INTRODUÇÃO À LINGUAGEM COBOL

O COBOL foi criado em 1959 durante o *CODASYL (Conference on Data Systems Language)*, um dos três comitês propostos numa reunião no Pentágono em Maio de 1959, organizado por Charles Phillips do Departamento de Defesa dos Estados Unidos.

O COBOL é uma linguagem de alto nível, isto é, semelhante a linguagem humana. É um acrônimo e significa **CO**mmon **B**usiness **O**riented **L**anguage (Linguagem Comum Orientada aos Negócios). Como seu nome indica, o objetivo desta linguagem é permitir o desenvolvimento de aplicações comerciais para pessoas sem conhecimento profundo de computadores. Por isso a linguagem COBOL usa frases normais da língua inglesa, e a estrutura de um programa COBOL se assemelha a um texto com divisões, seções, parágrafos e frases em inglês. Uma das características importantes do COBOL é sua auto documentação: um programador pode entender um programa COBOL pela simples leitura de sua codificação.

O padrão atual do COBOL é o COBOL2002. O COBOL2002 suporta conveniências modernas como Unicode, geração de XML e convenção de chamadas de/para linguagens como o C, inclusão como linguagem de primeira classe em ambientes de desenvolvimento como o .NET da Microsoft e a capacidade de operar em ambientes fechados como Java (incluindo COBOL em instâncias de EJB) e acesso a qualquer base SQL.

Como seu nome indica, o objetivo desta linguagem é permitir o desenvolvimento de aplicações comerciais. Depois de escrito o programa COBOL (**chamado de programa fonte**), é necessário traduzí-lo para a linguagem interna do computador (**linguagem de máquina**), convertendo-se então em um programa objeto. Esta conversão é feita através de um job executado no sistema operacional, chamado de compilador COBOL.

Teremos em seguida a definição de alguns termos importantes para o desenvolvimento do curso:

ASCII (American National Standard Code for Information Interchange):é uma codificação de caracteres de oito bits baseada no alfabeto inglês. Os códigos ASCII representam texto em computadores, equipamentos de comunicação, entre outros dispositivos que trabalham com texto. Desenvolvida a partir de 1960, grande parte das codificações de caracteres modernas a herdaram como base.

- **EBCDIC (Extended Binary Coded Decimal Interchange Code):**é uma codificação de caracteres8-bit que descende diretamente do código BCD com 6-bit e foi criado pela IBM como um padrão no início dos anos 1960 e usado no ibm 360.

- **Programa fonte :**é o conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica.

- **Programa objeto / Modulo de carga :**Existem linguagens que são compiladas e as que são interpretadas. As linguagens compiladas, após ser compilado o código fonte, transformam-se em software, ou seja, programas executáveis ou módulos de carga.

- **Compilador:**é usado principalmente para os programas que traduzem o código de fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível.

- **Linguagem de Alto Nível:**é linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo dalinguagem humana.

- **Linguagem de Baixo Nível:**trata-se de uma linguagem de programação que compreende as características da arquitetura do computador. Assim, utiliza somente instruções do processador, para isso é necessário conhecer os registradores da máquina.

INDENTAÇÃO (BOAS PRATICAS)

O processo de indentação (RECUO) consiste em alinhar comandos, de forma que fique mais fácil ao programador que estiver analisando o código, visualizar e, por decorrência, entender o conjunto de instruções. Algumas instruções trabalham com subconjuntos (blocos) de (outras) instruções; por meio da indentação colocam-se instruções que façam parte de um mesmo bloco num mesmo alinhamento.

O caso mais comum é o das instruções condicionais(IF), onde normalmente existe pelo menos um bloco de instruções que deve ser executado quando a condição for verdadeira; e, opcionalmente, outro bloco de instruções que devem ser executadas quando a condição for falsa, exemplo:

```
IF condição
  bloco para condição verdadeira
ELSE
  bloco para condição falsa
END-IF
```

Visualmente facilita-se bastante se deslocarmos os blocos algumas posições à direita (duas ou três posições são suficientes), para que fique destacado o ELSE e o END-IF, facilitando a análise do código fonte.

Se a especificação fosse feita sem indentação:

```
IF condição
COMPUTE A = (B * C ) ** 4
COMPUTE C = A / 0,005
ELSE
COMPUTE A = (B * C ) ** 5
COMPUTE C = A / 0,015
END-IF
```

Ficaria mais difícil analisar do que se houvesse sido especificado com indentação:

```
IF condição
   COMPUTE A = (B * C ) ** 4
COMPUTE C = A / 0,005
ELSE
   COMPUTE A = (B * C ) ** 5
   COMPUTE C = A / 0,015
END-IF.
```

A vantagem desta técnica é que fica muito mais evidente quando houver IFS encadeados :

Sem indentação :

```
IF condição
IF condição
COMPUTE A = (B * C ) ** 4
COMPUTE C = A / 0,005
ELSE
COMPUTE A = (B * C ) ** 8
COMPUTE C = A / 0,055
END-IF
ELSE
IF condição
COMPUTE A = (B * C ) ** 5
COMPUTE C = A / 0,007
ELSE
COMPUTE A = (B * C ) ** 9
COMPUTE C = A / 0,007
END-IF
END-IF.
```

Com indentação :

```
IF condição
   IF condição
COMPUTE A = (B * C ) ** 4
      COMPUTE C = A / 0,005
   ELSE
      COMPUTE A = (B * C ) ** 8
      COMPUTE C = A / 0,055
    END-IF
ELSE
   IF condição
       COMPUTE A = (B * C ) ** 5
       COMPUTE C = A / 0,007
   ELSE
       COMPUTE A = (B * C ) ** 9
       COMPUTE C = A / 0,007
   END-IF
END-IF.
```

FORMATO DO FONTE COBOL

Todo programa escrito na linguagem COBOL possui algumas regras a serem seguidas. Uma destas regras se refere ao formato das linhas de comando (instruções) dentro do seu editor de fonte.

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM    A    B
1....6 7 8...12.....................................72 73...80
```

Colunas de 1 a 6:	Área livre – pode ser usada para documentação / alteração / versão de programas
Coluna 7:	Área de indicação de comentários
Colunas de 8 a 11:	Área A ou Nível A para codificação das seções,divisões, parágrafos, variaveis
Colunas de 12 a 72:	Área B ou Nível B para codificação dos comandos da linguagem
Colunas de 73 a 80:	Cobol utiliza para numerar as linhas

AREA DE NUMERAÇÃO SEQÜENCIAL (COLUNAS DE 1 A 6)

Area livre. Pode ser usada para documentação de alterações no programa.

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM    A    B
1....6 7 8...12.....................................72 73...80
```

ÁREA DE INDICAÇÃO(COLUNA 7)

HÍFEN (-)

Se o hífen estiver nesta posição indica que existe uma continuação de uma cadeia de caracteres, (uma palavra ou frase), que foi iniciada na linha anterior. Uma literal que não caiba numa linha, para que seja continuada na próxima linha, precisa ter na próxima linha a indicação da continuação (hífen na coluna 7) e, em qualquer coluna a partir da 12, um apóstrofe ou aspas indicando o início da continuação.

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM    A    B
1....6 7 8...12.....................................7273...80
000001       DISPLAY "RELATORIO MENSAL DE VENDAS POR
      -      "AGENCIA".
```

ASTERISCO (*)

Nesta posição indica, para o compilador COBOL, que toda a linha deve ser tratada como uma linha de comentário.

Linhas em branco com * na coluna 7, são tratadas mais rapidamente pelo compilador COBOL, pois são ignoradas logo no início da compilação.

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM    A    B
1....6 7 8...12.....................................7273...80
000001 * ISTO E UM COMENTARIO
```

AREA A – COLUNAS DE 8 A 11

Posição a partir da qual se escrevem o nome de Divisão, Seção, Parágrafo e os níveis FD (FILE DESCRIPTION), 01 (grupo de variáveis) e 77 (variável individual).

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *-------------------------------------------------------------------*
       DATA                                DIVISION.
      *-------------------------------------------------------------------*
       FILE                                SECTION.
       WORKING-STORAGE                     SECTION.
```

ÁREA B - COLUNAS DE 12 A 72)

Posição a partir da qual se escrevem os valores dos parágrafos e as instruções COBOL

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *-------------------------------------------------------------------*
       PROCEDURE DIVISION.
      *-------------------------------------------------------------------*
       000-UNICA.
           PERFORM 010-INICIALIZAR
           PERFORM 030-PROCESSAR  UNTIL WS-FIM = "S"
           PERFORM 050-TERMINO
           STOP RUN
           .
```

Colunas 44 e 52

Há uma prática de programação na maioria dos ambientes de desenvolvimento COBOL mainframe que é o uso das colunas 44 e 52 para sempre colocar certas instruções.

Na COLUNA 44 escrevemos DIVISION, SECTION, PIC, TO do MOVE
Na COLUNA 52 escrevemos VALUE

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       DATA                                DIVISION.
       WORKING-STORAGE                     SECTION.
       77  WS-PRECO                        PIC 9(06)V99
                                                 VALUE ZERO.
       77  WS-TEXTO                        PIC X(30)
                                                 VALUE
           "CONTATE O ANALISTA RESPONSAVEL".
```

Não seguir a prática de programação do uso das colunas 44 e 52 não causará erro de compilação ou mesmo de lógica no programa, mas se há um padrão é melhor segui-lo.

DIVISÕES DO COBOL

De maneira semelhante a um livro com seus capítulos, parágrafos e itens, um programa COBOL é formado por uma hierarquia de divisão (DIVISION), seção (SECTION), parágrafo e instruções. Uma regra importante em COBOL é que todo nome de Divisão, Seção e Parágrafo deve ser terminada por ponto final (.). O código COBOL possui quatro divisões que devem ser utilizadas nesta ordem:

- **IDENTIFICATION DIVISION**

A IDENTIFICATION DIVISION possui informações documentais, como nome do programa, quem o codificou e quando essa codificação foi realizada.

- **ENVIRONMENT DIVISION**

A ENVIRONMENT DIVISION descreve o computador e os periféricos que serão utilizados pelo programa.

- **DATA DIVISION**

A DATA DIVISION descreve os layouts dos arquivos de entrada e saída que serão usadas pelo programa. Também define as áreas de trabalho e constantes necessárias para o processamento dos dados.

- **PROCEDURE DIVISION**

A PROCEDURE DIVISION contém o código que irá manipular os dados descritos na DATA DIVISION. É nesta divisão que o desenvolvedor descreverá o algoritmo do programa.

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................7273...80
         IDENTIFICATION DIVISION.
         ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
         INPUT-OUTPUT SECTION.
         DATA DIVISION.
         FILE SECTION.
         WORKING-STORAGE SECTION.
         LINKAGE SECTION.
         PROCEDURE DIVISION.
```

IDENTIFICATION DIVISION

Esta é a divisão de identificação do programa.
Não contêm Sections, mas somente alguns parágrafos pré-estabelecidos e opcionais. O único parágrafo obrigatório é o PROGRAM-ID (Nome do programa). O nome do programa deve ser uma palavra com até 8 caracteres (letras ou números), começando por uma letra.

