1. INTRODUÇÃO

A linguagem COBOL (**CO**mmon **B**usiness **O**riented **L**anguage – Linguagem Comum Orientada aos Negócios) foi desenvolvida no ano de 1959, por um grupo chamado Comitê CODASl (COference on DAta SYstem Languages). Esse grupo era formado por fabricantes e grupos de usuários, cujo objetivo era a criação de uma linguagem comum de programação, visando o uso em aplicações comerciais.

O COBOL For MVS & VM (versão mais recente do COBOL IBM) foi projetado de acordo com as especificações do padrão da instituição ANSI (American National Standards Institute – Instituto Nacional Americano de Padronizações) de 1985.

2. ESTRUTURA DA LINGUAGEM

A Linguagem COBOL é estruturada em :

- ☞ Divisões
- ☞ Seções
- ☞ Parágrafos
- ☞ Sentenças
  - * Cláusulas (nas três primeiras divisões)
  - * Comandos (na Procedure Division)

2.1. Divisões

Um programa Cobol é estruturado em quatro divisões de codificação obrigatória. Mesmo que uma delas não se faça necessário, pelo menos o seu nome deverá ser digitado.

- ☞ IDENTIFICATION DIVISION ⇨Informações sobre o programa
- ☞ ENVIRONMENT DIVISION ⇨ Definição do ambiente físico
- ☞ DATA DIVISION ⇨ Definição da estrutura lógica dos arquivos e áreas de trabalho
- ☞ PROCEDURE DIVISION ⇨ Procedimentos a serem executados

2.2. Seções

- ☞ Definidas na ENVIRONMENT DIVISION e DATA DIVISION conforme requeridas.
- ☞ Definidas na PROCEDURE DIVISION para especificar a segmentação do programa.

2.3. Parágrafos

Na PROCEDURE DIVISION são utilizados para agrupar sentenças, permitindo a alteração do fluxo lógico. Devem terminar com um ponto.

2.4. Sentenças

Formadas por uma ou mais cláusulas ou comandos. São terminadas por um ponto.

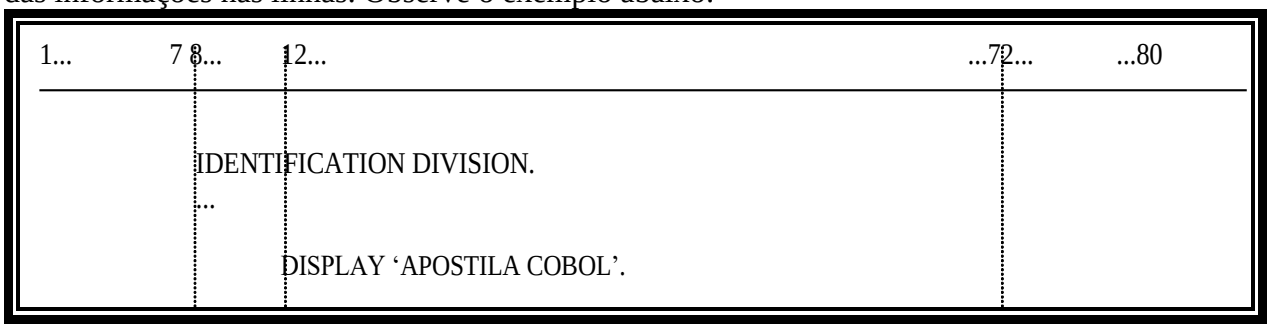
2.5. Pontuação e codificação

Um programa Cobol segue algumas regras que devem ser respeitadas durante a codificação dos programas:

- ☞ Ao final de cada nome de divisão, seção parágrafo deve ser colocado um ponto;
- ☞ Deve haver pelo menos um espaço entre cada palavra do programa.
- ☞ Quando na mesma linha, deve ser deixado pelo menos um espaço entre o ponto e o início da próxima sentença.
- ☞ Todas as palavras (instruções, cláusulas, etc.) devem ser digitadas em letras maiúsculas.

2.6. Folha de codificação Cobol

Ao se digitar um programa Cobol, devemos observar alguns aspectos relativos a disposição das informações nas linhas. Observe o exemplo abaixo:



Codificação das colunas:

- **1 a 6** ⇨ Utilizadas para numerar as linhas. Será de uso opcional.
- **Coluna 7** ⇨ Pode conter um hífen (-) para indicar que a ' é continuação da anterior, um asterisco (*) para indicar uma linha de comentário ou uma barra (/) para indicar um salto de página na listagem de compilação.
- **8 a 11** ⇨ Representam a margem A, onde começam os nomes de divisão, seção e parágrafos.
- **12 a 72** ⇨ Representam a margem B, onde estarão todos os comandos.
- **73 a 80** ⇨ Pode ser utilizada para comentários quaisquer, pois são ignoradas pelo compilador.

3. COMPONENTES DA LINGUAGEM

3.1. Palavras

Uma palavra em COBOL é um conjunto de caracteres de até 30 bytes de comprimento. Pode ser definida pelo programador ou ser um nome do sistema. Os identificadores e verbos de comando também são considerados palavras.

As palavras que tem um significado específico para o compilador COBOL são chamadas reservadas, não podendo dessa forma ser utilizadas para outros fins pelo programador. Exemplos de palavras reservadas: IDENTIFICATION, LABEL, RECORD, READ, WRITE.

As palavras reservadas podem ser dividias em constantes figurativas, palavras opcionais e palavras-chave, conforme descrição a seguir:

Constantes Figurativas

São palavras associadas a um valor em particular.

- ALL 'literal' ⇨ uma ou mais ocorrências de literal
- ZERO, ZEROS, ZEROES ⇨ valor zero
- SPACE, SPACES ⇨ brancos
- QUOTE, QUOTES ⇨ apóstrofe
- HIGH-VALUE, HIGH-VALUES ⇨ maior valor (HEXA FF)
- LOW-VALUE, LOW-VALUES ⇨ menor valor (HEXA 00)

Palavras opcionais

O uso destas palavras é opcional.

Exemplo

IF WS-CAMPO IS GREATER THAN 10 = IF WS-CAMPO GREATER 10

Logo, **IS** e **THAN** são opcionais.

