ESTRUTURAS DE CONTROLO

Estruturas Condicionais (seleção)

Na maioria das linguagens procedimentais **if-else** e **case/switch** são as únicas estruturas de seleção suportadas. O Cobol suporta versões avançadas em ambas as estruturas, mas também suporta uma variedade maior de tipos de condições, incluindo condições de relação, condições de classe, condições de sinal, condições complexas e nomes de condição.

Declaração IF

Permite indicar quais as circunstâncias (normalmente a satisfação ou não de uma condição) em que determinada instrução ou conjunto de instruções deve ser executada.

SINTAXE:

```
IF Condition THEN { StatementBlock | NEXT SENTENCE }
    [ ELSE { StatementBlock | NEXT SENTENCE } ] [END - IF]
```

- Os StatementBlock (s) podem incluir qualquer instrução COBOL válida, incluindo outras construções IF (encadeamento de estruturas condicionais);
- As condições são avaliadas como sendo verdadeiras (true) ou falso (false) e não aceitam 1 e 0, como acontece com outras linguagens de programação;
- O delimitador END-IF deve sempre ser usado porque explicita o fim da(s) instrução/instruções associada(s) ao IF.

Tipos de condição

Relação

Classe – referidas na ficha 1, na página 10

Sinal

Complexa

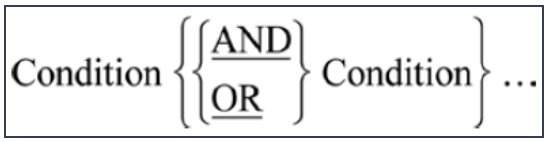
Nomes de condição - referidos na ficha 1, nos níveis especiais, nas páginas 15 e 16

Condições complexas

Formadas pela combinação de duas ou mais condições simples usando o operador de conjunção **OR** ou **AND**. Qualquer condição (simples, complexa, nome da condição) pode ser negada, precedendo-a com a palavra NOT.

Quando o NOT é aplicado a uma condição, ele alterna a avaliação verdadeiro / falso.
Por exemplo, se Num1 <10 é verdadeiro, então NOT Num1 <10 é falso.
Como outras condições no COBOL, uma condição complexa é avaliada como true ou false.

SINTAXE:



Precedência	Valor de condição
1	NOT
2	AND
3	OR

NOTA: Relembra as tabelas de verdade.

Exemplos:

1.

```
IF (Num1 > 0 AND Num1 < 25) AND (Num2 > 0 AND Num1 < 81) THEN
      DISPLAY “No ecrã”
END-IF
```

2.

```
IF Valor = “10” OR Valor = “11” OR Valor = “12” THEN
     DISPLAY “Passou”
```

É igual a:

```
IF Valor = “10” OR “11” OR “12” THEN
    DISPLAY “Passou”
```

3. **É igual a:**

`IF Num1 > Num2 AND Num1 > Num3 AND Num1 > Num4 THEN` `DISPLAY “O Num1 é o maior”`	`IF Num1 > Num2 AND Num3 AND Num4 THEN` `DISPLAY “O Num1 é o maior”`

Utilização do NOT

4.

```
IF (NOT Num1 < 25) OR (Num2 = 80 AND Num3 > 264) THEN
DISPLAY "FEITO"
END-IF
```

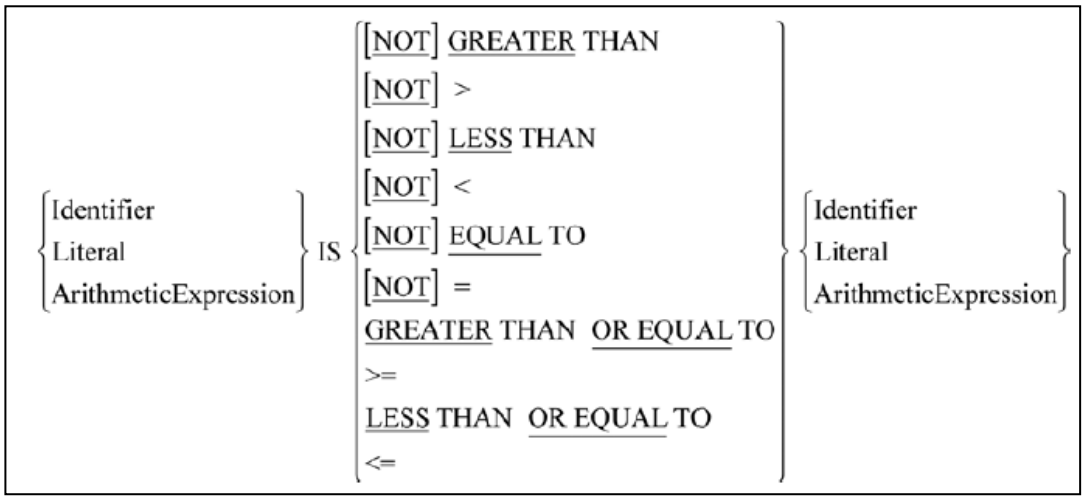
5.

```
IF NOT (Num1 < 25 OR Num2 = 80) AND Num3 > 264 THEN
DISPLAY "FEITO mas com possível resultado diferente"
END-IF
```