Esta divisão possui a seguinte estrutura:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.....................................7273.....80
         IDENTIFICATION   DIVISION.
         PROGRAM-ID.      NOME-DO-PROGRAMA.
       * [AUTHOR.         NOME-DO-PROGRAMADOR.]
       * [INSTALLATION.   NOME-DA-EMPRESA.]
       * [DATE-WRITTEN.   DATA-DA-CODIFICACAO.]
       * [DATE-COMPILED.  DATA-DA-COMPILACAO.]
       * [SECURITY.       COMENTÁRIO DO PROGRAMA.]
       * [REMARKS.        COMENTÁRIO DO PROGRAMA.]
```

No COBOL II, estes parâmetros são opcionais

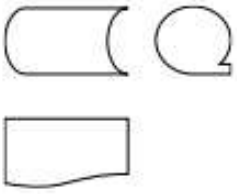
Todas as cláusulas que possuem o *** na coluna 7** , não possuem nenhum efeito na aplicação. São apenas parâmetros opcionais para documentação do programa.

A IDENTIFICATION DIVISION pode ser abreviada para ID DIVISION.

ENVIRONMENT DIVISION

Esta divisão é para a qualificação de ambiente, equipamentos e arquivos, que serão utilizados pelo programa. Possui duas SECTION e sua estrutura é a seguinte:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *-------------------------------------------------------------*
      ENVIRONMENT                          DIVISION.
      *-------------------------------------------------------------*
      CONFIGURATION                        SECTION.
      *-------------------------------------------------------------*
      SOURCE-COMPUTER.                     COMENTARIO.
      OBJECT-COMPUTER.                     COMENTARIO.
      SPECIAL-NAMES.
          DECIMAL-POINT IS COMMA.
      *-------------------------------------------------------------*
      INPUT-OUTPUT                   SECTION.
      *-------------------------------------------------------------*
      FILE-CONTROL.
          SELECT CADALUNO ASSIGN TO UT-S-CADALUNO
          FILE STATUS IS FS-CADALUNO.

          SELECT RELATO ASSIGN TO UT-S-RELATO
          FILE STATUS IS FS-RELATO.
```



CONFIGURATION SECTION

Esta seção destina-se a descrição dos equipamentos que serão utilizados pelo programa. Esta SECTION é composta por três parágrafos: **SOURCE-COMPUTER**,**OBJECT-COMPUTER** e **SPECIAL-NAMES**.

SOURCE-COMPUTER: identifica o computador onde foi confeccionado o programa.
OBJECT-COMPUTER: identifica o computador do ambiente de produção.
SPECIAL-NAMES: específica o sinal monetário, escolhe o tipo de ponto decimal, específica caracteres simbólicos e possibilitar adaptar o programa para se comunicar com programas de outras linguagens.

Têm comandos pré-definidos em Cobol, para especificar alfabeto, moeda, ou separador de decimal (virgula ou ponto), mas todos os comandos são opcionais. O separador de decimais é usado mais freqüentemente.

Formato:

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

Esta instrução informa que a vírgula (COMMA) será usada como separador de decimais. **DECIMAL-POINT IS COMMA deve estar na área B.**

INPUT-OUTPUT SECTION

Esta SECTION destina-se a configuração dos arquivos. No parágrafo FILECONTROL, informando ao COBOL o nome interno e externo dos arquivos, bem como a organização dos registros, modo de acesso e variável de FILE STATUS..

DATA DIVISION

A DATA DIVISION é a divisão do programa onde são descritos os layouts ou mapeamento dos registros de dados, incluindo as variáveis e constantes necessárias.A DATA DIVISION é composta pelas Sessões: FILE SECTION, WORKING-STORAGE SECTION e LINKAGE SECTION.

FILE SECTION

A FILE SECTION é usada para detalhar o conteúdo dos registros dos arquivos, utilizando-se da FILE DESCRIPTION (FD) que descreve as características do arquivo, que possuem alguns parâmetros importantes e o mapeamento dos campos do registro.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *-----------------------------------------------------------------*
       FILE                              SECTION.
      *-----------------------------------------------------------------*
       FD  CADALUN
           LABEL RECORD STANDARD
           BLOCK CONTAINS 0 RECORDS
           RECORDING MODE IS F
           RECORD CONTAINS 97 CHARACTERS
           DATA RECORD IS REG-CADALUN
           .
       01  REG-CADALUN.
           05 CLASSE-E                   PIC X(03).
           05 NUM-E                      PIC X(03).
           05 NOME-E                     PIC X(40).
           05 ENDE-E                     PIC X(40).
           05 NOTA1-E                    PIC 99V99 COMP-3.
           05 NOTA2-E                    PIC 99V99 COMP-3.
           05 NOTA3-E                    PIC 99V99 COMP-3.
           05 IDADE-E                    PIC 9(02).

       FD  RELATO
           LABEL RECORD OMITTED
           RECORDING MODE IS F
           RECORD CONTAINS 80 CHARACTERS
           DATA RECORD IS REG-RELATO
           .
       01  REG-RELATO                    PIC X(80).
```

Para todo arquivo declarado no **INPUT-OUTPUTSECTION,** é necessário haver uma descrição e um layout do mesmo na **FILE SECTION.**

WORKING-STORAGE SECTION

NA WORKING-STORAGE SECTION onde são definidas todas as variáveis que o programa irá utilizar. Não há parágrafos e os dados podem ser definidos como grupos hierárquicos ou independentes (**níveis 01 a 49**), ou dados independentes (**nível 77**) em qualquer ordem, desde que não se crie um nível 77 no meio de uma hierarquia de níveis causando seu rompimento.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *---------------------------------------------------------------*
       WORKING-STORAGE                    SECTION.1
      *---------------------------------------------------------------*
      * EXEMPLO DE VARIAVEIS INDEPENDENTES
      *---------------------------------------------------------------*
       77  WS-CONTADOR                    PIC 9(05) VALUE 0.
       77  WS-MSG                         PIC X(60) VALUE SPACE.
       77  WS-NOME                        PIC A(40).
      *---------------------------------------------------------------*
      * EXEMPLO DE VARIAVEIS DE GRUPO
      *---------------------------------------------------------------*
       01  WS-DATA-SISTEMA.
           05 WS-ANO                      PIC 9(02).
           05 WS-MES                      PIC 9(02).
           05 WS-DIA                      PIC 9(02).
       01  WS-DATA-FORMATADA.
           05 WS-DIA                      PIC 9(02).
           05 FILLER                      PIC X(01) VALUE "/".
           05 WS-MES                      PIC 9(02).
           05 FILLER                      PIC X(03) VALUE "/20".
           05 WS-ANO                      PIC 9(02).
      *---------------------------------------------------------------*
      * EXEMPLO DE VARIAVEIS DE GRUPO COM VARIOS NIVEIS
      *---------------------------------------------------------------*
       01  WS-CLIENTE.
           05 WS-CODCLI                   PIC X(04).
           05 WS-NOMECLI.
              10 WS-PRIMEIRO-NOME         PIC X(15).
              10 WS-SOBRENOME             PIC X(30).
           05 WS-ENDERECO.
              10 WS-LOGRADOURO.
                 15 WS-TIPO-LOGRADOURO    PIC X(10).
                 15 WS-NOME-LOGRADOURO    PIC X(30).
              10 WS-NUMERO                PIC 9(05).
              10 WS-COMPLEMENTO           PIC X(15).
              10 WS-BAIRRO                PIC X(25).
              10 WS-CIDADE                PIC X(25).
              10 WS-UF                    PIC X(02).
           05 WS-TELEFONECLI.
              10 WS-COD-PAIS              PIC 9(03).
              10 WS-COD-REGIAO            PIC 9(02).
              10 WS-NUM-TELEFONE          PIC 9(08).
```

LINKAGE SECTION.

A LINKAGE SECTION é a seção onde declaramos as variáveis usadas para receber dados do parâmetro PARM do JCL ou do programa PRINCIPAL, através da instrução CALL com USING.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *---------------------------------------------------------------*
      * RECEBENDO DADOS VIA PARM DO JCL
      *---------------------------------------------------------------*
       LINKAGE                            SECTION.
      *---------------------------------------------------------------*
       01  LS-PARAMETRO-RECEBIDO.
           05 LS-TAMANHO-DO-PARAMETRO      PIC S9(4) COMP.
           05 LS-DADOS-RECEBIDOS.
              10  LS-DATA-RELATORIO        PIC X(08).
      *---------------------------------------------------------------*
       PROCEDURE DIVISION USING LS-PARAMETRO-RECEBIDO.
      *---------------------------------------------------------------*
```

PROCEDURE DIVISION

Esta divisão controla a execução do programa, é onde colocamos os comandos oriundos do algoritmo planejado pelo programador. Os comandos (instruções) do COBOL são formados por um único verbo da língua inglesa, seguido dos parâmetros necessários, e que serão discutidos um a um nos parágrafos seguintes. As instruções de um programa COBOL podem ser reunidas em parágrafos, e estes em seções, definidas pelo programador com o fim de tornar o programa mais fácil de ser entendido

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *---------------------------------------------------------------*
       PROCEDURE                              DIVISION.
      *---------------------------------------------------------------*
       0000-GPXXNN02.
           PERFORM 1000-INICIALIZAR
           PERFORM 2000-PROCESSAR
             UNTIL WS-FIM = "S"
           PERFORM 3000-TERMINO
           STOP RUN
           .

       1000-INICIALIZAR.
           MOVE 0                             TO WS-CONT
           MOVE 0                             TO WS-CTLIDO
           MOVE "N"                           TO WS-FIM
           PERFORM 1500-LER-SYSIN
           .

       1500-LER-SYSIN.
           ACCEPT WS-TAB FROM SYSIN
           IF WS-TAB = 0
              MOVE "S"                        TO WS-FIM
           ELSE
              ADD 1 TO WS-CTLIDO
           END-IF
```

ESPECIFICAÇÃO PARA DADOS OU VÁRIAVEIS

A definição de um dado em COBOL é feito com o seguinte formato:

Nível	variável-1	Formato	Valor-inicial

NÍVEL

Os números de níveis definem a hierarquia dos campos dentro dos registros ou a hierarquia nas áreas auxiliares criadas pelo programador. O registro também deve ser numerado, pois ele é um item de grupo, A numeração para itens de grupo é “01”, por definição todos os itens de grupo serão itens alfanuméricos.

Dentro dos itens de grupo estão os itens elementares, e estes podem receber uma numeração entre “02 e “49”.

Exemplo:

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................................7273....80
         WORKING-STORAGE SECTION.
         01   REGISTRO-ALUNO.
              05  NOME                    PIC X(35).
              05  NASCIMENTO.
                  10  DD                  PIC 9(02).
                  10  MM                  PIC 9(02).
                  10  AAAA                PIC 9(04).
```

Os níveis de 50 a 99 tem uso específico, ou reservados para futuras expansões do Cobol. Nesta faixa há um nível de uso muito freqüente.

Número de Nível Especial 77

O nível 77 define áreas auxiliares independentes, onde estes não são subdivididos, são usados para contadores, acumuladores e indexadores.

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................................7273....80
         WORKING-STORAGE SECTION.
         77  WK-LIDOS                    PIC 9(03).
```

NOME-DO-DADO (variável)

Podemos usar qualquer palavra que não seja usada internamente no COBOL**(palavra reservada).** Esta palavra pode ter no **máximo 30 caracteres**, incluindo letras, números e hífen, sendo que pelo menos um dos caracteres deve ser uma letra.

Padroes de mercado:

- Todas as variáveis devem serm prefixadas de acordo com a section onde foi declarada:
Na WORKING STORAGE como WRK ou WK
Na LINKAGE como LNK ou LK
Na FILE como FS ..

FILLER: Palavra reservada que preenche determinados espaços definidos na Picture, sendo que não temos acesso ao item elementar, somente quando manipulamos o item de grupo à que ele esteja subordinado. Eles são usados freqüentemente para não poluirmos o programa fonte com nomes desnecessários de variáveis.

Exemplo:

```
01 WK-CLIENTE.
   03  FILLER              PIC X(10).       <- CLIENTE:
    03  WK-NOMECLIENTE PIC X(30).      <- JONAS BLOCH
```

Ex: CLIENTE: JONAS BLOCH

FORMATOS DA PICTURE:

A – O dado é alfabético e contém somente letras e espaços.
9 – O dado é numérico e contém somente números.
X – O dado é alfanumérico e pode conter letras, números e outros caracteres especiais.

A definição de um dado em COBOL tem o seguinte formato:

Nível	Nome-do-dado	Formato	Valor-inicial
02	SOMA-CREDITOS	PIC 9(6)V99	VALUE ZEROS.

Em memória este dado ficará assim:

1	2	3	4	5	6	7	8

Neste exemplo, o dado conterá 6 bytes inteiros e dois bytes decimais.

