Exit

Logic Flow - Looping

Objectives

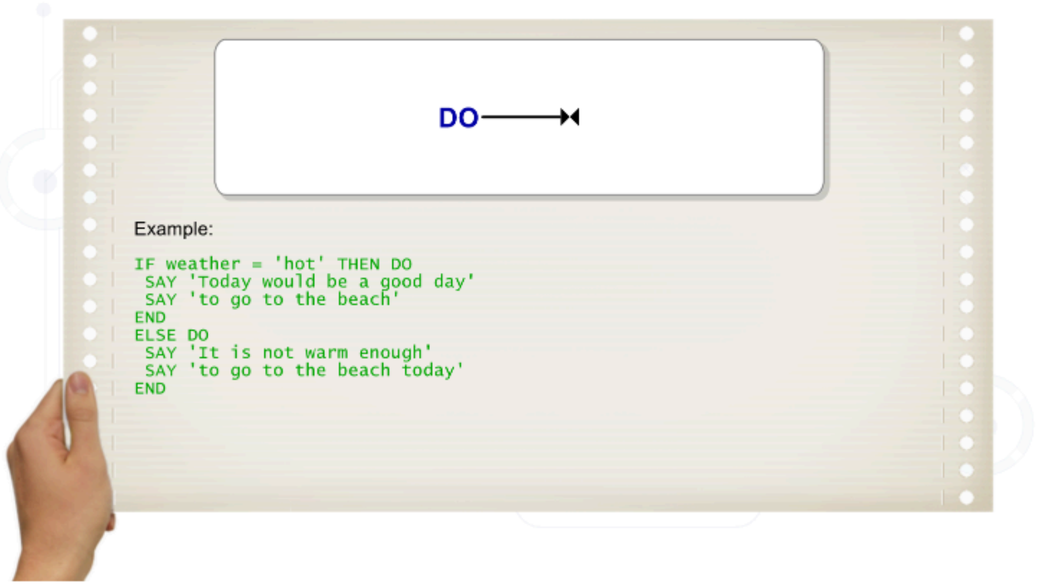
Logic Flow - Looping

In this module, you will discover how the DO keyword instruction is used by REXX to enable looping within an exec.

You will also look at the different types of loops and see how they are coded.

After completing this module, you will be able to recognize:

- Repetitive DO Loops
- Controlled Repetitive DO Loops
- Conditional DO Loops
- Compound DO Loops

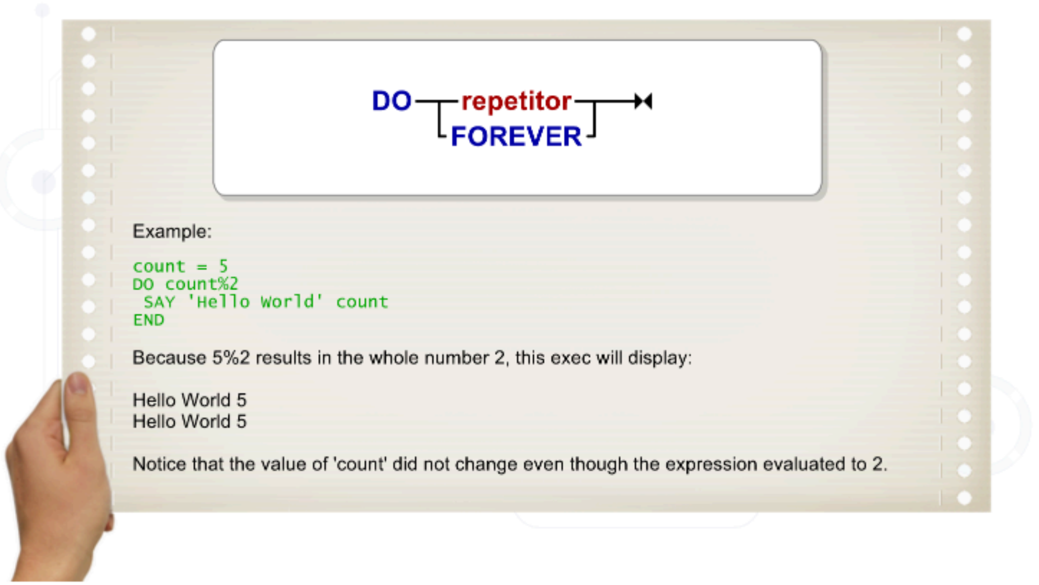


Example:

```
IF weather = 'hot' THEN DO
 SAY 'Today would be a good day'
 SAY 'to go to the beach'
END
ELSE DO
 SAY 'It is not warm enough'
 SAY 'to go to the beach today'
END
```

You have looked at simple DO groups in the form of the DO/END construct. This form of the DO instruction is the only one that does not loop.

It is used as a means of grouping a series of instructions together, particularly when used in conjunction with the IF/THEN/ELSE and SELECT/WHEN/OTHERWISE keyword instruction constructs.


DO
repetitor
FOREVER

Example:

```
count = 5
DO count%2
 SAY 'Hello World' count
END
```

Because 5%2 results in the whole number 2, this exec will display:

```
Hello World 5
Hello World 5
```

