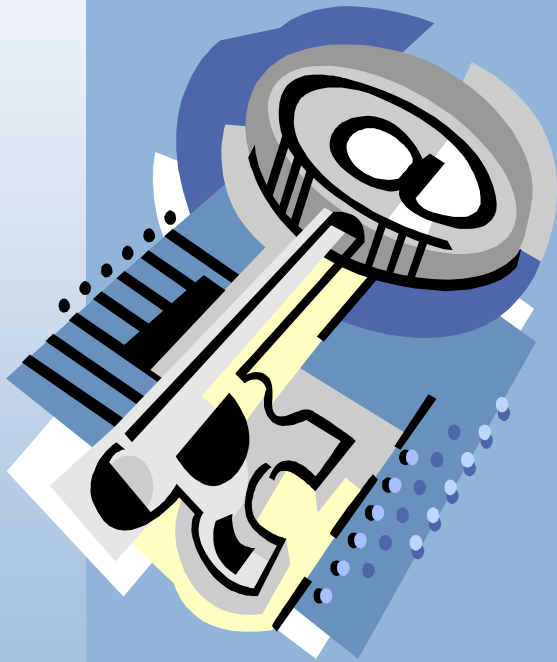


COBOL for the 21st Century

11th edition

John Wiley & Sons, Inc.



Chapter 9

Iteration: Beyond the Basic PERFORM

Chapter Objectives

To familiarize you with

- Simple PERFORM
- How PERFORM statements are used for iteration
- Options available with PERFORM

Simple PERFORM

Format

PERFORM [paragraph-name-1]

- Executes all instructions in named paragraph
- Then transfers control to instruction following PERFORM

Simple PERFORM

- Use to execute a paragraph from different points in a program
- Use to modularize program
 - Write each set of related instructions as separate module or paragraph
 - Use PERFORM paragraph-name to execute each module as needed

In-Line PERFORM

Format

PERFORM

. } Statements to
: } be executed
.

END-PERFORM

- Use when only a few statements are to be executed
- Modularize with PERFORM paragraph-name when many statements required

Nested PERFORM

- PERFORM may be one of instructions in range of another PERFORM

Perform 200-Paragraph

⋮

200-Paragraph.

Perform 500-Paragraph

← Nested
PERFORM

Nested In-Line PERFORM

- In-line PERFORMs can include nested in-line PERFORMs or PERFORMs with paragraph-name

Perform

...

Perform

...

End-Perform

...

End-Perform

Executing Group of Paragraphs

Format

PERFORM paragraph-name-1

{ THROUGH } paragraph-name-2
{ THRU }

- Use expanded format to execute all statements, including other paragraphs, from paragraph-name-1 through paragraph-name-2

Control Structures

- Sequence
 - instructions executed in order in which they appear
- IF-THEN-ELSE or selection
 - instructions executed depending on value of condition
- Iteration or looping
 - series of instructions executed repeatedly
 - either in-line or in different module

PERFORM UNTIL

Format

PERFORM [paragraph-name-1]
UNTIL condition-1

- Repeats statements in paragraph until condition is true
- Called iteration or loop

In-Line PERFORM UNTIL

- No paragraph name follows PERFORM
- Instead statements to be repeated placed between PERFORM UNTIL ...
END-PERFORM

Coding a Loop with PERFORM

- Often want to perform some action a certain number of times
- Use a field as a counter to count number of times action is repeated
- Set field to zero initially, then increment it by 1 each time action repeated
- When field equals number of times action is to be repeated, condition is met and loop ends

Loop Example

- Display the message 'Hello' 3 times
Move Zeros To Count
Perform Until Count = 3
 Display 'Hello'
 Add 1 To Count
End-Perform

Loop Example

- Count initialized to zero so not equal to 3 when condition checked first time
- Hello displayed on screen and Count incremented to 1

Move Zeros To Count

Perform Until Count = 3

Display 'Hello'

Add 1 To Count

End-Perform

Loop Example

- Condition checked again and still not met (1 not equal to 3) so in-line statements repeated again
- Hello displayed second time and Count incremented to 2

Move Zeros To Count

Perform Until Count = 3

Display 'Hello'

Add 1 To Count

End-Perform

Loop Example

- Count not equal to 3 so Hello displayed third time and Count incremented to 3
- Condition met when checked again so loop ends

Move Zeros To Count

Perform Until Count = 3

Display 'Hello'