Supondo que Num1 tem o valor 20, Num2 tem o valor 80 e que Num3 tem o valor 300. A tabela seguinte apresenta a avaliação da instrução IF

Condição:	IF (NOT Num1 <25)	OR	(Num2=80 AND Num3 > 264)
	(**NOT** **T**)	OR	(**T** **AND** **T**)
	(**F**)	**OR**	(**T**)
Resultado da avaliação:		**TRUE**	

Condições de Relação



Comandos			Significado
	=	EQUAL TO	Igual a
<>	NOT=	NOT EQUAL TO	Diferente de
	<	LESS THAN	Menor que
>=	NOT<	NOT LESS THAN	Maior ou igual
	>	GREATER THAN	Maior
<=	NOT>	NOT GREATER THAN	Menor ou igual

Exemplo:

```
IF Num1 < 10 THEN
      DISPLAY "Num1 < 10"
END-IF
IF Num1 LESS THAN 10
      DISPLAY "Num1 < 10"
END-IF
      IF Num1 GREATER THAN OR EQUAL TO Num2
      MOVE Num1 TO Num2
END-IF
IF Num1 < (Num2 + (Num3 / 2))
      MOVE ZEROS TO Num1
END-IF
```

Condições de sinal

É usada para descobrir se o valor de uma expressão aritmética é menor que, maior que ou igual a zero. As condições de sinal são uma forma mais curta de escrever certas condições de relação.

SINTAXE:

ArithmeticExpression IS [NOT] { POSITIVE / NEGATIVE / ZERO }

Exemplo:

IF (Num1 * 10 / 50) – 10 IS NEGATIVE DISPLAY "O resultado é negativo" END-IF	⇔ a	IF (Num1 * 10 / 50) – 10 LESS THAN ZERO DISPLAY "O resultado é negativo" END-IF

Condições de classe

A condição de classe é usada para testes, onde se deseja saber se uma variável é formada ou não por um tipo particular de dados.

NUMERIC Numérico, carateres de 0 a 9 (sinalizado ou não, como decimal ou inteiro).
ALPHABETIC Alfabético, carateres de A - Z, de a – z e espaços.
ALPHABETIC-UPPER Alfabético, carateres de A - Z, e espaços.
ALPHABETIC-LOWER Alfabético, carateres de a - z, e espaços.

Síntaxe:

```
IF VARIAVEL IS [NOT] {NUMERIC} {ALPHABETIC} {ALPHABETIC-UPPER}
{ALPHABETIC-LOWER} {nome-de classe}
```

Exemplo:

```
IF NOME IS NOT ALPHABETIC-UPPER
    DISPLAY "INSIRA APENAS DE A – Z E ESPAÇOS" END-IF
```

Nomes condicionais

Os nomes das condições são às vezes chamados de nível 88 porque são criados na DIVISÃO DE DADOS usando o nível especial número 88 (já referidos na ficha 1 página 15, rever também exemplos).

REGRAS

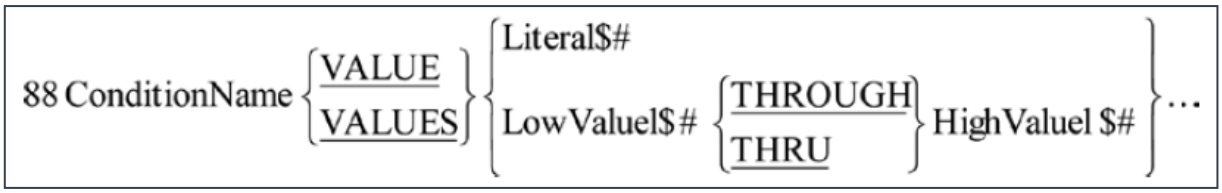
Os nomes de condição são sempre associados a um item de dados específico e são definidos imediatamente após a definição desse item de dados. Um nome de condição pode estar associado a um item de dados de grupo e dados elementares, ou até mesmo ao elemento de uma tabela. O nome da condição é definido automaticamente como verdadeiro ou falso no momento em que o valor de seu item de dado associado se alterar.