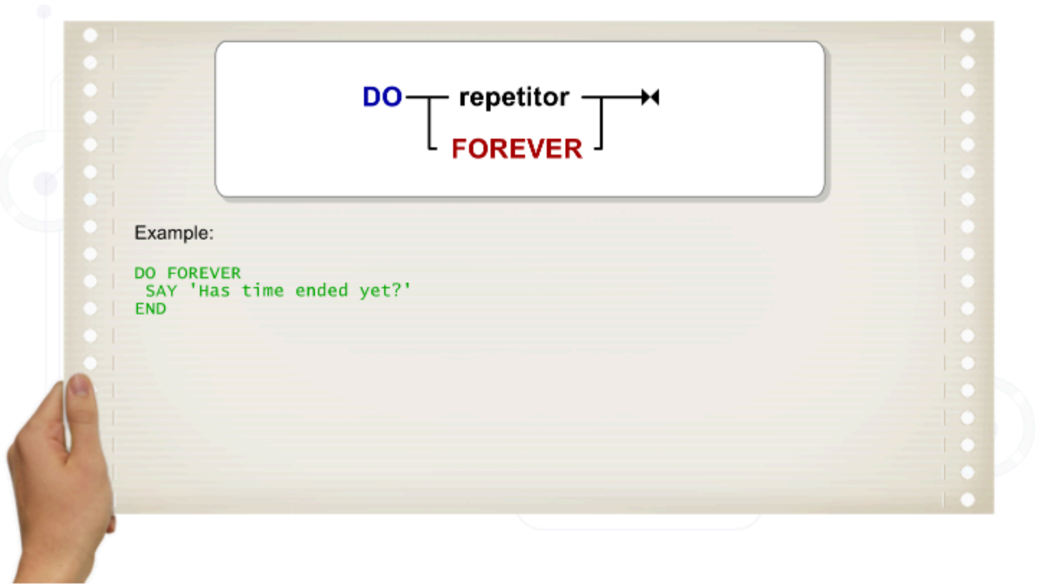
Notice that the value of 'count' did not change even though the expression evaluated to 2.

Alternatively, a number or repetitor can be used to create a simple DO loop that iterates or loops a set number of times. A "loop" is a series of instructions that are repeated one or more times within a DO/END construct. The repetitor can be any one of the following:

- A positive whole number or zero: `DO 5 /* loop 5 times */`

- A variable containing a positive whole number: `count = 5 DO count /* loop 5 times */`

- An expression that equates to a positive whole number: `DO 10/2 /* loop 5 times */`



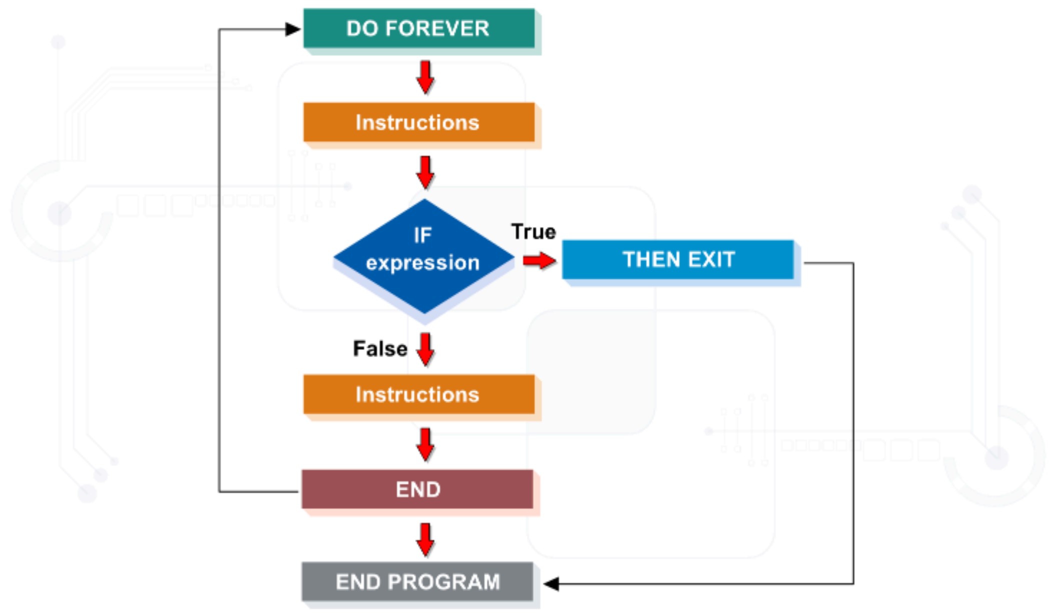
Example:

```
DO FOREVER
 SAY 'Has time ended yet?'
END
```

If the FOREVER operand is used, the loop will iterate infinitely, or until an EXIT or LEAVE instruction causes the loop to end.

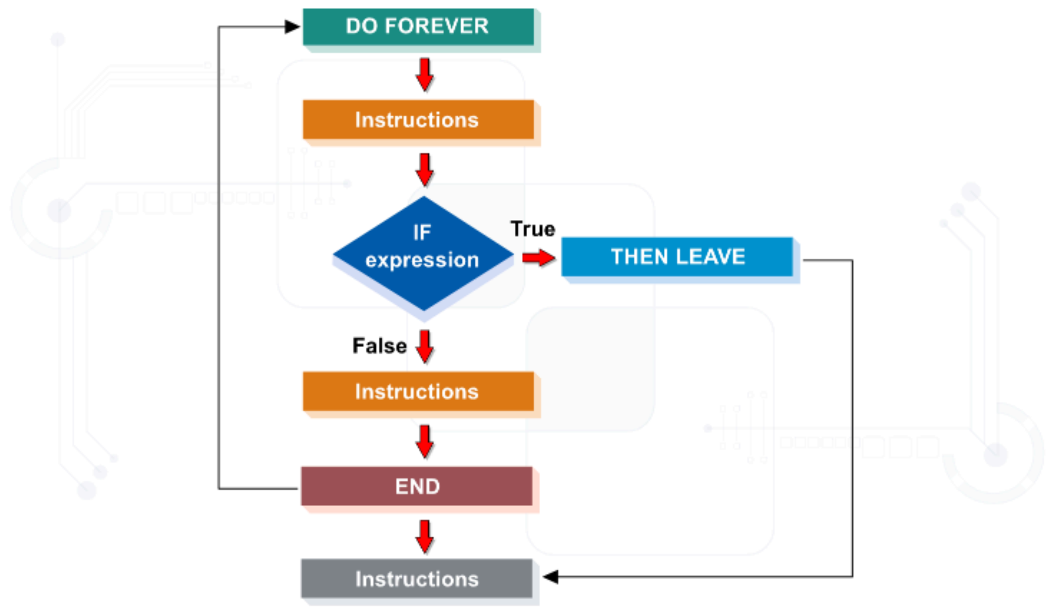
Obviously, a program that displays the same message as fast as the machine allows until time ends, or until the system programmer discovers it is wasting enormous amounts of CPU cycles and cancels the session, would be impractical under normal circumstances. You will look at the practical uses of the FOREVER parameter later.