Palavras-chave

Determinam a função da cláusula ou comando.

Exemplo

ADD, READ, IF

Regras para formação de nomes de palavras

- ☞ Pode conter apenas caracteres alfabéticos (A-Z), numéricos (0-9) e hifen.
- ☞ Devem começar com um caracter alfabético. Isso implica na existência de pelo menus um caractere alfabético na palavra.
- ☞ Não podem terminar com hífen.
- ☞ Deve haver pelo menos um espaço em branco entre as palavras.
- ☞ O tamanho máximo para palavras no COBOL é de 30 caracteres.

3.2. Nomes

São atribuídos pelo programador. Palavras reservadas não podem ser usadas como nomes.

Tipos :

- ☞ Nomes de dados
- ☞ Nomes de registro
- ☞ Nomes de arquivo
- ☞ Nomes de parágrafo
- ☞ Nomes de seção

3.3. Literais

Conjuntos de caracteres cujo valor é especificado. Existem dois tipos: as literais numéricas e as alfanuméricas.

Numéricas ⇨ são compostas por dígitos de 0 a 9 e/ou um sinal (+ ou -) e/ou um ponto decimal (vírgula), contém no máximo 18 dígitos.

Alfanuméricas ⇨ Contém todo tipo de caractere EBCDIC, num tamanho máximo de 120 caracteres, devendo estar entre apóstrofes. Um hífen deve ser colocado na coluna 7 para a continuação da literal na linha seguinte.

3.4. Comentários

Não são considerados pelo compilador, servindo somente para documentação.

REMARKS ⇨Parágrafo da IDENTIFICATION DIVISION

* na coluna 7 ⇨Linha toda como comentário

/ na coluna 7 ⇨ Salta uma página considerando o comentário

Pode-se ainda utilizar o recurso de comentários para informações que auxiliem na compreensão do programa. Para auxiliar na estética do programa, podemos utilizar as instruções SKIP e EJECT na coluna 12.

EJECT ⇨ Salta página

SKIP1 ⇨ Salta 1 linha

SKIP2 ⇨ Salta duas linhas

SKIP3 ⇨ Salta três linhas

Tais instruções serão executadas diretamente pelo compilador, refletindo seus resultados na listagem de compilação.

3.5. Palavras utilizadas junto a comandos

Condição

- NUMERIC
- ALPHABETIC
- POSITIVE
- NEGATIVE
- ZERO(ES)

Relação

- GREATER (>)
- LESS (<)
- EQUAL (=)
- NOT GREATER (<=)
- NOT LESS (>=)

Conecção

- AND
- OR

Negação

- NOT

Operadores

- + ⇨ Adição
- - ⇨ Subtração
- * ⇨ Multiplicação
- / ⇨Divisão
- ** ⇨ Exponenciação

3.6. Registradores Especiais

CURRENT-DATE
Campo de 8 bytes alfanuméricos
Formato : MM/DD/AA
Acesso : MOVE

TIME-OF-DAY
Campo de 6 bytes zonados
Formato : HHMMSS
Acesso : MOVE

DATE
Campo de 6 bytes zonados
Formato : AAMMDD
Acesso : ACCEPT

DAY

Campo de 5 bytes zonados
Formato : AADDD
Acesso : ACCEPT

TIME
Campo de 8 bytes zonados
Formato : HHMMSSCC
Acesso : ACCEPT

4. IDENTIFICATION DIVISION

Divisão de identificação do programa. Segue o formato:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. <NOME-DO-PROGRAMA>.
AUTHOR. <NOME-DO-PROGRAMADOR>.
```

5. ENVIRONMENT DIVISION

É a divisão onde configuramos o ambiente de trabalho do programa. Encontra-se dividida em duas seções. A configuration section (seção de configuração), onde configuramos parâmetros de ambiente do programa e a input-output section (seção de entrada e saída), onde selecionamos os arquivos que serão utilizados pelo programa. A estrutura é:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
        DECIMAL-POINT IS COMMA.
INPUT-OUTPUT SECTION.
SELECT <NOME-ARQUIVO> ASSIGN TO <DISK|TAPE|PRINTER>
        ORGANIZATION IS
                        <SEQUENTIAL|LINE SEQUENTIAL|INDEXED|RANDOM>
        ACCESS MODE IS <SEQUENTIAL|RANDOM|DYNAMIC>
        RECORD KEY IS <NOME-DE-CAMPO>
        ALTERNATE RECORD KEY IS <NOME-DE-CAMPO>
        FILE STATUS IS <VARIÁVEL>
```

6. DATA DIVISION

Seção na qual definimos todos os dados com os quais trabalharemos no programa. Não é possível utilizar qualquer variável, campo, registro ou arquivo sem que os mesmos não sejam primeiramente definidos nesta divisão.

6.1. Working-Storage Section

Na seção de armazenamento e trabalho, iremos definir as variáveis que serão utilizadas pelo nosso programa. Todas, sem exceção devem ser declaradas. A cláusula VALUE poderá inicializar uma variável já em sua criação.

Estrutura

```
DATA DIVISION.

WORKING-STORAGE SECTION.
NÍVEIS (01, 02-49, 77)        <IDENTIFICADOR> PICTURE      VALUE <VALOR>
```

6.2. File Section

É necessário que os arquivos selecionados na divisão de ambiente sejam declarados nesta seção.