Nomes de condição, só podem ter o valor verdadeiro ou falso. Se um nome de condição não está definido como true, é porque está definido como false.

Quando a cláusula VALUE é usada com nomes de condição, ela não atribui um valor. Em vez disso, identifica o(s) valor(es) que, se encontrados no item de dados associado, que tornará o nome da condição verdadeiro.

Para especificar uma lista de valores, as entradas são listadas após a palavra-chave VALUE. As entradas da lista podem ser separadas por vírgulas ou espaços, mas deve sempre terminar com um ponto final.

SINTAXE (relembrar):



Exemplos:

1. O nome da condição CityIsLimerick foi associado ao CityCode para que, se CityCode contiver o valor 2 (listado na cláusula CityIsLimerick VALUE), o nome da condição será automaticamente definido como verdadeiro.

   Neste fragmento do programa, DISPLAY e ACCEPT obtêm um código de cidade do utilizador. Assim que o valor em CityCode é alterado, o nome da condição CityIsLimerick será definido como verdadeiro ou falso, de acordo com o valor atribuído a CityCode.

```
…
WORKING-STORIDADE SECTION.
01 CityCode PIC 9 VALUE ZERO.
88 CityIsLimerick VALUE 2.     *>irá corresponder a true
PROCEDURE DIVISION.
…

DISPLAY "Enter a city code (1-6): "
      ACCEPT CityCode

IF CityIsLimerick
      DISPLAY "Hey, we're home."
END-IF
…
```

2. Neste exemplo, vários nomes de condição foram associados a CityCode. Cada nome de condição é definido como verdadeiro quando CityCode tomar o valor (fornecido pelo utilizador), correspondente ao seu valor atribuído na cláusula VALUE.

```
...
WORKING-STORIDADE SECTION.
01 CityCode PIC 9 VALUE ZERO.
88 CityIsDublin VALUE 1.
88 CityIsLimerick VALUE 2.
88 CityIsCork VALUE 3.
88 CityIsGalway VALUE 4.
88 CityIsSligo VALUE 5.
88 CityIsWaterford VALUE 6.
PROCEDURE DIVISION.
...

DISPLAY "Enter a city code (1-6): "
ACCEPT CityCode
IF CityIsLimerick
   DISPLAY "Hey, we're home."
END-IF
IF CityIsDublin
   DISPLAY "Hey, we're in the capital."
END-IF
...
```

3. Nomes de Condição Múltipla (IF…ELSE) com múltiplos valores (THRU) - ver exemplo 2, da página 16, da ficha 1). NOTA: A lista de valores associada a um nome de condição pode ser constituída por dados alfanuméricos!