Pressing the interrupt (PA1) or attention (ATTN) key will normally interrupt and halt a program suspected of being in an infinite loop.



DO loops are ideal when a program must iterate a known or unknown number of times, but must terminate as soon as the required result has been achieved. The two instructions that enable a DO loop to be terminated prematurely are EXIT and LEAVE.

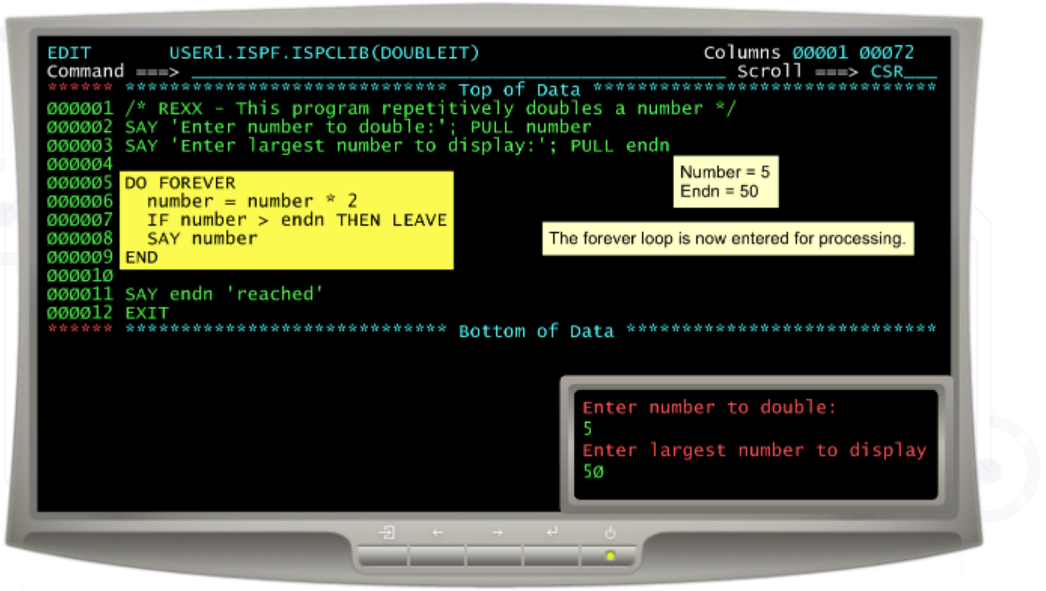
The least used of these is the EXIT instruction, which terminates the program immediately. This might be useful if an internal error is detected while in a loop, but has limited use in most program logic as the program ceases to run.



Unlike EXIT, the LEAVE instruction was designed specifically for loops. When encountered, it causes control to be passed to the instruction after the END clause, thereby terminating the loop but not the program.

As shown above, LEAVE is usually executed as the result of a conditional expression.

Repetitive Loops > LEAVE Example



```
EDIT       USER1.ISPF.ISPCLIB(DOUBLEIT)                   Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
000001 /* REXX - This program repetitively doubles a number */
000002 SAY 'Enter number to double:'; PULL number
000003 SAY 'Enter largest number to display:'; PULL endn
000004
000005 DO FOREVER
000006   number = number * 2
000007   IF number > endn THEN LEAVE
000008   SAY number
000009 END
000010
000011 SAY endn 'reached'
000012 EXIT
****** **************************** Bottom of Data ****************************
```

Number = 5
Endn = 50

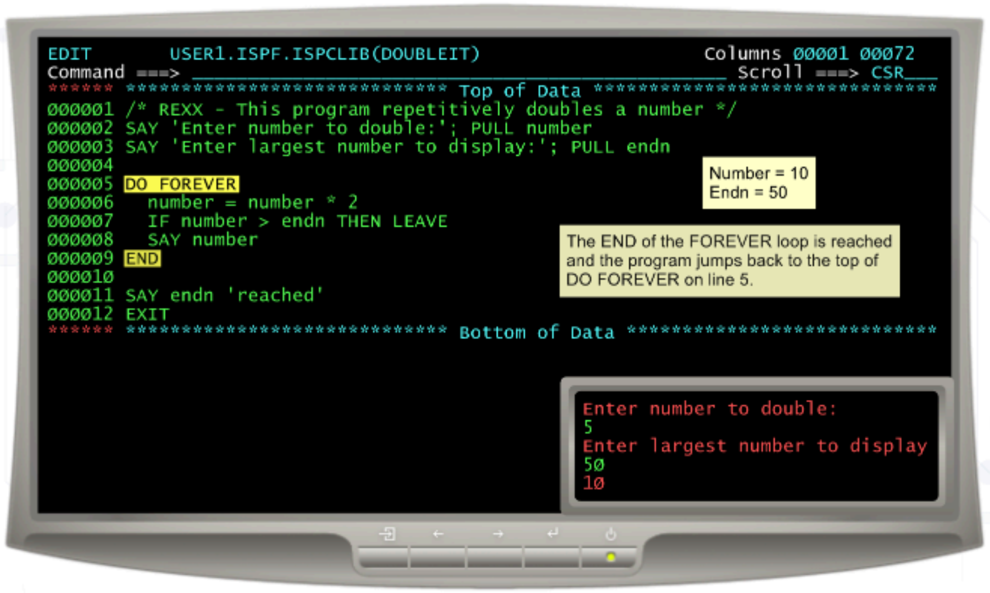
The forever loop is now entered for processing.

```
Enter number to double:
5
Enter largest number to display
50
```

The DOUBLEIT program shown here will repetitively double and display a number until a maximum value has been reached.

Click Play to see how the first section of this program asks for and accepts input from the terminal.

