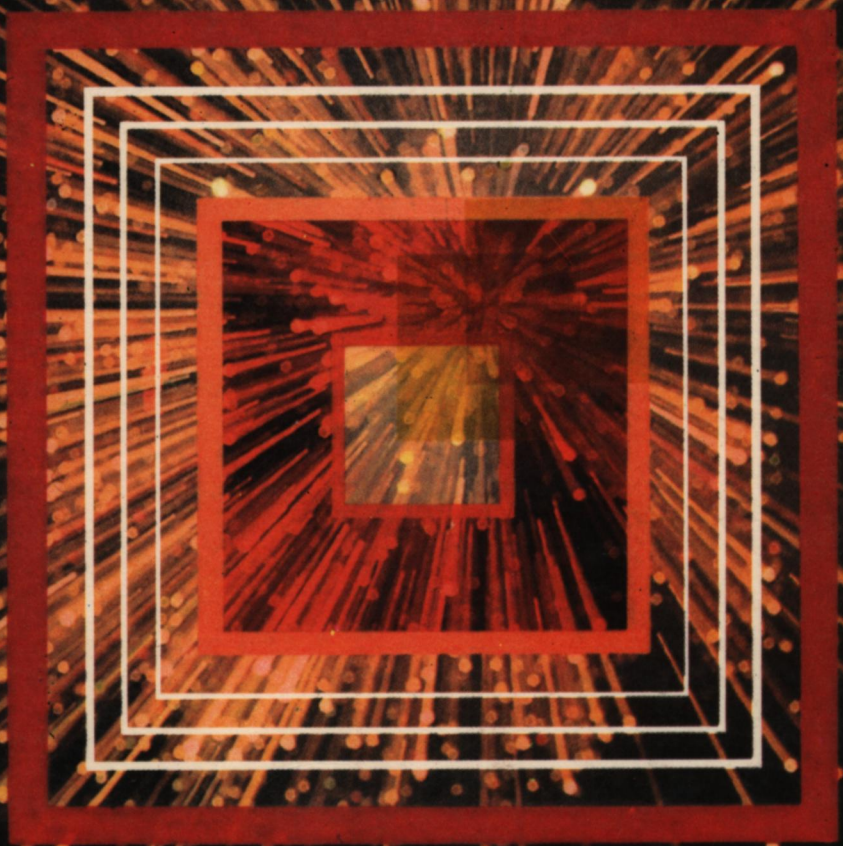
COBOL PARA MICROS

ELIANA PRAÇA



LTC LIVROS TECNICOS E CIENTIFICOS EDITORA S.A.

COBOL PARA MICROS

COBOL PARA MICROS

ELIANA PRAÇA

ANALISTA DE SISTEMAS DO NCE/UFRJ

LIVROS
TÉCNICOS E
CIENTÍFICOS EDITORA S.A.

Rio de Janeiro-RJ • São Paulo-SP

Revisor de provas:
Alberto Fernando de Araújo

Diagramação, paginação e desenhos:
Denilson Motta

Capa:
AG Comunicação Visual Assessoria e Projetos Ltda.

1ª edição: 1985
Reimpressão: 1986

CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional de Editores de Livros, RJ

Praça, Eliana.
P91c COBOL para micros / Eliana Praça. — Rio de Janeiro: LTC — Livros Técnicos e Cientfficos Editora S.A., 1985.

Apêndices

1. COBOL (Linguagem de programação para computadores) 2. Microcomputadores I. Título

84-0987 CDD — 001.6424

ISBN: 85-216-0405-X

LIVROS TECNICOS E CIENTIFICOS EDITORA S.A.

MATRIZ	FILIAL
Rua Vieira Bueno, 21 20.920 – Rio de Janeiro – RJ Brasil – End. Telegráfico: LITECE Tels.: 580-6055 Vendas: 580-9374	Rua Vitória, 486 – 2.º andar 01.210 – São Paulo – SP Tel.: (011) 223-6823 Caixa Postal 4.817

À Lucianna,
pelo tempo que lhe foi roubado.

À Mônica
por ajudar a concretizar dois
grandes sonhos.

PREFÁCIO

A idéia deste trabalho surgiu da inexistência, em Português, de textos didáticos relativos ao uso da linguagem COBOL em microcomputadores.

O presente texto refere-se ao COBOL 80, utilizado por micros da família INTEL 8080 que "rodam" com CP/M.

Gostaríamos de agradecer a José Antonio dos Santos Borges pelo incentivo dado na transformação da apostila no presente trabalho.

Eliana Praça

SUMÁRIO

NOTAÇÃO UTILIZADA

Para a descrição da linguagem esta publicação utiliza a seguinte notação:

1. Todas as palavras escritas em maiúsculas são palavras reservadas.
2. As palavras reservadas em negrito são palavras-chave e não podem ser omitidas da cláusula. As que não estão em negrito são palavras opcionais e são usadas apenas para tornar a cláusula mais documentacional.
3. As palavras escritas em letras minúsculas deverão ser substituídas por um nome dado pelo programador ou por um literal. Estas palavras aparecem sempre escritas entre ⟨ ⟩.
4. Palavras entre { } representam uma opção mutuamente exclusiva, isto é, apenas uma delas deverá ser utilizada.
5. Reticências (...) indicam que a cláusula pode ocorrer mais de uma vez.
6. Elementos opcionais (literais ou nomes dados pelo programador) são listados entre [ ].

Exemplo:

ADD { ⟨nome 1⟩ / ⟨literal 1⟩ } [{ ⟨nome 2⟩ / ⟨literal 2⟩ }]... { **TO** / **GIVING** } ⟨nome n⟩ [**ROUNDED**]

.ADD, TO, GIVING e ROUNDED são palavras reservadas.

.TO ou GIVING (apenas um) tem que ser usado.

.ROUNDED é opcional, mas o seu uso não é apenas para efeito de documentação (repare que ela está em negrito).

.⟨nome⟩ e ⟨literal⟩ serão fornecidos pelo programador. Repare que antes da palavra TO (ou GIVING) podem ser fornecidos vários nomes (ou literais).

.Assim sendo, são comandos válidos em COBOL:

```
ADD INPS IMP-RENDA GIVING DESCONTOS ROUNDED.
ADD 1 TO CONTA-PAGINA.
ADD SALARIO-FAMILIA HORAS-EXTRAS GRATIFICACAO TO
    SALARIO.
```

1

O COBOL

1.1 HISTÓRICO

COBOL (Common Business Oriented Language) é uma linguagem de Programação de alto nível projetada para aplicações comerciais.

As especificações originais para o COBOL foram esboçadas em maio de 1959, na "Conference on Data System Language" onde se reuniram representantes de vários fabricantes de computadores e usuários em geral, objetivando estabelecer uma linguagem padrão de programação para fins comerciais.

O projeto foi então iniciado e, em agosto de 1961, surgiu a primeira versão do COBOL. A partir daí o COBOL tem sido consideravelmente melhorado, resultando em versões que aumentam a potencialidade da linguagem.

1.2 A LINGUAGEM

COBOL é uma linguagem autodescritiva, com uma organização rígida e bem definida, com o propósito de facilitar a escrita e a leitura, tanto para o programador como para o leigo.

Um programa COBOL é sempre composto de entradas (frases), devidamente dispostas em divisões, seções e parágrafos. Assim sendo, para uma compreensão mais perfeita de sua organização, compararemos um programa COBOL com um livro:

Um livro geralmente está dividido em vários capítulos que são suas DIVISÕES, cada uma com uma função específica dentro do programa, porém sempre ligadas entre si; pode, assim, ter vários capítulos, enquanto um programa COBOL só possui quatro divisões.

Para melhor compreensão do texto o autor de um livro, baseando-se em seu estilo pessoal, pode dividir um capítulo em várias "partes", agrupando melhor os diferentes tópicos a serem abordados. Da mesma maneira, uma divisão em COBOL está dividida em partes que são as "SEÇÕES", de tal forma que cada seção tem uma função específica dentro da divisão, distribuindo melhor as tarefas e informações atribuídas a ela.

Dentro dessas partes estão as frases, agrupadas em parágrafos, dividindo definitivamente os assuntos do livro. Em COBOL também temos as frases e parágrafos que são, respectivamente, as "ENTRADAS" e "PARÁGRAFOS", e que possuem a função de informar o que realmente se deseja transmitir em cada parte do programa.

Desta forma, fica claro que é por meio de "ENTRADAS" que determinamos as tarefas que devem ser processadas, sendo uma entrada o item elementar de um programa COBOL.

1.3 AS DIVISÕES

Um programa COBOL possui quatro divisões que aparecem sempre nesta ordem:

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION

1.3.1 — IDENTIFICATION DIVISION

É a primeira divisão do programa. Sua função é identificar o programa de forma mais geral, processando informações tais como: o nome do programa, data da escrita, autor e comentários em geral.

Não possui seções e suas entradas estão agrupadas somente em parágrafos com nomes fixos.

Exemplo de IDENTIFICATION DIVISION:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  EXEMPLO1.
AUTHOR.  ELIANA.
INSTALLATION.  MICROLOGICA.
DATE-WRITTEN.  AGOSTO DE 1984.