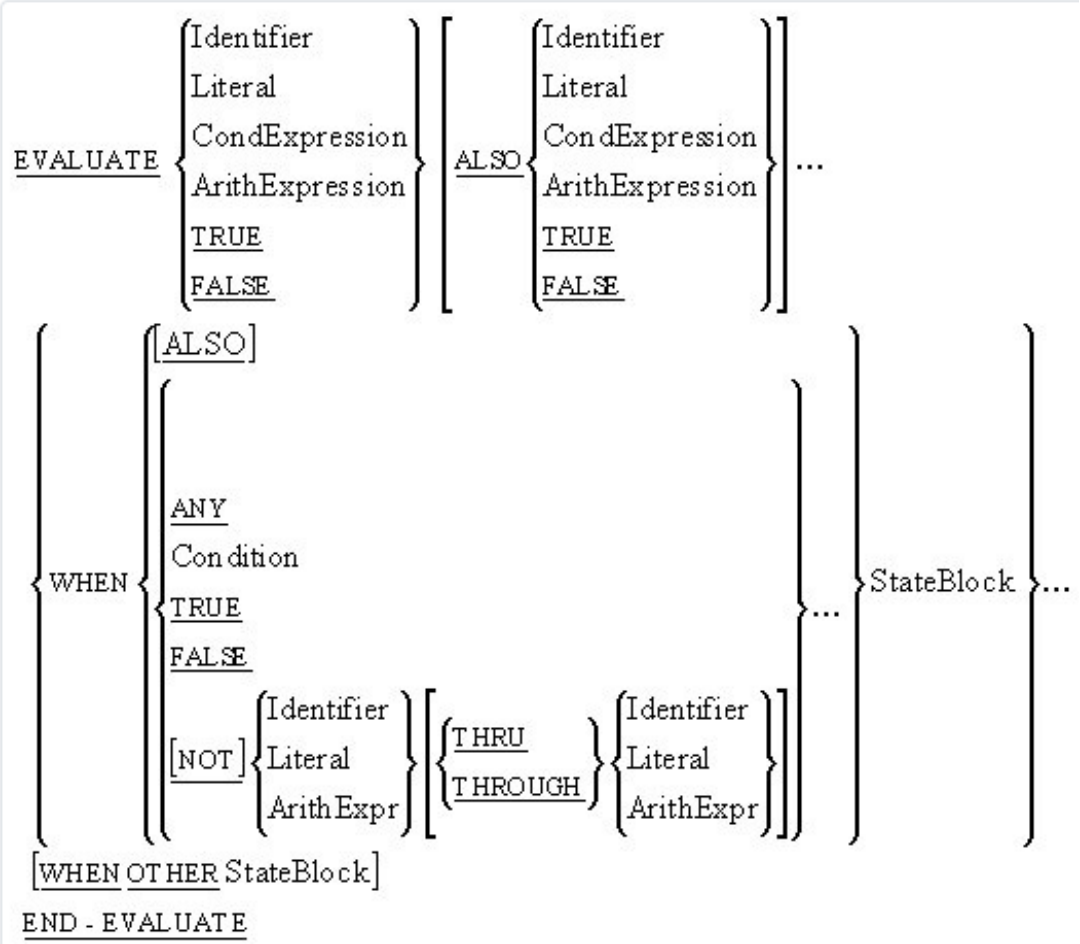
Verbo EVALUATE

Usado avaliar condições para a determinação de um resultado. Quando a sua sintaxe é usada na sua versão simplificada a cláusula WHEN equivale à estrutura SWITCH/CASE noutras linguagens de programação, representando uma alternativa mais simples para a criação de um encadeamento de instruções "IF". É, no entanto, mais poderosa do que a estrutura SWITCH e não se limita à avaliação de valores literais.

TIPOS relacionados com o verbo EVALUATE:

1. EVALUATE simples;
2. EVALUATE TRUE;
3. EVALUATE com THRU;
4. EVALUATE com múltiplas condições WHEN;
5. EVALUATE com múltiplas condições.

SINTAXE:



SINTAXE SIMPLIFICADA (EVALUATE WHEN):

```
EVALUATE <Expressãol>
        WHEN <condição 1>
                Instrução(ões)
        WHEN <condição 2>
                Instrução(ões)
                .
                .
                .
        WHEN <condição N>
        [WHEN other
                Instrução(ões)other]
END-EVALUATE.
```

Exemplos:

1.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EVAL02.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 IDADE        PIC 9(002).
77 GENERO       PIC X(001).
PROCEDURE DIVISION.
    DISPLAY "IDADE: "
    ACCEPT IDADE
    DISPLAY "GENERO: "
    ACCEPT GENERO
    MOVE FUNCTION UPPER-CASE (GENERO) TO GENERO
       EVALUATE IDADE ALSO GENERO
           WHEN < 18 ALSO "F"
            DISPLAY "MULHER MENOR DE IDADE"
           WHEN >= 18 ALSO "F"
            DISPLAY "MULHER MAIOR DE IDADE"
           WHEN < 18 ALSO "M"
            DISPLAY "HOMEM MENOR DE IDADE"
           WHEN >= 18 ALSO "M"
            DISPLAY "HOMEM MAIOR DE IDADE "
           WHEN OTHER
            DISPLAY "ERRO"
        END-EVALUATE
GOBACK.
```

Em alternativa à função UPPER-CASE
Poder-se-á usar na clausula WHEN
…ALSO "F" OR "f"

Saber mais…

O comando **FUNCTION UPPER-CASE (variável)**, transforma todo conteúdo de uma variável para maiúscula.

O comando **FUNCTION LOWER-CASE (variável)**, transforma todo conteúdo de uma variável para minúscula.

2.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. NOTAS.
DATA-DIVISION.
WORKING-STORAGE SECTION.
77 NOTA-ALUNO    PIC 9(003).
PROCEDURE DIVISION.
    DISPLAY “NOTA: “
    ACCEPT NOTA-ALUNO
       EVALUATE NOTA-ALUNO
           WHEN 0 THRU 60
             DISPLAY “Fraco”
           WHEN 61 THRU 99
             DISPLAY “Não Satisfaz”
           WHEN 100 THRU 139
             DISPLAY “Satisfaz”
           WHEN 140 THRU 179
               DISPLAY “Bom “
          WHEN 180 THRU 200
               DISPLAY “Muito bom “
           WHEN OTHER
             DISPLAY “Nota inválida (entre 0 e 200)”
       END-EVALUATE
GOBACK.
```