8 / 28



```
EDIT       USER1.ISPF.ISPCLIB(DOUBLEIT)                       Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
000001 /* REXX - This program repetitively doubles a number */
000002 SAY 'Enter number to double:'; PULL number
000003 SAY 'Enter largest number to display:'; PULL endn
000004
000005 DO FOREVER
000006   number = number * 2
000007   IF number > endn THEN LEAVE
000008   SAY number
000009 END
000010
000011 SAY endn 'reached'
000012 EXIT
****** **************************** Bottom of Data ****************************
```

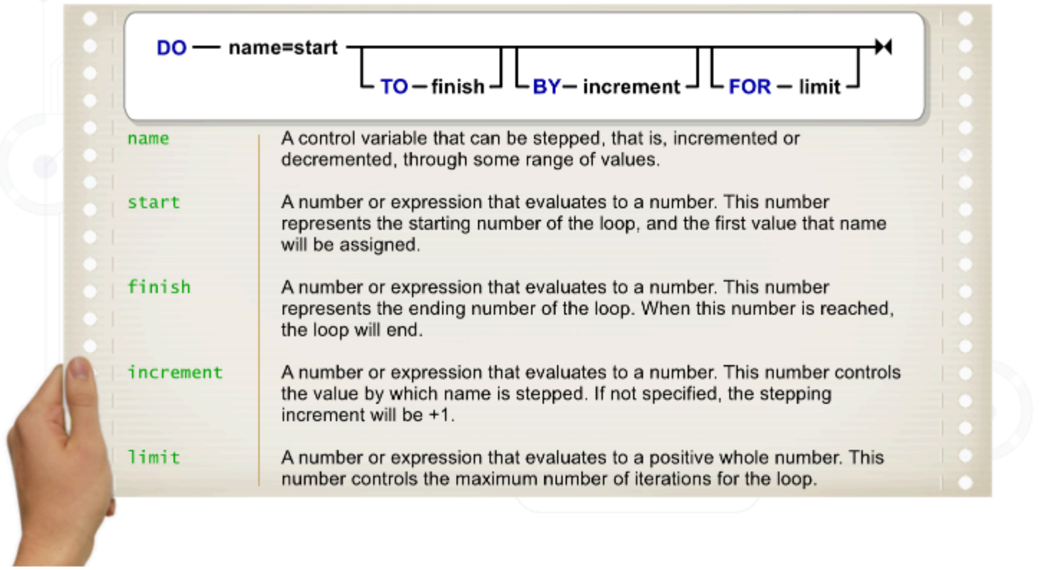
Number = 10
Endn = 50

The END of the FOREVER loop is reached and the program jumps back to the top of DO FOREVER on line 5.

```
Enter number to double:
5
Enter largest number to display
50
10
```

Now the program has entered the loop. As the FOREVER parameter has been used, checking is not performed each time the loop iterates.

Click **Play** to see the loop process being performed by the program.



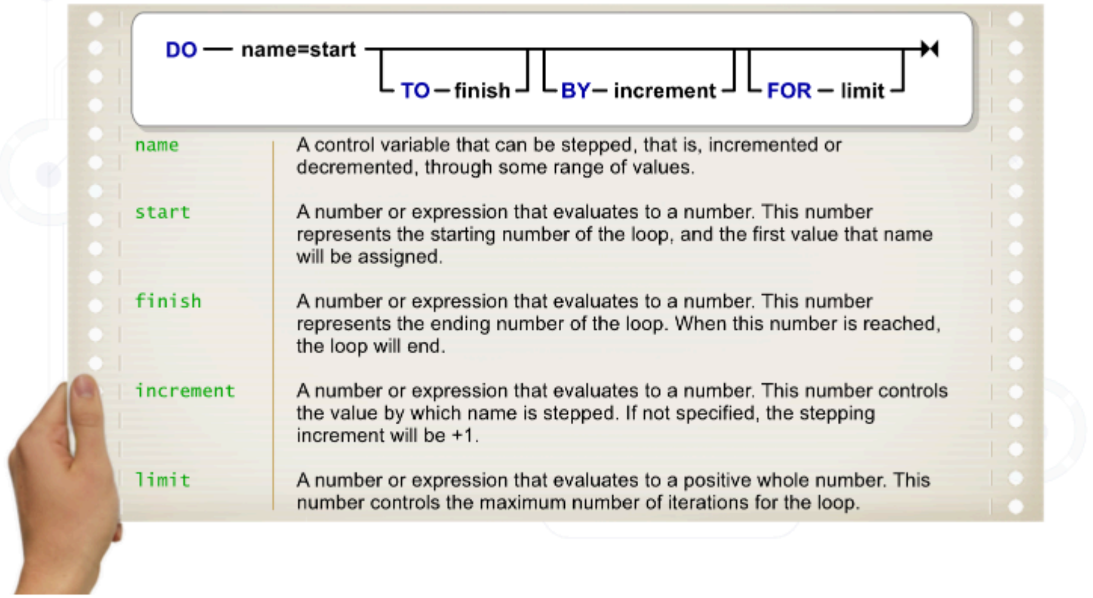
```
DO — name=start ─┬──────────┬─┬───────────────┬─┬────────────┬─▶◀
                 └TO─finish─┘ └BY─ increment─┘ └FOR ─ limit─┘
```

name	A control variable that can be stepped, that is, incremented or decremented, through some range of values.
start	A number or expression that evaluates to a number. This number represents the starting number of the loop, and the first value that name will be assigned.
finish	A number or expression that evaluates to a number. This number represents the ending number of the loop. When this number is reached, the loop will end.
increment	A number or expression that evaluates to a number. This number controls the value by which name is stepped. If not specified, the stepping increment will be +1.
limit	A number or expression that evaluates to a positive whole number. This number controls the maximum number of iterations for the loop.

An alternative to the FOREVER loop is a controlled repetitive loop, which is also called an iterative DO loop. A controlled repetitive DO loop provides a large number of optional parameters to control how the loop operates.

When the controlled repetitive DO loop is first entered, the control variable or name is assigned the value of `start` prior to the execution of the instructions preceding the END clause.

With each iteration, the value of `name` is stepped by adding the value of `increment` prior to the execution of the instruction list.

