1 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 CODASYL foi formado para recomendar as diretrizes de uma linguagem para negócios. Foi constituído por membros representantes de seis fabricantes de computadores e três órgãos governamentais, a saber: Burroughs Corporation, IBM, Minneapolis-Honeywell (Honeywell Labs), RCA, Sperry Rand, e Sylvania Electric Products, e a Força Aérea dos Estados Unidos, o David Taylor Model Basin e a Agência Nacional de Padrões (*National Bureau of Standards* ou NBS). Este comitê foi presidido por um membro do NBS. Um comitê de Médio Prazo e outro de Longo Prazo foram também propostos na reunião do Pentágono. Entretanto, embora tenha sido formado, o Comitê de Médio Prazo nunca chegou a funcionar; e o Comitê de Longo Prazo nem chegou a ser formado. Por fim, um subcomitê do Comitê de Curto Prazo desenvolveu as especificações da linguagem COBOL.

O COBOL foi definido na especificação original, possuia excelentes capacidades de autodocumentação, bons métodos de manuseio de arquivos, e excepcional modelagem de dados para a época, graças ao uso da cláusula PICTURE para especificações detalhadas de campos. Entretanto, segundo os padrões modernos de definição de linguagens de programação, tinha sérias deficiências, notadamente sintaxe prolixa e falta de suporte de variáveis locais, recorrência, alocação dinâmica de memória e programação estruturada. A falta de suporte à linguagem orientada a objeto é compreensível, já que o conceito era desconhecido naquela época.

O COBOL possui muitas palavras reservadas, e é difícil evitar de usar alguma inadvertidamente sem o uso de alguma convenção, como adicionando um prefixo a todos os nomes de variáveis. A especificação original do COBOL suportava até código auto-modificável através do famoso comando "ALTER X TO PROCEED TO Y". Entretanto, a especificação do COBOL foi redefinida de tempos em tempos para atender a algumas das críticas, e as últimas definições do COBOL corrigiram muitas destas falhas, acrescentando estruturas de controle melhoradas, orientação a objeto e removendo a possibilidade de codificação auto-modificável.

O COBOL provou ser durável e adaptável. 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.

No Brasil a área financeira e de seguros são os principais mercados de COBOL e está aquecido devido grandes compras e fusões das instituições.

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:

- **Byte:** Conjunto de 8 bits que formam uma posição de memória.

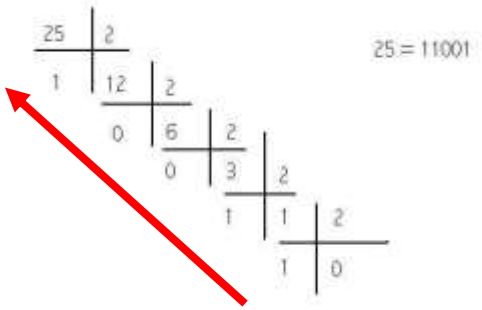
bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7

Conversão de números Decimais para números Binários:

A conversão do número inteiro, de decimal para binário, será feita da direita para a esquerda, isto é, determina-se primeiro o algarismos das unidades (o que vai ser multiplicado por 2^0) , em seguida o segundo algarismo da direita (o que vai ser multiplicado por 2^1) etc...

A questão chave, por incrível que pareça, é observar se o número é par ou ímpar. Em binário, o número par termina em 0 e o ímpar em 1. Assim determina-se o algarismo da direita, pela simples divisão do número por dois; se o resto for 0 (número par) o algarismo da direita é 0; se o resto for 1 (número ímpar) o algarismo da direita é 1.

Vamos converter 25 de decimal para binário.



Para saber o resultado em binário, basta verificar os restos das divisões de baixo para cima.

Exemplo:

Decimal	Binário
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

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:**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.

- **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 como se chama, na Ciência da Computação de linguagens de programação, uma 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.

2 INTERPRETAÇÃO DOS FORMATOS NA APOSTILA

1. Palavras sublinhadas são obrigatórias.
2. O símbolo | (pipe) indica que apenas uma das palavras é obrigatória.
3. Palavras em letras maiúsculas são palavras reservadas do COBOL.
4. A palavra "variável" significa um campo ou registro definido na DATA DIVISION.
5. As Chaves {} significam que uma das palavras em seu interior, separadas ou não pelo símbolo | (pipe), é obrigatória.
6. As reticências ou pontos (...) indicam que dois ou mais campos ou literais podem ser especificados.
7. A palavra "literal" significa uma constante numérica ou alfanumérica.
8. As palavras entre [] significam que são palavras reservadas do COBOL, mas sua utilização é opcional para o comando.

2.1 INDENTAÇÃO

O processo de indentação 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 :

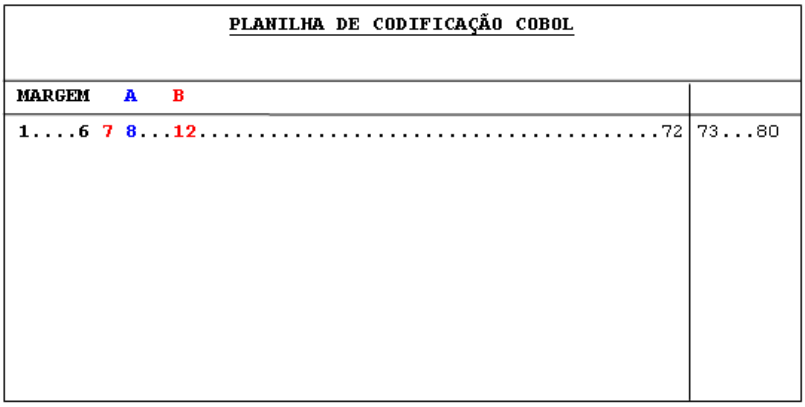
```
IF condição
IF condição
COMPUTE A = (B * C ) ** 4
COMPUTE C = A / 0,005
ELSE
COMPUTE A = (B * C ) ** 8
COMPUTE C = A / 0,055
END-IF
ELSE
IF condição
COMPUTE A = (B * C ) ** 5
COMPUTE C = A / 0,007
ELSE
COMPUTE A = (B * C ) ** 9
COMPUTE C = A / 0,007
END-IF
END-IF.
```