3.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. AdmissaoParque.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Idade   PIC 999 VALUE ZERO.
88 Infantil        VALUE 0 THRU 5.
88 Crianca         VALUE 6 THRU 12.
88 Visitante       VALUE 13 THRU 64.
88 Reformado       VALUE 66 THRU 105.
01 Altura   PIC 999 VALUE ZERO.
01 Admissao PIC 99.99$.
PROCEDURE DIVISION.
NovaIdade.
DISPLAY "Insira a idade: "
ACCEPT Idade
IF Idade>105 OR Idade<0
       GO TO NovaIdade
END-IF
DISPLAY "Insira a altura: "
ACCEPT Altura
EVALUATE TRUE      ALSO  TRUE
  WHEN  Infantil  ALSO  ANY           MOVE  0 TO Admissao
  WHEN  Crianca   ALSO  Altura >= 48  MOVE 15 TO Admissao
  WHEN  Crianca   ALSO  Altura < 48   MOVE 10 TO Admissao
  WHEN  Visitante ALSO  Altura >= 48  MOVE 25 TO Admissao
  WHEN  Visitante ALSO  Altura < 48   MOVE 18 TO Admissao
  WHEN  Reformado ALSO  ANY           MOVE 10 TO Admissao
END-EVALUATE
DISPLAY "Encargo de Admissao: " Admissao WITH NO ADVANCING
 STOP RUN.
```

Output1:
Insira a idade:
7
Insira a altura:
45
Encargo de Admissao: 10.00$

Output2:
Insira a idade:
9
Insira a altura:
52
Encargo de Admissao: 15.00$

Output3:
Insira a idade:
31
Insira a altura:
55
Encargo de Admissao: 25.00$

Estruturas Condicionais (iterações):

Na linguagem **C** eram usadas as construções **while** e **do..while** para a iteração pré-teste e pós-teste, e o ciclo **for** para contar as iterações.

O COBOL suporta todos estes diferentes tipos de iteração, mas possui apenas uma construção de iteração: o verbo **PERFORM**.

AS iterações pré-teste e pós-teste são suportadas pelo **PERFORM WITH TEST BEFORE** e **PERFORM WITH TEST AFTER.**

A contagem de iterações é suportada pelo **PERFORM…VARYING**.

O COBOL ainda apresenta variações que não são encontradas noutras linguagens. O PERFORM..VARYING, por exemplo, pode ter mais de um contador, e pode apresentar ambas as variações pré-teste e pós-teste.

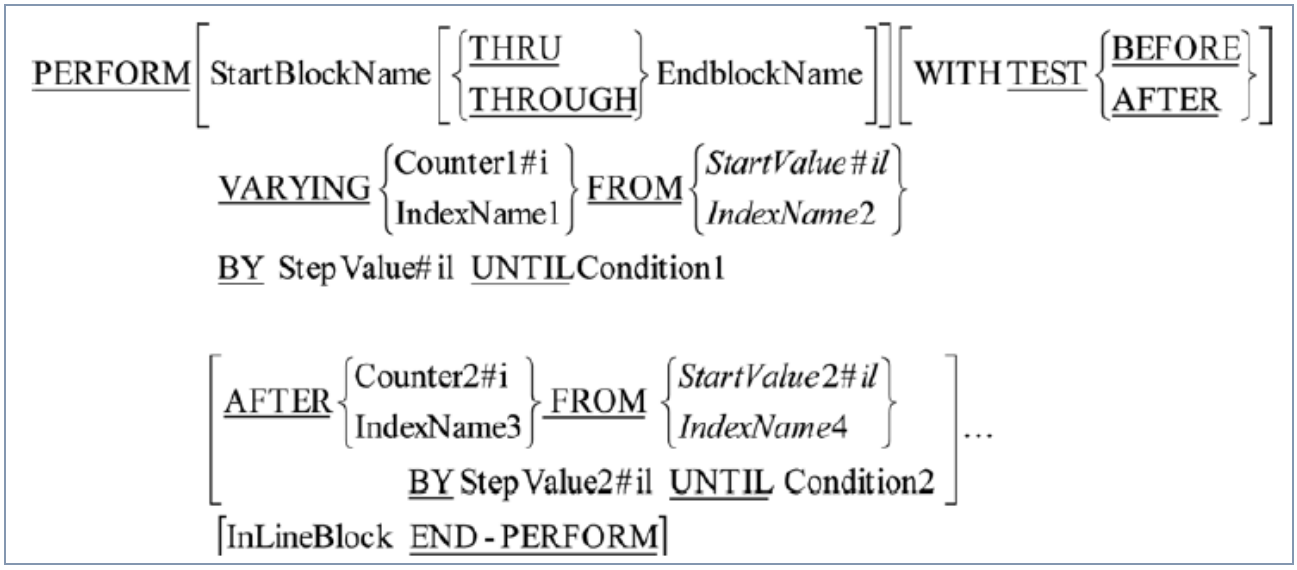
Enquanto na maioria das linguagens o alvo de loop é um bloco de código embutido- inline (isto é um bloco que está entre o início e o fim do ciclo), em COBOL ele pode também ser um bloco de código fora dos limites do ciclo, isto é, o bloco de instruções associado ao ciclo pode ser executado fora dos seus limites - outline.

Tabela de comparação entre a sintaxe das iterações entre linguagens de programação

	C, C++, Java	Cobol
Pré-teste	while {}	PERFORM WITH TEST BEFORE UNTIL
Pós-teste	do {} while	PERFORM WITH TEST AFTER UNTIL
Contagem	for	PERFORM..VARYING..UNTIL

PERFORM..VARYING

SINTAXE:



SINTAXE SIMPLIFICADA:

PERFORM VARYING <variável> FROM <valor1> BY <valor2> UNTIL <condição>

Exemplo:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ciclo.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 WS-A PIC 9 VALUE 0.
PROCEDURE DIVISION.
  PERFORM B-PARA VARYING WS-A FROM 1 BY 1 UNTIL WS-A=5
STOP RUN.
  B-PARA.
  DISPLAY 'IN B-PARA ' WS-A.
```

