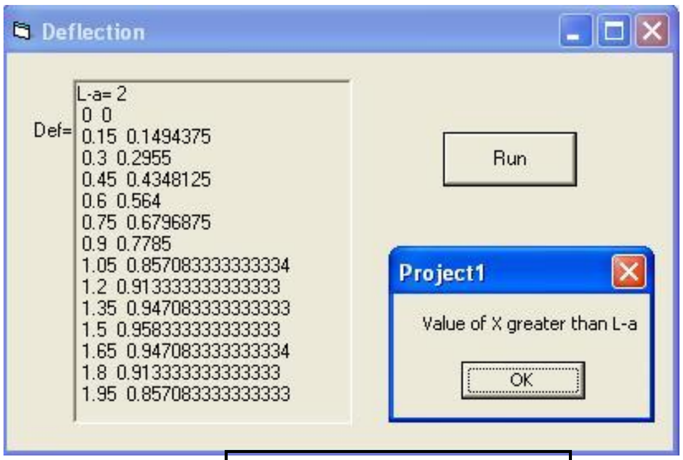
$$\Delta_x = \frac{P\,x}{6EI_G}\left(3L\,a - 3\,a^2 - x^2\right) \quad x < a$$

$$\Delta_x = \frac{P\,a}{6EI_G}\left(3L\,x - 3\,x^2 - a^2\right) \quad a \le x \le L - a$$



Solution:

```
Private Sub Command1_click( )
Dim L, P, E, IG, a, X, Df
L=Val ( Inputbox (“L=”) )
P=Val ( Inputbox (“P=”) )
IG=Val ( Inputbox (“IG=”) )
E= Val ( Inputbox (“E=”) )
a=Val ( Inputbox (“a=”) )
For X=0 To L Step 0.01 *L
If X< a Then
Df=p*X/(6*E*IG)*(3*L*a-3*a^2-X^2)
ElseIf X<= L- a Then
Df= p*a/(6*E*IG)*(3*L*X-3*X^2-a^2)
Else
Msgbox” Value of X greater than L-a” : Exit For
EndIf
Picture1.print X; Df
Next X
End Sub
```



```
L-a= 2
 0  0
 0.15  0.1494375
 0.3  0.2955
 0.45  0.4348125
 0.6  0.564
 0.75  0.6796875
 0.9  0.7785
 1.05  0.857083333333334
 1.2  0.913333333333333
 1.35  0.947083333333333
 1.5  0.958333333333333
 1.65  0.947083333333334
 1.8  0.913333333333333
 1.95  0.857083333333333
```

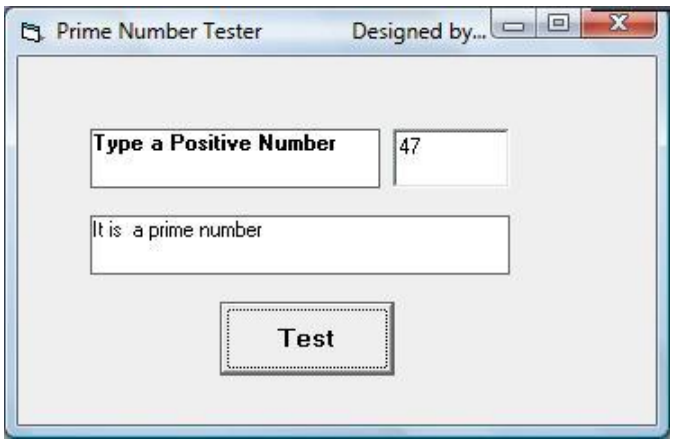
Value of X greater than L-a

L=3: P=1:a=1:E=1: IG=1

Example 5: Design a form with one command and two text boxes. Enter the value of integer number (N) in separate text box. Write a code program to check if the number (N) is a prime Number or not. Display the “It is not a prime number” or “It is a prime number” in separate text box.

Solution:

```
Private Sub Command1_Click()
Dim N, D As Single
Dim tag As String N
= Val(Text1.Text)
Select Case N
Case Is < 2
Text2.text = "It is not a prime
number" Case 2
Text2.text = "It is a prime number"
Case Is > 2
D = 2
Do
If N / D = Int(N / D) Then
Text2.text = "It is not a prime
number" tag = "Not Prime"
Exit Do
End If
D = D + 1
Loop While D <= N - 1
If tag <> "Not Prime" Then Text2.text = "It is a prime number"
```



```
End If
End Select
End Sub
```

Example 6: Create a Visual Basic Project to find the value of the following series.

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots$$

Solution:

```
Private Sub Command1_click()
Dim S as double, N , I , T
N=val(text1.text) : S= 0.0
For I=1 To N
T=1 / I^2
S=S+T
Next
Pi=SQR (S*6)
Text2.text=Str (Pi)
End Sub
```

Solution:

```
 Private Sub Command1_click()
Dim X, Sx, I, J, T, K, N, Fact
X = Val(Text1.Text): X = X * 3.14 / 180
N = 1: K = 1: Sx = 0#
  10Fact = 1
For I = 1 To 2 * N - 1
Fact = Fact * I
Next I
T = X ^ (2 * N - 1) / Fact
If Abs(T) >= 0.000001 Then
Sx = Sx + T * K
K = -K: N = N + 1
GoTo 10
Else
Text2.Text = Str(N)
Text3.Text = (Sx)
End If: End Sub
```

Example 7: Create a Visual Basic Project to find the value of the following series.

$$\text{Sum}=\sum_{i=1}^{N} a * i + b$$

Write the code program so that the value of constants (a, and b) are entered into text boxes. When the users click checkbox, calculate the value of series (where the total number of terms is equal 20). When the user unchecked the checkbox, the number of terms (N) is entered into input box and calculate the value of series. Display the value of series (Sum) in a separate text box.

Solution:

```
Private Sub Command1_Click ( )
Dim a, b, Sum, N
a = Val (Text1.Text)
b = Val (Text2.Text)
Sum = b
If Check1.Value = 1 Then
For I = 1 To 20
Sum = Sum + a * I
Next
Else
N = Val (inputbox (“No. of terms=”))
For I = 1 To N
Sum = Sum + a * I
Next
End If
Text3.Text = Str (Sum)
End Sub
```