Forma geral

```
FILE SECTION.
FD      <NOME-ARQUIVO>
        BLOCK CONTAIS <N> <CHARACTERS|RECORDS>
        RECORD CONTAINS <N> CHARACTERS
        LABEL RECORD IS <STANDARD|OMITTED>
        DATA RECORD IS <NOME-REGISTRO>
        RECORDING MODE IS <F|U|V|S>  (fix or variable)
NOME-REGISTRO
<DEFINIÇÕES DOS CAMPOS>
```

Onde:

- NOME-ARQUIVO ⇨ é o definido na cláusula SELECT da ENVIRONMENT DIVISION;
- BLOCK ⇨ determina o tamanho do bloco (registro físico) do arquivo, em número de registros ou caracteres. O bloco pode ter tamanho variável. Caso esta cláusula seja omitida, o arquivo será considerado desblocado;
- RECORD ⇨ determina o tamanho do registro (registro lógico) do arquivo;
- LABEL RECORD ⇨ é padrão (standard) no caso de arquivos em disco ou tape e omitido (omitted) no caso de arquivos de impressão ou em cartões que não possuem label;
- DATA RECORD ⇨ nome de dado de nível 01 utilizado posteriormente para identificar o registro ⇨
- RECORDING ⇨ determina o formato de registro que o arquivo possui. Basicamente, podemos declarar os seguintes formatos
  - F ⇨ todos os registros do arquivo são do mesmo tamanho e cada um está inteiramente contido em um bloco;
  - U ⇨ os registros podem ser fixos ou variáveis. Contudo, existe apenas um registro por bloco. Não existem campos de descrições de comprimento de registro ou bloco;
  - V ⇨ os registros podem ser fixos ou variáveis, inteiramente contidos em um bloco. Os blocos podem conter mais de um registro. Cada registro inclui a descrição dos campos de comprimento do registro e cada bloco inclui a descrição do tamanho do bloco. Este campos não são descritos na DATA DIVISION. Sua previsão é feita automaticamente e não estão disponíveis para o programador;

S ⇨ os registros podem ser fixos ou variáveis, podendo até mesmo ser maiores que o bloco. Sem um registro for maior que o espaço disponível em um bloco (SPANNED), um segmento do registro será escrito de modo a preencher o bloco. O resto é armazenado no próximo bloco (se necessário, será usado mais de um).

6.3. Linkage Section

É utilizada para definir as áreas de comunicação (commom area) entre programas. Tem formato parecido com a WORKING-STORAGE section com a diferença que os dados definidos serão utilizados por mais de um programa. Será vista com maiores detalhes no item comunicação entre programas.

6.4. Números de nível

6.4.1. Nível 01 – itens de grupo

São identificadores que agrupam variáveis. Podem ser utilizados para simplesmente melhorar o entendimento do programa ou para fins específicos, como grupo de valores que podem ser acessados individualmente ou de uma só vez. São indicados pelo número de nível 01.

Exemplo:

```
WORKING-STORAGE SECTION.
01      DATA-SISTEMA.
```

6.4.2. Níveis de 02 a 49 – itens elementares

São aqueles que estarão dentro dos itens de grupo, representados pelos números de nível de 02 a 49. São as variáveis simples. Pode-se utilizar o recurso de quebra de nível para dividir essas variáveis em outras e utilizar individualmente seus valores.

Exemplo:

```
WORKING-STORAGE SECTION.
DATA-SISTEMA.
05      WS-DIA      PIC    99      VALUE 0.
05      WS-MES      PIC    99      VALUE 0.
05      WS-ANO      PIC    99      VALUE 0.
```

6.4.3. Nível 77 – itens independentes

São idênticos aos itens elementares porém, não aceitam quebra de nível e estão fora de um grupo de variáveis.

6.4.4. Nível 66 – Cláusula RENAMES

Esse nível identifica que os itens devem conter a cláusula RENAMES.

6.4.5. Nível 88 – nomes de condições

São definidos no nível 88 e representam uma determinada condição conforme o valor de uma variável.

Exemplo:

```
02      CONFIRMACAO     PIC     X       VALUE ' '.
        88      CONFIRMADO      VALUE 'S'.
```

6.5. Tratamento de saídas

Os dados em Cobol podem ser tratados como informações que serão processadas apenas em memória ou que serão destinadas única e exclusivamente para exibição ao usuário. Para fazer tal configurações, faremos uso de PICTURES, que são características dos itens elementares e definem seu tipo de dado, tamanho e a maneira como serão apresentados em um dispositivo de saída.

Para valores numéricos, em pictures de exibição, utilizaremos principalmente os caracteres Z (que suprime a exibição de zeros não significativos) e , (vírgula, que exibe uma vírgula decimal – requer também a declaração DECIMAL-POINT IS COMMA).

Símbolos para PICTURES

X ..... Caractere alfanumérico do conjunto ASCII.
9 ..... Algarismo de um número (máx. 18 por número).
0 ..... Insere zero na posição indicada.
S ..... Sinal aritmético do número.
V ..... Posição da vírgula (ou ponto) decimal.
Z ..... Supressão de zeros não significativos.
B ..... Insere espaço(s) em branco na posição desejada.
A ..... Caractere alfabético ou espaço em branco.
, ... Insere vírgula na posição indicada, podendo haver apenas uma vírgula no caso de DECIMAL-POINT IS COMMA.
. ..... Insere ponto na posição indicada, podendo haver mais de um ponto no caso de DECIMAL-POINT IS COMMA.
/ ..... Insere uma barra na posição indicada.
- ..... Colocado à direita da PICTURE, insere um sinal de menos após o número, se este for negativo. Colocado à esquerda, adquire características flutuantes, sendo o sinal impresso antes do número, se este for negativo.
+..... Colocado à direita da PICTURE, insere um sinal de mais após o número, se este for positivo. Colocado à esquerda, adquire características flutuantes, sendo o sinal impresso antes do número, se este for positivo.
* ..... Substitui zeros não significativos por asteriscos.
$ ..... Insere um cifrão antes do primeiro algarismo significativo e supre os zeros não significativos.
CR ..... Colocado à esquerda, nas duas últimas posições, imprime um sinal de crédito (CR) à direita do número, se este for positivo.
DB ..... Colocado à esquerda, nas duas últimas posições, imprime um sinal de débito (DB) à direita do número, se este for negativo.

P ..... Fator de escala para especificar a posição do ponto decimal, representando uma ou mais posições, inteiras ou fracionárias, às quais não é reservado espaço em memória e cujo valor será sempre assumido zero. É contado na determinação do tamanho máximo da PICTURE em itens numéricos ou numérico-editados.