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.
```

2.2 FORMATO DO FONTE COBOL

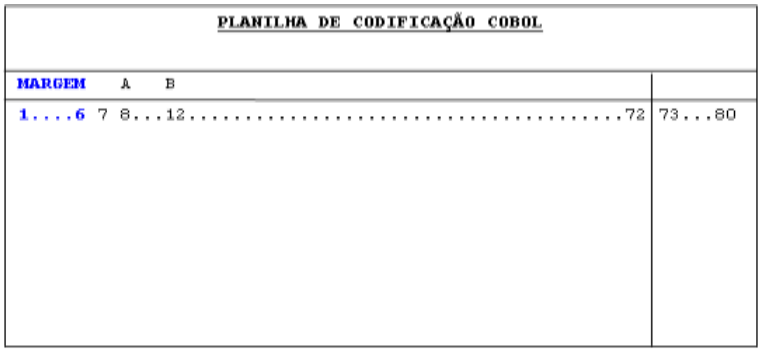
Todo programa escrito na linguagem COBOL possui algumas regras a serem seguidas. Uma destas regras se refere ao formato das linhas de comando (instruções) dentro do seu editor de fonte. Uma linha de comando COBOL pode ter até 65 caracteres, conforme o formato abaixo:



Colunas de 1 a 6:	Área de numeração seqüencial de linhas do editor
Coluna 7:	Área de indicação de comentários ou continuação
Colunas de 8 a 11:	Área A ou Nível A para codificação das palavras da linguagem
Colunas de 12 a 72:	Área B ou Nível B para codificação dos comandos da linguagem
Colunas de 73 a 80:	Não utilizadas na programação

AREA DE NUMERAÇÃO SEQÜENCIAL (COLUNAS DE 1 A 6)

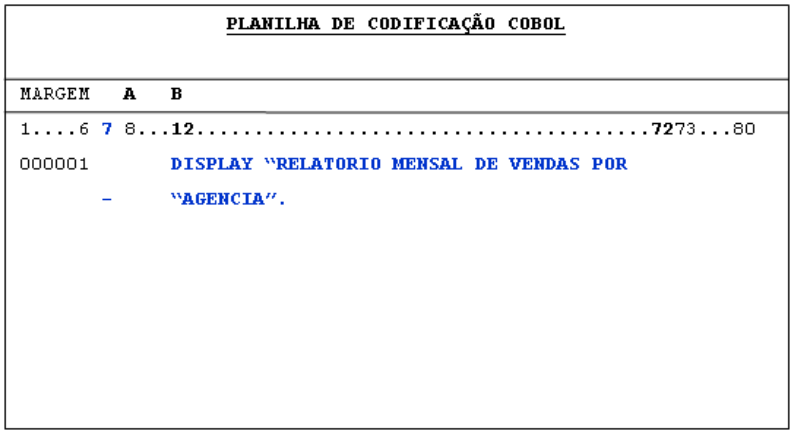
Normalmente consiste em seis dígitos em ordem crescente que são utilizados para numerar as linhas do programa fonte.



ÁREA DE INDICAÇÃO(COLUNA 7)

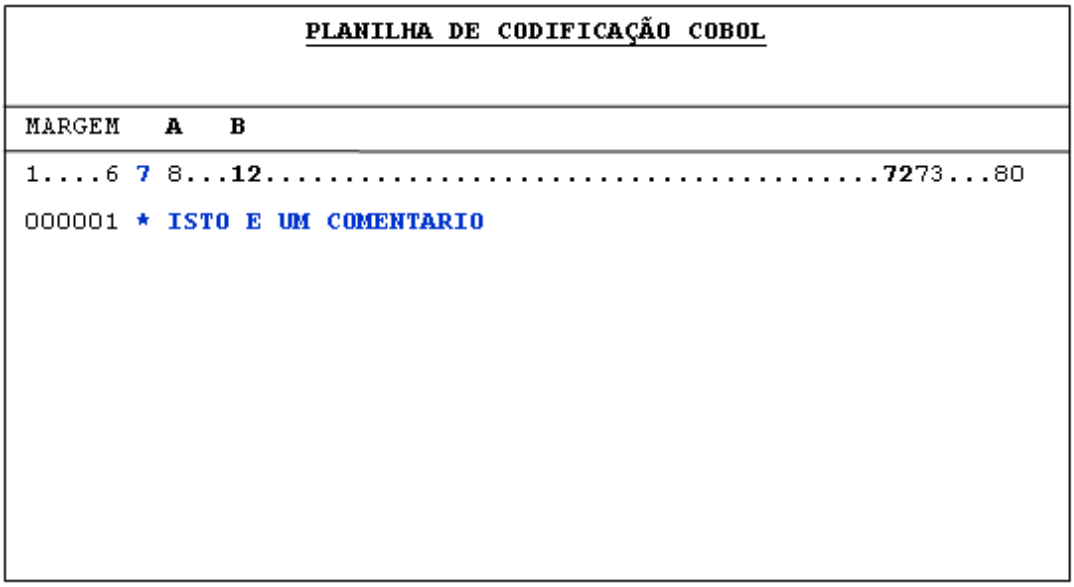
HÍFEN (-)

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.



ASTERISCO (*)

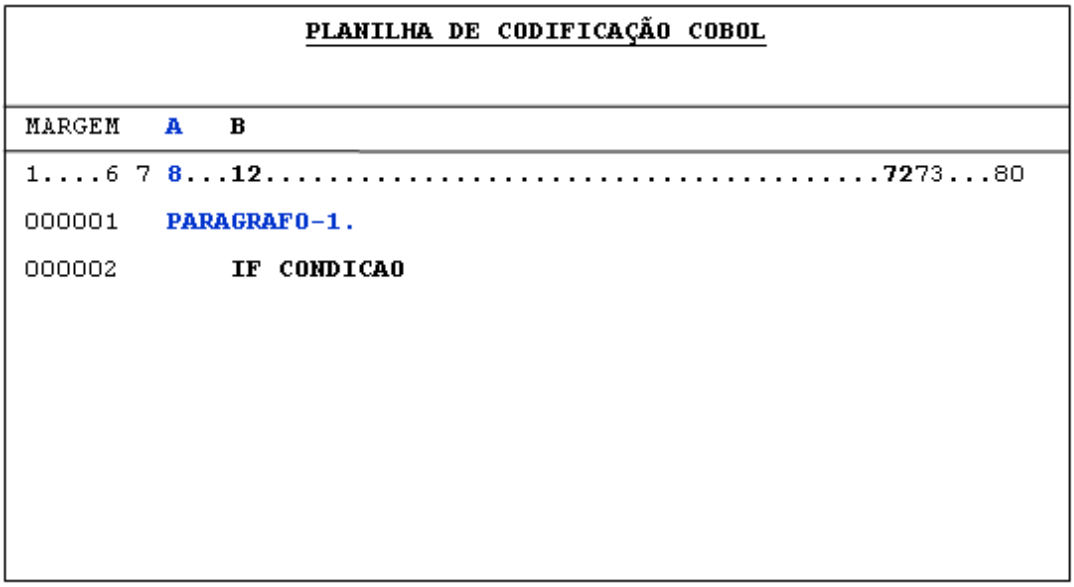
Nesta posição indica, para o compilador COBOL, que toda a linha deve ser tratada como uma linha de comentário.



ÁREA A – ÁREA B (COLUNAS DE 8 A 72)

Área A: Posição a partir da qual se escreve nome das **Divisões, Sessões, Parágrafos, Palavras Reservadas e Níveis de Dados**.

Área B: Posição a partir da qual se escrevem as **instruções** na linguagem COBOL.
Exemplo:

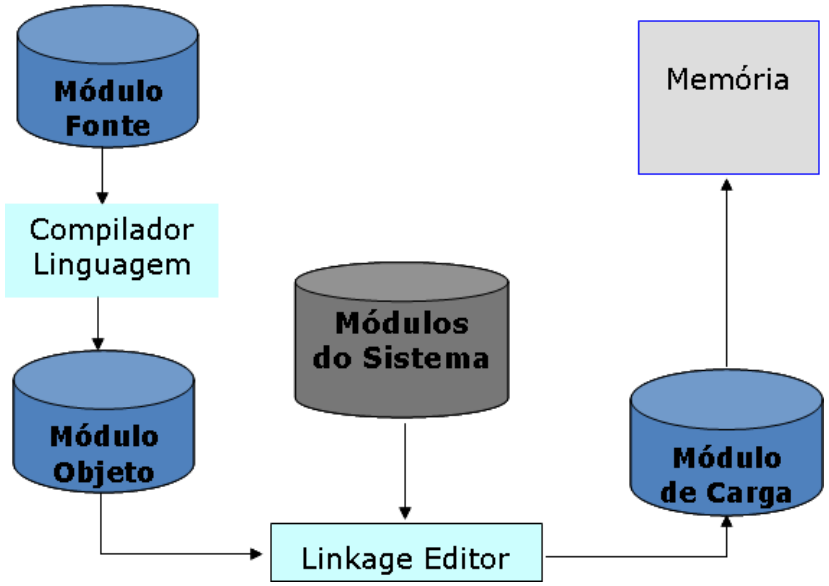


3 Compilação e Linkedição de programas

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



4 DIVISÕES DO COBOL

O COBOL consiste basicamente em quatro divisões separadas:

- **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.

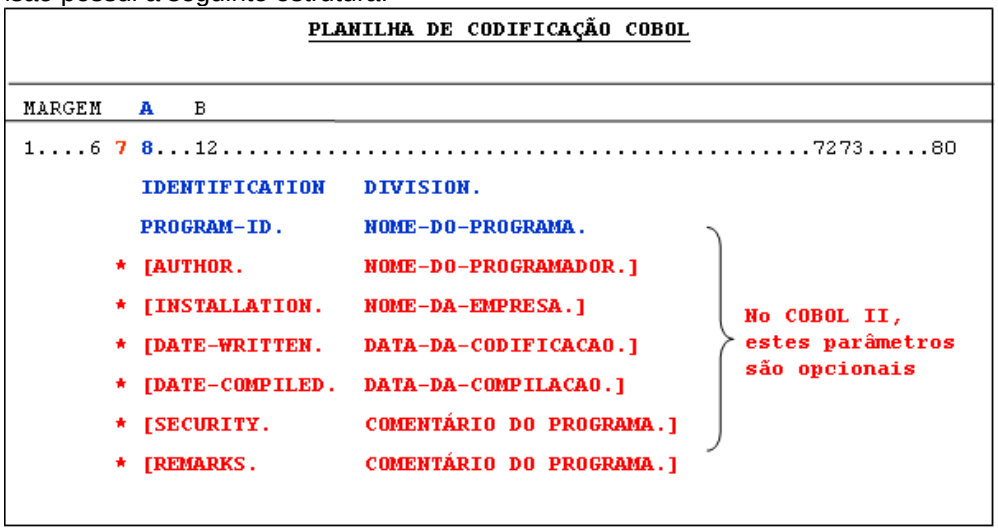
PLANILHA DE CODIFICAÇÃO COBOL