Quando usamos a cláusula de valor inicial (VALUE), no momento que esta variável é carregada em memória, o dado ficará assim:

0	0	0	0	0	0	0	0

A quantidade de caracteres contidos no dado é especificada no formato, podendo repetir a característica do formato, que representará a quantidade. Porém isto só é válido para formatos numéricos.
Por exemplo, se o item QUANT-PROD tem 5 algarismos, seu formato será:

```
01  QUANT-PROD        PIC 99999.
```

Pode-se abreviar esta repetição colocando o número de repetições entre parênteses:

```
01  QUANT-PROD        PIC 9(5).
```

Em uma variável numérica armazenada na memória, não existe o ponto e nem a vírgula decimal.

Se o item VALOR-PROD tiver, por exemplo, o valor de 2,35 fica na memória como 235.

Mas o programa COBOL precisa saber em que posição estava a vírgula que desapareceu (vírgula implícita). A vírgula implícita é definida no formato pela letra V, como abaixo:

```
01  VALOR-PROD       PIC 99999V99.
01  VALOR-PROD       PIC 9(5)V99.
```

Em um grupo de itens contidos em uma hierarquia (com níveis de 01 a 49) só podem ter a cláusula PIC, os itens no nível mais baixo da hierarquia (itens elementares).

Exemplo:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.....................................................7273....80
         WORKING-STORAGE SECTION.
         01   REGISTRO-ALUNO.
              05  NOME                    PIC X(35).
              05  NASCIMENTO.
                  10  DD                  PIC 9(02).
                  10  MM                  PIC 9(02).
                  10  AAAA                PIC 9(04).
```

A linguagem COBOL suporta itens **numéricos com até 18 algarismos**, e itens alfanuméricos até 32.768 caracteres (dependendo do sistema operacional).
Existem ainda formatos especiais da PIC para itens a serem exibidos ou impressos, chamados de mascaras de edição.

A cláusula PICTURE (ou PIC) tem alguns formatos próprios para fazer edição de variáveis numéricas no momento de uma impressão que são mostrados na tabela abaixo:

PICTURE	VALUE	IMPRESSÃO
$ZZZZ9,99	2	$ 2,00
$$$$$9,99	2	$2,00
$****9,99	2	$*****2,00
+ZZZZ9,99	-2	- 2,00
--------9,99	-2	-2,00
++++9,99	-2	-2,00
++++9,99	+2	+2,00
ZZ.ZZ9,99+	-2002	2.002,00-

BLANK WHEN ZERO

Esta cláusula, usada após a máscara de edição da PICTURE, envia espaços em branco para a impressora quando a variável numérica a ser impressa tem valor zero, independente do formato da máscara.

Exemplo:

03 VALOR PIC ZZ.ZZ9,99 BLANK WHEN ZERO.

VALUE - VALOR INICIAL

Esta cláusula é opcional em COBOL. Seu objetivo é definir um valor para a variável. Se ela for omitida, o item correspondente terá valores imprevisíveis. No caso de uma variável numérica, por exemplo, é conveniente que ele comece com o valor zero. O valor-inicial é definido no COBOL pelo formato:

Em COBOL existem 2 tipos de literais: numérica e alfa-numérica.

As literais numéricas são escritas colocando-se o valor na instrução, sem apóstrofe ou aspas.

Exemplo:

```
77   IDADE-MINIMA        PIC 99     VALUE 18.
77   IDADE-MAXIMA        PIC 9(2)   VALUE ZEROS.
```

As literais alfanuméricas devem ser colocados entre apostrofes (‘) ou aspas (“).

Exemplo:

```
77   NOME-RUA            PIC X(20)    VALUE  ‘RUA FIDALGA’.
```

Não se pode misturar em um programa COBOL o uso de apóstrofes com aspas, ou seja, se uma literal começou a ser escrito com apóstrofe, deve-se usar apóstrofe para terminar a literal.

Pode-se usar ainda como valor-inicial as **CONSTANTES FIGURATIVAS**, como por exemplo, ZEROS, SPACES, LOW-VALUES ou HIGH-VALUES.

ZEROS | ZERO | ZEROES – O item (deve ser numérico) será preenchido com algarismos 0 (zero).

SPACE | SPACES – O item (deve ser alfabético ou alfa-numérico) será preenchido com espaços.

LOW-VALUE | LOW-VALUES – (menor valor) Indica que este item na memória deve ter todos os seus bytes com todos os bits desligados.

HIGH-VALUE | HIGH-VALUES - (maior valor) Indica que este item na memória deve ter todos os seus bytes com todos os bits ligados.

Exemplo:

```
77   TOTAL          PIC 9(10)    VALUE  ZEROS.
```

Variável Zonada (PIC com USAGE DISPLAY)

É o formato padrão (default) das variáveis numéricas.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
*-------------------------------------------------------------------*
WORKING-STORAGE SECTION.
*-------------------------------------------------------------------*
77 WS-CTLIDO      PIC 9(03)     USAGE DISPLAY.
77 WS-PRECO-VENDA PIC 9(06)V99.
```

As variáveis zonadas não são boas para a realização de cálculos, pois o computador precisa primeiro converter os valores armazenados nessas variáveis para binário, depois realizar o cálculo (que sempre é em binário) e por fim, o resultado é convertido novamente para zonado.

O uso dessas variáveis é para exibição (USAGE DISPLAY).

Variável Binária (PIC com USAGE COMP ou COMP3)

É o formato das variáveis binárias. Pode-se abreviar para COMP. O número é primeiro convertido para binário antes de ser armazenado nesse tipo de variável.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
      *-------------------------------------------------------------*
       WORKING-STORAGE                    SECTION.
      *-------------------------------------------------------------*
       77  WS-CTLIDO                      PIC 9(03)
                                                 USAGE COMPUTATIONAL.
       77  WS-CTGRAV                      PIC 9(03)
                                                 USAGE COMP.
       77  WS-CTIMP                       PIC 9(03)
                                                 COMP.
```

COMANDOS

ACCEPT (RECEBENDO DADOS DO SISTEMA)

O comando ACCEPT recebe uma informação de dados, dependendo das cláusulas que completam o comando, que podem ser: **ESTRUTURA DE DADOS, DATA DO SISTEMA, DIAS DO ANO, DIA DA SEMANA e TEMPO.**

Esta opção recebe uma área de dados da SYSIN do JOB de execução do programa, e **NÃO recebe dados digitados** na tela, devemos assinalar uma área de entrada no JCL.

É importante lembrar que todo parâmetro SYSIN, passado via JCL, possui uma limitação de tamanho, que é de 72 bytes.

Sintaxe básica : ACCEPT variável FROM origem

```
1....6 7 8...12.....................................
         WORKING-STORAGE SECTION.
         01   ESTRUTURA-DE-DADOS.
              05  ITEM-DE-DAD01     PIC X(05).
              05  ITEM-DE-DAD02     PIC 9(03).
        PROCEDURE DIVISION.
            ACCEPT ESTRUTURA-DE-DADOS.
```

Obs.: Quando a opção FROM SYSIN é omitida o default FROM SYSIN é assumido.

Origens:

- **FROM SYSIN**

Cada vez que o programa COBOL executa a instrução ACCEPT, uma linha da SYSIN do JCL é carregada na variável. É necessário prever com cuidado quantas linhas terá a SYSIN do JCL, porque se o comando ACCEPT não encontrar uma linha para carregar na sua variável, o sistema operacional emitirá uma mensagem de erro para o operador e o programa ficará suspenso até a intervenção do operador.

- **FROM DATE [YYYYMMDD]**

```
Formato implícito PIC 9(06)
Formato YYMMDD- data gregoriana - 20/12/2009 virá como 091220
```

- **FROM DAY**

```
Formato implícito PIC 9(05)
Formato YYDDD- data Juliana 04/07/1981 será expresso como 81185
```

- **FROM DAY-OF-WEEK**

```
Formato implícito PIC 9(1), onde:
1 = Monday,
2 = Tuesday,
3 = Wednesday,
4 = Thursday,
5 = Friday,
6 = Saturday,
7 = Sunday.
```

- **FROM TIME** `Formato implícito PIC 9(08)`

`Formato HHMMSSCC – Hora, minuto, segundo, centésimos.`

`2:41 da tarde será expresso como 14410000`

DISPLAY

Exibe o conteúdo de uma variável podendo ser concatenada com uma literal, o conteúdo da variável será exibido num dispositivo de saída.

Exemplos:

```
        PROCEDURE DIVISION.
        PARAGRAFO-01 SECTION.
            ACCEPT WS-DATA FROM DATE.
            DISPLAY 'EXIBIR A DATA......= ' WS-DATA.
```

```
1....6 7 8...12.....................................................7273......80
        PROCEDURE DIVISION.
        PARAGRAFO-02 SECTION.
            DISPLAY  "PROGRAMA INICIANDO.............  ".
            DISPLAY  "TOTAL REGISTROS LIDOS.........: "  WS-REGLIDOS.
            DISPLAY  "TOTAL REGISTROS GRAVADOS......: "  WS-REGGRAV.
            DISPLAY  WS-DIA '/' WS-MES '/' WS-ANO.
```

STOP RUN

A instrução STOP RUN encerra a execução do programa.

Formato:

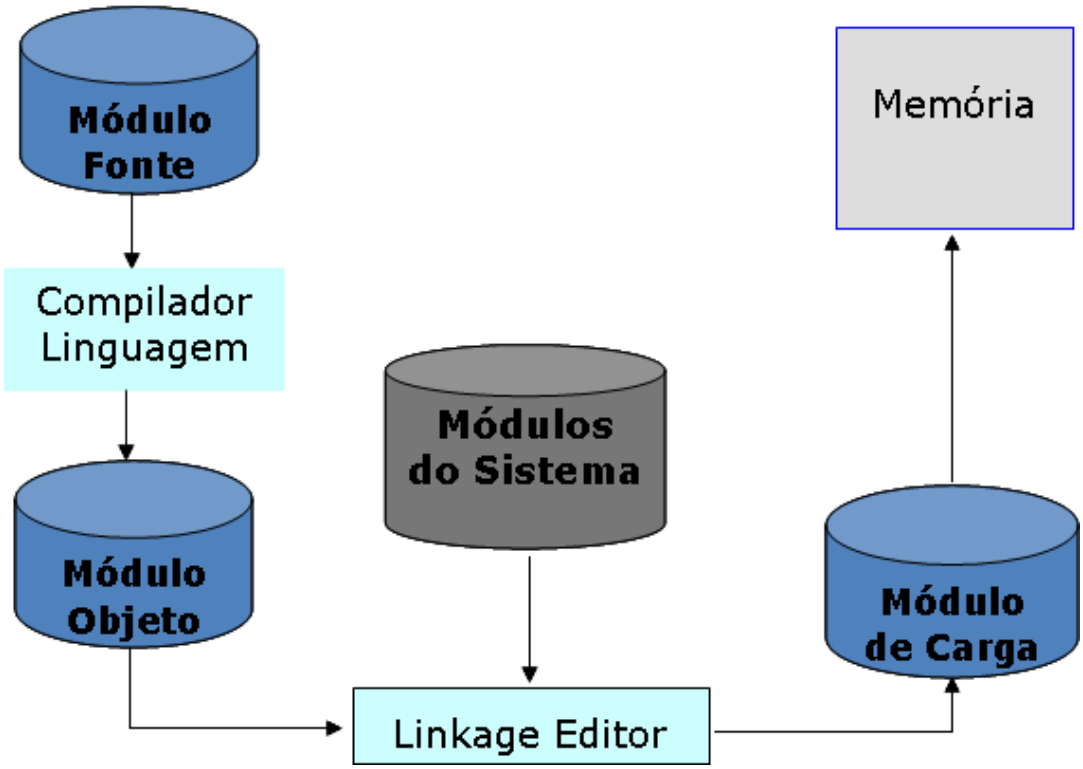
STOP RUN.

Compilação e Linkedição de programas

O compilador para linguagem COBOL, é responsável por traduzir as instruções e comandos da linguagem de alto nível, para a linguagem de baixo nível ou linguagem de máquina.
Todo programa só pode ser executado pelo sistema operacional, se o mesmo estiver em linguagem de máquina. Desta forma foi desenvolvido um utilitário que transforma os comandos e instruções em códigos binários, gerando um objeto executável.

Etapas da compilação e linkedição COBOL:

- Verificar os erros de sintaxe do código fonte.
- Transformar o código fonte em linguagem de máquina
- Gerar um módulo objeto
- Acoplar ao módulo objeto, os módulos do sistema operacional
- Gerar o objeto executável
- Disponibilizar este objeto executável em uma biblioteca de carga



HORA DE CONHECERMOS JCL

Comandos Aritméticos

As instruções para efetuar cálculos aritméticos em COBOL são:

```
ADD(Adição)
SUBTRACT(subtração)
MULTIPLY(multiplicação)
DIVIDE(divisão)
COMPUTE(calcular)
Todas as instruções aritméticas podem ser completadas com as opções
ROUNDED ou ON SIZE ERROR.
```

Formato básico para instruções aritméticas:

Instrução aritmética
[ROUNDED]
[[NOT] ON SIZE ERROR instrução imperativa …]
[END-{nome-da-instrução}]

Opção ROUNDED

Usa-se a opção ROUNDED quando se operam com números decimais e existe perda de algarismos no resultado. Para obter resultados arredondados, a opção ROUNDED pode ser especificada em qualquer instrução aritmética. Em todos os casos, ela vem imediatamente após o nome do operando resultante. O COBOL faz um arredondamento clássico para o resultado da instrução aritmética (valores perdidos menores que 5nn.. são truncados, e os maiores são arredondados para cima). Exemplo: ADD WS-VALOR1 TO WS-VALOR2 ROUNDED

Opção ON SIZE ERROR

Quando a variável que recebe o resultado da operação aritmética não tem tamanho suficiente para conter o resultado, o COBOL trunca o valor resultante (o valor perde algarismos à esquerda), e o COBOL não emite avisos ou código de erro. Para que se possa detectar esta situação é necessário codificar na instrução aritmética a clausula ON SIZE ERROR, onde podemos colocar uma mensagem de erro, parar o programa ou desviar para um parágrafo especial de tratamento de erro.

Exemplo:

```
ADD VALOR-1 TO VALOR-2
    ON SIZE ERROR
       DISPLAY 'ESTOUROU O CAMPO DE RESULTADO'
MOVE "S" TO WS-ESTOURO
  END-ADD.
```

Opção END-… (Delimitador de escopo)

Utilizado como delimitador em todas as instruções aritméticas do COBOL

```
ADD VALOR-1 TO VALOR-3
    ON SIZE ERROR
DISPLAY 'ESTOUROU O CAMPO DE RESULTADO'
MOVE "S" TO WS-ESTOURO
  END-ADD.
```

ADD

Acumula dois ou mais operandos numéricos e armazena resultados.

Formato:

ADD { variável-1 ... constante-1 ...} TO | GIVING
{ variável-de-resultado ... }

Regras:

1. Todos os campos e constantes que são parte da adição devem ser numéricos. Após a palavra GIVING, contudo, o campo pode ser um campo editado (campo numérico com mascara de edição).

2. O campo variável-de-resultado, após TO ou GIVING, deve ser um nome de dados, e não uma constante.

3. Pelo menos dois operandos deverão anteceder a palavra GIVING.

4. Ao usar **TO**, o conteúdo inicial do variável-de-resultado, que deve ser numérico, é somado junto ao dos outros campos (variável-1 ... constante-1...).

5. Ao usar o formato **GIVING**, o campo variável-de-resultado receberá a soma, mas seu conteúdo inicial não será parte da instrução ADD. Ele pode ser tanto um campo numérico como um campo editado.

Exemplos:

```
ADD    WS-VALOR  TO        WS-AC-TOTAL.
```

- Efetua: WS-AC-TOTAL = WS-AC-TOTAL + WS-VALOR.

```
ADD    WS-VALOR  TO        WS-AC-TOTAL1
                           WS-AC-TOTAL2.
```

- Efetua: WS-AC-TOTAL1 = WS-AC-TOTAL1 + WS-VALOR.
- Efetua: WS-AC-TOTAL2 = WS-AC-TOTAL2 + WS-VALOR.

```
ADD    WS-AC-TOTAL1
       WS-AC-TOTAL2
       WS-AC-TOTAL3 TO WS-AC-TOTGERAL.
```

- Efetua: WS-AC-TOTGERAL = WS-AC-TOTGERAL + WS-AC-TOTAL1 + WS-AC-TOTAL2 + WS-AC-TOTAL3.

ADD WS-VALOR1 WS-VALOR2 GIVING WS-AC-VALOR.

- Efetua: WS-AC-VALOR = WS-VALOR1 + WS-VALOR2.

SUBTRACT

Subtrai um ou mais operandos numéricos e armazena resultados.

Formato 1:

```
SUBTRACT { variável-1 ... constante-1 ...}
        FROM variável-de-resultado1 …
```

Formato 2:

```
SUBTRACT { variável-1 ... constante-1 ...}
        FROM variável-2|constante-2
        GIVING variável-de-resultado1 ...
```

Regras:

1. No Formato 1, o conjunto de operandos variável-1... constante-1, são subtraídos de variável de resultado1...

2. No Formato 2, a variável de resultado 1, armazenará o resultado de variável-2 constante-2 subtraídos de variável-1... constante-1...

3. Com o Formato 2, pode vir qualquer número de operandos imediatamente após a palavra SUBTRACT ou a palavra GIVING, mas depois da palavra FROM é permitido um único variável ou constante.

4. O campo variável de resultado1, após FROM ou GIVING, deve ser um nome de variável e não uma constante.

Exemplos:

SUBTRACT VL-CHEQUE FROM SALDO.

- Efetua: SALDO = SALDO – VL-CHEQUE.

SUBTRACT VL-CHEQUE FROM SALDO GIVING SALDO-ATUAL

- Efetua: SALDO-ATUAL = SALDO – VL-CHEQUE.

SUBTRACT TAXA FROM 100 GIVING COMPLEMENTO

- Efetua: COMPLEMENTO = 100 – TAXA.

DIVIDE

Efetua uma divisão, disponibilizando o quociente e, se indicado, o resto.

Formato 1:

```
DIVIDE {variável-1|constante-1}  INTO
       {variável-2|constante-2}
       [ GIVING variável-de-resultado …]
       [ REMAINDER variável-de-resto ]
```

Formato 2:

```
DIVIDE {variável-1|constante-1} BY
       {variável-2|constante-2}
       GIVINGvariável-de-resultado ...
       [ REMAINDER variável-de-resto ]
```

Regras:

1. Observar que GIVING é opcional com INTO, mas obrigatório com BY.

2. No Formato 1, variável-2 ou constante-2 é o dividendo e variável-1 ou constante-1 é o divisor. Se a opção GIVING não for utilizada, o resultado fica em variável-2(dividendo).

3. No Formato-1, se a opção GIVING não for utilizada, o dividendo terá de ser variável-2(não poderá ser usado constante-2).

4. No formato 2, variável-1 ou constante-1 é o dividendo e variável-2 ou constante-2 é o divisor.

5. A opção REMAINDER é utilizada quando se faz necessário guardar o resto da divisão em outro variável. Neste caso, a Picture da variável utilizada como resultado não poderá ter casas decimais.

Exemplos:

DIVIDE 2 INTO WS-NUMERO

- Efetua WS-NUMERO = WS-NUMERO / 2

DIVIDE WS-VL-DOLAR INTO WS-VL-REAIS
GIVING WS-RESULT

- Efetua WS-RESULT = WS-VL-REAIS / WS-VL-DOLAR

DIVIDE WS-NUM BY 2
GIVING WS-RESULT
REMAINDER WS-RESTO

- Efetua WS-RESULT = WS-NUM / 2 (resto em WS-RESTO)

MULTIPLY

Efetua a multiplicação entre variáveis.

Formato:

```
MULTIPLY {variável-1|constante-1} BY
              {variável-2|constante-2}
                                    [ [GIVING variável-de-resultado1 ...]
[END-MULTIPLY]  ]
```

Regras:

1. Observe a colocação das reticências (...). O resultado será armazenado em todos os variáveisderesultado após GIVING.

2. Se a opção GIVING não for utilizada, o resultado irá para variável-2, e neste caso não poderá ser usado constante-2.

Exemplos:

MULTIPLY 2 BY VALOR ROUNDED

- Efetua VALOR = 2 * VALOR (com arredondamento)

MULTIPLY VALOR BY 2 GIVING DOBRO

- Efetua DOBRO = VALOR * 2

MULTIPLY 2 BY VALOR GIVING DOBRO

Efetua DOBRO = 2 * VALOR

COMPUTE

Com a instrução COMPUTE, as operações aritméticas podem ser combinadas em fórmulas sem as restrições impostas para o campo receptor quando é usado ADD, SUBTRACT, MULTIPLY e DIVIDE. Quando as operações aritméticas são combinadas, a instrução COMPUTE é mais eficiente que as instruções aritméticas escritas em série.

Formato:

```
COMPUTEvariável-de-resultado [ROUNDED] =
        formula-aritmética
        [ [NOT] ON SIZE ERROR instrução-imperativa]
```

Regras:

1. A opção ROUNDED e ON SIZE ERROR segue a mesma regra utilizada para expressões aritméticas.

2. Os símbolos que podem ser utilizados em uma instrução COMPUTE, conforme sua ordem de prioridade de execução, são:
   ( ) Parênteses
   **** Exponenciação**
   *** Multiplicação**
   / Divisão
   + Adição
   - Subtração

3. O sinal de igual, assim como os símbolos aritméticos, devem ser precedidos e seguidos de um espaço. Assim, para calcular B+C+D**2 e colocar o resultado em A, use a seguinte instrução: COMPUTE A = B + C + D ** 2. Na abertura e fechamento de parênteses não é obrigatório o uso de espaço.

4. A ordem em que são executadas as operações em uma expressão aritmética que contenha mais de um operador segue a seguinte prioridade:

   1 º ( ) Expressões dentro de parênteses

   2 º Exponenciação

   3 º Multiplicação, divisão

   4 º Adições e Subtrações

☺ Quando houver operadores de mesma prioridade, eles serão executados da esquerda para a direita.

Exemplos:

COMPUTE WS-RESULT ROUNDED = (AA + BB) / CC

A fórmula A = $B^2 + C^2$

ficará em Cobol da seguinte forma:

COMPUTE A = (B ** 2 + C ** 2)

MOVIMENTAÇÃO DE CAMPOS

MOVE

A instrução MOVE transfere dados de uma área de memória para uma ou mais áreas. Se o campo receptor dos dados for numérico, ocorrera um alinhamento numérico, caso contrário, ocorrerá um alinhamento alfanumérico conforme as regras abaixo:

- Alinhamento alfabético/alfanumérico
  Os dados são acomodados no campo receptor alinhando-se da esquerda para a direita. Se o campo emissor for maior que o receptor, os BYTES mais a direita, em excesso, serão truncados no campo receptor. Se o emissor for menor que o receptor, os BYTES faltantes para preencher o campo receptor serão preenchidos com SPACES.

- Alinhamento Numérico
  Os dados são acomodados no campo receptor alinhando-se da direita para a esquerda. Se o campo emissor for maior que o receptor, os BYTES mais a esquerda do campo emissor serão truncados.

  - Se o emissor for menor que o receptor, os bytes faltantes para preencher o campo receptor serão preenchidos com zeros.

  - Os campos: EMISSOR e RECEPTOR podem ser itens de grupo ou elementares.
  - A clausula ALL (opcional) quando usada faz com que o campo emissor seja repetido Várias vezes no campo receptor até preenchê-lo completamente.

Exemplos de MOVE com constantes figurativas

Comando	Campo emissor (constante figurativa)	Campo receptor formato:	Conteúdo do campo receptor após o MOVE (expresso em hexa)
MOVE ZERO TO TOTSAL	ZERO	PIC 9(7) COMP-3	00.00.00.0C
MOVE ZEROS TO TOTSAL	ZEROS	PIC 9(7) COMP-3	00.00.00.0C
MOVE ZEROES TO TOTSAL	ZEROES	PIC 9(7) COMP-3	00.00.00.0C
MOVE ZERO TO TOTSAL	ZERO	PIC 9(5)	F0.F0.F0.F0.F0
MOVE ZEROS TO TOTSAL	ZEROS	PIC 9(5)	F0.F0.F0.F0.F0
MOVE ZEROES TO TOTSAL	ZEROES	PIC 9(5)	F0.F0.F0.F0.F0
MOVE SPACE TO SIGLA	SPACE	PIC X(02)	40.40
MOVE SPACES TO SIGLA	SPACES	PIC X(02)	40.40
MOVE HIGH-VALUES TO CHAVE-ARQUIVO	HIGH-VALUES	PIC X(03)	FF.FF.FF
MOVE LOW-VALUES TO CHAVE-ARQUIVO	LOW-VALUES	PIC X(03)	00.00.00
MOVE ALL '*' TO WS-MSG	ALL '*'	PIC X(05)	5C.5C.5C.5C.5C
MOVE ALL '*A' TO WS_MSG	ALL '*A'	PIC X(05)	5C.C1.5C.C1.5C

- **ZEROS** – O item (deve ser numérico) será preenchido com algarismos 0 (zero).
- **SPACES** – O item (deve ser alfabético ou alfa-numérico) será preenchido com espaços.
- **LOW-VALUES** - indica que este item na memória deve ter todos os seus bytes com todos os bits desligados. Um item nestas condições terá o menor valor possível em todas as comparações.
- **HIGH-VALUES** - indica que este item na memória deve ter todos os seus bytes com todos os bits ligados. Um item nestas condições terá o maior valor possível em todas as comparações.

Formatos:

MOVE	**nome-do-campo-entrada**	**TO**	**nome-do-campo-saida**
MOVE	**nome-do-registro-entrada**	**TO**	**nome-do-registro-saida**
MOVE	**nome-do-campo-entrada**	**TO**	**variável**

Regras:

1. A movimentação só é permitida para campos ou variáveis que possuam o mesmo formato.
2. Pode-se movimentar conteúdo numérico para as variáveis de edição.
3. Pode-se movimentar conteúdos numéricos inteiros para variáveis ou campos decimais

Exemplo:

```
PROCESSA    SECTION.

    INITIALIZE REG-ARQSAI.
    MOVE  ARQENT-NOME        TO    ARQSAI-NOME.
    MOVE  ARQENT-ENDERECO    TO    ARQSAI-ENDERECO.
    MOVE  ARQENT-TELEFONE    TO    ARQSAI-TELEFONE.
    MOVE  ARQENT-CODIGO      TO    ARQSAI-CODIGO.
    MOVE  ARQENT-IDADE       TO    ARQSAI-IDADE.
```

ESTRUTURA LÓGICA - DECISÃO

Comandos de Decisão

IF / ELSE / END-IF.

Formato:

```
IF CONDIÇÃO
        instruções (para condição verdadeira)
[ELSE
        instruções (para condição falsa)
END–IF]
```

Regras:

- Se o teste da condição foi VERDADEIRO, o bloco de instruções situado após o comando IF será executado, até que se encontre um PONTO, um ELSE ou um END-IF.
- Se o teste da condição foi FALSO será executado o bloco de instruções situado após o ELSE, até que se encontre um PONTO ou um END-IF. Não havendo ELSE dentro do contexto do IF(conjunto de instruções terminadas por PONTO ou END-IF), não serão executadas instruções para o teste de condição FALSO do IF.
- PONTO e END-IF indicam o fim da especificação da instrução IF. As instruções que estão após o PONTO e o END-IF, portanto, são executadas tanto para os casos de condição VERDADEIRO quanto para os casos de condição FALSO.

❑ **CONDIÇÃO**

Pode ser uma condição simples ou composta. As tabelas de operadores lógicos e relacionais abaixo podem ser utilizadas para compor a condição a ser testada.

Operadores Relacionais

Descrição	Em COBOL	
Igual	=	EQUAL
Diferente	NOT =	NOT EQUAL
Maior que	>	GREATER
Menor que	<	LESS
Maior ou igual a	>=	NOT LESS
Menor ou igual a	<=	NOT GREATER

Operadores Lógicos

Os operadores lógicos com exceção do **NOT** devem ser utilizados em condições compostas.

Os operadores lógicos são:

OPERADOR	DESCRIÇÃO
Not	**"NÃO":NOT**<condição VERDADEIRO> é igual a FALSO e **NOT**<condição FALSO> é igual a VERDADEIRO.
And	**"E"**: Condições associadas com AND resultam VERDADEIRO quando todas forem VERDADEIRO.
Or	**"OU"**: Condições associadas com **OR** resultam VERDADEIRO bastando apenas uma delas ser VERDADEIRO.

Categorias	Notação alternativa
Teste de sinal	IS POSITIVE
	IS NEGATIVE
Teste de classe	IS NUMERIC
	IS ALPHABETIC

Exemplos:
Suponha que temos três variáveis A = 5, B = 8 e C =1, os resultados das expressões seriam:

Expressões	Resultado
A = B AND B > C	Falso
A NOT = B OR B < C	Verdadeiro
NOT (A > B)	Verdadeiro
A < B AND B > C	Verdadeiro
A >=B OR B = C	Falso
NOT(A <= B)	Falso

A tabela abaixo mostra todos os valores possíveis criados pelos três operadores lógicos (AND, OR e NOT)

OPERADOR AND (E)		
A	**B**	**A AND B**
V	V	V
V	F	F
F	V	F
F	F	F

OPERADOR OR (OU)		
A	**B**	**A OR B**
V	V	V
V	F	V
F	V	V
F	F	F

OPERADOR NOT (NÃO)	
A	**NOT**A
V	F
F	V

CONTINUE

A instrução CONTINUE pode ser usada quando nada deve ser executado no caso da instrução IF avaliar VERDADEIRO, ou quando nada deve ser executado após a cláusula ELSE (condição FALSO).

Formato:

CONTINUE

Exemplo:

```
IF A > B
             { CONTINUE  }
ELSE
              DISPLAY   'A MENOR OU = B'
END-IF.
DISPLAY "COMPARAÇÃO EFETUADA".
```

Exemplo 1: (IF SEM ELSE)

```
IF WS-CAMPO1 EQUAL WS-CAMPO-2
     PERFORM 10-00-PESQUISA
END-IF.
```

Exemplo 2: (IF COM ELSE)

```
IF WS-CAMPO1 < WS-CAMPO-2
    ADD 1 TO WAC-CAMPO3
ELSE
   COMPUTE WS-TOTAL = WS-VALOR1 + WS-VALOR2
END-IF.
```

Exemplo 3:

```
IF    WS-CAMPO1 > WS-CAMPO-2
      CONTINUE
ELSE
      ADD WS-CAMPO4 TO WS-TOTAL
END-IF.
```

Exemplo 4: (NINHOS DE IF COM CONTINUE)

```
IF WS-CAMPO1 = WS-CAMPO-2
    MOVE "N" TO WS-CAMPO3 ]
       IF WS-CAMPO < WS-CAMPO4
            ADD 1 TO WS-CAMPO
            PERFORM 15-00-SITUAÇÃO1
       END-IF
       IF WS-CAMPO5 > WS-CAMPO6
            PERFORM  15-00-SITUAÇÃO2
       ELSE
                IF WS-CAMPO7 EQUAL "H"
                     CONTINUE
                END-IF
       END-IF
ELSE
    PERFORM 20-00-SITUAÇÃO3
END-IF.
```

USO DE FLAGS (NIVEL 88)

O uso de flags é muito comum na programação COBOL e ela é implementada com o nível 88 seguido de um nome de condição.

Um nome de condição é uma palavra definida pelo usuário, estabelecida na DATA DIVISION, dando um nome a um valor específico que pode ser assumido por um identificador. Por exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       WORKING-STORAGE SECTION.
       01 WS-ESTADO-CIVIL PIC X(01).
              88 SOLTEIRO            VALUE "S".
              88 CASADO              VALUE "C".
              88 DIVORCIADO          VALUE "D".
              88 VIUVO               VALUE "V".
```

Quando a variável WS-ESTADO-CIVIL for igual a “S”, chamamos essa condição de SOLTEIRO. O item de nível 88 não é o nome de uma variável, mas o nome de uma condição.

Refere-se especificamente ao item elementar imediatamente precedente. SOLTEIRO é um nome de condição aplicado a variável WS-ESTADO-CIVIL, uma vez que WS-ESTADO-CIVIL é imediatamente precedente ao item de número 88.

O nome de condição é sempre codificado no nível 88 e tem apenas uma cláusula VALUE associada a ele. Como o nome de condição não é um nome de campo, ele não tem uma cláusula PICTURE.

Então no lugar de testar a variável WS-ESTADO-CIVIL contra um conteúdo, por exemplo “S”, como mostra o código abaixo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8

       IF WS-ESTADO-CIVIL = "S"
              DISPLAY "O ESTADO CIVIL EH SOLTEIRO"
       END-IF
```

Podemos testar diretamente o nome da condição, como mostra o código abaixo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
          IF SOLTEIRO
              DISPLAY "O ESTADO CIVIL EH SOLTEIRO"
          END-IF
```

EVALUATE

Pode ser utilizado em substituição de ninhos de IF's sobre uma única variável.

Na instrução EVALUATE, a comparação de faixa só pode ser feita com a claúsula THRU, não podendo ser usados os operadores relacionais (=, <, >).

Formato:

```
EVALUATE variável
        WHEN valor-1 [THRU valor-2 …]
                Instrução- 1
        [WHEN OTHER
                Instrução- 2]
END-EVALUATE.
```

Exemplos:

Usando ninhos de IF's sobre uma única variável:

```
IF SIGLA-UF =  'SP'
    DISPLAY 'SÃO PAULO'
ELSE
    IF SIGLA-UF =  'SC'
          DISPLAY 'SANTA CATARINA'
    ELSE
          IF SIGLA-UF =  'RS'
                DISPLAY 'RIO GRANDE DO SUL'
          ELSE
                DISPLAY 'OUTRO ESTADO'
          END-IF
    END-IF
END-IF.
```

Usando EVALUATE para o mesmo propósito:

```
EVALUATE SIGLA-UF
    WHEN 'SP' DISPLAY 'SÃO PAULO'
    WHEN 'SC' DISPLAY 'SANTA CATARINA'
    WHEN 'RS' DISPLAY 'RIO GRANDE DO SUL'
    WHEN OTHER DISPLAY 'OUTRO ESTADO'
END-EVALUATE.
```

Usando EVALUATE com a cláusula THRU:

```
EVALUATE WS-SALDO
    WHEN ZEROS THRU 10000  DISPLAY "CLIENTE COMUM"
    WHEN 10001 THRU 20000  DISPLAY "CLIENTE ESPECIAL"
    WHEN > 20000  DISPLAY "CLIENTE SUPER-ESPECIAL"
END-EVALUATE.
```

Usando EVALUATE com a cláusula TRUE:

```
EVALUATE TRUE
    WHEN A>B DISPLAY "A EH MAIOR"
    WHEN B>A DISPLAY "B EH MAIOR"
END-EVALUATE.
```

ESTRUTURA LÓGICA – REPETIÇÃO

Toda estrutura de repetição envolve um teste de condição. De acordo com o resultado da condição, VERDADEIRO ou FALSO, serão executadas instruções, sendo que uma delas deverá configurar o teste de condição para o oposto do resultado verificado, se as instruções foram executadas para VERDADEIRO, esta instrução deverá configurar a condição para FALSO e vice-versa, sem a qual, esta estrutura entrará em looping. Após a execução de todas as instruções, será obrigatória uma instrução de desvio do fluxo lógico para este mesmo teste de condição.

Exemplo:

```
      MOVE ZEROS TO ACUM.
TESTE.
      IF ACUM =  10
            STOP RUN
      ELSE
            ADD 1 TO ACUM
            DISPLAY ACUM
            GO TO TESTE
      END-IF.
```

Programas bem projetados e estruturados são aqueles que possuem uma série de construções lógicas, em que a ordem na qual as instruções são executadas é padronizada. Em programas estruturados, cada conjunto de instruções que realiza uma função específica é definido em uma rotina ou um parágrafo. Ele consiste em uma série de instruções relacionadas entre si e em programação estruturada, os parágrafos são executados por meio da instrução PERFORM.