Add 1 To Count

End-Perform

Coding a Loop

- Precede loop by instruction to initialize field to be tested
- Include `PERFORM UNTIL ...` that repeats until field tested reaches desired value
- Include instruction in loop to change value of field tested so that condition is eventually met

Condition Tested First

- Condition tested *before* paragraph or in-line statements executed even once
- If condition met on first test, paragraph or statements executed zero times

Example

Move 6 To X

Perform 300-Process-Rtn

Until $X > 5$

Paragraph
executed 0 times



Ending PERFORM UNTIL

- Loop stops when condition is true
- One of instructions in loop should change identifier used in condition

Example

Move 0 To Y

Perform Until $Y > 10$

...

Add 1 To Y ←

End-Perform

Changes Y so condition eventually met

Avoid Loop Errors

- Consider this loop
 - Move Zeros To Count
 - Perform Until Count = 5
 - Display Out-Message
 - End-Perform
- Error occurs because no instruction included to change Count from zero
- DISPLAY executed over and over again because condition never met

Avoid Loop Errors

- Loop that executes repeatedly without end called infinite loop
- On mainframe program automatically terminated after fixed period of time
- On PCs press interrupt keys (e.g., Escape key, Ctrl + Break keys) to terminate program

Alternative Loop

Move 1 To Count

Perform Until Count > 3

Display 'Hello'

Add 1 To Count

End-Perform

- Initialization value for Count now 1
- Condition uses '>' instead of '='

Alternative Loop

- Loop still displays Hello 3 times when Count = 1, 2 and 3
- When Count = 4, loop condition met
- Testing for '>' or '>=' less prone to infinite loops than testing for '='
- If value of Count exceeds 3 but is never exactly equal to 3, loop will still terminate

PERFORM ... TIMES

- Executes a sequence of steps a fixed number of times
- No counter needed
- Loop below executes paragraph 300-Print-Rtn 5 times

Perform 300-Print-Rtn
5 Times

PERFORM ... TIMES

- May use field whose value represents number of times to repeat loop
- Field must be numeric, containing only positive integers or 0
- Loop below performs 300-Print-Rtn ten times

Move 10 To How-Many
Perform 300-Print-Rtn
How-Many Times

PERFORM ... TIMES

- Also used with in-line loop
- Loop below executes MULTIPLY statement 3 times

Move 2 To Num

Perform 3 Times

Multiply 2 By Num

End-Perform

- Num equals 16 when loop ends

Loop Example

Sum even integers from 2 through 10

- Initialize a field to first number to be added (2)
- Increment field by 2 so it equals even numbers (2, 4, 6, 8, 10)
- Use this field's value to
 - Test in condition
 - Add to a total field to find sum

Code for Loop Example

- Sum even integers from 2 through 10

Move 0 To Total

Initialize field to be tested

Move 2 To Count

Perform Until Count > 10

Test field until it reaches desired value

Add Count To Total

Add 2 To Count

Change field tested so condition eventually met

End-Perform

Display 'Total=', Total

Result: Total = 30

Nested PERFORMs

- One of statements in PERFORM loop may be another PERFORM loop
- A loop within another loop is called a nested loop

Nested PERFORM Example

- Assume 50 records will be read in as 5 groups of 10 records
- Amount fields of each group are to be added and a total printed
- Five totals, one for each group of 10 records will be printed

Nested PERFORM Pseudocode

```
Perform 5 Times ← Outer loop
  Perform 10 Times ← Inner loop
    Read record from file and
      add its amount to group total
    End-Read
  End-Perform
Perform Print-Group-Total
End-Perform
```

Nested PERFORM Example

- Outer loop repeats these steps 5 times
 - Performs inner loop to read in 10 records
 - Prints group total
- Inner loop repeated 50 Times or 10 times each time outer loop is repeated
- Notice that step to print group total is not part of inner loop
 - Executed only 5 times or once each time outer loop executed

TIMES vs UNTIL

- Use `PERFORM ... TIMES` if you know in advance the number of times loop statements are to be executed
- Use `PERFORM ... UNTIL` if number of times loop repeated is needed for output or calculations

PERFORM VARYING

Format

PERFORM VARYING identifier-1

FROM { identifier-2
integer-1 }

BY { identifier-3
integer-2 }

UNTIL condition-1

statement-1 ...