```
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.
```

4.1 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:

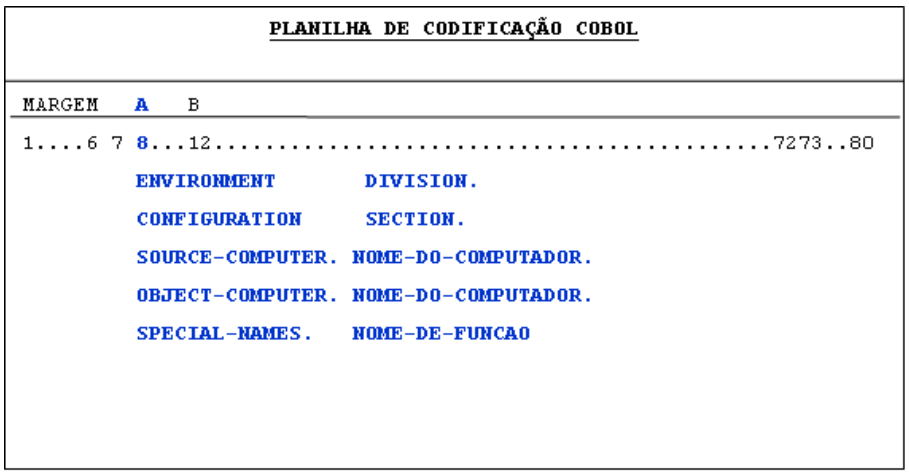


Todas as cláusulas que possuem a palavra “comentário” à direita, não possuem nenhum efeito na aplicação. São apenas parâmetros opcionais para documentação do programa.

A IDENTIFICATION DIVISION pode ser abreviada para ID DIVISION.

4.2 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:



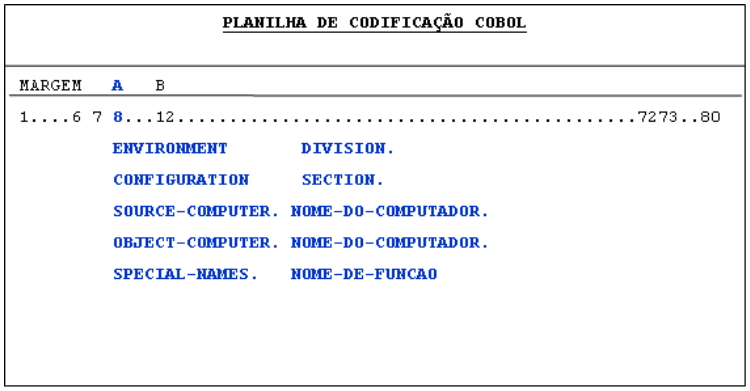
4.3 CONFIGURATION SECTION

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**. A declaração destes parágrafos eram necessárias até a versão do MVS/ESA, pois um programa só podia ser executado em apenas uma máquina. Após as versões OS/390 e z/OS, e a nova versão do COBOL II, estes parágrafos, tornaram-se obsoletos, pois um programa pode ser executado em qualquer máquina.

SOURCE-COMPUTER: identifica o computador onde foi confeccionado o programa.

OBJECT-COMPUTER: identifica o computador do ambiente de produção.

SPECIAL-NAMES:especifica 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.



PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................7273..80
         ENVIRONMENT       DIVISION.
         CONFIGURATION     SECTION.
         SOURCE-COMPUTER.  NOME-DO-COMPUTADOR.
         OBJECT-COMPUTER.  NOME-DO-COMPUTADOR.
         SPECIAL-NAMES.    NOME-DE-FUNCAO
```

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.

4.4 INPUT-OUTPUT SECTION

Esta seção destina-se a declaração dos arquivos de entrada e saída que será utilizado pelo programa.

Formato:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.............................................7273....80

         INPUT-OUTPUT   SECTION.
         FILE-CONTROL.
             SELECT NOME-DO-ARQUIVO ASSIGN TO NOME-EXTERNO
```

4.5 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 SECTIONe 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.

Formato:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.............................................7273....80
         DATA                DIVISION.
         FILE                SECTION.
         FD   NOME-DO-ARQUIVO
              BLOCK CONTAINS nn CHARACTERS
              RECORD CONTAINS nn CHARACTERS
              LABEL RECORD IS [OMITTED] ou [STANDARD]
              [RECORDING MODE IS [F] ou [V].

         01   NOME-DO-REGISTRO.
              05  NOME-DO-CAMPO1          FORMATO E TAMANHO DO CAMPO
```

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.

Formato:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.........................................7273....80
         WORKING-STORAGE SECTION.
         77  WK-CONTADOR            PIC 9(03)   VALUE ZEROS.
         01  WK-LINHA.
             05  FILLER             PIC X(10)   VALUE 'CURSO'.
             05  WK-ALUNO           PIC X(35)   VALUE SPACES.
```

PROCEDURE DIVISION

Nesta divisão é onde serão codificados os comandos necessários para a execução do programa. As instruções de um programa COBOL podem ser reunidas em parágrafos, definidos pelo programador com o fim de tornar o programa mais fácil de ser entendido. Os comandos podem ser verbos, ações ou tomadas de decisão.

```
                    PLANILHA DE CODIFICAÇÃO COBOL
MARGEM   A   B
1....6 7 8...12.............................................7273....80
         PROCEDURE DIVISION.
         PARAGRAFO-1 SECTION.
             IF CONDICAO
                TRATAR ALGO-X
             ELSE
                TRATAR ALGO-Y
             END-IF.
```

5 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 Níveis Especiais77 e 88

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

O nível 88 define nomes de condições que devem ser associados à valores definidos ao conteúdo de um determinado campo, pois podemos associar um valor a um nome fantasia.

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM    A    B
1....6 7 8...12.....................................7273.....80
         WORKING-STORAGE SECTION.
         77  WK-LIDOS                    PIC 9(03).
         01  WK-ESTADO-CIVIL             PIC 9(01).
             88  SOLTEIRO                VALUE 1.
             88  CASADO                  VALUE 2.
```

Nota:

Podemos fazer a pergunta pelo campo WK-ESTADO CIVIL , SOLTEIRO ou CASADO,

```
IF    WK-ESTADO-CIVIL = 1
      DISPLAY 'SOLTEIRO'
END-IF
Ou
IF    CASADO
      DISPLAY '2'
END-IF
```

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.

Exemplo:

77 WK-CAMPO-01-ATUAL

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-CAMPO-ATUAL.
   03  FILLER
   03  WK-CAMPO-01.
```

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.
Ou
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.
```

6 COMANDOS

6.1 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.

Para que o programa possa receber estes dados, não basta apenas codificar o comando ACCEPT, temos que codificar uma variável na WORKING-STORAGE.

Sintaxe básica
ACCEPT variável FROM origem

Exemplo:

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................................7273......80
         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:

- **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.

- **DATE [YYYYMMDD]**

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

- **DAY**

```
Formato implícito PIC 9(05)
Formato YYDDD- data Juliana
```

04/07/1981 será expresso como 81185

- **DAY-OF-WEEK**

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

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

Formato HHMMSSCC – Hora, minuto, segundo, centésimos.