**********************************************************
* PROGRAMA-EXEMPLO DO CURSO DE COBOL *
**********************************************************
```

1.3.2 — ENVIRONMENT DIVISION

É a divisão que interliga o programa-lógico com o computador, descrevendo principalmente os equipamentos utilizados. É a única parte do programa que sofre grandes alterações quando processamos o programa em computadores diferentes, pois é nesta divisão que são descritos os equipamentos físicos que o programa utiliza.

São também fornecidas informações de ordem lógica dos arquivos usados, tais como método de acesso, disposição dos arquivos e outras, possibilitando a ligação lógico-física destes arquivos com o sistema operacional.

A ENVIRONMENT DIVISION é subdividida em duas seções, com parágrafos predefinidos:

— CONFIGURATION SECTION, que trata das características do sistema.
— INPUT-OUTPUT SECTION, que controla os arquivos usados pelo programa.

Exemplo de ENVIRONMENT DIVISION:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  SDE-42.
OBJECT COMPUTER.  SDE-42.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT DADOS ASSIGN TO DISK.
     SELECT LISTAG ASSIGN TO PRINTER.
```

1.3.3 — DATA DIVISION

É a divisão que trata e descreve os dados que serão manipulados durante o processamento.

A DATA DIVISION subdivide-se em 3 seções:

```
FILE-SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
```

Cada uma dessas seções é opcional, podendo ser omitida se não for necessária. Contudo, se a seção for incluída ela deve respeitar a ordem.

Exemplo de DATA DIVISION:

```
DATA DIVISION.
FILE SECTION.
FD   DADOS
     LABEL RECORD IS STANDARD
     VALUE OF FILE-ID IS "DADOS.DAT"
     RECORD CONTAINS 80 CHARACTERS
     DATA RECORD IS CADAUM.
01   CADAUM.
     03 NUMERO-FUNC          PIC 9(5).
     03 NOME-FUNC            PIC X(40).
     03 ENDERECO-FUNC        PIC X(35).
```

```
FD  LISTAG
    LABEL RECORD IS OMITTED
    RECORD CONTAINS 132 CHARACTERS
    DATA RECORD IS LINHA.
01  LINHA                          PIC  X(132).
WORKING-STORAGE SECTION.
77  CONTA-LINHA                    PIC  9(5) VALUE ZEROS.
77  CONTA-LIDOS                    PIC  9(5) VALUE 0.
01  IMPRESSÃO.
    03 FILLER                      PIC  X(38) VALUE SPACES.
    03 NUMERO                      PIC  9 (5).
    03 FILLER                      PIC  X (10) VALUE SPACES.
    03 NOME                        PIC  X(40).
    03 FILLER                      PIC  X(39) VALUE SPACES.
```

1.3.4 — PROCEDURE DIVISION

É a divisão do programa que especifica as ações necessárias para o processamento dos dados em geral: controle de execução, entrada e saída, movimento dos dados, etc. Os procedimentos descritos na PROCEDURE DIVISION estão, geralmente, agrupados em parágrafos cujos nomes são fornecidos pelo programador. Tais parágrafos podem, ainda, estar agrupados em SECTIONS também criadas pelo programador.

Exemplo de PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
UNICA SECTION.
ABRE-ARQUIVOS.
   OPEN INPUT DADOS
        OUTPUT LISTAG.
LEITURA.
    READ DADOS AT END GO TO ACABOU.
    ADD 1 TO CONTA-LIDOS.
    MOVE NUMERO-FUNC TO NUMERO.
    MOVE NOME-FUNC TO NOME.
    WRITE LINHA FROM IMPRESSAO.
    ADD 1 TO CONTA-LINHA.
    GO TO LEITURA.
```