END-PERFORM

PERFORM VARYING

- Repeatedly executes statements in loop while varying value of a field
- First identifier-1 is given FROM value
- Condition then tested
- Executes statements in loop if condition not met
- Then adds BY value to identifier-1 and repeats condition test

PERFORM VARYING Example

Perform Varying Ctr From 1 By 1

Until Ctr > 5

Display 'Ctr = ', Ctr

End-Perform

- Sets Ctr to 1, since Ctr > 5 not true, executes DISPLAY statement
- Increments Ctr by 1, tests condition again

PERFORM VARYING Execution

<u>CTR</u>	<u>Condition</u>	<u>Output</u>
1	1 > 5 false	Ctr = 1
2	2 > 5 false	Ctr = 2
3	3 > 5 false	Ctr = 3
4	4 > 5 false	Ctr = 4
5	5 > 5 false	Ctr = 5
6	6 > 5 true	(loop ends)

PERFORM VARYING Examples

- Finds sum of odd numbers from 1 to 25
Move 0 To Total
Perform Varying Ctr From 1 By 2
 Until Ctr > 25
 Add Ctr To Total
End-Perform
Display 'Total = ', Total

Output:
Total = 169

PERFORM VARYING Examples

- Statements to be repeated may also be in separate paragraph

Perform 300-Process-Rtn

Varying Ctr From 1 By 1

Until Ctr > 20

- Executes 300-Process-Rtn 20 Times

Nested PERFORM VARYING

- May include a PERFORM VARYING loop as one of statements in another PERFORM VARYING loop
- Each time outer loop is repeated, inner loop is executed until its condition is met
- Following example prints the times tables for numbers 1 to 9

Print Times Tables

Perform Varying N1 From 1 By 1

Outer loop

Until N1 > 9

Perform Varying N2 From 1 By 1

Inner loop

Until N2 > 9

Compute Product = N1 * N2

Display N, ' * ' M ' = ', Product

End-Perform

End-Perform

Print Times Tables Execution

<u>N1</u>	<u>N2</u>	<u>Output</u>
1	1	$1 * 1 = 1$
1	2	$1 * 2 = 2$
...
1	9	$1 * 9 = 9$
2	1	$2 * 1 = 2$
2	2	$2 * 2 = 4$
...
2	9	$2 * 9 = 18$

Outer loop first time

Inner loop repeats 9 times

Outer loop second time

Inner loop repeats 9 times

Print Times Tables Execution

- Outer loop repeated seven more times
- Each time, statements in inner loop are repeated 9 times
 - N2 initialized to 1 and incremented by 1 each time through inner loop until $N2 > 9$
- Outer loop ends after printing 9's table

PERFORM UNTIL loop

- Condition tested before statements in loop executed first time
- If condition met on first test, statements not executed at all
- Can specify that condition be tested *after* instructions executed first time
- Then instructions always executed at least once

PERFORM WITH TEST AFTER

Format

PERFORM [paragraph-name-1]

[WITH TEST { BEFORE
AFTER }]

UNTIL condition-1

TEST AFTER Example

Example

Perform With Test After

Until Opt-Num ≥ 1 And ≤ 5

Display 'Select option (1-5)'

Accept Opt-Num

End-Perform

TEST AFTER Example

- Condition is not checked before loop begins
- DISPLAY and ACCEPT for user to enter Opt-Num always executed at least once
- Checks Opt-Num *after* user types in value for first time
- If Opt-Num not a value from 1 to 5, loop is repeated

Chapter Summary

- Formats of PERFORM Statement

- Simple PERFORM

- In-Line PERFORM

- PERFORM ... END-PERFORM

- PERFORM paragraph-name-1

- [THRU paragraph-name-2]

- Causes execution of instructions in named paragraph(s)

- After paragraph executed, control returned to statement after PERFORM

Chapter Summary

- Formats of PERFORM Statement
 - PERFORM UNTIL repeats instructions until a condition is met
 - Condition may be tested before or after instructions are executed
 - PERFORM ... TIMES
 - Use when you know exact number of times loop statements are to be executed

Chapter Summary

- Formats of PERFORM Statement
 - PERFORM VARYING
 - Automatically initializes and changes value of loop counter
- Nested PERFORMS (PERFORM statements within PERFORM statements) allowed

Chapter Summary

- In-line PERFORMs permitted with all PERFORM options
 - Code does not need to be in separate paragraph
 - Terminated with END-PERFORM