Output:

IN B-PARA 1

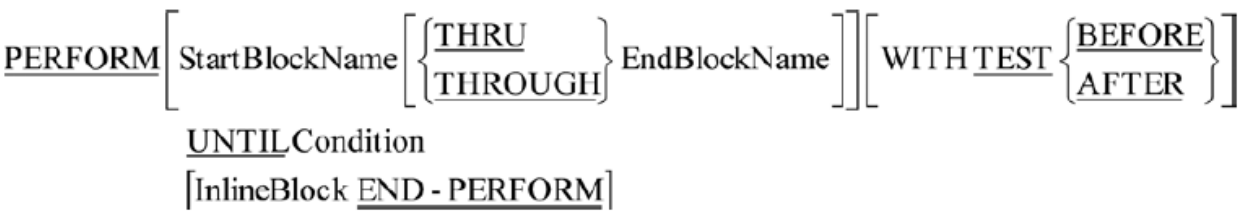
IN B-PARA 2

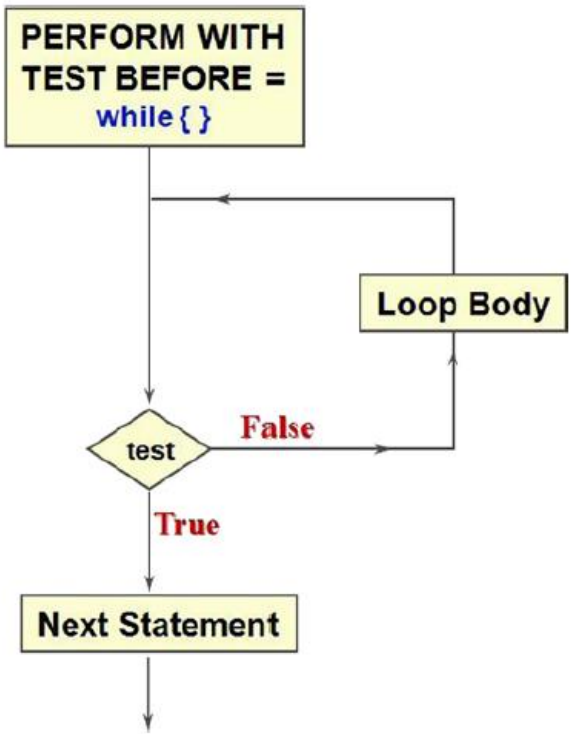
IN B-PARA 3

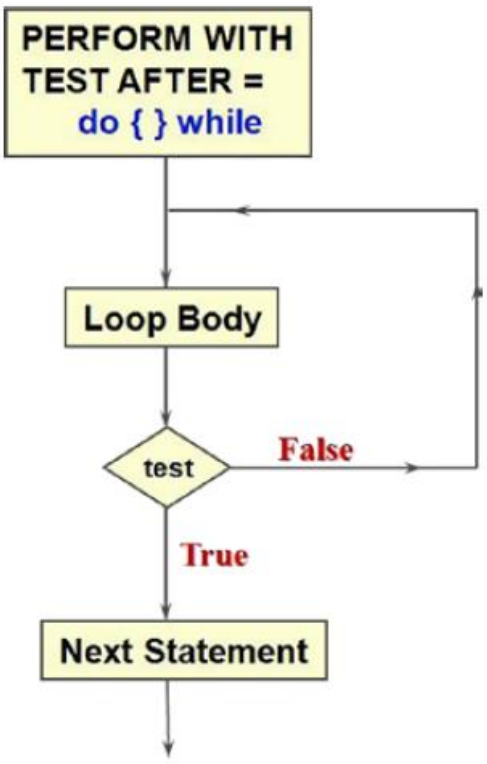
IN B-PARA 4

***>VER PÁG. 2 DA FICHA 3**

PERFORM UNTIL







Exemplo (PERFORM WITH TEST BEFORE UNTIL):

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TestBEfore.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 NOME PIC X(015) VALUE '.'.

PROCEDURE DIVISION.
PERFORM WITH TEST BEFORE UNTIL NOME = SPACES
DISPLAY "COMO TE CHAMAS NOME: "
ACCEPT NOME
DISPLAY NOME
END-PERFORM.
GOBACK.
```

```
Output(inseridos Ana, Rui e espaço):
COMO TE CHAMAS NOME:
Ana
Ana
COMO TE CHAMAS NOME:
Rui
Rui
COMO TE CHAMAS NOME:
```

Exemplo (PERFORM WITH TEST AFTER UNTIL):

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TesteDepois.

DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 WS-CNT PIC 9(1) VALUE 0.

PROCEDURE DIVISION.
 PERFORM B-PARA WITH TEST AFTER UNTIL WS-CNT>3.
STOP RUN.
   B-PARA.
      DISPLAY 'WS-CNT: 'WS-CNT.
      ADD 1 TO WS-CNT.
```

```
Output:
WS-CNT: 0
WS-CNT: 1
WS-CNT: 2
WS-CNT: 3
```

PERFORM TIMES

Nessa estrutura o parágrafo será executado "n" vezes.

Exemplo (A execução do programa é desviada para a marca B-PARA onde o parágrafo é executado 3 vezes após o qual a execução do programa é devolvida à instrução que se segue ao PERFORM – STOP RUN, terminando por isso a execução do programa):

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
PROCEDURE DIVISION.
 PERFORM B-PARA 3 TIMES.
  STOP RUN.
  B-PARA.