PERFORM

A instrução PERFORM permite que o controle passe temporariamente para um parágrafo diferente e depois retorne para o parágrafo original de onde a instrução PERFORM foi executada.

Há dois tipos de instrução PERFORM.

PERFORM OUT–LINE: Um parágrafo é especificado, ou seja, é dado um PERFORM em um PARÁGRAFO ou SECTION.

```
            PERFORM 010-INICIALIZAR
```

PERFORM IN–LINE – Executa as instruções que estão logo abaixo do comando PERFORM. Deve ser delimitado pela frase END-PERFORM.

Há 3 formatos de PERFORM IN-LINE.

> PERFORM com opção TIMES – executa as instruções uma determinada quantidade de vezes.

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       PERFORM 5 TIMES
       DISPLAY "*----------------------*"
       END-PERFORM
```

PERFORM com opção UNTIL – executa as instruções até que uma determinada situação ocorra. Se nada for informado, é feito o teste da condição ANTES de executar as instruções, mas é possível especificar se o teste deve ser feito antes ou depois com a opção WITH TEST BEFORE ou WITH TEST AFTER, respectivamente.

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
             MOVE 0 TO WS-CONT
             PERFORM UNTIL WS-CONT > 10
               DISPLAY "VALOR DO CONTADOR = " WS-CONT
               COMPUTE WS-CONT = WS-CONT + 1
             END-PERFORM
```

PERFORM com opção VARYING – executa as instruções variando o conteúdo de uma variável, de um valor inicial (FROM), incrementando um valor (BY) até que uma situação ocorra (UNTIL).

Exemplos

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
             PERFORM VARYING WS-CONT FROM 1 BY 1 UNTIL WS-CONT > 10
                  DISPLAY "VALOR DO CONTADOR = " WS-CONT
             END-PERFORM
```

É preciso tomar cuidado ao dividir a PROCEDURE DIVISION em seções, pois quando se utiliza o PERFORM para chamar uma seção, todos os parágrafos da seção são executados

LÓGICA ESTRUTURADA

Programação Estruturada é o resultado de um trabalho da IBM com o objetivo de padronizar as estruturas lógicas de programas.

Uma das premissas é que todo programa tem procedimentos iniciais, procedimentos repetitivos controlados por condições simples ou compostos e procedimentos finais.

Outra premissa descreve que é muito mais fácil dividir um problema em partes (módulos) logicamente conectadas entre si e desenvolver cada parte isoladamente do que resolver o problema como um todo.

Uma terceira premissa é a de eliminar comandos de desvio do fluxo lógico de execução (GO TO), substituindo-os por comandos de execução de módulos (PERFORM).

Baseado na primeira premissa, os três primeiros comandos de um programa COBOL são:

```
PERFORM INICIO.
PERFORM PROCESSAR UNTIL <condição>.
PERFORM FINALIZA.
```

Estes três comandos constituem o módulo principal do programa e controlam o fluxo de execução do mesmo.

Na **rotina inicio**, realizamos as seguintes tarefas:

- Inicialização das variáveis (contadores e acumuladores)
- Abertura dos arquivos de entrada e saída
- Leitura do primeiro registro dos arquivos de entrada

Na **rotina-processar**, realizamos as seguintes tarefas:

- Seleção de registros para o processamento, pelos operadores relacionais (=, <, >, >=, <= e NOT =) e operadores lógicos (AND, OR e NOT)
- Processamento dos campos numéricos pelos operadores aritméticos (+,-,*,/ e **)
- Movimentação de campos da entrada para a saída
- Gravação do registro de saída
- Leitura do próximo registro do arquivo de entrada

Na **rotina-finaliza**, realizamos as seguintes tarefas:

- Exibição de uma estatística de processamento, mostrando registros lidos, gravados, desprezados, selecionados, etc. (contadores e acumuladores)
- Fechamento dos arquivos de entrada e de saída
- Exibição de mensagem avisando do término normal do processamento

Exemplo:

```
*            PROGRAMA -  MAIOR E MENOR

IDENTIFICATION DIVISION.
PROGRAM-ID.    ALUNO01.
AUTHOR.       ALUNO.
DATA           DIVISION.
WORKING-STORAGE SECTION.
01 WS-NUMERO      PIC 9(03) VALUE ZEROS.
01 WS-MAIOR       PIC 9(03) VALUE ZEROS.
01 WS-MENOR      PIC 9(03) VALUE 999.

  PROCEDURE      DIVISION.

  PRINCIPAL.
  PERFORM INICIO.
  PERFORM PROCESSA UNTIL WS-NUMERO EQUAL ZEROS.
  PERFORM FINALIZA.
STOP RUN.

INICIO.
DISPLAY '            INICIO DO PROCESSAMENTO              '.
ACCEPT WS-NUMERO.

PROCESSA.
      DISPLAY 'O NUMERO EH =>'   WS-NUMERO.
      PERFORM MAIOR-MENOR.
      ACCEPT WS-NUMERO.
*
MAIOR-MENOR.
IF WS-NUMERO GREATER WS-MAIOR
           MOVE WS-NUMERO TO WS-MAIOR
      END-IF.
      IF WS-NUMERO LESS WS-MENOR
           MOVE WS-NUMERO TO WS-MENOR
END-IF.
*
FINALIZA.
      DISPLAY '            TERMINO DO PROCESSAMENTO              '.
      DISPLAY 'O MAIOR NUMERO EH  => '   WS-MAIOR.
    DISPLAY 'O MENORNUMERO EH  => '   WS-MENOR.
```

ARQUIVOS SEQÜENCIAIS

Arquivo é um meio de armazenar informações que foram processadas e que poderão ser utilizadas novamente. Para isso precisamos ver alguns conceitos de lógica.

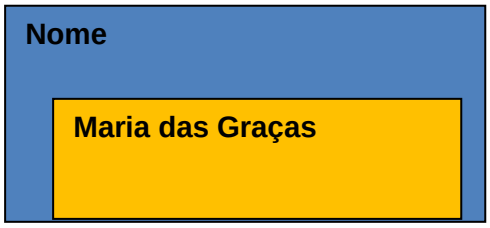
ARQUIVO é um conjunto de registros armazenados de forma seqüencial.
Exemplo: O arquivo de Clientes da Empresa, onde estão armazenados os dados de todos os clientes da empresa.

REGISTRO é um conjunto de campos.

Exemplo: Registro de Clientes (definem uma ocorrência da entidade Cliente).

COD-CLI	NOME	ENDEREÇO	FONE
00001	MARIA DAS GRAÇAS	RUA DAS DORES,1400	9888-9876

CAMPO é a menor parte da informação em memória.
Exemplo: Campo Nome, Campo Endereço



Os arquivos estudados neste modulo são arquivos seqüenciais utilizados para processamento BATCH. Os arquivos seqüenciais devem estar ordenados por um campo-chave afim de serem processados.

Os arquivos seqüenciais podem ser armazenados em disco ou fita magnética.Um arquivo seqüencial não admite leitura direta de um registro, regravação e exclusão de um registro.

HORA DE VOLTAR AO JCL

COMANDOS PARA PROCESSAMENTO DE ARQUIVOS SEQUENCIAIS.

Regras:

- Os arquivos devem ser declarados na INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION), com a instrução SELECT.
- O conteúdo do arquivo deve ser descrito na FILE SECTION (DATA DIVISION) com a clausura FD e definição do(s) registro(s) em nível 01.
- Na PROCEDURE DIVISION é necessário escrever instruções para gravar ou ler estes arquivos.

1.1 ENVIRONMENT DIVISION - INPUT-OUTPUT SECTION

Esta seção destina-se a declarar os arquivos que o programa usa.

- **FILE-CONTROL – Cláusula SELECT**

No parágrafo FILE-CONTROL, usado para definir arquivos, usamos uma instrução SELECT para declarar cada um dos arquivos usados pelo programa.

Formato:

SELECT nome-arquivo-lógico ASSIGN TO nome-arquivo-fisico.
FILE-STATUS is FS- nome-arquivo-lógico

Nome-arquivo-logico.

É o nome do arquivo que será usado dentro do programa, pode ser diferente do nome físico e pode ter até 30 caracteres.

Nome-arquivo-fisico.

O nome externo pode ter no máximo 8 caracteres e será usado no JOB no tipo de cartão DD, para associá-lo a um arquivo físico, chamado de DDNAME.

Exemplo:

//DDNAME DD DSN= Nome do Arquivo Físico

No exemplo abaixo mostramos a ENVIRONMENT DIVISION de um programa que irá acessar um arquivo CLIENTES.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT CADCLI ASSIGN TO CLIENTES
        FILE STATUS IS FS-CADCLI.
```

Neste exemplo escolhemos como ***nome-arquivo-lógico*** dentro da instrução SELECT a palavra **CADCLI. No JCL o DDNAME (arquivo fisico) deverá ser CLIENTES.**

//CLIENTES DD DSN=GR.ALUNO.CLIENTES,DISP=SHR

CADCLI - será usado em todos os comandos do programa como referência para este arquivo.

DATA DIVISION - FILE SECTION

A FILE SECTION é a Section usada para detalhar o arquivo e o conteúdo dos registros dos arquivos que o programa irá ler/gravar. Isto é feito usando a clausula FD (FILE DESCRIPTION).

FD (FILE DESCRIPTION)

```
FD    nome-do-arquivo.
  01    nome-do-registro. ...
        [03   nome-de-campo ...]
```

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       DATA                    DIVISION.
       FILE                   SECTION.
       FD  ESTOQUE-PRODUTO
           LABEL RECORD STANDARD
           BLOCK CONTAINS 0 RECORDS
           RECORDING MODE IS F
           RECORD CONTAINS 44 CHARACTERS
           .
       01  REG-ESTOQUE-PRODUTO.
           05 CODPROD    PIC X(04).
           05 DESCPROD   PIC X(25).
           05 QTDEST     PIC 9(03).
           05 QTDMIN     PIC 9(03).
           05 QTDMAX     PIC 9(03).
           05 PRECOPROD  PIC 9(04)V99.
```

FD significa FILE DESCRIPTION (descrição do arquivo) e deve ser seguida do nome lógico do arquivo. Deve ser escrito na margem A.

LABEL RECORD STANDARD indica a utilização de etiquetas (label) padronizadas.
É uma cláusula opcional. O label de um arquivo consiste de um registro especial, que contém campos como: nome do arquivo, data de criação, o tempo que deve durar, local do arquivo no disco, informações sobre código de acesso ao arquivo para evitar acessos não autorizados, etc. Labels de arquivos em fitas precedem e sucedem o próprio arquivo na fita. Label de arquivos em discos são armazenados na VTOC (Volume Table of Contents). A utilização de labels garantem que os programas processem os registros corretos.

BLOCK CONTAINS 0 RECORDS indica a quantidade de registros lógicos que existem em um registro físico (bloco). Isso é conhecido como blocagem do arquivo.

O FATOR DE BLOCO é a quantidade de registros lógicos dentro de um registro físico, neste exemplo o FATOR DE BLOCO é 5.
Quando não sabemos o fator de bloco, colocamos 0 na cláusula BLOCK CONTAINS, assim, o próprio sistema determina o fator de bloco.

RECORDING MODE IS F indica que todos os registros serão gravados com o mesmo tamanho. O F indica FIXED (fixo). **É uma cláusula obrigatória.**

Observação:
Não usar value dentro da sintaxe...
Utilizar RECORDING MODE IS F.

ABERTURA DE ARQUIVOS

A linguagem COBOL exige que todo arquivo antes do processamento de leitura, seja aberto como entrada e que todo arquivo, antes da gravação de seus registros, seja aberto como saída.

OPEN

Todo arquivo antes de ser processado deve ser aberto pelo comando OPEN. Este comando abre o contato com o dispositivo físico do arquivo e reserva, na memória (WORKING-STORAGE SECTION), áreas necessárias para a troca de dados entre o computador e o dispositivo externo. Indica quais arquivos serão de entrada e quais serão de saída.

Há 3 modos de abertura de arquivo, que são INPUT, OUTPUT e EXTEND.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           OPEN INPUT CADCLI
           OPEN OUTPUT VCADPRD
           OPEN EXTEND CADALUN
```

• INPUT indica que o arquivo será lido pelo programa. Não é possível gravar um arquivo aberto como INPUT.