Exemplos

PIC X(01) ..... Alfanumérico de 1 byte
PIC 9(02) ..... Numérico sem sinal de 2 bytes e 2 dígitos
PIC S9(05) ..... Numérico com sinal de 5 bytes e 5 dígitos
PIC S9(09) COMP ..... Binário com sinal de 4 bytes e 9 ou 10 dígitos
PIC S9(03) COMP-3 ..... Compactado com sinal de 2 bytes e 3 dígitos

MEMÓRIA	PICTURE	VALOR REAL
503	9(3) ou 999	503
125	9V(2)	1,25
-245	S9V9(2)	-2,41
ALBA123	X(7)	ALBA123
BABA52	BBX(6)	∅∅BABA52
ZIL15A	X(2)BX(4)	Zi∅L15A
BOZZANO	X(5)	BOZZA
1234	9(2)	34

OBSERVAÇÕES IMPORTANTES

1) Sempre que forem exibidos valores numéricos, fazer com que seja exibido ao menos um zero (ex: zz9,99 para 0,00)

2) Todos os dados alfanuméricos serão alinhados à esquerda, ao passo que os dados numéricos serão alinhados à direita, tendo as posições livres preenchidas com zeros.

PIC X(10) para “PACO”

P	A	C	O						

PIC 9(8)V99 para 12,34

0	0	0	0	0	0	1	2	3	4

6.6. Cláusulas

Serão citadas apenas as mais utilizadas. São elas:

VALUE ⇨ Utilizada para atribuir valores iniciais aos campos ou para uso no nível 88 com valor de condição. Não utilizar na FILE SECTION, LINKAGE SECTION (exceto para itens de nível 88), nem para itens relacionados com OCCURS ou REDEFINES.

Exemplos

```
03 AC-LIDOS                PIC S9(17) COMP-3        VALUE 0.
```

Neste exemplo o acumulador está sendo inicializado com zeros.

```
03      CH-ACHOU           PIC X(01)                VALUE 'N'.
        88 ACHOU-REG                                VALUE 'S'.
```

Neste exemplo a palavra chave ACHOU-REG é um nome de condição que estará validado quando o campo CH-ACHOU tiver o conteúdo 'S'.

REDEFINES ⇨ Utilizada para redefinir um campo com uma configuração diferente da original. Veremos este conceito em tabelas.

OCCURS ⇨ Utilizada para especificar tabelas cujos elementos podem ser referenciados pela indexação ou subscrição. Cria uma tabela.

USING ⇨ Utilizada para indicar os campos definidos em um programa chamador que serão passados para um programa chamado como parâmetros.

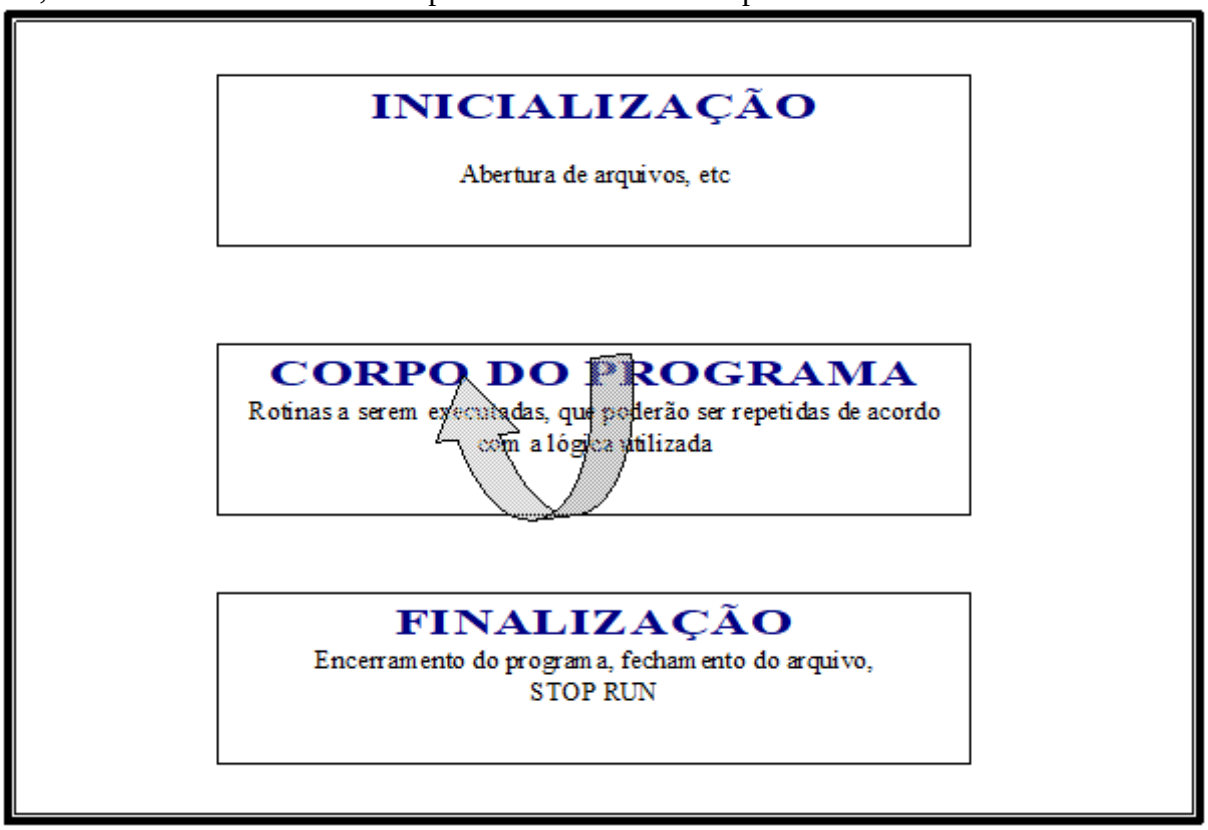
C01 ⇨ Utilizada na SPECIAL-NAMES. Permite que seja definido um nome a ser utilizado quando for necessário saltar uma página. Observe o exemplo:

```
SPECIAL-NAMES.
        DECIMAL-POINT IS COMMA
        C01 IS SALTO.
```

FILLER ⇨ Indica um item elementar cuja informação nunca será referenciada durante a execução do programa. Sempre será definido pela picture X.