      DISPLAY 'IN B-PARA'.
```

```
PERFORM PARAGRAPH1 THRU PARAGRAPH2
```

Exemplo:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UsarTHRU.
PROCEDURE DIVISION.
   PERFORM DISPLAY 'IN A-PARA'
   END-PERFORM.
   PERFORM C-PARA THRU E-PARA.
   DISPLAY 'IN B-PARA'.
   STOP RUN.
   C-PARA.
   DISPLAY 'IN C-PARA'.
   D-PARA.
   DISPLAY 'IN D-PARA'.
   E-PARA.
   DISPLAY 'IN E-PARA'.
```

Output:

IN A-PARA

IN C-PARA

IN D-PARA

IN E-PARA

IN B-PARA

RESUMO:

Inline perform	Outline perform
UNTIL: PERFORM UNTIL Condition Statements END-PERFORM.	**UNTIL:** PERFORM {Paragraph/Section} [{THROGH/THRU} {Paragraph/Section} UNTIL Condition
WITH TEST BEFORE: PERFORM WITH TEST BEFORE UNTIL Condition Statements END-PERFORM.	**WITH TEST BEFORE:** PERFORM {Paragraph/Section} [{THROGH/THRU} {Paragraph/Section} WITH TEST BEFORE UNTIL Condition
WITH TEST AFTER: PERFORM WITH TEST AFTER UNTIL Condition Statements END-PERFORM.	**WITH TEST AFTER:** PERFORM {Paragraph/Section} [{THROGH/THRU} {Paragraph/Section} WITH TEST AFTER UNTIL Condition.

O comando CONTINUE

É usado para especificar que não há instruções a serem executadas.

No exemplo seguinte é exibida a percentagem de estudantes que conseguiram aprovação em todos os conteúdos ou é exibida a quantidade de conteúdos aos quais não obtiveram aprovação.
Se o valor obtido pelo estudante estiver abaixo de 35, enquanto são adicionadas as classificações para o cálculo da percentagem, o comando CONTINUE irá simplesmente passar o controlo à iteração seguinte sem interromper o processo.