```
ACABOU.
    DISPLAY "REGISTROS LIDOS "  CONTA-LIDOS.
    DISPLAY "LINHAS IMPRESSAS "  CONTA-LINHA.
    CLOSE DADOS LISTAG.
    STOP RUN.
```

1.4 ELEMENTOS DA LINGUAGEM

1.4.1 — Símbolos

São caracteres especiais que têm significado particular para o compilador. São em número limitado e sua utilização obedece a uma série de regras de pontuação.

Os símbolos podem ser divididos em quatro categorias:

. Aritméticos

+adição	* multiplicação	** exponenciação
– subtração	/ divisão	= igualdade
(	)	

. Pontuação

. , ; ( )

. Relação

= > <

. Edição

B 0 + – CR DB Z * $, . /

Para a reunião correta dos símbolos existem as seguintes regras de pontuação:

1. Todas as entradas em COBOL, assim como as frases em Português, terminam com um ponto.
2. A mudança de linha é feita da forma mais livre possível, sendo possível até no meio de uma palavra (para saber como fazê-lo veja o item 1.5).
3. Uma palavra (ou literal) pode ser delimitada por um espaço em branco, ponto, abre-parênteses, vírgula ou ponto-e-vírgula.
4. Um ou mais espaços em branco podem seguir um fecha-parênteses ou preceder um abre-parênteses.
5. O ponto, a vírgula ou o ponto-e-vírgula usados como um símbolo de pontuação podem ser precedidos por um ou mais espaços em branco e devem ser seguidos por, pelo menos, um espaço em branco.
6. Sinais de relação, assim como os sinais aritméticos, devem estar sempre entre espaços em branco.
7. Quando o ponto, a vírgula, o símbolo + ou o símbolo – são usados em cláusulas de edição não podem ser precedidos nem seguidos de espaço em branco.
8. O ponto-e-vírgula e a vírgula podem ser usados para separar as cláusulas de uma entrada.

1.4.2 — Nomes dados pelo programador

São os nomes que o programador atribui a seus dados, parágrafos e seções.

Para a criação correta de um nome devem ser observadas as seguintes regras:

1. Um nome pode ter no máximo 30 (trinta) caracteres.
2. Pode conter as letras (A-Z), os dígitos (0-9) e hifens (-).
3. Embora possa conter hifens, um nome não pode começar nem terminar por um. O primeiro caractere de um nome deve ser sempre uma letra.
4. Não pode haver espaços em branco no meio dos nomes. O espaço em branco, assim como os símbolos de pontuação, indicam o término do nome.
5. Um nome não pode ser escrito como uma palavra reservada (veja o próximo item).

Exemplo:

Válidos:

DADO-DE-ENTRADA
NOME-DO-FUNCIONARIO

Inválidos:

NOME DO ALUNO
DATA
SPACES

1.4.3 — Palavras Reservadas

São palavras que possuem significado especial para o compilador. Possuem funções específicas e, quando usadas, não podem ser alteradas.

Existem cerca de duzentas palavras reservadas mas em pouco tempo o programador conhece quase a sua totalidade.

O programador não poderá dar o nome de POSITIVE para qualquer um de seus dados, pois POSITIVE é uma palavra reservada e, se tal ocorrer, o compilador vai emitir uma mensagem de erro avisando que onde era esperado um nome foi encontrada a palavra POSITIVE.

A palavra PICTURE é uma palavra reservada, e como tal não pode ser nome de variável; porém a palavra PIKTURE poderá perfeitamente ser nome de qualquer variável dentro do programa.

Da mesma maneira, EQUAL e SIGN são palavras reservadas, porém EQUAL-SIGN poderá ser usada como nome de variável.

Obs.:

No apêndice A, são encontradas todas as palavras reservadas do COBOL-80.

1.4.4 — Literais

Os literais são, basicamente, os diversos valores que uma variável pode assumir. São autodescritivos, não precisando, portanto, de definição separada dentro do programa.

Os literais podem ser numéricos ou não numéricos. Para a formação de um literal numérico devem ser observadas as seguintes regras:

1. Um literal numérico possui no máximo 18 caracteres e, no mínimo, um. Pode ser formado pelos dígitos 0 a 9, possuir um sinal (+ ou –) e um ponto decimal.

O valor de um literal numérico é a quantidade algébrica representada por seus caracteres.

2. Se houver sinal, este deverá ser o caractere mais à esquerda do literal. Os sinais + ou – determinam o sinal do cálculo e são, evidentemente, mutuamente exclusivos. O sinal + é opcional.

3. O ponto decimal do literal pode aparecer em qualquer lugar, exceto como caractere mais à direita. Números inteiros são escritos sem o ponto decimal.

Exs.:

Literais numéricos válidos:

100
3.0
159
– 47.5
+3.2892

Literais numéricos inválidos:

132.
3.79+
A237
1.E+72

Um literal não numérico pode ser formado por qualquer caractere exceto o delimitador utilizado. Este delimitador pode ser a aspa ou o apóstrofe. Espaços em branco inseridos entre as aspas são considerados pertencentes ao literal. Um literal não numérico não pode conter mais de 120 caracteres (o delimitador não é considerado) e deve conter pelo menos um caractere.

Exs.:

– Literais não numéricos válidos:

"13A29+" 'CURSO DE COBOL' "3.879" (diferente de 3.879)
"SPACES" 'NOME-DO-PROGRAMADOR' "(+-*/**)"
"BOB 'S"

– Literais não numéricos inválidos:

1439 'VALOR' "COBOL
"BOB"S"

1.4.5 — Constantes figurativas

A constante figurativa é um tipo especial de literal. Ela representa um valor conhecido.

São constantes figurativas:

ZERO	– Representa o valor 0.
SPACE	– Representa o espaço em branco.
LOW-VALUE	– Representa o menor valor que o dado pode assumir.
HIGH-VALUE	– Representa o maior valor que o dado pode assumir.
QUOTE	– Representa a aspa.

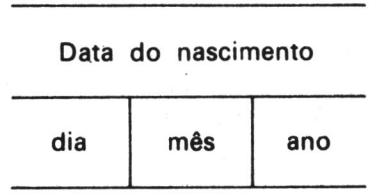
1.4.6 — Número de nível

O conceito de número de nível está intimamente ligado ao conceito de estrutura; portanto, a boa compreensão de sua aplicação depende de conhecimentos ainda não adquiridos. Logo, a sua definição aqui fornecida é simplesmente de ordem conceitual.

Números de nível são números que servem para atribuir níveis aos itens de dados, relacionando-os uns com os outros, isto é, estabelecendo a ligação entre estes dados.

Um exemplo clássico é a data do nascimento de uma pessoa que é composta de dia, mês e ano. Assim, diz-se que a data do nascimento está em um nível, e dia, mês e ano, em outro.

É importante notar-se que dia, mês e ano e data do nascimento, apesar de estarem em níveis diferentes, representam a mesma coisa.



Um número de nível está relacionado com um grupo de dados, partindo do mais externo (o mais global), até o mais interno ou mais elementar (sem subdivisões).

Os números que podem ser usados como números de nível são de 01 a 49, 77 e 88. Os números de 01 a 49 são usados para designar os níveis dos dados que formam registros, isto é, pertencem a alguma estrutura. O nível 01 é sempre considerado como um todo, ou seja, a parte mais externa da estrutura, aquela que é subdividida, enquanto que os de 02 a 49 são atribuídos àqueles que são partes dos registros, ou das estruturas, isto é, às diversas subdivisões que o nível 01 puder englobar.

O nível 77 é usado em entradas que descrevem dados independentes, isto é, aqueles que não fazem parte de registros, ou ainda, aqueles que não são subdivisíveis.

O nível 88 é usado em entradas que atribuem nomes a valores específicos que os dados podem tomar.

Resumindo, teremos:

01 a 49 – Para descrever os registros, dados que se subdividem (estruturas).

77 – Para dados independentes, não subdivisíveis.

88 – Para relacionar nomes a certos valores que os dados poderão assumir.

1.4.7 — PICTURE

As "PICTURES" são usadas para descrição dos dados e suas características, como:

- O tipo de dado (alfabético, numérico ou alfanumérico)
- Tamanho do dado (número de caracteres ou dígitos)
- Sinal do dado (+ ou –)
- Localização do ponto decimal

As "PICTURES" também são usadas para edição, considerando-se a eliminação, a substituição ou a inclusão de caracteres que deverá ser feita para a formação do dado.

Uma PICTURE é uma cadeia de, no máximo, 30 caracteres, composta de um ou mais dos seguintes caracteres:

A P S V X 9 B 0 + – CR DB Z * $, . /

Um estudo mais aprofundado sobre "PICTURES" será feito mais tarde.

1.5 FOLHA DE CODIFICAÇÃO COBOL

A folha de codificação COBOL é dividida em quatro campos:

. Colunas 1 a 6

Usadas para numerar linhas de um programa. A numeração é feita em ordem crescente. Esta numeração é opcional.

. Coluna 7

Se preenchida, só pode conter um dos símbolos abaixo:

/ Usada para indicar que haverá mudança para o topo de uma nova página na listagem do programa-fonte.

* Usado para indicar que toda aquela linha é de comentário.

– Usado para indicar continuação de literais não numéricos ou de palavras (veja obs.: 1 e 2).

. Colunas 8 a 72

Usadas para as entradas do programa. Estas colunas estão agrupadas em duas margens: Margem A (colunas 8 a 11) e Margem B (colunas 12 a 72) (veja obs.: 3).

. Colunas 73 a 80

Usadas para identificação do programa. Esta identificação é opcional.

1.5.1 — Observações

1. Um literal não numérico é continuado da seguinte forma: coloca-se um hífen na coluna 7 continuando-se o literal na margem B, começando-o com o delimitador utilizado. A linha anterior é considerada até a coluna 72.

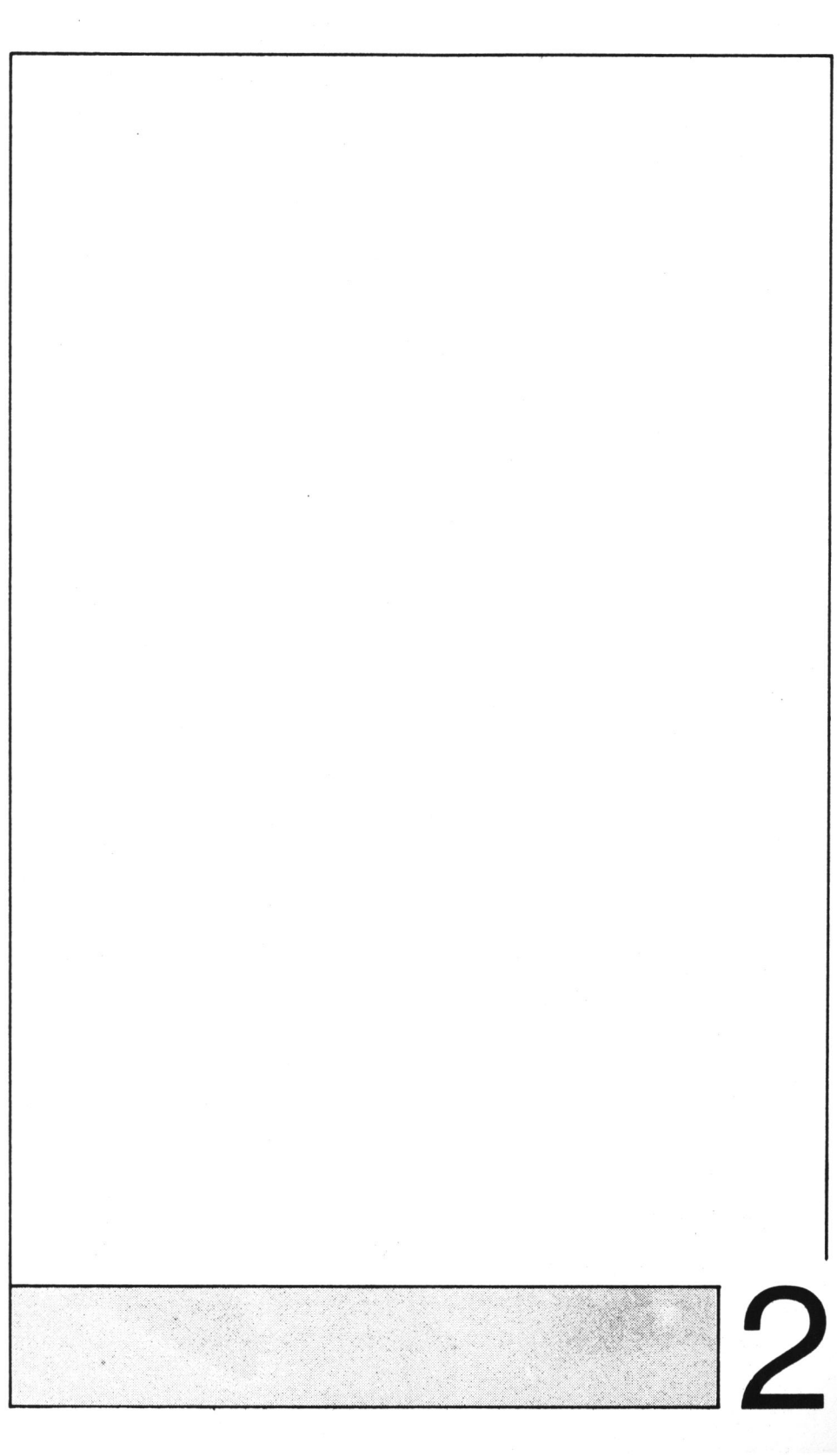
Exs.:

```
77 EXEMPLO PIC X(46) VALUE "EXEMPLO DE CONTINUACAO
-    "DE LITERAL NAO-NUMERICO".
```

2. A continuação de palavras é feita de forma semelhante à descrita acima. Apenas não é necessária a presença do delimitador (visto que ela não o possui).
3. Somente são codificados na margem A nomes de divisões, seções e parágrafos, descrições de arquivos e números de nível. As demais entradas não são permitidas na margem A.

Exs.:

```
PROCEDURE DIVISION.
UNICA SECTION.
ABRE-ARQUIVOS.
    OPEN INPUT DADOS
         OUTPUT LISTAG.
LEITURA.
    READ DADOS AT END GO TO ACABOU.
    ADD 1 TO CONTA-LIDOS.
    MOVE NUMERO-FUNC TO NUMERO.
    MOVE NOME-FUNC TO NOME.
    WRITE LINHA FROM IMPRESSAO.
    ADD 1 TO CONT-LINHA.
    GO TO LEITURA.
ACABOU.
    DISPLAY "REGISTROS LIDOS " CONTA-LIDOS.
    DISPLAY "LINHAS IMPRESSAS " CONTA-LINHA.
    CLOSE DADOS LISTAG.
    STOP RUN.
```



IDENTIFICATION DIVISION

É a primeira divisão de um programa COBOL. Sua função é identificar o programa de uma forma geral.

Esta divisão não possui seções. Suas entradas estão agrupadas em parágrafos de nomes fixos. Embora sendo obrigatório o uso da divisão, todos os seus parágrafos são opcionais.

2.1 ESTRUTURA DA DIVISÃO

IDENTIFICATION DIVISION.
PROGRAM-ID. nome-do-programa.

AUTHOR. nome-do-programador.
INSTALLATION. local onde foi feito o programa.
DATE-WRITTEN. data em que foi escrito o programa.
DATE-COMPILED. data em que o programa foi compilado.
SECURITY. comentários sobre a segurança do programa e/ou de seus arquivos.

Apenas o parágrafo PROGRAM-ID é obrigatório, sendo os outros opcionais. Contudo, se eles estão presentes, devem aparecer nesta ordem.

O nome do programa, dado no parágrafo PROGRAM-ID, deve começar com letra.

3

ENVIRONMENT DIVISION

A ENVIRONMENT DIVISION é a segunda divisão de um programa COBOL. É usada para especificar o computador usado para a compilação, o computador a ser usado pelo programa objeto e para correlacionar os arquivos lógicos com os equipamentos periféricos.

3.1 ESTRUTURA DA DIVISÃO

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computador usado para a compilação.

OBJECT-COMPUTER. computador usado para execução.
SPECIAL-NAMES. relaciona nomes internos a equipamentos.
INPUT-OUTPUT SECTION.
FILE-CONTROL. nomeia e associa arquivos a periféricos.
I-O-CONTROL. define técnicas de controle.

A ENVIRONMENT DIVISION é formada por duas seções: a CONFIGURATION SECTION e a INPUT-OUTPUT SECTION.

A CONFIGURATION SECTION trata das características do computador-fonte e do computador-objeto, enquanto que a INPUT-OUTPUT SECTION trata das informações necessárias ao controle da transmissão e manipulação de dados entre o meio externo e o programa-objeto.

3.2 A CONFIGURATION SECTION

Esta seção é dividida em três parágrafos: SOURCE-COMPUTER, OBJECT-COMPUTER e SPECIAL-NAMES.

3.2.1 — O parágrafo SOURCE-COMPUTER

A função deste parágrafo é determinar o computador a ser usado na compilação do programa.

Obs.:

Este parágrafo possui a cláusula DEBBUGGIN MODE que não será vista nesta publicação.

3.2.2 — O parágrafo OBJECT-COMPUTER

A função deste parágrafo é determinar o computador a ser usado na execução do programa.

3.2.3 — O parágrafo SPECIAL-NAMES

Este parágrafo é usado tanto para interligar nomes internos do programa a nomes externos, como para atribuir nomes mnemônicos a nomes de equipamentos.

Sua forma geral é:

SPECIAL-NAMES.
[**CURRENCY SIGN IS** ⟨LITERAL⟩]
[**DECIMAL-POINT IS COMMA**]
[**PRINTER IS** ⟨mnemonico⟩]

O literal que aparece na cláusula CURRENCY SIGN é usado na cláusula PICTURE para representar o símbolo $. O literal deve ser um literal não numérico de apenas um caractere diferente de:

. aspas
. dígito (0 a 9)
. qualquer dos caracteres usados na cláusula PICTURE.

Se a cláusula não está presente o $ é usado na cláusula PICTURE.

A cláusula DECIMAL-POINT IS COMMA substitui o ponto decimal pela vírgula nas cláusulas de descrição de dados (PICTURE) e em literais numéricos.

A terceira cláusula serve para definir o nome a ser usado no comando DISPLAY...UPON... (veremos na PROCEDURE DIVISION).

3.2.4 — Exemplo de CONFIGURATION SECTION

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  SDE-42.
OBJECT-COMPUTER.  SDE-42.
SPECIAL-NAMES.
    CURRENCY SIGN IS "E"
    DECIMAL-POINT IS COMMA
    PRINTER IS IMPRESSORA.
```

3.3 A INPUT-OUTPUT SECTION

A INPUT-OUTPUT SECTION descreve os arquivos a serem usados pelo programa-objeto e a forma de usar estes arquivos.

Esta seção é composta de dois parágrafos:

FILE-CONTROL e I-O-CONTROL.

3.3.1 — O parágrafo FILE-CONTROL

A função deste parágrafo é fornecer o nome interno de cada arquivo, associando-o ao periférico correspondente.

Sua forma geral, simplificada, é:

```
FILE-CONTROL.
  SELECT <nome-arq> ASSIGN TO { DISK    }
                              { PRINTER }
                    [ RESERVE <inteiro> {AREA  } ]
                                        {AREAS }
                    [ FILE-STATUS IS <nome de dado 1> ]
                    [ ACCESS MODE IS {SEQUENTIAL} ]
                                    {RANDOM    }
                                    {DYNAMIC   }
                    [ ORGANIZATION IS {SEQUENTIAL} ]
                                      {INDEXED    }
                                      {RELATIVE   }
                    [ RECORD KEY IS <nome de dado 2> ]
                    [ RELATIVE KEY IS <nome de dado 3> ].
```

3.3.1.1 — A entrada SELECT.

Usada para nomear os arquivos manipulados pelo programa. Todos os arquivos devem ser nomeados uma e apenas uma vez.

A cláusula ASSIGN é usada para associar um arquivo com o seu "periférico".

Exemplos:

1. SELECT CARTAO ASSIGN TO DISK.

Indica que o arquivo de nome CARTAO está (ou será) gravado em disco.

2. SELECT SAIDA ASSIGN TO PRINTER.

Indica que o arquivo de nome SAIDA será impresso em listagem.

A cláusula RESERVE permite que se altere o número de buffers do arquivo. O valor ⟨inteiro⟩ pode variar de 1 a 7, mas se for maior que 2, apenas 2 buffers são reservados.

A cláusula FILE-STATUS é usada para atribuir o código de sucesso (ou não) de uma operação de leitura e/ou escrita no arquivo a ⟨nome de dado 1⟩. Este código é composto de dois caracteres; assim, ⟨nome de dado 1⟩ deve estar definido na WORKING-STORAGE SECTION ou na LINKAGE SECTION com PIC XX (veremos mais adiante).

Para arquivos seqüenciais o status pode ter os seguintes valores:

00 – Operação bem sucedida
10 – Fim de arquivo
30 – Erro
34 – Falta de espaço em disco

A cláusula ACCESS MODE determina o modo de acesso ao arquivo em disco. Se usamos a opção SEQUENTIAL os registros são manipulados seqüencialmente, isto é, o próximo registro a ser lido ou gravado é o registro contíguo ao registro corrente. Se a opção RANDOM é especificada, o sistema obtém cada registro randomicamente, isto é, o próximo registro a ser lido ou gravado é endereçado pelo valor da RECORD KEY ou da RELATIVE KEY.

A opção DYNAMIC não será estudada nesta publicação.

A ausência da cláusula ACCESS MODE significa que ACCESS MODE IS SEQUENTIAL.

A cláusula ORGANIZATION determina a organização do arquivo. ORGANIZATION IS SEQUENTIAL determina que o arquivo está organizado seqüencialmente, implicando que ACCESS MODE IS SEQUENTIAL deve ser usada. ORGANIZATION IS INDEXED determina que o arquivo é indexado, fazendo com que ACCESS MODE possa ter qualquer das 3 opções. ORGANIZATION IS RELATIVE determina que o arquivo é relativo e qualquer das 3 opções de ACCESS MODE pode ser usada. A ausência da cláusula ORGANIZATION determina que ORGANIZATION IS SEQUENTIAL.

A cláusula RECORD KEY é utilizada para arquivos indexados e o valor de ⟨nome de dado 2⟩ determinará o registro a ser lido (ou gravado). Cada registro deve ter um único valor para a RECORD KEY. A descrição de ⟨nome de dado 2⟩ deve fazer parte da descrição do registro do arquivo.

A cláusula RELATIVE KEY é usada para arquivos relativos e ⟨nome de dado 3⟩ deve ser descrito como um campo binário e não pode pertencer à descrição de registro de nenhum arquivo.

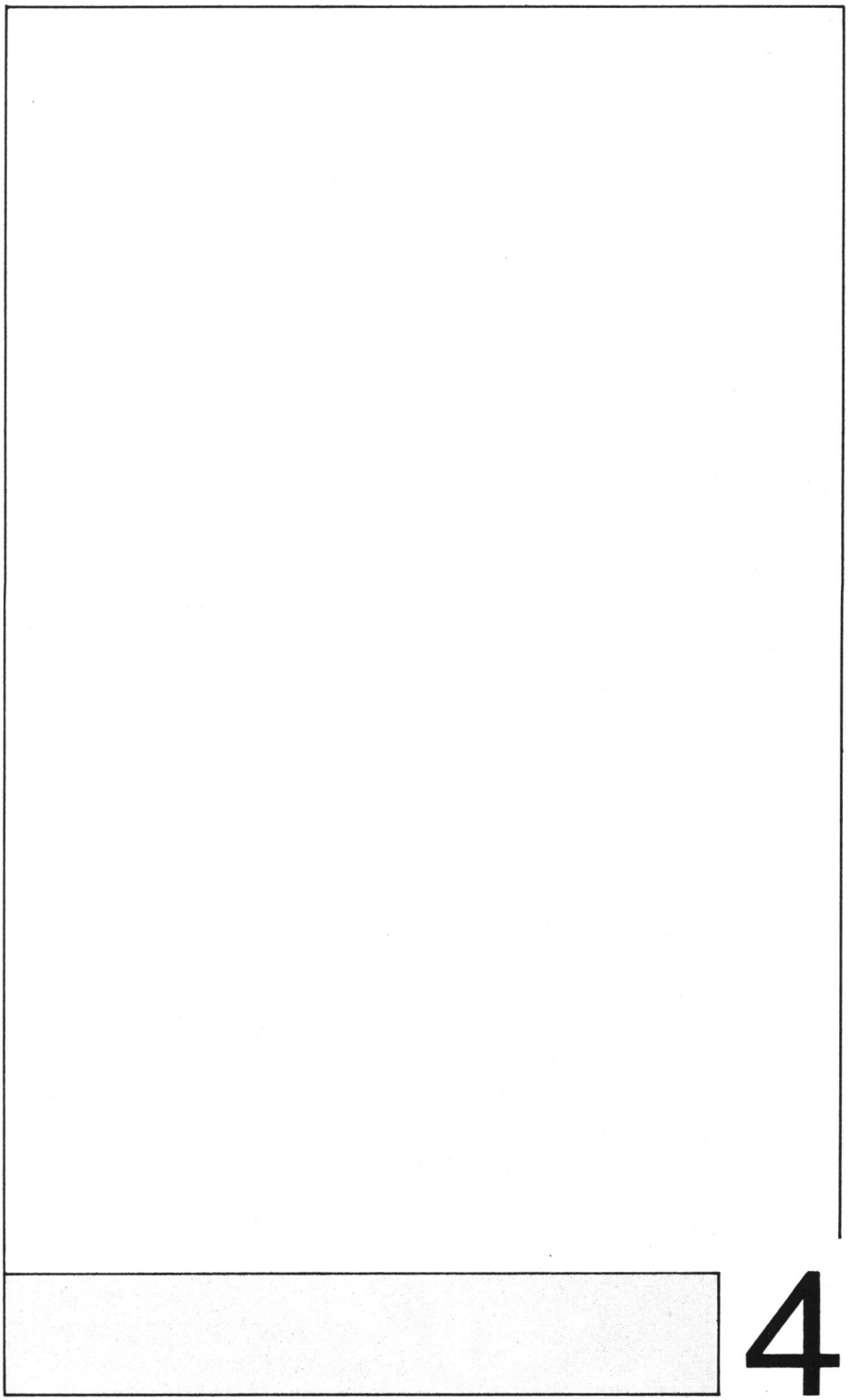
Maiores informações a respeito de organização de arquivos podem ser encontradas no Apêndice B desta publicação.