7. PROCEDURE DIVISION

Quando aplicarmos a lógica de programação para resolver um problema em um programa Cobol, deixaremos nossa divisão de procedimentos com três partes a saber:



7.1. Instruções aritméticas

COMPUTE

Calcula uma expressão armazenando seu resultado em uma variável numérica.

Sintaxe:

```
COMPUTE <VARIÁVEL> = <EXPRESSÃO>
```

Exemplos:

```
COMPUTE D = B ** 2 – 4 * A * C
COMPUTE R = (B * (-1) + D ** 0.5) / (2 * A)
```

ADD

```
ADD B A TO C.( C = C + B + A).
ADD B A GIVING C.( C = B + A).
```

SUBTRACT

```
SUBTRACT B FROM A.(A = A – B)
SUBTRACT B FROM A GIVING C.(C = A – B)
```

MULTIPLY

```
MULTIPLY A BY B.(B =  B * A)
MULTIPLY A BY B GIVING C.(C = A * B)
```

DIVIDE

```
DIVIDE A INTO B. (B = B / A)
DIVIDE A INTO B GIVING C. (C = B / A, B = B / A)
DIVIDE A BY B GIVING C. (C = B / A)
DIVIDE A BY B GIVING C REMAINDER RESTO.
```

7.2. Instrução IF

Executa um bloco de instruções dependendo de uma determinada expressão lógica.
Sintaxe:

```
IF <condição>
    <sentença 1 | NEXT SENTENCE>
ELSE
   <sentença 2 | NEXT SENTENCE>.
```

Obs:

- ponto final indica o final da estrutura lógica.
- No caso de IFs aninhados, os IFs internos não possuem ponto.
- A instrução NEXT SENTENCE pode ser usada para saltar para a próxima sentença (instrução) imediatamente posterior ao IF.

Teste de condição de classe

```
IF <identificador> IS [NOT] <NUMERIC | ALPHABETIC>
```

Operadores Relacionais

```
IF <identificador 1> IS [NOT] GREATER (>) <identificador 2>
IF <identificador 1> IS [NOT] LESS (<) <identificador 2>
IF <identificador 1> IS [NOT] EQUAL (=)<identificador 2>
```

Teste de condição de sinal

```
IF <identificador> IS [NOT] <POSITIVE | NEGATIVE | ZERO>
```

7.3. Instrução Perform

Determina a execução de um ou mais parágrafos localizados em pontos diversos da PROCEDURE DIVISION. Após a execução do(s) procedimento(s) o fluxo de execução do programa volta para a instrução imediatamente posterior a linha do perform.

SINTAXES:

```
PERFORM <PARÁGRAFO>
PERFORM <PARÁGRAFO1> THRU <PARÁGRAFO2>
PERFORM <PARÁGRAFO> <N> TIMES
PERFORM <PARÁGRAFO> UNTIL <CONDIÇÃO>
PERFORM <PARÁGRAFO> VARYING <VARIÁVEL>
        FROM <VL.INICIAL> BY <INCREMENTO>
        UNTIL <CONDIÇÃO>
```

É possível utilizar elementos das várias sintaxes em uma única linha de comando, conforme se tornar necessário.

EXEMPLOS

```
PERFORM PROCESSA.
PERFORM LE-ARQUIVO THRU ATUALIZA-ARQUIVO.
PERFORM LE-ARQUIVO UNTIL FIM-ARQUIVO.
PERFORM LE-MOV THRU ATUALIZA-MOV
        UNTIL WS-PROCESSO IS EQUAL “OK”.
PERFORM CALC-SALARIO 10 TIMES.
```

OBSERVAÇÕES IMPORTANTES

Atentar sempre para a lógica do programa. Deve ser utilizada a maneira mais eficiente e procurar deixar o arquivo no menor tamanho possível.

Procurar sempre fazer documentações. Por exemplo, em um relatório, exibir o total de registros impressos e/ou processados. Isso ajuda inclusive na manutenção do sistema. O usuário não poderá contestar uma falha de sistema em uma determinada situação que necessite manutenção.

7.4. Terminando um programa

- ☞ Instrução **STOP RUN** ⇨ Termina a execução de um programa retornando o controle ao sistema operacional.
- ☞ Instrução **GOBACK** ⇨ Termina a execução de um programa retornando o controle ao chamador ou ao sistema operacional.

7.5. Instrução MOVE.

Atribui um valor a uma ou mais variáveis de memória ou campo de arquivos.

Sintaxe:

```
MOVE <VALOR> TO <VARIÁVEIS>
```

Exemplos

```
MOVE  0              TO   NUMERO.
MOVE 'LUIZ'          TO   NOME.
MOVE 'SÃO PAULO'     TO   TIME MELHOR CAMPEAO BOM.
MOVE 0               TO   TOTAL SUBTOTAL CONTADOR.
```

7.6. Instrução GO TO

Desvia o fluxo de execução do programa para um parágrafo ou seção, sem retornar para o chamador. É um comando que deve ser utilizado com extremo cuidado para evitar que a lógica de programação torne-se desestruturada. Utilizaremos o comando GO TO apenas dentro de um laço (de mais de um parágrafo ou seção) de forma a quebrá-lo sem contudo desviar o fluxo para fora.

Sintaxe:

```
GO TO <PARÁGRAFO(S)|SEÇÃO(ÕES)> DEPENDING ON <NOME-DADO>
```

O NOME-DADO:

- ☞ deve conter entre 1 e n e deve haver n PARÁGRAFOS|SEÇÕES;
- ☞ deve ser um item número elementar.

Exemplos:

O exemplo abaixo desvia o fluxo de execução do programa para um parágrafo ou seção chamado LEITURA –ARQUIVO-FIM:

```
GO TO LEITURA-ARQUIVO-FIM.
```

No próximo exemplo, é acessada da console uma variável chamada CASO. E o fluxo de execução do programa é desviado para um parágrafo ou seção dependendo do valor dessa variável. Se ela valer 1 é desviado para CASO-1, se valer 2 para CASO-2, se 3 para CASO-3 e, se 4 é desviado para CASO-4.

```
ACCEPT CASO FROM CONSOLE.
GO TO CASO-1 CASO-2 CASO-3 CASO-4 DEPENDING ON CASO.
...
CASO-1.
...
GOTO SEGUE.
CASO-2.
...
GOTO SEGUE.
CASO-3.
...
GOTO SEGUE.
CASO-4.
...
SEGUE.
...
```

7.7. Instrução EXIT

Dentro de um laço com vários parágrafos ou seções, poderá haver um momento em que se faz necessário ignorar uma parte das instruções. Neste caso, podemos encerrar a iteração de vez ou fazer novamente um teste.