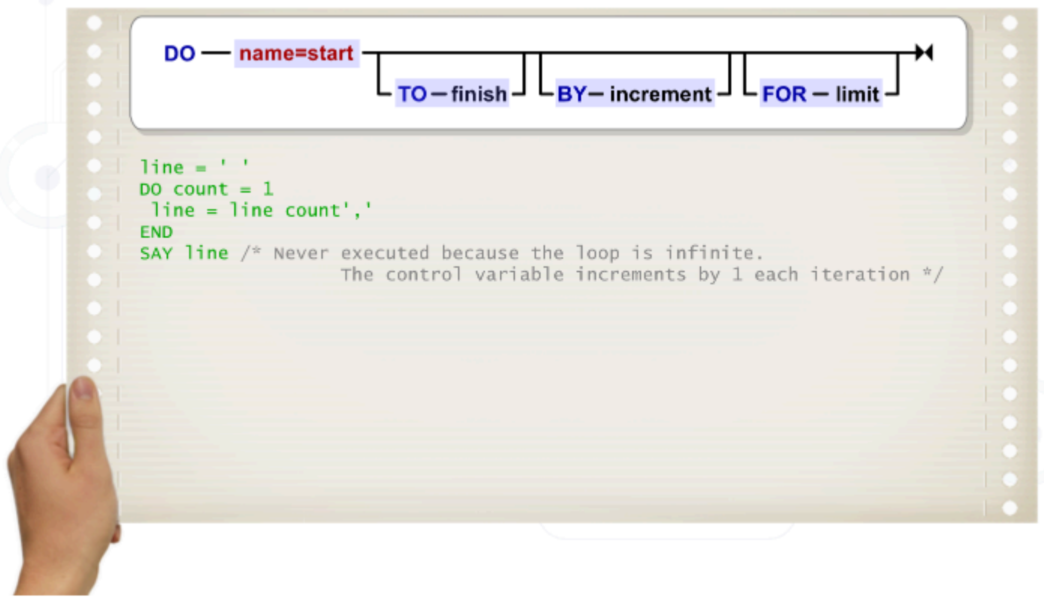


```
DO — name=start ─┬──────────────┬─┬────────────────┬─┬─────────────┬─▸◂
                 └─TO — finish ─┘ └─BY — increment ─┘ └─FOR — limit ─┘
```

name	A control variable that can be stepped, that is, incremented or decremented, through some range of values.
start	A number or expression that evaluates to a number. This number represents the starting number of the loop, and the first value that name will be assigned.
finish	A number or expression that evaluates to a number. This number represents the ending number of the loop. When this number is reached, the loop will end.
increment	A number or expression that evaluates to a number. This number controls the value by which name is stepped. If not specified, the stepping increment will be +1.
limit	A number or expression that evaluates to a positive whole number. This number controls the maximum number of iterations for the loop.

The loop completes processing until one of the termination clauses is satisfied:

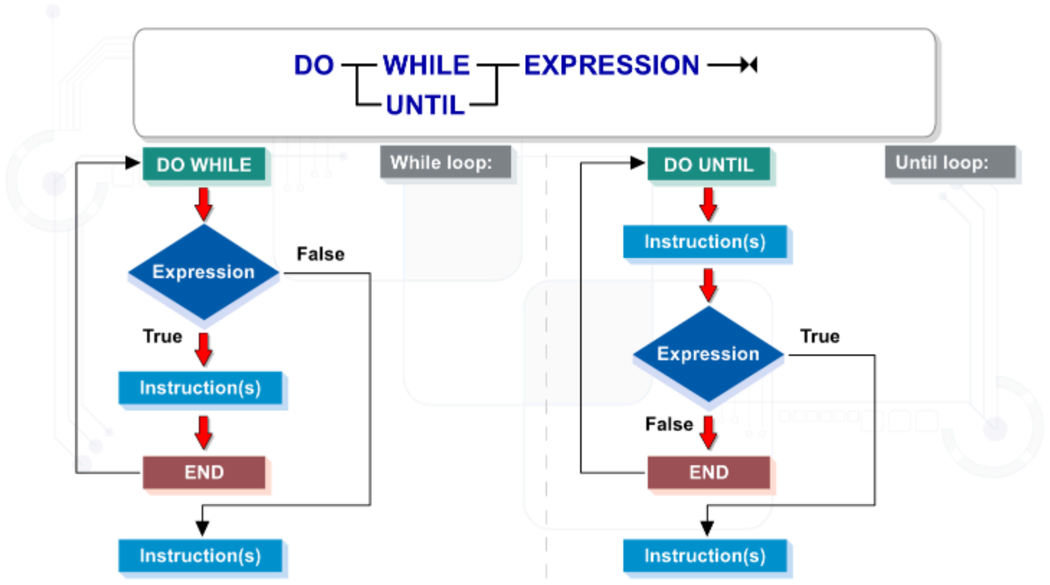
- If increment is a positive number, looping stops when `name` is greater than `finish`.
- If increment is a negative number, looping stops when `name` is less than `finish`.
- If the BY clause is not specified, the default `increment` is 1.
- The loop can be limited to a maximum number of iterations by using the value of `limit`.
- The FOR clause, if used, takes precedence over the normal execution of the TO and BY clauses if `limit` is reached



```
line = ' '
DO count = 1
 line = line count','
END
SAY line /* Never executed because the loop is infinite.
                 The control variable increments by 1 each iteration */
```

Here are some examples of iterative DO loops and their results.

Mouse-over each parameter to see how they are used.