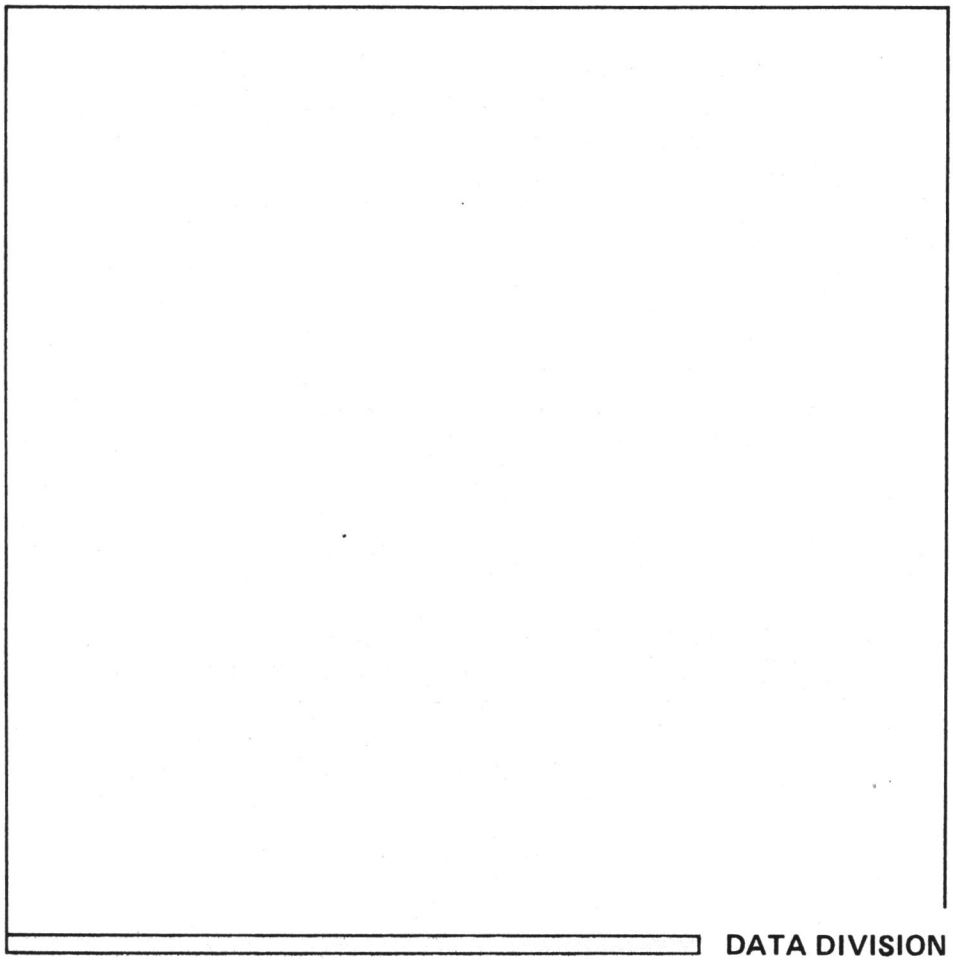
3.3.2 — O parágrafo I-O-CONTROL

A principal função deste parágrafo é especificar os arquivos que serão usados em horas diferentes no programa. O uso destes arquivos deve ser mutuamente exclusivo. Assim, os buffers e o mesmo espaço de memória podem ser usados para eles.

A cláusula que faz com que isto seja possível possui a seguinte forma geral:

SAME AREA FOR ⟨nome arq 1⟩ ⟨nome arq 2⟩ ...





DATA DIVISION

A DATA DIVISION é a terceira divisão de um programa COBOL. É usada para descrever todos os dados a serem usados pelo programa, sejam eles externos (informações gravadas em arquivos) ou internos (só existem durante a execução do programa).

4.1 ESTRUTURA DA DIVISÃO

DATA DIVISION.
FILE SECTION.
{ entrada de descrição de arquivo }
{ entrada de descrição de registro } ...

WORKING-STORAGE SECTION.
[entrada de descrição de item independentes] ...
[entrada de descrição de registro] ...

A DATA DIVISION é formada por três seções:

FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.

Cada seção é opcional e pode ser omitida se não for necessária. Contudo, se a seção for incluída, a ordem é importante.

Nesta publicação estudaremos apenas duas seções:

FILE SECTION.
WORKING-STORAGE SECTION.

A FILE SECTION preocupa-se em descrever os dados que serão lidos e/ou gravados externamente (dados que existiam antes do programa e dados que continuarão a existir após o programa ser executado) enquanto que a WORKING-STORAGE SECTION descreve os dados internos, isto é, aqueles que só existem durante a execução do programa e por este são desenvolvidos.

4.2 ORGANIZAÇÃO DAS ENTRADAS NA DATA DIVISION

Toda entrada da DATA DIVISION começa por um indicador de nível ou por um número de nível, seguido por um ou mais espaços, seguidos pelo nome do dado o qual é seguido por uma série de cláusulas independentes que descrevem o dado. A última cláusula termina sempre por ponto.

Exemplo:

```
03 DESCONTOS          PIC 9(5)V99 VALUE ZERO.
```

Onde:

03 → Número de nível
DESCONTOS → Nome do dado
PIC 9(5)V99 VALUE ZERO. → cláusulas de descrição

4.2.1 – Indicador de nível

O indicador de nível FD (File Description) é usado para indicar o início de uma entrada de descrição de arquivo. Cada indicador deverá começar na margem A (coluna 8) seguido, na margem B (coluna 12), pelo nome do arquivo e suas cláusulas descritivas.

Exemplo:

```
FD CADASTRO RECORD CONTAINS 550 CHARACTERS.
```

Temos:

FD → Indicador de nível
CADASTRO → Nome do arquivo
RECORD CONTAINS 550 CHARACTERS. → cláusula descritiva

4.2.2 – Número de nível

Os números de nível são usados para estruturar as diversas subdivisões lógicas de um registro.

Um registro é composto de itens-grupo e itens elementares. Um item-grupo é aquele que possui subdivisões e um item elementar é aquele que não as possui.

Itens-grupos são usados para referenciar uma série de itens elementares, isto é, itens elementares estão reunidos em itens-grupos. Vários itens-grupos, por sua vez, podem estar reunidos em outros itens-grupos maiores. Logo, um item elementar pode pertencer a vários itens-grupos.

Exemplo:

```
01 FUNCIONARIO.
   03 DADOS-PESSOAIS.
      05 NOME
      05 SEXO
      05 NATURALIDADE
      05 NACIONALIDADE
```

```
05 FILIACAO.
   07 PAI
   07 MAE
05 ENDERECO.
   07 RUA
   07 BAIRRO
   07 TELEFONE
   07 CIDADE
05 DATA-DO-NASCIMENTO.
   07 DIA
   07 MES
   07 ANO
05 IDENTIDADE.
   07 NUMERO
   07 ORGAO-EXPEDIDOR
   07 ESTADO
05 CPF
03 DADOS-FUNCIONAIS.
   05 PROFISSAO
   05 CARGO
   05 FUNCAO
   05 DATA-DE-ADMISSAO.
      07 DIA
      07 MES
      07 ANO
   05 NUM-DEPENDENTES
   05 ULTIMA-PROMOCAO.
      07 MES
      07 ANO
   05 FERIAS
```

São itens-grupos:

FUNCIONARIO, DADOS-PESSOAIS, FILIACAO, ENDERECO, DATA-DO-NASCIMENTO, IDENTIDADE, DADOS-FUNCIONAIS, DATA-DE-ADMISSAO e ULTIMA-PROMOCAO.

Repare que um item-grupo termina ao encontrar-se um número de nível menor ou igual ao número de nível do grupo. Um item-grupo pode ter no máximo 4095 caracteres.