A instrução EXIT cria um ponto de fim em uma seqüência deste tipo. Ela deve ser colocada isoladamente em um parágrafo ou seção, para o qual o fluxo de execução do programa será desviado.

Exemplo

```
LOGICA-PRINCIPAL.
...
        PERFORM LE-MOV THRU LE-MOV-EXIT.
...
LE-MOV.
        READ LE-MOV AT END GOTO LE-MOV-EXIT.
        PERFORM LE-CONTA UNTIL CC-CONTA EQUAL MO-CONTA.
        IF MO-TIPOLANC EQUAL 'D'
                SUBTRACT MO-VALOR FROM CC-SALDO
        ELSE
                ADD MO-VALOR TO CC-SALDO.
        REWRITE REG-CONTA.
        GOTO LE-MOV.
LE-MOV-EXIT.
        EXIT.
```

7.8. USAGE

Em muitos sistemas, em que a quantidade de informações armazenadas é muito grande, podemos fazer uma considerável economia de espaço no dispositivo de armazenamento ao utilizarmos do recurso de compactação de dados numéricos disponível pelo Cobol.

COMPUTATIONAL

O campo binário oferece uma maior capacidade de representação dentro de um byte. Será definido dessa forma através do uso da cláusula COMP. Em um campo de quatro casas o valor 6859 é armazenado em dois bytes e com representação hexadecimal X'1ACB'. A definição do campo por exemplo ficaria na seguinte forma:

```
05 CPO-BINARIO PIC 9(04) USAGE COMPUTATIONAL.
```

A linha anterior pode ser abreviada para:

```
05 CPO-BINARIO PIC 9(04) COMP.
```

O uso deste campo tem aplicação prática para representar valores até 9.999, embora os dois bytes binários represente um valor máximo de 32.768.

A definição PIC 9(09) COMP, tem sua aplicação para valores até 999.999.999, embora os quatro bytes binários possam também representar um valor máximo de 2.147.483.648.

COMPUTATIONAL-3

Trata-se de um campo decimal compactado, onde cada algarismo é representado em meio byte e o meu byte mais à direita contém o sinal do campo.

COMPUTATIONAL-1

Especificado para item de ponto flutuante de precisão simples, 4 bytes de comprimento.

COMPUTATIONAL-2

Especificado para item de ponto flutuante de precisão dupla, 8 bytes de comprimento.

8. TABELAS

Uma tabela é uma região da memória que pode conter diversos valores e é referenciada por um único nome. A tabela possui o chamado subscrito ou índice que identifica a posição do elemento desejado na lista de valores. Uma variável poderá ser utilizada para acessar um determinado valor em uma tabela. Nesta caso, ela será chamada de indexador.

Na PROCEDURE DIVISION o número correspondente ao subscrito da tabela será passado para o COBOL entre parênteses logo após o nome do dado.

Por exemplo,

```
DISPLAY MESES(4)
```

exibe o elemento que está na quarta posição de uma tabela chamada MESES.

8.1.1. Cláusula OCCURS

Especifica que um dado será repetido n vezes. Não é permitida para os níveis 01 e 77.

Sintaxe:

```
<número de nível>   nome-de-dado OCCURS <n> TIMES
                    INDEXED BY <nome-do-índice>.
```

Onde:

- ☞ nome-de-dado ⇨ é o nome da variável ou campo que será repetido;
- ☞ n ⇨ é um número inteiro maior que zero indicando quantas vezes o dado será repetido;
- ☞ nome-do-índice ⇨ é um dado que será utilizado como índice da tabela na PROCEDURE DIVISION.

Exemplo

No exemplo a seguir temos os 12 últimos saldos médios de um determinado cliente :

```
01    WS-GRUPO.
        03  WS-CIF            PIC  9(011).
        03  WS-SALDO-MEDIO-1   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-2   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-3   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-4   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-5   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-6   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-7   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-8   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-9   PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-10  PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-11  PIC S9(013)V99  COMP-3.
        03  WS-SALDO-MEDIO-12  PIC S9(013)V99  COMP-3.
```

Com o uso de tabela, o código seria simplificado para :

```
01    TB-GRUPO.
        03  TB-CIF            PIC  9(011).
        03  TB-SALDO-MEDIO    OCCURS 12
                              PIC S9(013)V99  COMP-3.
```

TB-SALDO-MEDIO (1) equivale, então, a WS-SALDO-MEDIO-1,
TB-SALDO-MEDIO(2) equivale a WS-SALDO-MEDIO-2 e assim por diante.

8.1.2. Manipulando um índice de tabela

Setando seu valor

```
SET <NOME-ÍNDICE> TO <VALOR>
```

Incrementando de N

```
SET <NOME-ÍNDICE> UP BY <N>
```

Decrementando de N

```
SET <NOME-ÍNDICE> DOWN BY <N>
```

O exemplo a seguir cria uma tabela e zera todos os seus elementos na PROCEDURE DIVISION.

```
WORKING-STORAGE SECTION.
01      VARIAVEIS.
        05      SALDOS      PIC     S9(7)V99      OCCURS      50      TIMES
                        INDEXED BY IND.
...
PROCEDURE DIVISION.
...
        SET IND TO 0.
        PERFORM  ZERA-TAB 50 TIMES.
        ...
ZERA-TAB.
        SET IND UP BY 1.
        MOVE ZERO TO SALDOS(IND).
```

8.1.3. Cláusula REDEFINES

É utilizada para redefinir um campo com uma configuração diferente da original. O campo redefinido não pode ser maior que o original. Ambos devem ter o mesmo número de nível. Entre eles não pode haver a definição de outro campo com número de nível menor ou igual ao deles.