• OUTPUT indica que o arquivo será gravado e é vazio no momento da abertura. Não é possível ler um arquivo aberto como OUTPUT.

• EXTEND indica que o arquivo será gravado e não é vazio no momento da abertura, ou seja, serão acrescentados registros após o último.

O nome do arquivo utilizado no comando OPEN é o nome lógico do arquivo, que aparece logo após o SELECT da FILE-CONTROL.

É possível abrir todos os arquivos com um único comando OPEN, como mostra o exemplo abaixo.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           OPEN INPUT CADCLI
                      MOVCLI
                OUTPUT NEWCAD
                      RELATO
```

A recomendação, porém, é abrir um arquivo de cada vez e testar o FILE STATUS para saber se houve sucesso ou não na operação.

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           OPEN INPUT CADCLI
           IF                    FS-CADCLI NOT = "00"
              MOVE "ERRO ABERTURA CADCLI"   TO WS-MSG
              MOVE FS-CADCLI     TO WS-FS


              GO TO 999-ERRO
           END-IF
```

No exemplo anterior, FILE STATUS IS FS-CADCLI , indica uma variável, com formato PIC X(02), que conterá um código de FILE STATUS, indicando se a operação de I-O (Input-Outupt), tais como OPEN, READ, WRITE e CLOSE foi realizada com sucesso.

Essa variável, uma vez indicada, é carregada automaticamente pelo sistema operacional, toda vez que uma operação de I-O for realizada.

A **boa prática de programação** sugere que todos os comandos de I-O devem ser seguidos de um teste na variável de FILE STATUS para verificar possíveis erros. A variável de FILE STATUS pode receber, entre outros, os seguintes valores:

FILE STATUS	**SIGNIFICADO**
'00'	SUCCESSFUL COMPLETION
'10'	END OF FILE
'35'	OPEN, FILE NOT PRESENT
'39'	OPEN, FILE ATTRIB CONFLICTING
'41'	OPEN, FILE IS OPEN
'42'	CLOSE, FILE IS CLOSED
'46'	SEQUENTIAL READ WITHOUT POSITIONING
'47'	READING FILE NOT OPEN AS INPUT/IO/EXTEND
'48'	WRITE WITHOUT OPEN IO
'9x'	PERMISSION NOT OK

GRAVAÇÃO DE REGISTROS

WRITE

A gravação consiste na transferência de um registro da memória, para o arquivo. A gravação é feita registro a registro. Cada novo comando de gravação grava o conteúdo do registro da memória em seguida ao último registro gravado no arquivo.

A instrução WRITE grava um registro após o último registro gravado em um arquivo de acesso seqüencial.

Formatos:

WRITE nome-de-registro-1 (Necessita movimentação dos campos)

WRITE nome-de-registro-1 [FROM variável-1].

Regras:

1. O arquivo de acesso seqüencial associado à instrução WRITE deve ser aberto no modo OUTPUT ou EXTEND.

2. **nome-de-registro-1**: Deve ser o nome do registro lógico (nível 01) da FD na DATA DIVISION.

3. **FROM variável-1**: O conteúdo do variável-1 é copiado para o nome-de-registro-1 antes de ocorrer a gravação. Depois da execução com sucesso da instrução WRITE, o registro continua disponível no variável-1. Precisa existir uma estrutura de varíaveis ESPELHO.

Exemplo:

```
*=============================================================
*      ROTINA DE GRAVACAO DO ARQUIVO DE SAIDA                 *
*=============================================================
 GRAVA-SAIDA SECTION.

     WRITE  REG-ARQSAI.

     IF  WS-FS-ARQSAI  NOT EQUAL  ZEROS
         DISPLAY '======================================='
         DISPLAY ' ERRO NA GRAVACAO DO ARQSAI            '
         DISPLAY ' FILE STATUS  = ' WS-FS-ARQSAI
         DISPLAY '======================================='
         STOP RUN.

      ADD    1      TO   WS-CONT-GRAVA.

GRAVA-SAIDA-FIM.
      EXIT.
```

FECHAMENTOS DE ARQUIVOS

CLOSE

Fecha o arquivo especificado. Todos os arquivos abertos devem ser fechados.
É possível fechar todos os arquivos com um único comando CLOSE, como mostra o exemplo abaixo.

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           CLOSE CADCLI MOVCLI NEWCLI
```

Porém a recomendação é testar o FILE STATUS para verificar o sucesso da operação. Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           CLOSE CADCLI
           IF FS-CADCLI NOT = "00"
               MOVE "ERRO FECHAMENTO CADCLI" TO WS-MSG
               MOVE FS-CADCLI  TO WS-FS
               GO TO 999-ERRO
           END-IF
```

LEITURA / GRAVAÇÃO DE ARQUIVOS SEQÜENCIAIS

READ

A instrução READ obtém um registro lógico de um arquivo. A cada novo comando READ, o próximo registro lógico do arquivo será lido. Após cada comando READ, todos os campos descritos na FD do arquivo estarão preenchidos com os valores do registro lido.

Quando a instrução READ for executada, o arquivo associado deve estar aberto no modo INPUT.

Formato:

READ nome-arquivo-logico

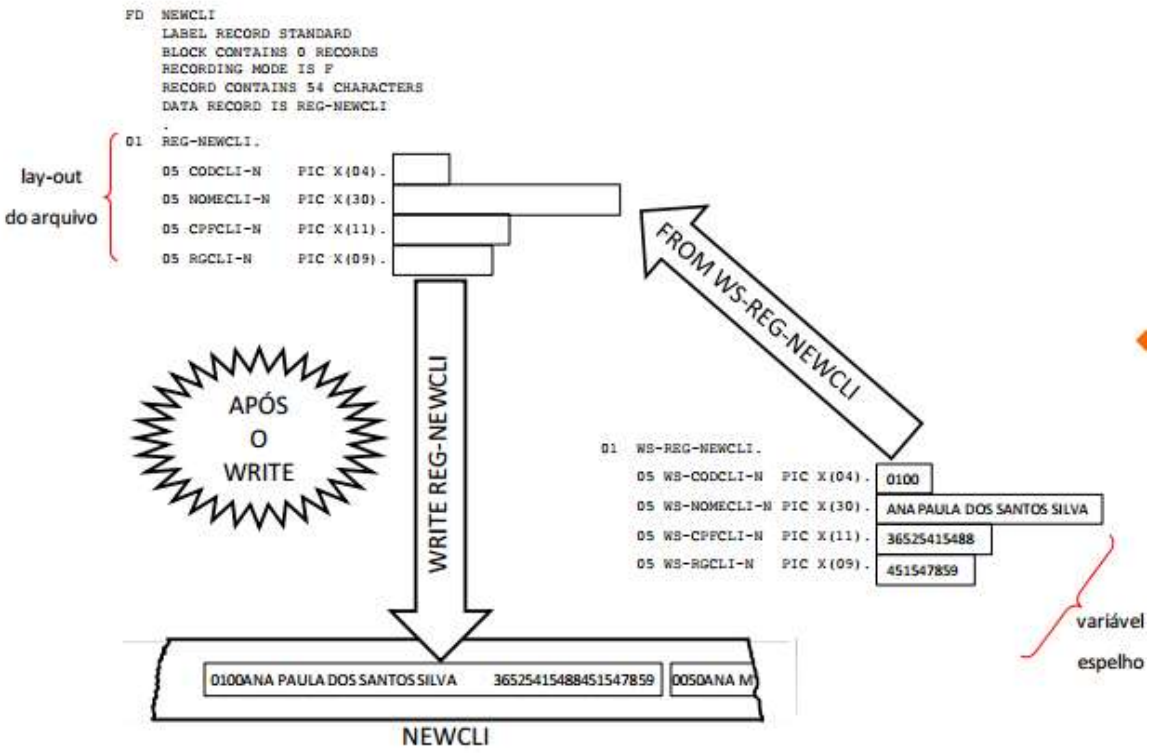
Exemplo:

```
    *=============================================================
    *     ROTINA DE LEITURA DO ARQUIVO DE ENTRADA            *
*=============================================================
     LER-ARQENT  SECTION.

        READ  ARQENT.

        IF  WS-FS-ARQENT  NOT EQUAL  ZEROS  AND  10
           DISPLAY '======================================='
           DISPLAY ' ERRO NO READ DO ARQUIVO ARQENT       '
           DISPLAY ' FILE STATUS  = ' WS-FS-ARQENT
           DISPLAY '======================================='
           STOP RUN.

        ADD  1   TO   WS-CONT-LIDOS.
```

WRITE com váriavel ESPELHO

Faz a inclusão de um registro lógico no arquivo. Os dados podem vir da variável espelho especificado na cláusula FROM.

O WRITE **move** os dados do layout do arquivo LÓGICO para o arquivo FISICO, portando o WRITE é destrutivo, porém os dados armazenados na variável espelho não são apagadas.



A recomendação é testar o FILE STATUS para saber se houve sucesso ou não na operação. Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
           WRITE REG-NEWCLI FROM WS-REG-NEWCLI
           IF FS-NEWCLI NOT = "00"
               MOVE "ERRO GRAVAVAO NEWCLI" TO WS-MSG
               MOVE FS-CADCLI TO WS-FS
               GO TO 999-ERRO
           END-IF
```

INICIALIZAÇÃO DE CAMPOS E CONJUNTO DE VARIÁVEIS

INITIALIZE

Efetua a inicialização (atribuição de valores) de uma variável (ou um conjunto de variáveis).

- Como default variáveis numéricas são inicializadas com zeros e variáveis alfanuméricas são inicializadas com espaços.

- Se a variável especificada for um item de grupo, todos os seus subitens serão inicializados de acordo com seu formato: os que forem numéricos, serão inicializados com zero (respeitando-se seu formato : zonado, compactado ou binário); se a variável for alfa-numérica ou alfabética, ela será inicializada com espaços.

Formato:

```
INITIALIZE variável ou Nível "01" do registro.
```

Exemplos:

```
    *=============================================================
    *     ROTINA DE PROCESSAMENTO                    *
    *=============================================================
     PROCESSA    SECTION.

           INITIALIZE REG-ARQSAI.
        MOVE  ARQENT-NOME          TO    ARQSAI-NOME.
        MOVE  ARQENT-ENDERECO   TO    ARQSAI-ENDERECO.
        MOVE  ARQENT-TELEFONE    TO   ARQSAI-TELEFONE.
        MOVE  ARQENT-CODIGO       TO    ARQSAI-CODIGO.
        MOVE  ARQENT-IDADE         TO    ARQSAI-IDADE.
```

ENCERRAMENTO DE PARÁGRAFOS

EXIT

A instrução EXIT provê um ponto de encerramento comum para uma serie de parágrafos. A instrução EXIT não tem efeito na compilação nem na execução do programa. É, portanto, usado com a finalidade de documentar o programa.

A instrução EXIT deve ser única dentro do seu parágrafo.

Formato:

```
EXIT.
```

Exemplo:

```
     ROT-MESTRE  SECTION.
        PERFORM  INICIO    THRU   INICIO-FIM.
        PERFORM  PROCESSA   THRU   PROCESSA-FIM   UNTIL
              WS-FS-ARQENT = 10.
        PERFORM  FINALIZA   THRU   FINALIZA-FIM.
STOP RUN.
     ROT-MESTRE-FIM.
EXIT.
```

TABELAS DE MEMÓRIA – CLÁUSULA OCCURS

TABELAS – OCCURS

Alguns algoritmos mais avançados exigem a definição de uma mesma variável várias vezes, aumentando o trabalho de codificação do programa correspondente tanto na DATA DIVISION, como também as instruções resultantes na PROCEDURE DIVISION.

Por exemplo, em um algoritmo para acumular as vendas do ano separadas por mês, precisamos definir 12 campos de total na DATA DIVISION, e a PROCEDURE DIVISION deverá ter 12 testes do mês da venda para decidir em que total deve ser feito a soma.

Exemplo:

```
DATA DIVISION.
03 TOTAL-01      PIC 9(8)V99.
03 TOTAL-02      PIC 9(8)V99.
...
03 TOTAL-12      PIC 9(8)V99.
PROCEDURE DIVISION.
          …
IF MES = 01
   ADD VENDAS TO TOTAL-01
ELSE
IF MES = 02
        ADD VENDAS TO TOTAL-02
ELSE
...
IF MES = 12
        ADD VENDAS TO TOTAL-12
```

A linguagem COBOL possui um recurso para resolver este problema. Na DATA DIVISION a variável será definida somente uma vez, acompanhada da cláusula OCCURS que definirá quantas vezes a variável deve ser repetida. A sintaxe da definição do item com OCCURS é:

Formato:

```
NÍVEL variável-1        PIC 9,X OU A OCCURS n [TIMES].
```

Regras:

1. A cláusula OCCURS só pode ser usada em variáveis de nível 02 a 49.
2. Quando uma variável de uma tabela (definida com OCCURS) for usada na PROCEDURE DIVISION, ela precisa ser acompanhada de um indexador (subscrito) que definirá qual ocorrência da tabela está sendo referida. Este subscrito deve estar dentro de parênteses e pode ser um literal numérico ou uma variável numérica com valores inteiros.

Por ex: **ADD VENDAS TO TOTAL-MENSAL(5).** Neste caso a soma esta sendo feita na quinta ocorrência de total-mensal.

Exemplo:

A codificação do algoritmo do exemplo acima ficará reduzida agora a:

```
DATA DIVISION.
01    TOTAIS-GERAIS.
        03  TOTAL-MENSAL      PIC 9(8)V99   OCCURS 12 TIMES.
...
 PROCEDURE DIVISION.
     ...
     ADD VENDAS TO TOTAL-MENSAL (MES-VENDA).
```

1.2 NÍVEIS DE TABELAS

Em COBOL podemos definir um item de uma tabela como uma nova tabela, e assim sucessivamente até um nível de 3 tabelas. Por exemplo, para obter o total de vendas separado por estado, e em cada estado por tipo de produto, e para cada produto por mês de venda, montaremos a DATA DIVISION como abaixo:

```
DATA DIVISION.
01 TOTAIS-VENDA.
03 VENDAS-ESTADO   OCCURS 27 TIMES.
    05   VENDAS-PRODUTO   OCCURS 5 TIMES.
         07   VENDAS-MÊS    PIC 9(8)V99 OCCURS 12 TIMES.
```

Este código montará na memória uma tabela com 3 níveis de 1620 totais (27 estados X 5 produtos X 12 meses). Para acessar um total desta tabela será necessário um conjunto de 3 indexadores:

```
        PROCEDURE DIVISION.
        ....
            ADD VENDAS TO
            VENDAS-MÊS (CD-ESTADO, CD-PRODUTO, MÊS-VENDA).
```

Importante:

Os indexadores dentro dos parênteses devem estar na mesma seqüência da definição das tabelas (mesma hierarquia).

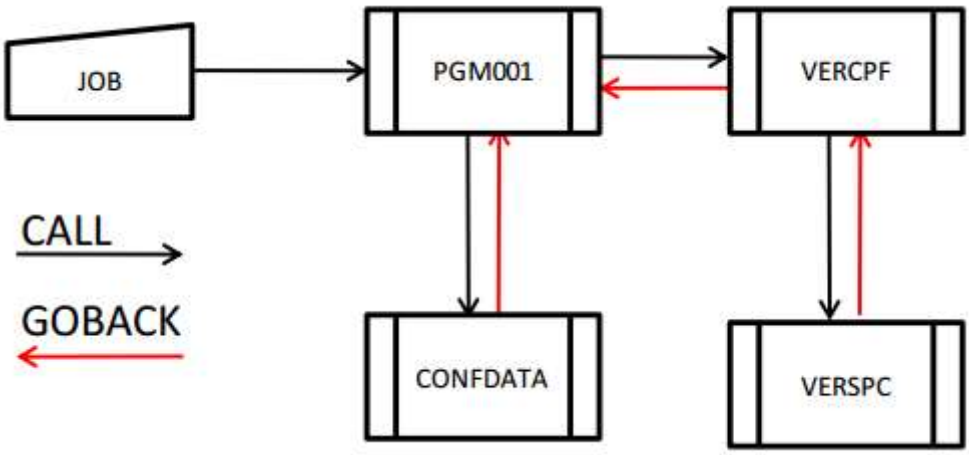
CHAMADAS A OUTROS PROGRAMAS

Um programa COBOL pode chamar outros programas para execução, passando dados que podem ou não serem alterados. Esse é um procedimento muito comum.

Por exemplo, um programa que verifica a consistência do número do CPF é feito uma única vez e todos os programas que precisam verificar se um número de CPF é correto podem chamá-lo, passando o número e recebendo como resposta se ele é correto ou não.

O Comando CALL chama outro programa para execução. O programa que emite o CALL é o programa PRINCIPAL e o programa que é executado a partir deste CALL é o SUB-PROGRAMA. **O SUB-PROGRAMA não pode terminar com STOP RUN,** mas sim com um GOBACK ou EXIT PROGRAM.

A figura abaixo mostra outro exemplo de chamada de programas. Um JOB inicializa um programa, que pode chamar sequencialmente um ou mais programas, estes por sua vez, podem chamar outros e etc.



Em geral, nos ambientes de desenvolvimento, existem grandes bibliotecas de SUB-PROGRAMAS para uso pelos programadores, objetivando a modularização dos sistemas, reuso de software, otimização do desenvolvimento, etc.

O Comando CALL chama um SUB-PROGRAMA, que pode ou não receber dados do programa PRINCIPAL, por referência (quando o endereço do dado é passado e o SUB-PROGRAMA pode alterar esse dado) ou por valor (quando o SUBPROGRAMA recebe uma cópia do dado e não pode alterar o dado original).

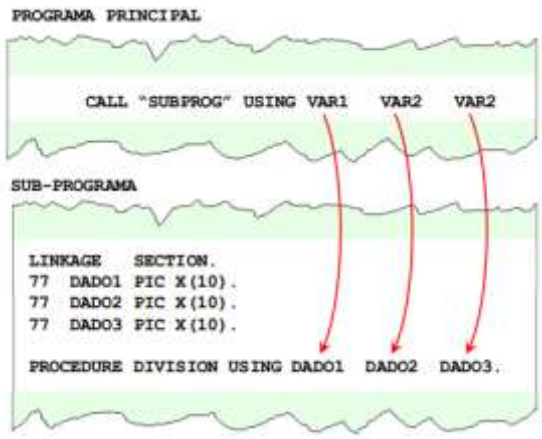
Chamada por Referência – No momento do CALL o endereço do item de dado é passado ao SUB-PROGRAMA. Isso é transparente para o programador. O valor do item de dado pode ser alterado pelo SUB-PROGRAMA e o programa PRINCIPAL pode usar esse novo valor. A cláusula **BY REFERENCE** é default e não é obrigatório informá-la.

```
CALL 'VERCPF' USING BY REFERENCE WS-DADOS-ENVIADOS
     ou
CALL 'VERCPF' USING WS-DADOS-ENVIADOS
```

Chamada por valor – Neste caso o SUB-PROGRAMA recebe uma cópia do item de dado e, mesmo que esse valor seja alterado, o programa PRINCIPAL temacesso apenas ao dado original. A cláusula **BY CONTENT** não é default e deve ser informada sempre que desejarmos realizar uma chamada por valor.

```
CALL 'VERCPF' USING BY CONTENT WS-DADOS-ENVIADOS
```

Os dados são carregados no SUB-PROGRAMA nas variáveis declaradas na LINKAGE SECTION. A mesma quantidade de variáveis usadas no CALL USING deve ser declarada na LINKAGE SECTION. Essas mesmas variáveis devem também ser informadas na PROCEDURE DIVISION com a cláusula USING, como mostra a figura abaixo. As variáveis podem também estar separadas por vírgula



CALL ESTÁTICO – O SUB-PROGRAMA é chamado como uma constante, com seu nome informado entre aspas ou apóstrofe. O programa PRINCIPAL e o SUBPROGRAMA fazem parte de um mesmo módulo (LOAD MODULE), assim quando o controle passa para o SUB-PROGRAMA ele já está na memória e as chamadas subsequentes do SUB-PROGRAMA o encontram no último estado, exceto quando o programa chamado possui o atributo **IS INITIAL** no parágrafo PROGRAM-ID.

Se você alterar e compilar o SUB-PROGRAMA, o programa principal também precisará ser recompilado, pois ele tem uma cópia anterior do sub-programa. Isso pode se tornar um problema, pois se um programa muito utilizado for sempre chamado de forma estática por diversos programas e precisar sofrer alteração, todos os demais programas precisam ser recompilados.

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
*---------------------------------------------------------------*
WORKING-STORAGE                                    SECTION.
*---------------------------------------------------------------*
01 WS-DADOS-ENVIADOS.

                05 WS-NUMERO-CPF     PIC X(11).
                05 WS-RESULTADO      PIC X(01).

*---------------------------------------------------------------*
PROCEDURE DIVISION.
*---------------------------------------------------------------*
050-CHAMA-PROGRAMA-CPF.
       MOVE "15524514585"                TO WS-NUMERO-CPF
       MOVE SPACES TO WS-RESULTADO
O NOME DO SUB-PROGRAMA ESTA ENTRE ASPAS
       CALL "VERCPF" USING WS-NUMERO-CPF WS-RESULTADO
```

CALL DINÂMICO – O nome do SUB-PROGRAMA é carregado em uma variável, que é usada para emitir o CALL. O SUB-PROGRAMA é um módulo independente e só é carregado na memória quando é chamado. As chamadas sucessivas encontram o SUB-PROGRAMA no último estado, exceto quando o programa chamado possui o atributo **IS INITIAL** no parágrafo PROGRAM-ID.

Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
*-----------------------------------------------------------------*
PROCEDURE DIVISION.
*-----------------------------------------------------------------*
050-CHAMA-PROGRAMA-CPF.
      MOVE "VERCPF" TO WS-PROGRAMA
      MOVE "15524514585" TO WS-NUMERO-CPF
      MOVE SPACES TO WS-RESULTADO
       O NOME DO SUB-PROGRAMA ESTA NA VARIAVEL WS-PROGRAMA
       CALL WS-PROGRAMA USING WS-NUMERO-CPF  WS-RESULTADO
```

É possível detectar a falha do CALL com a opção ON EXCEPTION, como mostra o exemplo abaixo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       CALL "PROGRAMA" USING WS-DADOS
            ON EXCEPTION DISPLAY "FALHA NA CHAMADA AO PROGRAMA"
       END-CALL.
```

GOBACK ou EXIT PROGRAM

Finaliza o SUB-PROGRAMA e devolve o controle ao PROGRAMA PRINCIPAL, executando a próxima instrução após o comando CALL. Exemplo:

```
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
*-----------------------------------------------------------------*
LINKAGE SECTION.
*-----------------------------------------------------------------*
01 LS-DADOS-RECEBIDOS.
                05 LS-NUMERO-CPF     PIC X(11).
                05 LS-RESULTADO      PIC X(03).

*-----------------------------------------------------------------*
PROCEDURE DIVISION USING LS-DADOS-RECEBIDOS.
*-----------------------------------------------------------------*
080-VERIFICA-CPF.
*-----------------------------------------------------------------*
* SUPONDO QUE O CPF ESTAVA ERRADO MOVEREMOS NOK PARA LS-RESULTADO
*-----------------------------------------------------------------*
       MOVE "NOK" TO LS-RESULTADO
       GOBACK
.
```

Se um programa, que não foi chamado por nenhum outro programa, possuir a instrução EXIT PROGRAM, essa instrução será ignorada e a próxima instrução após o EXIT PROGRAM será executada.

Já o GOBACK realmente encerra o programa, não importando se há mais instruções no mesmo parágrafo e abaixo dele, devolvendo o controle ao programa principal. Algumas empresas preferem usar GOBACK no lugar de STOP RUN, fazendo com que qualquer programa possa ser chamado por outro.