São itens elementares:

NOME, SEXO, NATURALIDADE, NACIONALIDADE, PAI, MAE, RUA, BAIRRO e todos os outros não citados como itens-grupo.

Quando queremos fazer referências a todo o grupo citamos FUNCIONARIO, que engloba todos os outros itens.

Ao fazermos referências a ENDEREÇO, estamos nos referindo, simultaneamente, a RUA, BAIRRO, TELEFONE e CIDADE.

Pelo exemplo mostrado podemos concluir que os números de nível mostram a organização dos dados dentro do registro e, à medida que os dados se subdividem, seus números de nível devem aumentar, não necessitando porém de serem números consecutivos.

Os números de nível usados para a descrição de registros devem pertencer ao intervalo (01 a 49). Existem ainda os números de nível especiais (77 e 88), usados para a descrição de dados elementares.

77 – Usado em itens independentes (não pertencem a nenhuma estrutura).

88 – Usado para atribuir nomes a valores específicos.

Os números de nível menores que 10 podem ser escritos com apenas um algarismo.

4.3 QUALIFICADORES

Todo nome dado pelo programador usado em um programa COBOL deve ser único, a fim de que não exista dúvida quanto ao dado a que estamos nos referindo.

Reparem no exemplo dado em 4.2.2 que, quando o programador, em seu programa, fizer referência ao dado denominado DIA não saberemos exatamente a que DIA ele se refere: DIA do nascimento ou DIA de admissão. O mesmo comentário podemos fazer em relação ao MES e ANO (que ainda existem em ULTIMA-PROMOCAO).

A fim de tornar as referências precisas, devemos fazer uso de um dos qualificadores da linguagem COBOL: IN ou OF, que podem ser usados indistintamente.

Assim sendo, se o programador quisesse fazer referência ao MES de nascimento do funcionário deveria usar MES OF DATA-DO-NASCIMENTO. Ao se referir ao ANO da última promoção poderia usar ANO OF ULTIMA-PROMOCAO ou ANO IN ULTIMA-PROMOCAO.

Os qualificadores também podem ser usados para distinguir dados de mesmo nome que pertencem a registros diferentes.

Exemplo:

```
01 REG-CAD.
   03 NOME
   03 CURSO
   03 ENDERECO
01 REG-MOV.
   03 NOME
   03 TITULO
   03 SALARIO
```

Ao fazermos referência ao dado NOME, devemos qualificá-lo:

```
NOME OF REG-CAD,   NOME IN REG-CAD.
NOME OF REG-MOV,   NOME IN REG-MOV.
```

Podemos também utilizar os qualificadores, no caso de dois ou mais parágrafos de mesmo nome, em seções diferentes.

Exemplo:

```
PRIMEIRA SECTION.
LEITURA.
        ...
        ...
SEGUNDA SECTION.
LEITURA.
        ...
        ...
```

Ao fazermos referência ao parágrafo LEITURA fora da seção a que pertence, devemos qualificá-lo: LEITURA IN PRIMEIRA. LEITURA OF SEGUNDA.

A referência dentro da seção considera que o parágrafo referido é aquele a que ela pertence.

Um parágrafo pode ter apenas um qualificador, mas um nome de variável pode ter até 5 qualificadores. Nomes de arquivos não podem ser qualificados.

4.4 FILE SECTION

Conforme já visto, é nesta seção que são descritos os dados que serão lidos e/ou gravados externamente, ou seja, dados que existiam antes da

execução do programa e dados que continuarão a existir após o programa ser executado.

4.4.1 — A entrada FD

Esta entrada é usada para descrever os arquivos usados pelo programa.

Cada arquivo nomeado pela entrada SELECT da INPUT-OUTPUT SECTION deverá ser descrito por uma entrada FD.

Sua forma geral, simplificada, é:

```
FD <nome do arquivo>
   LABEL {RECORD  IS  } {OMITTED  }
         {RECORDS ARE} {STANDARD }
   [ VALUE OF FILE-ID "<d:nomearq.tip>" ]
   [ BLOCK CONTAINS <inteiro 1> {CHARACTERS} ]
                                {RECORDS    }
   [ RECORD CONTAINS <inteiro 2> TO <inteiro 3> CHARACTERS ]
   [ DATA {RECORD IS} <nome de dado 1> [, <nome de dado 2> ]].
          {RECORDS ARE}
```

A cláusula LABEL RECORD determina a existência (STANDARD) ou não (OMITTED) de "label" no arquivo. Arquivos associados a PRINTER devem usar a opção OMITTED e arquivos em DISK devem usar a opção STANDARD.

A cláusula VALUE OF FILE-ID é usada apenas para arquivos em DISK e serve para dar nome ao arquivo (nome com o qual ele ficará gravado ou será procurado no disco). O nome do arquivo é fornecido por "d:nomearq.tip". Esta cláusula é obrigatória para arquivos em disco.

A cláusula BLOCK CONTAINS é usada para especificar o tamanho do bloco do arquivo. Seu uso é opcional quando o arquivo tem fator de blocagem igual a 1. Quando o tamanho do bloco é dado em caracteres a palavra reservada CHARACTERS não precisa constar da cláusula. Para o COBOL-80 esta cláusula não tem nenhum efeito.

A cláusula RECORD CONTAINS é usada para especificar o tamanho do registro. Para registros de tamanho fixo, <inteiro 3> não pode ser usado, mas é obrigatório para registros de tamanho variável. Neste caso <inteiro 2> representa o tamanho mínimo e <inteiro 3> o tamanho máximo que o registro pode assumir. Esta cláusula é usada apenas para documentação, pois o tamanho do registro é dado pelas cláusulas descritivas de seus campos.

A cláusula DATA RECORD é usada para nomear o registro do arquivo, servindo apenas para documentação.

Exemplo:

Descreva a entrada FD para o arquivo de nome CADASTRO que possui as seguintes características:

. Tamanho de registro – 150 caracteres
. Fator de blocagem – 10
. Nome de registro – REG-CAD

```
FD CADASTRO
    BLOCK CONTAINS 10 RECORDS
    RECORD CONTAINS 150 CHARACTERS
    DATA RECORD IS REG-CAD.
```

Obs.:

No COBOL-80 os arquivos não são blocados.

4.4.2 — Descrição de registros e dados

Cada entrada FD deve ser seguida por, pelo menos, uma entrada de número de nível 01.

A forma geral de uma entrada de descrição de registro é:

```
⟨numero de nivel⟩ {⟨nome de dado 1⟩}
                  {FILLER}
[REDEFINES ⟨nome de dado 2⟩]
[{PICTURE} IS ⟨cadeia de caracteres⟩]
 {PIC}
[USAGE IS {COMP}]
          {COMPUTATIONAL}
          {COMP-3}
          {COMPUTATIONAL-3}
          {DISPLAY}
          {INDEX}
```

[**OCCURS** ⟨inteiro 1⟩ TIMES
[{**ASCENDING**} KEY IS ⟨nome de dado 3⟩ [, ⟨nome de dado 4⟩]...]
 {**DESCENDING**}
...[**INDEXED** BY]⟨nome de indice 1⟩[, ⟨nome de indice 2⟩]...]]
[{**JUSTIFIED**} RIGHT]
 {**JUST**}
[**BLANK** WHEN ZERO]
[{**VALUE** IS }⟨literal⟩ [{**THRU**}⟨literal 2⟩]...].
 {**VALUES** ARE } {**THROUGH**}

Existe ainda outra forma de entrada de descrição de registro:

88 ⟨nome de condicao⟩ {**VALUE IS**} ⟨literal 1⟩ [{**THROUGH**} ⟨literal 2⟩].

O número de nível da opção 1 pode ser qualquer um pertencente ao intervalo (01 a 49) ou 77.

As cláusulas descritivas podem estar em qualquer ordem, exceto que ⟨nome de dado 1⟩ ou FILLER devem vir imediatamente após o número de nível e a cláusula REDEFINES, quando usada, deve seguir imediatamente ⟨nome de dado 1⟩.

As cláusulas PICTURE, JUSTIFIED e BLANK WHEN ZERO não podem ser especificadas para itens de grupo.

A opção 2 (número de nível 88) será vista no item 4.4.4.

Veremos agora cada uma das cláusulas descritivas:

. BLANK WHEN ZERO

Esta cláusula permite o branqueamento de um item quando seu valor é zero. Só pode ser usada para itens numéricos de edição de tamanho fixo.

. FILLER

Usada para especificar uma parte do registro que não será referenciada pelo programa.

. JUSTIFIED RIGHT

Faz com que o dado seja alinhado pela direita no campo receptor. Quando o campo é menor que o dado, os caracteres mais à esquerda são truncados. Esta cláusula não pode ser especificada para itens numéricos ou numéricos de edição, não podendo também ser especificada para itens de tamanho variável.

OCCURS

Esta cláusula é usada para a definição de tabelas de uma ou mais dimensões.

O valor de ⟨inteiro 1⟩ (vide forma geral) deve ser inteiro e positivo.

A cláusula OCCURS não pode ser usada em entradas iniciadas pelo número de nível 01 ou 77.

A opção INDEXED é usada quando se deseja definir um índice para trabalhar com a tabela. Cada dimensão deve conter uma opção INDEXED. Uma tabela pode ter, no máximo, 3 dimensões.

Exemplo:

Suponha que se possua uma tabela com a quantidade de alunos matriculados no Curso MICROLÓGICA a cada mês do ano de 1983.