2:41 da tarde será expresso como 14410000

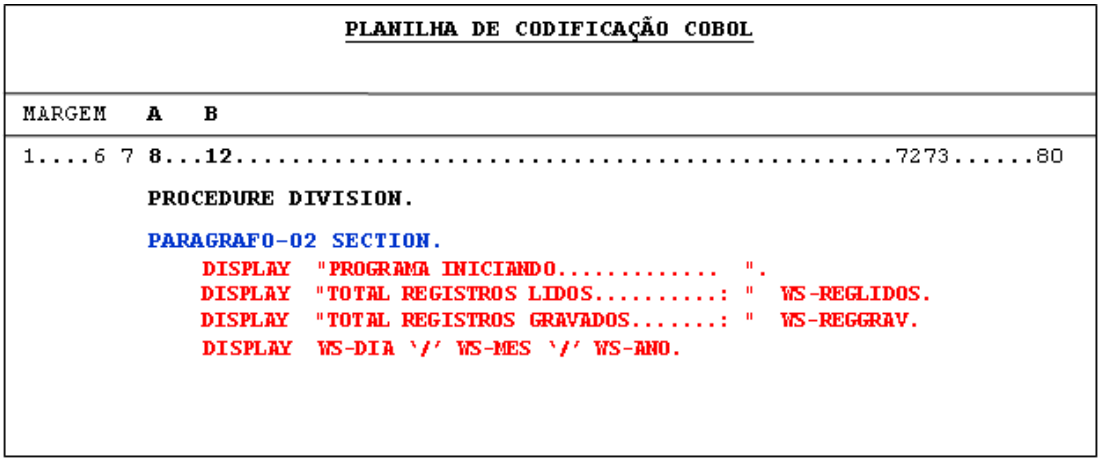
6.2 DISPLAY

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

Exemplos:

PLANILHA DE CODIFICAÇÃO COBOL

```
MARGEM   A   B
1....6 7 8...12.....................................................7273......80
         PROCEDURE DIVISION.
         PARAGRAFO-01 SECTION.
             ACCEPT WS-DATA FROM DATE.
             DISPLAY 'EXIBIR A DATA......= ' WS-DATA.
```



PLANILHA DE CODIFICAÇÃO COBOL

MARGEM A B

```
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.
```

6.3 STOP RUN

A instrução STOP RUN encerra a execução do programa.

Formato:

STOP RUN.

7 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}]
```

7.1 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

7.2 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.
```

7.3 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 comoum 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 ...}
        FROMvariável-de-resultado1 …
```

Formato 2:

```
SUBTRACT { variável-1 ... constante-1 ...}
        FROMvariável-2|constante-2
        GIVINGvariável-de-resultado1 ...
```

Regras:

1. No Formato 1, o conjunto de operandos variável-1... constante-1, são subtraídos de variávelderesultado1...

2. No Formato 2, ovariávelderesultado1, 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. Todas as constantes e campos que são parte da instrução SUBTRACT devem ser numéricos. Depois da palavra GIVING, contudo, o campo pode ser um campo numérico editado.

5. O campo variávelderesultado1, após FROM ou GIVING, deve ser um nome de variável e não uma constante.
6. Ao usar o formato GIVING, o campo variávelderesultado1 receberá o resultado da subtração, mas seu conteúdo inicial não será considerado, ou seja, sobreposto. Ele pode ser tanto um campo numérico como um campo numérico editado.

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
COMPUTE WS-RESULT = (((AA + BB) / CC) / DD) * (EE ** 2)
```

A fórmula $A = \sqrt{B^2 + C^2}$

ficará em Cobol da seguinte forma:

```
COMPUTE A = (B ** 2 + C ** 2)  ** (0,5).
```

8 Estrutura Lógica - Decisão

8.1 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. Astabelas de operadores lógicos e relacionais abaixo podem ser utilizadas para compor a condição a ser testada.

8.2 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

8.3 Operadores Lógicos

Os operadores lógicos com exceção do **NOT** devemser 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

8.4 CONTINUE OU NEXT SENTENCE

A instrução CONTINUE ou NEXT SENTENCE 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 | NEXT SENTENCE

Exemplo:

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

Exemplo 1: (IF SEM ELSE)

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

Exemplo 2: (IF COM ELSE)

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

Exemplo 3: (IF COM NEXT SENTENCE)

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

Exemplo 4: (NINHOS DE IF COM CONTINUE)

```
IFWS-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.
```

8.5 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
        WHENvalor-1 [THRUvalor-2 …]
                Instrução- 1
        [WHENOTHER
                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.
```

9 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 ouum parágrafo. Ele consiste em uma série de instruções relacionadas entre si e em programação estruturada, os parágrafos são executados por meio da instrução PERFORM.

9.1 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.

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

2. **PERFORM IN–LINE:** As instruções acionadas estão logo abaixo do comando PERFORM. Estas instruções devem ser delimitadas pelo comando END-PERFORM.

- Há 4 Formatos de PERFORM.
    1. PERFORM básico
    2. PERFORM com opção TIMES
    3. PERFORM com opção UNTIL
    4. PERFORM com opção VARYING

Formato 1 (básico):

PERFORM parágrafo

Formato 2 (opção TIMES):

PERFORM parágrafo N TIMES

O parágrafo referido é executado N TIMES, onde N pode ser uma constante ou variável numérica.

Exemplo: PERFORM com opção TIMES

```
20-00-CALCULA-TOTAL.
        MOVE ZEROS TO WS-TOTAL
        PERFORM 25-00-CALCULO 3 TIMES.
20-99-EXIT. EXIT.
```

Formato 3 (opção UNTIL):

PERFORM parágrafo UNTIL condição

O parágrafo referido é executado repetidamente até que a condição especificada pela opção UNTIL seja verdadeira.

Exemplo 1:**PERFORM com opção UNTIL**

```
        00-00-MAIN-LINE.
                MOVE “N” TO WS-FIM .
                PERFORM 10-00-INICIALIZAR.
                PERFORM 20-00-LER-ARQUIVO.
                PERFORM 30-00-PROCESSAR UNTILWS-FIM = “S”.
                PERFORM 40-00-FINALIZAR.
        00-00-EXIT. EXIT.
```

Exemplo 2: **PERFORM com opção UNTIL (IN-LINE)**

```
                PERFORM UNTIL WS-FIM = “S”
                        IF WS-LIN > 50
                        PERFORM 50-00-CABECALHO
                        END-IF
                PERFORM 60-00-ROT-DETALHE
                        PERFORM 20-00-LER-ARQUIVO
                END-PERFORM.
```

Formato 4 (opção VARYING):

PERFORM parágrafo VARYING campo FROM n BY m UNTIL condição

Executa o parágrafo especificado, até que a condição especificada seja satisfeita. Antes de executar o bloco de instruções pela primeira vez, atribui o valor N a variável CAMPO. Após cada execução do bloco, antes de voltar a executá-lo, incrementa M à variável CAMPO.

Exemplo1:**PERFORM com opção VARYING**

```
30-00-PROCESSAR.
                PERFORM 50-00-ROT-CABECALHO.
                PERFORM 60-00-ROT-DETALHE VARYING WS-LIN
                        FROM 1 BY 1 UNTIL WS-LIN = 60.
        ...
        60-00-ROT-DETALHE.
                PERFORM 70-00-IMPRIMIR-DETALHE.
                PERFORM 20-00-LER-ARQUIVO.
```

10 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.

Exemplo:

```
*       PROGRAMA -  MAIOR E MENOR NUMERO
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 FROM SYSIN.
*
PROCESSA.
      DISPLAY 'O NUMERO EH =>'   WS-NUMERO.
      PERFORM MAIOR-MENOR.
      ACCEPT WS-NUMERO FROM SYSIN.
*
  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.
```

11 Arquivos Seqüenciais

Arquivo é um meio de armazenar informações que foram processadas e que poderão ser utilizadas novamente. Para isso precisamos relembrar os conceitos aplicados no módulo de lógica.
Vamos rever !!!!

11.1 Conceitos Básicos

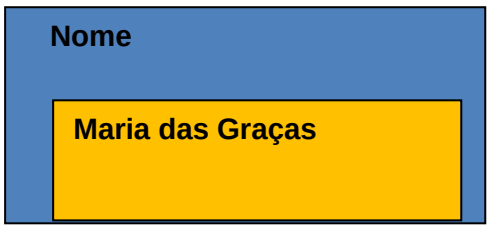
ARQUIVO é um conjunto de registros armazenados de forma seqüencial.
Exemplo: O arquivo de Clientes da Empresa, onde estão armazenados os dados de todos os clientes da empresa.

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.

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.

11.2 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.

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.

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.....................................7273...80
         ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
         SPECIAL-NAMES.
             DECIMAL-POINT IS COMMA.
         INPUT-OUTPUT SECTION.
         FILE-CONTROL.
             SELECT CLIENTES ASSIGN TO CLIENTES.
```

Neste exemplo escolhemos como ***nome-arquivo-lógico*** dentro da instrução SELECT a palavra **CLIENTES. CLIENTES**será usado em todos os comandos do programa como referência para este arquivo.