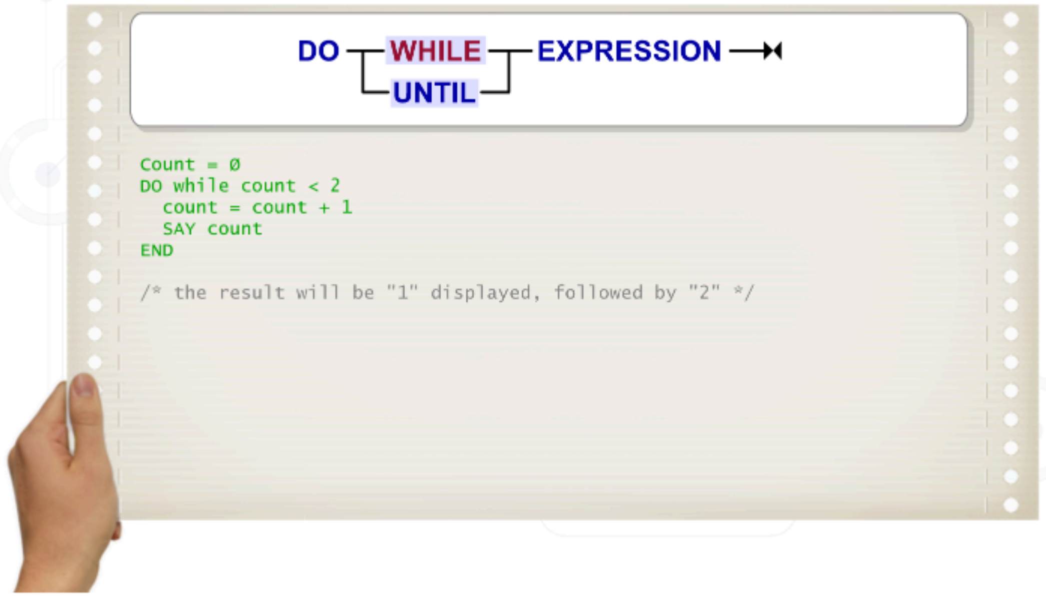


Another form of loop is the conditional DO loop, which has two mutually exclusive condition tests that can be performed.

WHILE loops test that the expression is true before processing any instructions and looping; this is referred to as "pre-processing" because the test is done before the loop is processed.

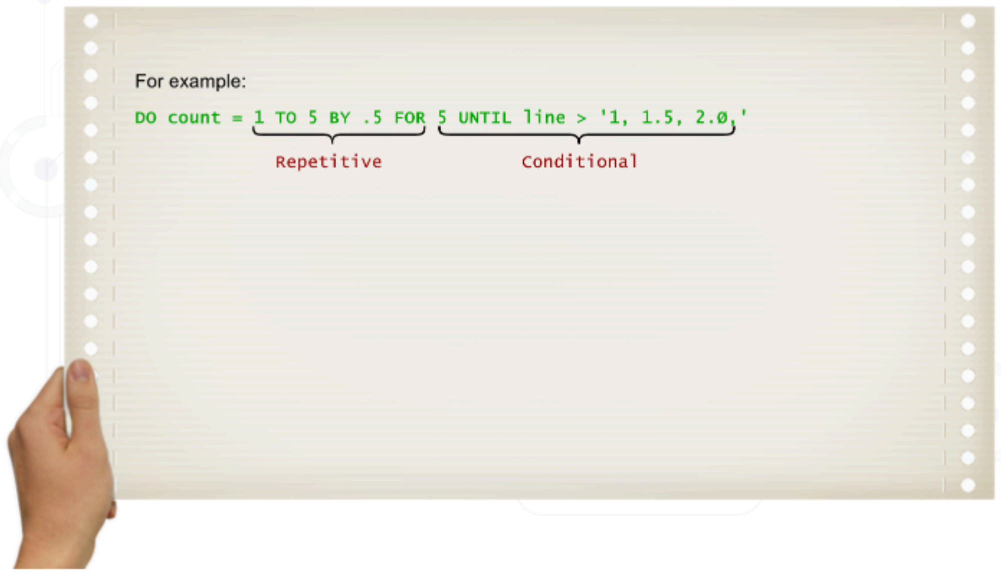
UNTIL loops process all instructions before testing the expression and will only loop if it is false; this is referred to as "post-processing" because the test is done after the loop has been processed.

Note that UNTIL loops will always process at least once.



```
Count = 0
DO while count < 2
  count = count + 1
  SAY count
END

/* the result will be "1" displayed, followed by "2" */
```

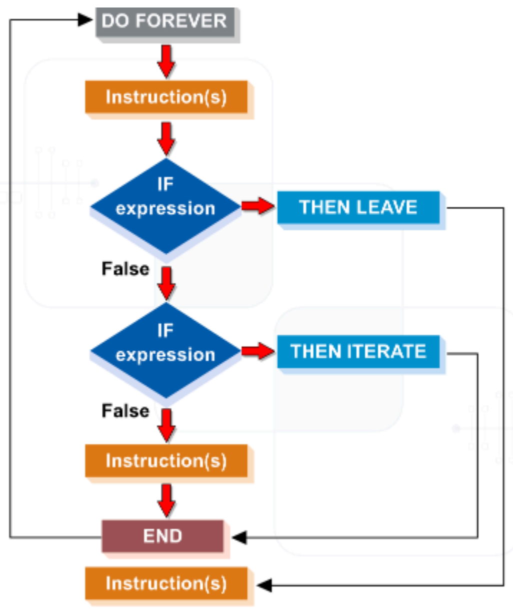
Mouse-over the WHILE and UNTIL parameters to see an example of their use.



For example:

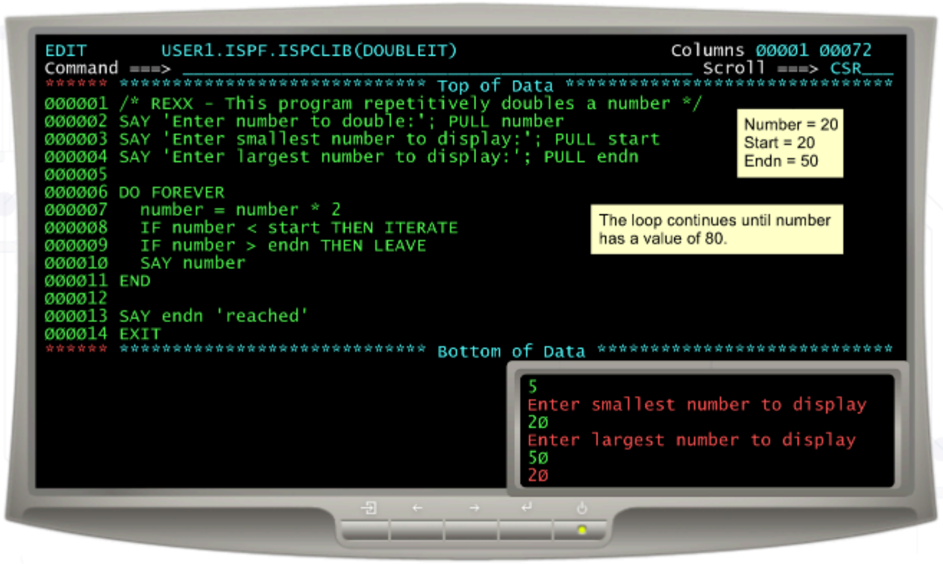
$$\texttt{DO count = }\underbrace{\texttt{1 TO 5 BY .5 FOR}}_{\text{Repetitive}}\ \underbrace{\texttt{5 UNTIL line > '1, 1.5, 2.0,'}}_{\text{Conditional}}$$