Para trabalhar com estes dados poderíamos ter:

```
01 TABELA-DE-MATRÍCULAS.
   03 JANEIRO    PIC 9(5).
   03 FEVEREIRO  PIC 9(5).
        ⋮
   03 DEZEMBRO PIC 9(5).
```

Assim, se quiséssemos saber a quantidade de alunos de um determinado mês (MARCO, por exemplo) bastaria sabermos o valor armazenado em MARCO. Acontece, porém, que isto não é muito prático.

Para facilitar nosso trabalho poderíamos, então, construir uma tabela:

```
01 TABELA-DE-MATRÍCULAS.
   03 MATR-MES      PIC 9(5) OCCURS 12 TIMES.
```

Assim, se quiséssemos saber a quantidade de alunos de MARCO bastaria conhecermos o conteúdo de MATR-MES (3).

Suponha, agora, que possuamos esta mesma tabela para os anos de 83 e 84:

1983	1984
JANEIRO	JANEIRO
FEVEREIRO	FEVEREIRO
MARCO	MARCO
...	...
...	...
DEZEMBRO	DEZEMBRO

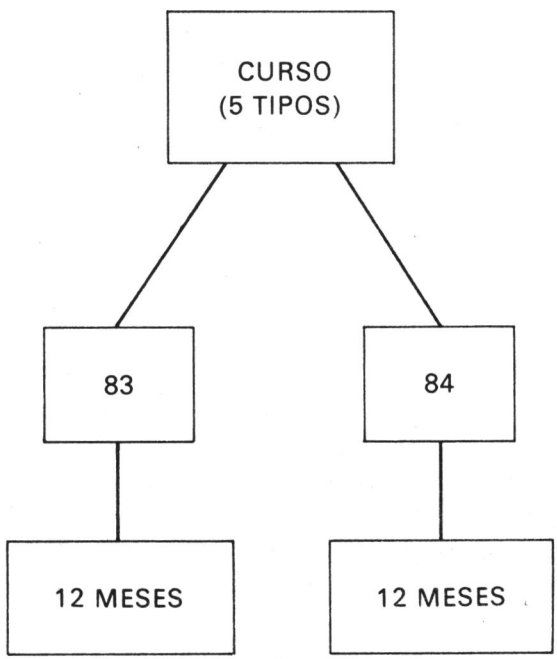
Uma boa forma de construí-la seria:

```
01 TABELA-DE-MATRICULAS.
   03 ANO OCCURS 2 TIMES.
      05 MATR-MES PIC 9(5) OCCURS 12 TIMES.
```

Assim:

MATR-MES(1 , 1) – representa as matrículas de JANEIRO de 1983.
MATR-MES(2 , 5) – representa as matrículas de MAIO de 1984.
MATR-MES(1 , 8) – representa as matrículas de AGOSTO de 1983.

Suponha agora que necessitássemos separar o total de matrículas destes alunos de acordo com o tipo de curso (5 cursos, por exemplo).



Poderíamos ter:

```
01 TABELA-DE-MATRICULAS
   03 TIPO-DE-CURSO OCCURS 5 TIMES.
      05 ANO OCCURS 2 TIMES.
         07 MATR-MES PIC 9(5) OCCURS 12 TIMES.
```

Assim:

MATR-MES (5 , 2 , 2) – matrículas no Curso 5 em fevereiro de 84.
MATR-MES (1 , 1 , 3) – matrículas no Curso 1 em março de 83.
MATR-MES (3 , 2 , 4) – matrículas no Curso 3 em abril de 84.
etc..

. PICTURE

Esta cláusula será estudada no item 4.4.3.

. REDEFINES

Esta cláusula é usada quando se deseja referenciar uma área de memória por mais de um nome com diferentes formas e tamanhos.

Algumas regras devem ser observadas para seu uso:

– A cláusula REDEFINES, quando especificada, deve seguir imediatamente o ⟨nome de dado 1⟩.

– Os números de nível de ⟨nome de dado 1⟩ e ⟨nome de dado 2⟩ devem ser idênticos e diferentes de 88.

– Quando aplicada em outro nível que não o 01, a área a ser redefinida deve ter o mesmo tamanho da área da nova descrição. FILLERs podem ser usados para evitar a violação desta regra.

– A redefinição acaba quando um número de nível menor ou igual ao de ⟨nome de dado 1⟩ ou ⟨nome de dado 2⟩ é encontrado.

– Múltiplas redefinições de uma mesma área são permitidas. Todas as áreas devem redefinir a área inicialmente definida.

– A entrada de descrição de ⟨nome de dado 2⟩ não pode conter as cláusulas OCCURS e REDEFINES e a cláusula VALUE, exceto em entradas de ⟨nome de condição 1⟩ (nível 88).

Exemplo:

```
01 CHAVE.
   03 PARTE1              PIC XX.
   03 PARTE2              PIC XXX.
01 CHAVE2 REDEFINES CHAVE.
   03 PARTE-INICIAL       PIC X.
   03 PARTE-FINAL         PIC X(4).
```

. USAGE

Esta cláusula é usada para especificar o modo de armazenamento interno de um item. Na ausência desta cláusula o item é armazenado na forma DISPLAY. Se especificarmos a cláusula num item-grupo ela é válida para todos os itens elementares do grupo.

Veremos agora a forma de armazenamento de cada opção:

– DISPLAY

Armazenamento em código ASCII: 8 bits por caractere.

– INDEX

Usado para definir índices de tabelas. As cláusulas VALUE e PICTURE não podem ser utilizadas em itens com USAGE INDEX.

– COMPUTATIONAL ou COMP

Armazenamento em representação binária. Deve ser usado em itens envolvidos em operações aritméticas. Só pode ser usado em itens numéricos que pertençam ao intervalo (-32768 a 32767).

– COMPUTATIONAL-3 ou COMP-3

Usado para armazenar dados numéricos na forma compactada. Nesta forma o número ocupa (N + 2)/2 bytes de memória, ou seja, em cada byte são armazenados 2 algarismos: O byte a mais é usado para armazenar o sinal do número.

. VALUE

Esta cláusula é usada para inicializar (dar valor inicial) itens da WORKING-STORAGE SECTION bem como para dar valor(es) a um nome condicional (definido em entrada de número de nível 88).

Algumas regras devem ser obedecidas para uso desta cláusula:

– O tipo de item deve concordar com o tipo do valor associado a ele, ou seja, itens numéricos inicializam-se com valores numéricos e itens alfabéticos ou alfanuméricos com valores não numéricos.

– O tamanho do item, bem como sua PICTURE, devem ser compatíveis com o valor de inicialização.

– A cláusula VALUE não pode ser usada para itens que contenham em sua descrição as cláusulas OCCURS ou REDEFINES e em itens cuja USAGE seja INDEX.

```
01 CONTADORES.
   03 PAGINAS     PIC 9(5)     VALUE 0.
   03 LINHAS      PIC 99       VALUE 60.
```

4.4.3 — PICTURE

Esta cláusula é usada para a descrição de características gerais e de edição de um item elementar, não podendo ser usada em itens cuja USAGE seja INDEX.

A ⟨cadeia de caracteres⟩ consiste em uma certa combinação de, no máximo, 30 caracteres permitidos em COBOL e determina implicitamente a categoria do dado.

Existem 5 categorias de dados:

. Alfabético

É aquele cuja ⟨cadeia de caracteres⟩ só contém o caractere A. Seu conteúdo deverá ser necessariamente uma das 26 letras do alfabeto e o espaço em branco.

. Numérico

É aquele cuja ⟨cadeia de caracteres⟩ só contém os símbolos 9, P, S e V. Seu conteúdo deverá ser necessariamente um dos 10 algarismos arábicos, podendo incluir um sinal. Um item numérico pode ter, no máximo, 18 algarismos.

. Alfanumérico

É aquele cuja ⟨cadeia de caracteres⟩ é restrita a certas combinações dos símbolos A, X e 9. O item é tratado como se toda a cadeia fosse de X. Seu conteúdo poderá ser qualquer caractere válido em COBOL.

. Alfanumérico de edição

É aquele cuja ⟨cadeia de caracteres⟩ é restrita a certas combinações dos símbolos A, X, 9, B, 0 e 1.

. Numérico de edição

É aquele cuja ⟨cadeia de caracteres⟩ é restrita a certas combinações dos símbolos B, P, V, Z, 0, 9, Vírgula, Ponto, +, –, CR, DB, * e $. O número máximo de posições representadas na ⟨cadeia⟩ deve ser 30.

A repetição de símbolos em uma ⟨cadeia de caracteres⟩ pode ser indicada através de uma constante numérica, inteira e positiva entre parênteses, sucedendo o caractere que terá sua ocorrência repetida.