11.3 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. Vimos anteriormente, na INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION), que para cada arquivo a ser tratado no programa havia uma instrução SELECT especificando e definindo um nome lógico para o arquivo. Na FILE SECTION precisamos agora detalhar cada um destes arquivos. 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:

```
                    PLANILHA DE CODIFICAÇÃO COBOL

MARGEM   A   B
1....6 7 8...12.....................................7273...80
         DATA DIVISION.
         FILE SECTION.
         FD  CLIENTES.
         01  REG-CLIENTES.
             03  COD-CLIENTE      PIC  9(8).
             03  NOME-CLIENTE     PIC  X(20).
             03  ENDER-CLIENTE    PIC  X(40).
             03  VALOR-CLIENTE    PIC  9(10)V99.
```

Observação: Não usar value dentro da sintaxe...

11.4 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.

Formato:

OPEN [INPUT | OUTPUT] Nome-arquivo1 ...

Regras:

1. **INPUT:** Permite abrir o arquivo apenas para operações de leitura.

2. **OUTPUT:** Permite abrir o arquivo para operações de gravação. Esta operação pode ser especificada quando o arquivo estiver sendo criado. Esta opção não permite comandos de leitura no arquivo.

3. **NOME-Arquivo1 ...** : Nome lógico do(s) arquivo(s) definido(s) na clausula SELECT e na FD:

4. Teste de file status é obrigatório.

Exemplo:

```
*=============================================================
      *      ROTINA DE ABERTURA DOS ARQUIVOS                    *
*=============================================================
       INICIO  SECTION.

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

           OPEN  OUTPUT ARQSAI.
           IF  WS-FS-ARQSAI  NOT EQUAL  ZEROS
               DISPLAY '======================================='
               DISPLAY ' ERRO NO OPEN DO ARQUIVO ARQSAI        '
               DISPLAY ' FILE STATUS  = ' WS-FS-ARQSAI
               DISPLAY '======================================='
STOP RUN.
```

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 váriasvezes no campo receptor até preenchê-lo completamente.

Exemplos de MOVEcom 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:

```
*=============================================================
      *      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.
```

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.

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

Efetua o fechamento de arquivos.

Formato:

```
CLOSE nome-de-arquivo ...
```

Exemplos:

```
     *=============================================================
     *     ROTINA DE FINALIZACAO E FECHAMENTO DOS ARQUIVOS       *
*=============================================================
      FINALIZA    SECTION.

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

         CLOSE  ARQSAI.
         IF  WS-FS-ARQSAI  NOT EQUAL  ZEROS
DISPLAY '======================================='
```

```
           DISPLAY ' ERRO NO CLOSE DO ARQUIVO ARQSAI      '
DISPLAY ' FILE STATUS  = ' WS-FS-ARQSAI
           DISPLAY '======================================'
           STOP RUN.
```

Leitura 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
        [INTO nome-area-working]
[END-READ].
```

Regras:

1. **INTO:** Os campos do registro lido serão movidos da área de leitura do arquivo para uma área de trabalho.

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.
```

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.
```

12 Relatórios

A impressão de relatórios é o registro de informações processadas pelo programa em um meio de armazenamento de dados chamado de formulário. Para efetuarmos a impressão de relatórios devemos nos preocupar com os seguintes aspectos:

- Características do formulário
- Controle de linhas, numeração e salto de página
- Impressão de cabeçalho e estética da página

Características do Formulário

A maioria dos formulários possui um formato padrão, isto é, a **quantidade de linhas por página e de caracteres** por linha são constantes.

Controle de Linhas, numeração e salto de Páginas

Uma preocupação com o controle de linhas é não permitir que a impressora imprima fora do espaço útil do papel reservado para o texto, pois além de esteticamente não ficar bom, haveria perda de informações. Para controlarmos o número de linhas impressas, devemos criar um contador de linha e de páginas e não deixar o valor desses contadores ultrapassarem o número desejado de linhas por página.

Exemplo:

```
ID                    DIVISION.
      *-----------------------------------------------------------------*
        PROGRAM-ID.           EXEMPLO2.
AUTHOR.               ALUNO.
      *-----------------------------------------------------------------*
       ENVIRONMENT            DIVISION.
       CONFIGURATION          SECTION.
       SPECIAL-NAMES.
           DECIMAL-POINT IS COMMA.
      *-----------------------------------------------------------------*
       INPUT-OUTPUT           SECTION.
       FILE-CONTROL.
           SELECT PECAS       ASSIGN TO ENTRADA.
           SELECT RELATORIO   ASSIGN TO RELATO.
      *-----------------------------------------------------------------*
       DATA                   DIVISION.
      *-----------------------------------------------------------------*
       FILE                   SECTION.
      *-----------------------------------------------------------------*
       FD  PECAS
           RECORDING MODE F.
       01  REG-PECAS.
02  CODIGO                      PIC 9(005).
02  NOME                        PIC X(020).
           02  DESCRICAO                   PIC X(020).
           02  PRECO                       PIC 9(007)V99.
           02  ESTOQUE                     PIC 9(007).
*-----------------------------------------------------------------*
       FD  RELATORIO
           RECORDING MODE F.
```

```
       01  REG-RELATO                      PIC X(132).
      *-----------------------------------------------------------------*
       WORKING-STORAGE SECTION.
      *-----------------------------------------------------------------*
       77  WS-FIM                          PIC X VALUE SPACES.
01  ACUMULADORES.
           02  NPAG-WS                     PIC 9(02)    VALUE ZEROS.
           02  NLINHAS-WS                  PIC 9(02)    VALUE ZEROS.
           02  NITENS-WS                   PIC 9(03)    VALUE ZEROS.
           02  NVALOR-WS                   PIC 9(7)V99  VALUE ZEROS.
*-----------------------------------------------------------------*
       01  LIN1.
           02  FILLER                      PIC X(5)  VALUE SPACES.
02  DATA-LIN1.
               03  DIA                     PIC 9(02) VALUE ZEROS.
03  FILLER               PIC X     VALUE "/".
               03  MES                     PIC 9(02) VALUE ZEROS.
               03  FILLER                  PIC X     VALUE "/".
               03  ANO                     PIC 9(02) VALUE ZEROS.
           02  FILLER                      PIC X(20) VALUE SPACES.
           02  FILLER                      PIC X(18) VALUE
               "RELATORIO DE PECAS".
           02  FILLER                      PIC X(20) VALUE SPACES.
           02  FILLER                      PIC X(5)  VALUE"PAG. ".
02  NPAG-LIN1               PIC Z9    VALUE SPACES.
      *-----------------------------------------------------------------*
       01  LIN2.
           02  FILLER                      PIC X(33) VALUE
               "     CODIGO  NOME                ".
           02  FILLER                      PIC X(22) VALUE
               "  DESCRICAO".
           02  FILLER                      PIC X(02) VALUE SPACES.
           02  FILLER                      PIC X(13) VALUE
               "      PRECO  ".
           02  FILLER                      PIC X(9)  VALUE
               "  ESTOQUE".
      *-----------------------------------------------------------------*
       01  DETALHE.
           02  FILLER                      PIC X(5)   VALUE SPACES.
           02  CODIGO-DET                  PIC ZZ.ZZ9 VALUE SPACES.
           02  FILLER                      PIC X(2)   VALUE SPACES.
           02  NOME-DET                    PIC X(20)  VALUE SPACES.
           02  FILLER                      PIC X(2)   VALUE SPACES.
           02  DESC-DET                    PIC X(20)  VALUE SPACES.
           02  FILLER                      PIC X(2)   VALUE SPACES.
           02  PRECO-DET                   PIC ZZ.ZZZ.ZZ9,99
  VALUE SPACES.
           02  FILLER                      PIC X(2)   VALUE SPACES.
           02  EST-DET                     PIC Z.ZZZ.99
    VALUE SPACES.
      *-----------------------------------------------------------------*
       01  LINTOT1.
           02 FILLER                       PIC X(21) VALUE
              "     TOTAL DE ITENS: ".
           02 TOTALITENS                   PIC Z99   VALUE SPACES.
      *-----------------------------------------------------------------*
       01 LINTOT2.
          02 FILLER                        PIC X(22) VALUE
"     VALOR DOS ITENS: ".
           02 TOTALVALOR                   PIC $ZZ.ZZ9,99 VALUE SPACES.
```

```
      *-----------------------------------------------------------------*
       01 DATA-ACCEPT.
          02 ANO                          PIC 9(02) VALUE ZEROS.
          02 MES                          PIC 9(02) VALUE ZEROS.
          02 DIA                          PIC 9(02) VALUE ZEROS.
      *-----------------------------------------------------------------*
       PROCEDURE             DIVISION.
      *-----------------------------------------------------------------*
       PRINCIPAL.
      *-----------------------------------------------------------------*
            PERFORM INICIAR.
            PERFORM PROCESSAR UNTIL WS-FIM = "S".
            PERFORM FINALIZAR.
            STOP RUN.