Exemplo: ...

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STD-DET OCCURES 6 TIMES INDEXED BY STD-INDEX.
   05 STD-MARKS              PIC 9(03).
01 TOTAL-MARKS               PIC 9(03) VALUE ZERO.
01 STD-PERCENT               PIC 9(03).9(02).
01 I                         PIC 9(01).
01 J                         PIC 9(01) VALUE ZERO.
PROCEDURE DIVISION.
    MOVE ZEROES     TO TOTAL-MARKS.
    PERFORM VARYING I FROM 1 BY 1
      UNTIL I > 6
      SET STD-INDEX TO 1
      ACCEPT STD-MARKS (STD-INDEX)
      IF STD-MARKS (STD-INDEX) < 35
         CONTINUE
      ELSE
         ADD STD-MARKS (STD-INDEX) TO TOTAL-MARKS
         SET STD-INDEX DOWN BY 1
COMPUTE J = J + 1
      END-IF
    END-PERFORM.
    IF J < 6
       COMPUTE J = I -(J + 1)
       DISPLAY 'STUDENT FAILED IN 'J 'SUBJECTS
    ELSE
       COMPUTE STD-PERCENT = TOTAL-MARKS/6
       DISPLAY 'STUDENT PERCENTAGE : ' STD-PERCENT.
    END-IF.
    STOP RUN.
```

Exercícios:

1. Escreve uma expressão com a instrução IF que utilize o verbo SET para atribuir o valor TRUE ao nome InvalidoCodigo se DeptCodigo tiver o valor 1, 6 ou 8.

2. Supõe que a variável DeptCodigo, apresentada na questão anterior é descrita como:

01 DeptCodigo PIC 9.

Escreve uma condição de nível 88 com o nome InvalidoCodigo que tem, automaticamente, o valor true quando a declaração ACCEPT DeptCodigo aceita qualquer valor diferente de 1, 6 ou 8.

3. Considere os cinco grupos de fragmento de declarações IF, apresentados a seguir. Indique quais são os grupos cujas declarações produzem o mesmo efeito (no sentido de que avaliam verdadeiro ou falso).

IF Num1 = 1 OR Num1 NOT = 1 …
IF NOT (Num1 = 1 AND Num1 = 2) …

IF TransCode IS NOT = 3 OR Total NOT > 2550 …
IF NOT (TransCode IS = 3 OR Total NOT > 2550) …

IF Num1 = 31 OR Num2 = 12 AND Num3 = 23 or Num4 = 6 …
IF (Num1 = 31 OR (Num2 = 12 AND Num3 = 23)) or Num4 = 6 …

IF Num1 = 15 OR Num1 = 12 OR Num1 = 7 AND City = “Lisboa” …
IF (Num1 = 15 OR Num1 = 12 OR Num1 = 7) AND City = “Lisboa” …

IF (Num1 = 1 OR Num2 = 2) AND (Num2 = 6 or Num2 = 8) …
IF Num1 = 1 OR Num2 = 2 AND Num2 = 6 or Num2 = 8 …

4. Elabora um programa, em COBOL, que peça ao utilizador os dois lados de um retângulo e calcule a sua área. Se a área for menor do que 100 exiba “Área pequena” e caso contrário “Área grande”.

5. Elabora um programa, em COBOL, que aceite um número dado pelo utilizador e incremente-o de 2 em 2 exibindo os resultados no ecrã até que este número seja maior do que 50.

6. Escreve um programa que analise se um número é par ou ímpar.

7. Escreve um programa que determine as classes de nadadores, de acordo com a sua idade:

   a. Bebé: até 4 anos
   b. Infantil A: de 5 a 7
   c. Infantil B: de 8 a 11
   d. Juvenil A: de 12 a 13
   e. Juvenil B: de 14 a 17
   f. Adulto: de 18 a 64
   g. Sénior: maior de 65

8. Refaz o exercício 5, da ficha n.º 3, mas agora considerando que o utilizador pode escolher a operação a efetuar (utiliza a declaração IF).

Bibliografia/webgrafia:
http://www.mainframestechhelp.com/tutorials/cobol/
Beginning COBOL for Programming, Michael Coughlan, Editora Apress
Mainframe Apostila de Cobol, G &P Treinamentos em http://www.csis.ul.ie/cobol/
https://www.apostilando.com/apostila/2962/manual-pratico-de-programacao-em-cobol
https://www.tutorialspoint.com/cobol/
Cobol Estruturado, Lawrence R. Newcomer, Schaum-McGraw-Hill