A compound DO loop is a combination of both repetitive and conditional DO loops. The compound loop ends when the repetition counter is reached or the condition is met, whichever comes first.



DO loops normally iterate when the END clause is reached, but you may not want to process any further instructions within a loop during a particular iteration.

The ITERATE keyword instructs REXX to pass control to the END statement and iterate the loop. As shown in the diagram, ITERATE is usually executed as the result of a conditional expression.



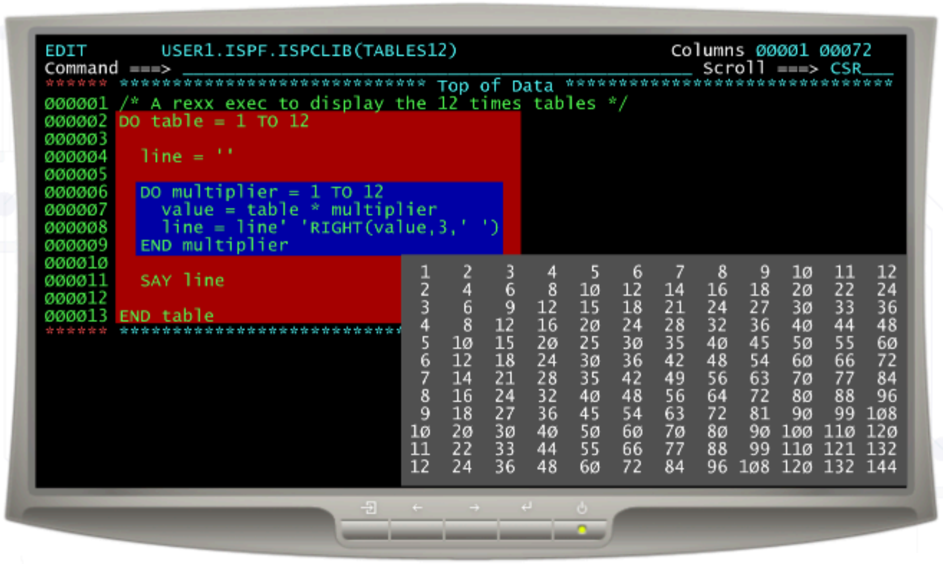
```
EDIT       USER1.ISPF.ISPCLIB(DOUBLEIT)                    Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
/* REXX - This program repetitively doubles a number */
SAY 'Enter number to double:'; PULL number
SAY 'Enter smallest number to display:'; PULL start
SAY 'Enter largest number to display:'; PULL endn

DO FOREVER
  number = number * 2
  IF number < start THEN ITERATE
  IF number > endn THEN LEAVE
  SAY number
END

SAY endn 'reached'
EXIT
****** **************************** Bottom of Data ****************************
```

```
5
Enter smallest number to display
20
Enter largest number to display
50
20
```

This version of the DOUBLEIT program has been modified to use the ITERATE instruction. A "minimum" is now requested and no numbers will be displayed until a minimum value has been reached.

We will assume that when this program is executed, the user enters 5 as the number to double (number), 20 as the smallest number to display (start), and 50 as the largest number to display (end).

Click **Play** to see how the FOREVER loop is now processed with the LEAVE and ITERATE keywords.



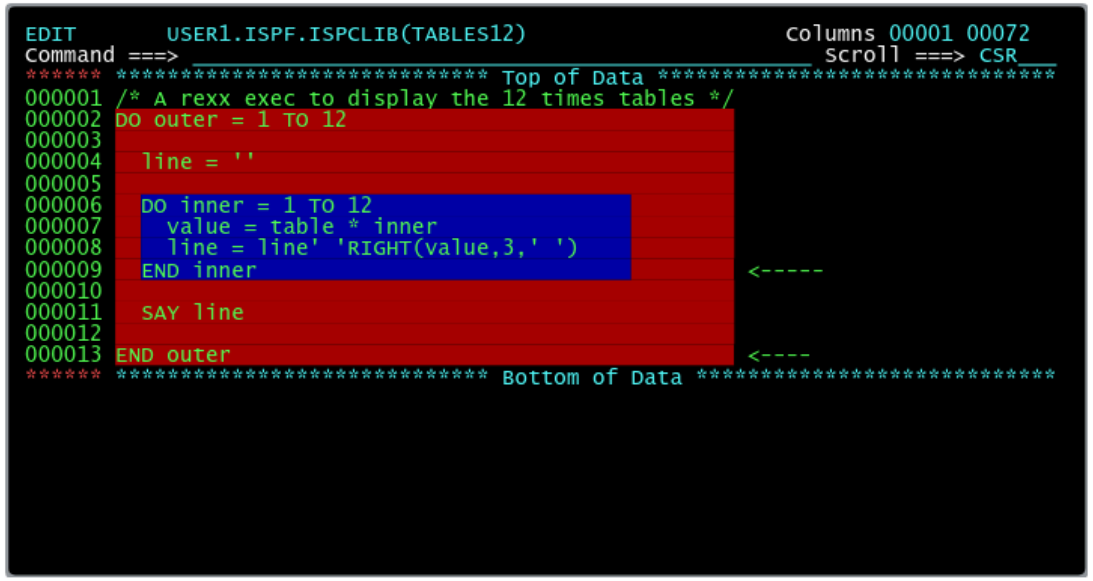
```
EDIT       USER1.ISPF.ISPCLIB(TABLES12)                  Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
000001 /* A rexx exec to display the 12 times tables */
000002 DO table = 1 TO 12
000003
000004   line = ''
000005
000006   DO multiplier = 1 TO 12
000007     value = table * multiplier
000008     line = line' 'RIGHT(value,3,' ')
000009   END multiplier
000010
000011   SAY line
000012
000013 END table
****** ****************************
```

```
  1   2   3   4   5   6   7   8   9  10  11  12
  2   4   6   8  10  12  14  16  18  20  22  24
  3   6   9  12  15  18  21  24  27  30  33  36
  4   8  12  16  20  24  28  32  36  40  44  48
  5  10  15  20  25  30  35  40  45  50  55  60
  6  12  18  24  30  36  42  48  54  60  66  72
  7  14  21  28  35  42  49  56  63  70  77  84
  8  16  24  32  40  48  56  64  72  80  88  96
  9  18  27  36  45  54  63  72  81  90  99 108
 10  20  30  40  50  60  70  80  90 100 110 120
 11  22  33  44  55  66  77  88  99 110 121 132
 12  24  36  48  60  72  84  96 108 120 132 144
```