FIM-ROTINA-PRINCIPAL.
      *-----------------------------------------------------------------*
       INICIAR.
*-----------------------------------------------------------------*
            MOVE "N"                          TO WS-FIM.
MOVE ZEROS                        TO ACUMULADORES.
            MOVE 99                           TO NLINHAS-WS.
      *
            OPEN INPUT  PECAS.
OPEN OUTPUT RELATORIO.
      *
            ACCEPT DATA-ACCEPT                FROM DATE.
            MOVE CORR DATA-ACCEPT             TO DATA-LIN1.
READ PECAS
                 AT END
                    MOVE "S"                  TO WS-FIM
            END-READ.

       FIM-INICIAR.
            EXIT.
*-----------------------------------------------------------------*
       PROCESSAR.
      *-----------------------------------------------------------------*
            MOVE CODIGO                       TO CODIGO-DET.
MOVE NOME                         TO NOME-DET.
            MOVE DESCRICAO                    TO DESC-DET.
            MOVE PRECO                        TO PRECO-DET.
            MOVE ESTOQUE                      TO EST-DET.
      *
            IF NLINHAS-WS > 35
               PERFORM CABECALHO.
            WRITE REG-RELATO  FROM DETALHE.
            ADD 1                             TO NLINHAS-WS.
            ADD 1                             TO NITENS-WS.
            ADD PRECO                         TO NVALOR-WS.
            READ PECAS
                 AT END
                    MOVE "S"                  TO WS-FIM
            END-READ.

       FIM-PROCESSAR.
            EXIT.
```

```
*-----------------------------------------------------------------*
       FINALIZAR.
      *-----------------------------------------------------------------*
            MOVE NVALOR-WS                  TO TOTALVALOR.
MOVE NITENS-WS                  TO TOTALITENS.
            WRITE REG-RELATO FROM LINTOT1 AFTER 2 LINES.
            WRITE REG-RELATO FROM LINTOT2 AFTER 2 LINES.
CLOSE PECAS RELATORIO.

       FIM-FINALIZAR.
EXIT.
      *-----------------------------------------------------------------*
       CABECALHO.
      *-----------------------------------------------------------------*
           ADD 1                            TO NPAG-WS.
           MOVE NPAG-WS                     TO NPAG-LIN1.
           WRITE REG-RELATO FROM LIN1 AFTER PAGE.
           WRITE REG-RELATO FROM LIN2 AFTER 2 LINES.
MOVE 5                          TO NLINHAS-WS.
       FIM-CABECALHO.
           EXIT.
```

COMANDOS PARA PROCESSAMENTO DE RELATÓRIOS

O Cobol trata a impressão de relatórios de maneira similar à gravação de arquivos, ou seja, enviar uma linha de relatório para a impressora é idêntico a gravar um registro em um arquivo. Por isso precisamos definir o relatório na INPUT-OUTPUT SECTION (ENVIRONMENT DIVISION) com a instrução SELECT, precisamos definir o conteúdo das linhas de impressão na FILE SECTION (DATA DIVISION) com a instrução FD, e na PROCEDURE DIVISION devemos usar as instruções OPEN, WRITE e CLOSE para controlar a impressão. No entanto, obviamente **existem diferenças entre um arquivo e um relatório**, **e os seguintes detalhes devem ser observados em um programa:**

- Diferentemente dos arquivos, onde todos os registros podem ter o mesmo layout, em um relatório as linhas de detalhe podem ser diferentes (incluindo sub-totais, títulos de grupos etc.). Além disso, sempre existe um cabeçalho em cada folha ou ainda linhas de rodapé.

- O programa precisa controlar a mudança de página. Para isto o programa deve criar uma variável para contar linhas, que deve ser incrementada a cada comando WRITE do relatório. Quando esta variável atinge o numero de linhas disponíveis na folha, o programa deve comandar o salto para a nova folha e imprimir as linhas de cabeçalho.

- É comum haver totalizações em vários níveis (subtotais, totais gerais etc.). Estes totais são emitidos quando muda a identificação de grupo dentro dos registros lidos. Por exemplo, em um relatório de vendas com totais por mês, o programa deve comparar o mês do registro lido com o mês do registro anterior para verificar se são diferentes. Nestas mudanças de variável de grupo (geralmente conhecido como QUEBRA), o programa deve emitir uma linha de total.

WRITE {BEFORE | AFTER}
A instrução WRITE tem um formato próprio para impressões de relatórios.

Formato:

```
WRITE nome-registro-1 [FROM variável-1]
        {BEFORE | AFTER} n [n LINES|PAGE]
 [END-WRITE].
```

Regra:

1. **BEFORE:** Quando a opção BEFORE for utilizada o Cobol interpreta o comando da seguinte forma: GRAVAR nome-registro-1 ANTES DE SALTAR N LINHAS ou PAGINA.

2. **AFTER:** Quando a opção AFTER for utilizada o Cobol interpreta o comando da seguinte forma: GRAVAR nome-registro-1 DEPOIS DE SALTAR N LINHAS ou PÁGINA.

3. **FROM:** A opção FROM pode ser utilizada para transferir as linhas do relatório a serem impressas (definidas na Working-Storage Section), após serem preparadas, para o registro do arquivo de impressão.

Exemplo:

```
    *-----------------------------------------------------------------*
     CABECALHO.
    *-----------------------------------------------------------------*
        ADD 1                    TO NPAG-WS.
        MOVE NPAG-WS             TO NPAG-LIN1.
        WRITE REG-RELATO FROM LIN1 AFTER PAGE.
        WRITE REG-RELATO FROM LIN2 AFTER 2 LINES.
MOVE 5                   TO NLINHAS-WS.
     FIM-CABECALHO.
        EXIT.
```

12.1 Totalização e Quebra de Relatórios

Totalização aplica-se a arquivos com vários registros com a mesma chave.

Exemplo:
Arquivo de lançamentos para contas correntes, onde uma conta pode ter vários lançamentos. Pode ser necessário um relatório que totalize por conta.

```
IF NLINHAS-WS > 35
           PERFORM CABECALHO.
        WRITE REG-RELATO  FROM DETALHE.
        ADD 1                    TO NLINHAS-WS.
        ADD 1                    TO NITENS-WS.
        ADD PRECO                TO NVALOR-WS.
```

13 Lógica Balanceada

Trata-se do processamento sincronizado de dois ou mais arquivos ordenados pela mesma chave. Exemplo: arquivo de contas correntes e movimentos, ambos com a chave: CONTA. A lógica balanceada irá gravar um novo arquivo de contas correntes com os saldos atualizados pelos valores dos lançamentos das respectivas contas. Com a lógica balanceada, este processamento será efetuado com apenas uma leitura de cada arquivo, verificando as condições de igual, maior ou menor, para cada registro lido.

A lógica balanceada também é conhecida como BALANCE LINE ou MERGE.

Exemplo:

```
 IDENTIFICATION DIVISION.
       PROGRAM-ID. EXEMPLO3.
       AUTHOR. ALUNO.
      *OBJETIVO: BALANCE LINE ENTRE O ARQUIVO CCANT E MOV.
*================================================================
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           DECIMAL-POINT IS COMMA.
       INPUT-OUTPUT  SECTION.
       FILE-CONTROL.
           SELECT  CCANT  ASSIGN  TO  CCANT
              FILE STATUS  IS  WS-FS-CCANT.

           SELECT  MOV  ASSIGN  TO  MOV
              FILE STATUS  IS  WS-FS-MOV.

           SELECT  ATUA   ASSIGN  TO   ATUA
              FILE STATUS  IS  WS-FS-ATUA.
      *================================================================
       DATA DIVISION.
       FILE SECTION.
       FD  CCANT
           BLOCK CONTAINS  0  RECORDS
           RECORD CONTAINS  30 CHARACTERS
           LABEL RECORD IS STANDARD
           RECORDING MODE IS F.