- ☞ No campo redefinido não é permitido o uso da cláusula VALUE ou OCCURS.
- ☞ Não é permitido redefinir um campo de nível 77 ou 88.
- ☞ São permitidas múltiplas redefinições de um mesmo campo.
- ☞ Campos subordinados a um campo redefinido podem ter a cláusula OCCURS.

Exemplo

```
03    WS-ANO              PIC X(36) VALUE
       'JANFEVMARABRMAIJUNJULAGOSETOUTNOVDEZ'.
03    FILLER REDEFINES WS-ANO.
      05  WS-MESES        PIC X(03) OCCURS 12 TIMES.
```

8.1.4. Zerando uma tabela por expansão

Uma maneira diferente de zerar uma tabela sem a necessidade de utilizar um contador é a expansão da tabela. O exemplo abaixo cria uma tabela de 100 elementos e inicializa todos eles:

Na DATA DIVISION:

```
03      TAB-ZERA.
        05      FILLER      PIC     9(04)               VALUE       ZERO.
        05      FILLER      PIC     X(50)               VALUE       SPACES.
        05      FILLER      PIC     S9(5)   COMP-3      VALUE       ZEROS.
        05      TB-AGE.
                07      FILLER OCCURS 100 TIMES.
                        09      TB-COD      PIC     9(04).
                        09      TB-NOME     PIC     X(50).
                        09      TB-VALOR    PIC     S9(5)V99 COMP-3.
```

Na PROCEDURE DIVISION, a linha

```
MOVE TAB-ZERA TO TB-AGE.
```

inicializa todos elementos de TB-AGE com os valores definidos em TAB-ZERA.
A declaração comum seria:

```
05    TB-AGE OCCURS 100 TIMES.
      09    TB-COD     PIC   9(04).
      09    TB-NOME    PIC   X(50).
      09    TB-VALOR   PIC   S9(5)V99 COMP-3.
```

9. MANIPULAÇÃO DE ARQUIVOS

Como sabemos, os arquivos são as estruturas utilizadas pelo sistema operacional para armazenar informações em um dispositivo externo. Este dispositivo pode ser um disco ou uma fita.

Em COBOL, a manipulação dos arquivos dependerá também de meios externos. Os chamados métodos de acesso. Para a utilização dos comandos abaixo, supomos que no computador haja uma partição com o VSAM (Virtual Storage Access Metod) que dá suporte ao trabalho com arquivos de organização indexada. Todos os arquivos VSAM só podem ser armazenados em disco, por se tratarem de arquivos de acesso direto. Apenas os arquivos seqüenciais poderão ser gravados em fitas magnéticas

9.1.1. Instrução OPEN

Abre um arquivo para utilização.

Sintaxe

```
OPEN <INPUT|OUTPUT|EXTEND> <NOME-DO-ARQUIVO>
```

Modos de abertura:

- **INPUT** ⇨ só permite leitura do arquivo
- **OUTPUT** ⇨ só permite gravação no arquivo, criando o arquivo se o mesmo já existir. Se o mesmo já existir, todos os registros serão destruídos.
- **I-O** ⇨ permite tanto leitura quanto gravação no arquivo.
- **EXTEND** ⇨ abre um arquivo que já exista para permitir apenas gravações, criando o arquivo se o mesmo não existir.

9.1.2. Instrução CLOSE

Fecha o arquivo, finalizando gravações que possam estar pendentes.

Sintaxe

```
CLOSE <NOME-DO-ARQUIVO>
```

9.1.3. Instrução WRITE

Sintaxe para arquivos seqüenciais:

```
WRITE <NOME-REGISTRO> FROM <NOME-DADO>
```

Sintaxe para arquivos VSAM:

```
WRITE <NOME-REGISTRO> FROM <NOME-DADO>
        INVALID KEY <COMANDO IMPERATIVO>
```

Obs: a condição INVALID KEY será atingida quando a gravação não for bem sucedida. Iremos optar pela utilização desta cláusula ou pelo teste da variável de file status.

Sintaxe para impressora

```
WRITE <NOME-REGISTRO> FROM <NOME-DADO1>
        <BEFORE|AFTER> <N>LINES | PAGE
```

EXEMPLOS

```
MOVE CABEC TO LINHA.
WRITE LINHA AFTER 1.

WRITE LINHA FROM CABEC AFTER 1.

MOVE REG-FITA TO REG-DISCO.
WRITE REG-DISCO.

WRITE REG-DISCO FROM REG-FITA
```

9.1.4. Instrução READ

Leitura seqüencial

```
READ <NOME-ARQUIVO> NEXT RECORD INTO <NOME-DADO>
        AT END <COMANDO IMPERATIVO>
```

Leitura aleatória (arquivos VSAM)

```
READ <NOME-ARQUIVO> KEY IS <NOME-DADO1> RECORD
        INTO <NOME-DADO2> INVALID KEY <COMANDO IMPERATIVO>
```

9.1.5. Instrução REWRITE

Regrava o registro atual.

Sintaxe:

```
REWRITE <NOME-REGISTRO> FROM <NOME-DADO>
                INVALID KEY <COMANDO IMPERATIVO>
```

Lembre-se que a cláusula INVALID KEY só pode ser utilizada para arquivos de organização indexada.

9.1.6. Instrução DELETE

Remove o registro atual do arquivo indexado (VSAM). Não possível a utilização deste comando em arquivos seqüenciais.

```
DELETE <NOME-ARQUIVO> RECORD
            INVALID KEY <COMANDO IMPERATIVO>
```

9.1.7. Tabela de valores de file status

A FILE STATUS (situação do arquivo) é uma variável de duas posições que deve ser referenciada no SELECT do arquivo na ENVIRONMENT DIVISION e declarda como qualquer outra na WORKING-STORAGE SECTION.

No ambiente de microcomputador, a FILE STATUS deve ser obrigatoriamente declarada como alfanumérica de duas posições (PIC XX). No ambiente de grande porte, essa declaração pode também ser numérica (PIC 99).

A cláusula VALUE não deve ser utilizada para inicializar a variável, pois a mesma terá seu valor automaticamente atualizado a cada operação que for realizada com o arquivo VSAM.