As with IF/THEN/ELSE constructs, DO loops can also be nested within other DO loops.

This program produces a 12 x 12 matrix of a multiplication table. The outer loop controls which line of the matrix is to be built. The inner loop builds the output line with all the multiplication values and displays it. The RIGHT function on line 8 will be covered in a later module.

Click **Play** to see the result of running this REXX program.

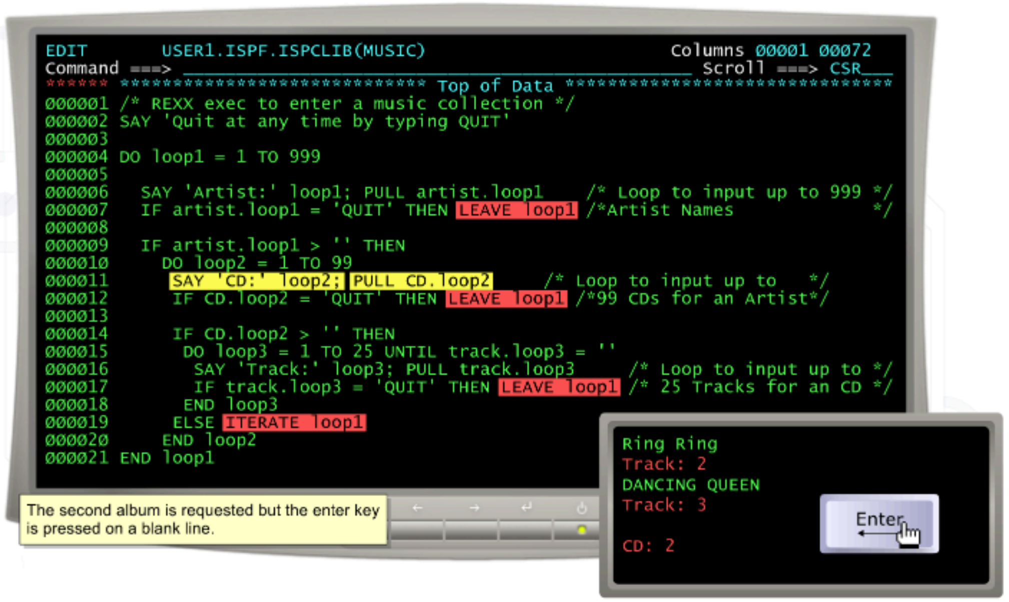


```
EDIT       USER1.ISPF.ISPCLIB(TABLES12)                    Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** ***************************** Top of Data ******************************
000001 /* A rexx exec to display the 12 times tables */
000002 DO outer = 1 TO 12
000003
000004   line = ''
000005
000006   DO inner = 1 TO 12
000007     value = table * inner
000008     line = line' 'RIGHT(value,3,' ')
000009   END inner                                <-----
000010
000011   SAY line
000012
000013 END outer                                  <----
****** **************************** Bottom of Data ****************************
```

In this example, both END clauses have been coded with the name of the loop control variables.

This option is available for controlled repetitive DO loops and is good practice because it aids debugging when working with multiple nested loops.

This parameter is purely for documentation and debugging purposes and has no effect on processing.



```
EDIT       USER1.ISPF.ISPCLIB(MUSIC)                      Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR___
****** **************************** Top of Data ******************************
/* REXX exec to enter a music collection */
SAY 'Quit at any time by typing QUIT'

DO loop1 = 1 TO 999

  SAY 'Artist:' loop1; PULL artist.loop1    /* Loop to input up to 999 */
  IF artist.loop1 = 'QUIT' THEN LEAVE loop1 /*Artist Names            */

  IF artist.loop1 > '' THEN
    DO loop2 = 1 TO 99
     SAY 'CD:' loop2; PULL CD.loop2     /* Loop to input up to   */
     IF CD.loop2 = 'QUIT' THEN LEAVE loop1 /*99 CDs for an Artist*/

     IF CD.loop2 > '' THEN
      DO loop3 = 1 TO 25 UNTIL track.loop3 = ''
       SAY 'Track:' loop3; PULL track.loop3     /* Loop to input up to */
       IF track.loop3 = 'QUIT' THEN LEAVE loop1 /* 25 Tracks for an CD */
      END loop3
     ELSE ITERATE loop1
    END loop2
END loop1
```

The second album is requested but the enter key is pressed on a blank line.

```
Ring Ring
Track: 2
DANCING QUEEN
Track: 3

CD: 2
```

When working with controlled repetitive DO loops, control variables can be used on the LEAVE and ITERATE instructions to alter the flow of active loops. This is useful for modifying the flow of nested DO loops.

Click Play to see how this REXX program would run and the role that control variables would play.