       01  REG-CCANT.
03  CCANT-CONTA       PIC X(3).
           03  CCANT-NOME        PIC X(20).
           03  CCANT-SALDO       PIC S9(5)V9(2).

       FD  MOV
           BLOCK CONTAINS  0  RECORDS
           RECORD CONTAINS  10 CHARACTERS
           LABEL RECORD IS STANDARD
           RECORDING MODE IS F.

01  REG-MOV.
03  MOV-CONTA         PIC X(3).
03  MOV-SALDO         PIC S9(5)V9(2).

       FD  ATUA
           BLOCK CONTAINS  0  RECORDS
```

```
           RECORD CONTAINS  30 CHARACTERS
           LABEL RECORD IS STANDARD
           RECORDING MODE IS F.

01  REG-ATUA.
           03  ATUA-CONTA       PIC X(3).
           03  ATUA-NOME        PIC X(20).
           03  ATUA-SALDO       PIC S9(5)V9(2).

WORKING-STORAGE SECTION.

       77  WS-FS-CCANT          PIC  9(2)  VALUE ZEROS.
       77  WS-FS-MOV            PIC  9(2)  VALUE ZEROS.
77  WS-FS-ATUA           PIC  9(2)  VALUE ZEROS.

PROCEDURE  DIVISION.
       PRINCIPAL SECTION.

           PERFORM INICIO.
           PERFORM PROCESSA  UNTIL  HIGH-VALUES = CCANT-CONTA
                   AND MOV-CONTA.
           PERFORM FIM.
STOP RUN.

       INICIO  SECTION.

           OPEN  INPUT  CCANT.
           IF WS-FS-CCANT NOT EQUAL  ZEROS
              DISPLAY 'ERRO NO OPEN DO CCANT = ' WS-FS-CCANT
                  MOVE 99 TO RETURN-CODE
              STOP RUN.

           OPEN  INPUT  MOV.
           IF WS-FS-MOV  NOT EQUAL ZEROS
              DISPLAY 'ERRO NO OPEN DO MOV = ' WS-FS-MOV
                  MOVE 99 TO RETURN-CODE
              STOP RUN.

           OPEN  OUTPUT  ATUA.
           IF WS-FS-ATUA  NOT EQUAL ZEROS
DISPLAY 'ERRO NO OPEN DO ATUA = ' WS-FS-ATUA
                  MOVE 99 TO RETURN-CODE
              STOP RUN.
           PERFORM LER-CCANT.
PERFORM LER-MOV.
```

```
       PROCESSA  SECTION.

           IF CCANT-CONTA = MOV-CONTA
              ADD MOV-SALDO TO CCANT-SALDO
PERFORM LER-MOV
           ELSE
              IF CCANT-CONTA < MOV-CONTA
                 MOVE REG-CCANT TO REG-ATUA
                 WRITE REG-ATUA
                 PERFORM LER-CCANT
              ELSE
                 PERFORM LER-MOV
              END-IF
           END-IF.

       FIM SECTION.
```

```
           CLOSE  CCANT MOV ATUA.

           IF WS-FS-CCANT NOT EQUAL  ZEROS
DISPLAY 'ERRO NO CLOSE DO CCANT = ' WS-FS-CCANT.
                MOVE 99 TO RETURN-CODE
              STOP RUN.

           IF WS-FS-MOV  NOT EQUAL  ZEROS
DISPLAY 'ERRO NO CLOSE DO MOV = ' WS-FS-MOV.

           IF WS-FS-ATUA  NOT EQUAL  ZEROS
              DISPLAY 'ERRO NO CLOSE DO ATUA = ' WS-FS-ATUA.
```

```
LER-CCANT SECTION.

           READ CCANT.

           IF WS-FS-CCANT NOT EQUAL  ZEROS AND 10
              DISPLAY 'ERRO NO READ DO CCANT = ' WS-FS-CCANT
MOVE 99 TO RETURN-CODE
              STOP RUN
           ELSE
              IF WS-FS-CCANT EQUAL 10
                 MOVE HIGH-VALUES TO CCANT-CONTA
              END-IF
           END-IF.
```

```
       LER-MOV  SECTION.

           READ MOV.

           IF WS-FS-MOV  NOT EQUAL  ZEROS AND 10
              DISPLAY 'ERRO NO READ DO MOV = ' WS-FS-MOV
      ------>MOVE 99 TO RETURN-CODE
              STOP RUN
           ELSE
              IF WS-FS-MOV  EQUAL 10
                 MOVE HIGH-VALUES TO MOV-CONTA
END-IF
           END-IF.
```

14 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).
```

14.1 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).

15 Chamando um Sub-Programa

O comando CALL é usado para incorporar ao programa principal (LINK) um sub-programa. A sua principal função é a redundância de códigos em vários programas, podendo assim, vários programas usarem a mesma sub-rotina (sub-programa), não necessitando reescrever os códigos a cada novo programa.

Formatos:

- **Chamada estática:**

```
CALL 'SUBPRG1' USING WS-AREA.
```

- **Chamada Dinâmica:**

```
CALL NOME-DO-SUBPROGRAMA USING WS-AREA
```

Parâmetros:

USING : Indica a área que servirá de comunicação entre o programa principal e o sub-programa, definida na WORKING-STORAGE SECTION.

Chamada estática: esta chamada incorpora na linkedição do programa principal o executável do sub-programa, a desvantagem dessa modalidade é que se houver alterações no sub-programa, devemos compilar todos os programas principais que usarem o sub-programa.

Chamada dinâmica: esta chamada incorpora o executável do sub-programa ao programa principal em tempo de execução do mesmo, tornando assim, disponível a última versão do sub-programa, se este sofreu modificações.

Procedimentos no Sub-Programa:

Esta instrução é usada para sub-programas (programas chamados por CALL num programa principal) para devolver o controle (retornar) ao programa principal, logo após a instrução CALL.

Parâmetros:

USING: Indica a área que servirá de comunicação entre o sub-programa e o principal, será definida na LINKAGE SECTION e referenciada na PROCEDURE DIVISION, note que a área tem que ter o mesmo formato, podendo ter nomes diferentes.

EXIT/GOBACK: Retorna para o programa que chamou, passando a área da linkage.

ANEXO A FILE STATUS

Campo com o código de retorno dos comandos executados sobre o arquivo. Devem ter formato PIC XX e os valores retornados estão na tabela abaixo:

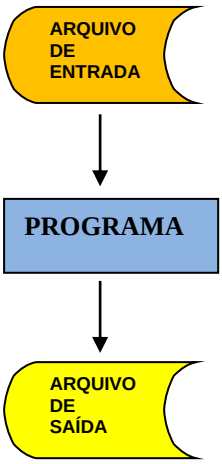
STATUS	DESCRIÇÃO
'00'	'SUCCESSFUL COMPLETION'
'02'	'DUPLICATE KEY, NON UNIQ. ALT INDX'
'04'	'READ, WRONG LENGTH RECORD'
'05'	'OPEN, FILE NOT PRESENT'
'07'	'CLOSE OPTION INCOMPAT FILE DEVICE OPEN IMPLIES TAPE; TAPE NOT USED'
'10'	'END OF FILE'
'14'	'RRN > RELATIVE KEY DATA'
'20'	'INVALID KEY VSAM KSDS OR RRDS'
'21'	'SEQUENCE ERROR, ON WRITE OR CHANGING KEY ON REWRITE'
'22'	'DUPLICATE KEY'
'23'	'RECORD OR FILE NOT FOUND'
'24'	'BOUNDARY VIOLATION. WRITE PAST END OF KSDS RECORD. COBOL 370: REL: REC# TOO BIG. OUT OF SPACE ON KSDS/RRDS FILE'
'30'	'PERMANENT DATA ERROR. DATA CHECK, PARITY CHK, HARDW'
'34'	'BOUNDARY VIOLATION. WRITE PAST END OF ESDS RECORD OR NO SPACE TO ADD KSDS/RRDS RECORD. OUT OF SPACE ON SEQUENTIAL FILE'
'35'	'35' 'OPEN, FILE NOT PRESENT'
'37'	'OPEN MODE INCOMPAT WITH DEVICE'
'38'	'OPENING FILE CLOSED WITH LOCK'
'39'	'OPEN, FILE ATTRIB CONFLICTING'
'41'	'OPEN, FILE IS OPEN'
'42'	'CLOSE, FILE IS CLOSED'
'43'	'DELETE OR REWRITE & NO GOOD READ FIRST'
'44'	'BOUNDARY VIOLATION/REWRITE REC TOO BIG'
'46'	'SEQUENTIAL READ WITHOUT POSITIONING'
'47'	'READING FILE NOT OPEN AS INPUT/IO/EXTEND'
'48'	'WRITE WITHOUT OPEN IO'
'49'	'DELETE OR REWRITE WITHOUT OPEN IO'
'90'	'UNKNOWN'
'91'	'VSAM - PASSWORD FAILURE'
'92'	'LOGIC ERROR/OPENING AN OPEN FILE OR READING OUTPUT FILE OR WRITE INPUT FILE OR DEL/REW BUT NO PRIOR READ '
'93'	'VSAM - VIRTSTOR. RESOURCE NOT AVAILABLE'
'94'	'VSAM - SEQUENTIAL READ AFTER END OF FILE OR NO CURRENT REC POINTER FOR SEQ'
'95'	'VSAM - INVALID FILE INFORMATION OR OPEN OUTPUT (LOAD) WITH FILE THAT NEVER CONTAINED DATA'
'96'	'VSAM - MISSING DD STATEMENT IN JCL'
'97'	'VSAM - OPEN OK, FILE INTEGRITY VERIFIED FILE SHOULD BE OK'
OTHER	'UNKNOWN REASON'