Código	**Significado**
00	Operação completada com sucesso
10	Fim do arquivo (AT END)
21	Erro na estrutura do arquivo
22	Chave duplicada
23	Registro não encontrado
24	Sem espaço no dispositivo para gravação
30	Erro permanente
91	Erro na estrutura do arquivo

Observações

- ☞ Código 21 ⇨ quando em um acesso seqüencial a gravação não obedece uma seqüência ascendente de chave, no arquivo indexado.
- ☞ Código 30 ⇨ nas operações de OPEN INPUT e OPEN I-O, representa que o arquivo não foi encontrado no dispositivo.
- ☞ Código 91 ⇨nas operações OPEN INPUT e OPEN I-O ocorre quando a estrutura do arquivo está destruída. No caso de OPEN INPUT, o arquivo continua aberto e a instrução READ pode funcionar “normalmente”. No caso de OPEN I-O o arquivo não é considerado aberto.
- ☞ Códigos 21, 22, 23 e 24 ⇨ detectados pela função INVALID KEY.

9.2. Instrução START

Posiciona um registro indexado para início de uma leitura seqüencial a partir de um valor de chave especificado. Só pode ser utilizado nos modos de acesso seqüencial ou dinâmico.

Sintaxe

```
START NOME-ARQUIVO KEY   IS <GREATER | NOT LESS | EQUAL>
      NOME-DADO
```

Onde:

- ☞ NOME-DADO deve ser a RECORD KEY declarada e o valor a ser procurado no arquivos deve ser armazenado previamente ali.
- ☞ Se a frase KEY IS ... for omitida, será procurado o primeiro registro no arquivo em que a chave tenha valor igual ao armazenado no identificador definido como RECORD KEY.

A utilização deste comando pressupõe que o arquivo tenha sido aberto com INPUT ou I-O.

Observação: o comando START apenas posiciona no registro desejado, não disponibilizando os dados deste registro. É necessário utilizar o comando READ (no formato seqüencial) para poder obter as informações desejadas.

10. COMUNICAÇÃO ENTRE PROGRAMAS

10.1. Instrução CALL

A instrução CALL chama um outro programa compilado para execução. Após o término do programa chamado, o fluxo de execução retorna para a linha imediatamente posterior a da instrução.

Podem haver áreas de memória que serão utilizadas para o envio de instruções do programa chamador ao chamado em outras que receberão o resultado do processamento das informações.

Todas as variáveis que mandamos para um programa externo, serão chamadas de parâmetros ou argumentos.

Sintaxe

```
CALL PROGRAMA USING P1 P2 P3 ... Pn
                    ON OVERFLOW <comando-imperativo>
```

Onde:

- ☞ P1 até Pn são os parâmetros que estão sendo enviados para o programa. Alguns dos identificadores da lista já possuem valores que serão utilizados no processamento. Outros não possuem valores nenhum ou se possuírem serão ignorados, pois tratam-se dos identificadores que serão formatados pelo programa chamado.
- ☞ O comando-imperativo especificado na cláusula ON OVERFLOW será executado caso a memória seja insuficiente para acomodar o programa chamado. Neste caso não haverá a transferência de controle. Se esta situação acontecer e não houver o tratamento ON OVERFLOW, acontecerá um ABEND do programa.

10.2. PROCEDURE DIVISION para programas chamados por outros

Quando um programa recebe parâmetros de outro, é necessário declarar os identificadores que receberão esses valores.

Sintaxe

```
PROCEDURE DIVISION USING P1 P2 P3 ... Pn
```

Onde P1 a Pn são os parâmetros recebidos de 1 a n que serão processados ou formatados por este programa.

10.3. LINKAGE SECTION

Como comentamos anteriormente, trata-se de uma seção da DATA DIVISION onde definiremos a área de dados que será utilizada por mais de um programa. Assemelha-se com a WORKING-STORAGE SECTION, porém com algumas características próprias:

- ☞ somente será permitido o uso de identificadores em nível 01 e 77;
- ☞ não será permitido o uso da cláusula VALUE.

Quando se deseja ter acesso a um campo de dados descrito na LINKAGE SECTION, o nome de dados no cabeçalho PROCEDURE DIVISION é tornado igual ao nome de dados correspondente na instrução CALL, sendo assim determinado o endereço do campo no programa que chama.

Exemplo

Suponha o programa:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CHAMA.

ENVIRONMENT DIVISION.

DATA DIVISION.
77      N1          PIC   99     VALUE 1.
77      N2          PIC   99     VALUE 1.
77      SOMA        PIC   999    VALUE 0.

PROCEDURE DIVISION.
INICIO.
        DISPLAY 'NO PROGRAMA CHAMADOR' .
        CALL CHAMADO USING N1, N2, SOMA
                ON OVERFLOW PERFORM ERRO.
        DISPLAY 'RESULTADO : '  SOMA.
        STOP RUN.
ERRO.
        DISPLAY 'MEMORIA INSUFICIENTE PARA CARREGAR O PROGRAMA'
                        .
```

Agora suponha o segundo programa que será chamado pelo primeiro

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CHAMADO.

ENVIRONMENT DIVISION.

DATA DIVISION.
LINKAGE SECTION.
77      A1      PIC 99.
77      A2      PIC 99.
77      RES     PIC 9999.

PROCEDURE DIVISION USING A1 A2 RES.
INICIO.
        DISPLAY 'NO PROGRAMA CHAMADO' .
        COMPUTE RES = A1 + A2.
        GOBACK.
```

Agora note:

- ☞ Os parâmetros enviados pelo comando CALL devem ser recebidos por identificadores correspondentes na mesma ordem na PROCEDURE DIVISION através da cláusula USING.
- ☞ Observe que mesmo as variáveis que são utilizadas para receber os resultados do processamento precisam obrigatoriamente ser enviadas pelo comando CALL.
- ☞ O programa chamado usa o comando GOBACK para retornar ao chamador. Se fosse utilizado o comando STOP RUN, o controle retornaria para o sistema operacional, quebrando desta forma a estrutura de processamento.