ANEXO B - Fluxograma

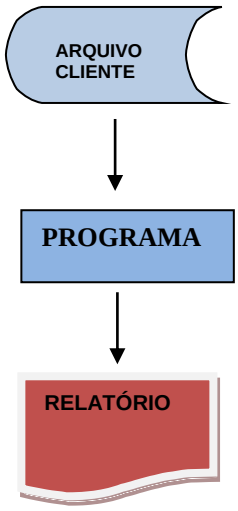
O fluxograma é a representação gráfica do programa e os arquivos que serão processados pelo mesmo, sem desenvolver a lógica do programa.

Estes dois exemplos de fluxograma dão uma visão geral dos arquivos envolvidos em cada um dos programas. O primeiro diz que haverá um arquivo de entrada que será usado para leitura, um processamento principal e um arquivo de saída que será usado para gravação. Já o segundo exemplo diz que haverá um arquivo de leitura, um processamento, e a saída será gerado um relatório.

1º Exemplo



2º Exemplo




```
ID  DIVISION.
       PROGRAM-ID. EXEMPLO.
       AUTHOR. ALUNO.
      *================================================================
      *OBJETIVO: LER UM ARQUIVO SEQUENCIAL E GRAVAR INFORMACOES EM
      *          UM ARQUIVO DE SAIDA
      *================================================================
       ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
       SPECIAL-NAMES.
           DECIMAL-POINT IS COMMA.
       INPUT-OUTPUT  SECTION.
       FILE-CONTROL.
           SELECT  ARQENT   ASSIGN   TO   ARQENT
             FILE STATUS       IS   WS-FS-ARQENT.

           SELECT  ARQSAI   ASSIGN   TO   ARQSAI
             FILE STATUS       IS   WS-FS-ARQSAI.

      *================================================================
       DATA  DIVISION.
       FILE  SECTION.
       FD  ARQENT
           BLOCK CONTAINS 0 RECORDS
           RECORD CONTAINS 100 CHARACTERS
           LABEL RECORD IS STANDARD
```

```
           RECORDING MODE IS F.

       01  REG-ARQENT.
           02  ARQENT-NOME                PIC X(20).
           02  ARQENT-ENDERECO            PIC X(30).
           02  ARQENT-TELEFONE            PIC X(09).
           02  ARQENT-CODIGO              PIC 9(06).
           02  ARQENT-IDADE               PIC 9(03).
           02  FILLER                     PIC X(32).

       FD  ARQSAI
           BLOCK CONTAINS 0 RECORDS
           RECORD CONTAINS 150 CHARACTERS
           LABEL RECORD IS STANDARD
           RECORDING MODE IS F.

01  REG-ARQSAI.
           02  ARQSAI-NOME                PIC X(20).
           02  ARQSAI-ENDERECO            PIC X(30).
           02  ARQSAI-TELEFONE            PIC X(09).
           02  ARQSAI-CODIGO              PIC 9(06).
           02  ARQSAI-IDADE               PIC 9(03).
02  FILLER                     PIC X(82).

      *=============================================================
       WORKING-STORAGE  SECTION.
       77  WS-FS-ARQENT    PIC 9(02)   VALUE ZEROS.
       77  WS-FS-ARQSAI    PIC 9(02)   VALUE ZEROS.
       77  WS-CONT-LIDOS   PIC 9(05)   VALUE ZEROS.
       77  WS-CONT-GRAVA   PIC 9(05)   VALUE ZEROS.

      *=============================================================
       PROCEDURE DIVISION.
       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.

      *=============================================================
      *      ROTINA DE ABERTURA DOS ARQUIVOS                        *
*=============================================================
       INICIO  SECTION.

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

           OPEN  OUTPUT ARQSAI.
           IF  WS-FS-ARQSAI  NOT EQUAL  ZEROS
               DISPLAY '======================================='
               DISPLAY ' ERRO NO OPEN DO ARQUIVO ARQSAI        '
```

```
               DISPLAY ' FILE STATUS  = ' WS-FS-ARQSAI
               DISPLAY '======================================='
               STOP RUN.

PERFORM LER-ARQENT   THRU LER-FIM.

       INICIO-FIM.
           EXIT.

      *================================1================================
      *      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.

LER-FIM.
           EXIT.

      *================================================================
      *      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.

           PERFORM  GRAVA-SAIDA   THRU    GRAVA-SAIDA-FIM.
           PERFORM  LER-ARQENT    THRU    LER-FIM.

       PROCESSA-FIM.
           EXIT.

      *================================================================
      *      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.

      *=================================================================
      *      ROTINA DE FINALIZACAO E FECHAMENTO DOS ARQUIVOS       *
*=================================================================
       FINALIZA    SECTION.

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

           CLOSE  ARQSAI.
           IF  WS-FS-ARQSAI  NOT EQUAL  ZEROS
DISPLAY '======================================='
               DISPLAY ' ERRO NO CLOSE DO ARQUIVO ARQSAI       '
DISPLAY ' FILE STATUS  = ' WS-FS-ARQSAI
               DISPLAY '======================================='
               STOP RUN.
           DISPLAY '================================================='
           DISPLAY 'QTDE. REG. LIDOS NO ARQENT    = ' WS-CONT-LIDOS
           DISPLAY 'QTDE. REG. GRAVADOS NO ARQSAI = ' WS-CONT-GRAVA
           DISPLAY '================================================='

       FINALIZA-FIM.
           EXIT.
```

ANEXO C – JCL´s de apoio

JCL de Compilação – Programa Cobol

```
//BRADXXCO JOB 'COMPCOBOL',CLASS=C,MSGCLASS=X,NOTIFY=&SYSUID
//        JCLLIB  ORDER=GR.GERAL.PROCLIB
//COBCOMP  EXEC DFHCOBOL,USER=BRADXX,PROG='BRADXX01'
//LKED.SYSIN DD *
  NAME BRADXX01(R)
```

JCL de Execução – Programa Cobol

```
//BRADXXEX  JOB 'EXECCOB',CLASS=C,MSGCLASS=X,NOTIFY=&SYSUID
//EXECUTA  EXEC PGM=BRADXX01
//STEPLIB  DD   DSN=GR.GERAL.LOADLIB,DISP=SHR
```

JCL de Execução **(com SYSIN)** – Programa Cobol

```
//BRADXXEX  JOB 'EXECCOB',CLASS=C,MSGCLASS=X,NOTIFY=&SYSUID
//EXECUTA  EXEC PGM=BRADXX01
//STEPLIB  DD   DSN=GR.GERAL.LOADLIB,DISP=SHR
//SYSIN     DD  *
LINHA1
LINHA2
.....
```

JCL de Execução **(ARQUIVO SEQUENCIAL)** – Programa Cobol

```
//BRADXXSQ  JOB 'ARQSEQ',CLASS=C,MSGCLASS=X,NOTIFY=&SYSUID
//EXECUTA  EXEC PGM=BRADXX01
//STEPLIB  DD   DSN=GR.GERAL.LOADLIB,DISP=SHR
//CLIENTES DD   DSN=GR.BRADXX.CLIENTES,DISP=OLD
```