Exemplo:

A(7) é o mesmo que AAAAAAA
9(5)V99 é o mesmo que 99999V99

Veremos agora a função dos símbolos usados na cláusula PICTURE:

A – Representa uma letra ou um espaço em branco.
B – Representa a inserção de um espaço em branco na posição correspondente.
P – Especifica a posição de um ponto decimal suposto, quando o ponto não pertencer ao mesmo valor do item dado. O caractere P não é contado no tamanho do dado. Os símbolos S e V são os únicos caracteres que podem aparecer à esquerda de P na ⟨cadeia⟩. Como a posição de P implica num ponto decimal suposto, o símbolo do ponto decimal V é redundante.
S – Usado para indicar a presença ⟨mas não a representação nem, necessariamente, a posição) de um sinal operacional. Deve ser o caractere mais à esquerda da ⟨cadeia⟩ e não é contado na determinação do tamanho do item elementar, a menos que o item seja COMP-3.
V – Usado para indicar a posição do ponto decimal suposto. Só pode aparecer uma vez na ⟨cadeia⟩. Como não representa a posição de caractere não é contado no tamanho do item.
X – Representa qualquer caractere válido em COBOL.
Z – Representa a posição de um caractere numérico que, se for zero, será substituído por espaço em branco. A substituição termina ao encontrar um ponto decimal ou um algarismo diferente de zero. Só pode ser usado em posições à esquerda da ⟨cadeia⟩ e é contado no tamanho do item.
9 – Representa um caractere numérico. O número máximo de 9's na PICTURE deve ser 18.
0 – Representa a inserção de um zero na posição correspondente.
, – Representa a inserção de uma vírgula na posição correspondente. Não pode ser o último caractere na ⟨cadeia⟩.
. – Representa a inserção do ponto na posição correspondente, servindo também para a determinação de alinhamento. Não pode ser o último caractere na ⟨cadeia⟩.

Nota:

Se a cláusula DECIMAL-POINT IS COMMA for estabelecida no parágrafo SPECIAL-NAMES do programa, as funções do ponto e da vírgula serão permutadas.

+ – – Usados para a edição do sinal do número ou como símbolos de inserção flutuante. Estes símbolos são mutuamente exclusivos.
CR DB – Usados para a edição de símbolos de controle, representando a inserção do símbolo na posição correspondente. Estes símbolos são mutuamente exclusivos. Só podem ser usados como caracteres mais à direita da PICTURE e são impressos apenas se o valor do item for negativo.
* – Representa a posição de um caractere numérico que, se for zero, será substituído pelo asterisco. A substituição funciona como no símbolo Z.

$ – Representa a inserção "flutuante" do $. O símbolo dólar na ⟨cadeia de caracteres⟩ é representado tanto pelo símbolo $ como por um caractere único especificado na cláusula CURRENCY SIGN do parágrafo SPECIAL-NAMES.
/ – Representa a inserção da barra na posição correspondente.

Agora que conhecemos a função de cada um dos símbolos que podem aparecer na cláusula PICTURE, vamos estudar as regras de edição. Só podemos realizar edição através de dados alfanuméricos de edição e numéricos de edição.

Itens alfanuméricos de edição estão sujeitos a apenas um tipo de edição: a simples inserção dos caracteres b e 0, por intermédio dos símbolos B e 0 em sua PICTURE.

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
A(4)BA(5)BA(4)	JOAOMELLONETO	JOAO MELLO NETO
X(3)000X(2)	MARIA	MAROOOIA
X(2)BX(2)BX(2)	141278	14 12 78
000A(5)	MARIA	000MARIA
X(3)B099	TL-69	TL- 069

Itens numéricos de edição estão sujeitos tanto à inserção quanto à substituição de caracteres. A inserção pode ser de quatro tipos: simples, especial, fixa e flutuante. A substituição pode ser de dois tipos: substituição de zeros por espaços ou por asteriscos.

Veremos agora cada um destes tipos:

. Inserção simples

É obtida usando-se os seguintes caracteres de inserção: vírgula, B e zero. Os caracteres de inserção são contados no tamanho do item e representam a posição dentro do item na qual o caractere será inserido.

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
999,90	4321	432,10
9,9	21	2,1
99B99B99	141278	14 12 78
99,90	1234	23,40

. Inserção especial

É obtida usando-se o ponto como caractere de inserção. O resultado é a edição de um ponto na mesma posição que este ocupava na PICTURE.

Além de ser um caractere de inserção, o ponto representa também o ponto decimal para efeito de cálculo de alinhamento, sendo contado no tamanho do item.

O uso do ponto e do suposto ponto decimal (representado por V) em uma mesma ⟨cadeia⟩ é errado e implica em redundância.

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
99.99	10.20	10.20
99.9	0.15	00.1
9.99	0.15	0.15
999.99	222.22	222.22
9.99	123.45	3.45
999	222.22	222
999V99	222.22	22222

. Inserção fixa:

É obtida usando-se $ + – CR e DB. Estes quatro últimos símbolos são mutuamente exclusivos em uma mesma ⟨cadeia⟩.

Este tipo de inserção implica numa edição do caractere na mesma posição que ele ocupa na ⟨cadeia de caracteres⟩ da PICTURE.

$ – Deve ser o caractere mais à esquerda da PICTURE, exceto quando precedido por + ou –. É contado no tamanho do item.

+ ou – – Quando usados deverão representar a posição mais à esquerda da PICTURE. São contados no tamanho do item.

CR ou DB – Devem representar as duas posições mais à direita da PICTURE. São contados no tamanho do item.

Os símbolos de controle de sinal (+, –, CR ou DB) produzem resultados dependendo do valor do item, como se segue:

SIMBOLO DE CONTROLE	VALOR	RESULTADO DA EDICAO
+	–	–
+	+	+
–	–	–
–	+	b
CR	+	bb
CR	–	CR
DB	+	bb
DB	–	DB

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
+99	+33	+33
+99	−33	−33
−99	+33	b33
−99	−33	−33
99CR	+33	33bb
99CR	−33	33CR
99DB	+33	33bb
99DB	−33	33DB

. Inserção flutuante

Em uma ⟨cadeia de caracteres⟩ da PICTURE, existem dois tipos de inserção flutuante:

1. Todas ou algumas posições numéricas à esquerda do ponto decimal são representadas por símbolos de PICTURE flutuante de inserção.

2. Toda a PICTURE é composta por caracteres de inserção flutuante.

Caso 1: Se os caracteres de inserção flutuante aparecerem somente à esquerda do ponto decimal, um caractere de inserção será editado na primeira posição imediatamente antes do primeiro dígito não nulo do item.

Caso 2: Se todas as posições numéricas na ⟨cadeia de caracteres⟩ da PICTURE forem de inserção flutuante, o resultado depende do valor do dado. Se o valor for zero, todo o valor editado será composto por espaços (b). Caso contrário, o resultado editado é análogo ao Caso 1.

Para evitar truncamentos que alterem a ordem de grandeza do dado quando do uso de PICTURE's flutuantes, o programador deverá compor o tamanho da PICTURE como sendo:

a) O número de caracteres do campo emissor, mais

b) O número de caracteres de inserção (que não os de inserção flutuante), do campo receptor, mais

c) um caractere para inserção flutuante.

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
$$$$.99	.123	$.12
$$$$.99	4.44	$4.44
$$$9.99	.123	$0.12
$, $$$, 999.99	−1234.56	$1,234.56
$$$.$$	00.00	bbbbbb
$$$.99	00.00	$.00
$$99.99	1.234	$01.23
$$$$.99	1234	$234.00
$$$$.$$	+31.20	$31.20
$$,$$$,$$$.99CR	−1234567	$1,234,567.00CR
$$,$$$,$$$.99DB	+1234567	$1,234,567.00
++++.99	+3.14	+3.14
++++.99	−3.14	−3.14
++++.99	+00.20	+.20
++++.++	−30.40	−30.40
----.99	+2.34	2.34
----.99	−2.34	−2.34
+++,+++,999.99	−123456.789	−123,456.78
---,---,999.99	+123456.789	b123,456.78
++,+++,+++.++	000.00	bbbbbbbbbbbb

. Substituição

A substituição de zeros por espaços em branco ou por asteriscos é realizada por intermédio do uso dos caracteres Z e * na ⟨cadeia⟩.

Quando é usado Z, a substituição dos zeros será feita por espaços (b); os zeros serão substituídos por * quando do uso de * na ⟨cadeia⟩.

Os símbolos + − * Z e $ são mutuamente exclusivos quando usados como caracteres de inserção flutuante em uma mesma ⟨cadeia⟩.

Cada símbolo de substituição é contado no tamanho do item.

Exemplos:

PICTURE	VALOR DO DADO	RESULTADO EDITADO
ZZZZ.ZZ	0000.00	bbbbbbb
****.**	0000.00	****.**
ZZZ.99	0000.00	bbb.00
****.99	0000.00	****.00
ZZ99.99	0000.00	bb00.00

PICTURE	VALOR DO DADO	RESULTADO EDITADO
ZZZZ.ZZ	0012.34	bb12.34
****.**	0310.34	*310.34
ZZZZZZ.ZZ	+123.456	bbb123.45
******.**	−123.45	***123.45
,*,***.**	+12345678.9	12,345,678.90
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$bbb12,345.67bb
$B*,***,***.**BBDB	−12345.67	$b***12,345.67bbDB

4.4.4 — O número de nível 88

Este número de nível é utilizado para definir um ⟨nome de condição⟩.

As entradas iniciadas pelo nível 88 são sempre associadas a outro item contendo número de nível, exceto a outro ⟨nome de condição⟩ ou a um índice.

Toda entrada de nível 88 deve possuir a cláusula VALUE, que é a única cláusula permitida nesta entrada.

Exemplo:

```
03 ESTADO-CIVIL      PIC 9.
   88 SOLTEIRO       VALUE 1.
   88 CASADO         VALUE 2.
   88 VIUVO          VALUE 3.
```

Ao nos referirmos a SOLTEIRO é como se nos referíssemos a ESTADO-CIVIL = 1.

```
03 NIVEL-ESCOLAR     PIC 99.
   88 PRIMEIRO-GRAU  VALUE 1 THRU 4.
   88 SEGUNDO-GRAU   VALUE 9 THRU 11.
   88 UNIVERSITARIO  VALUE 12, 13, 14, 15.
```

4.5 WORKING-STORAGE SECTION

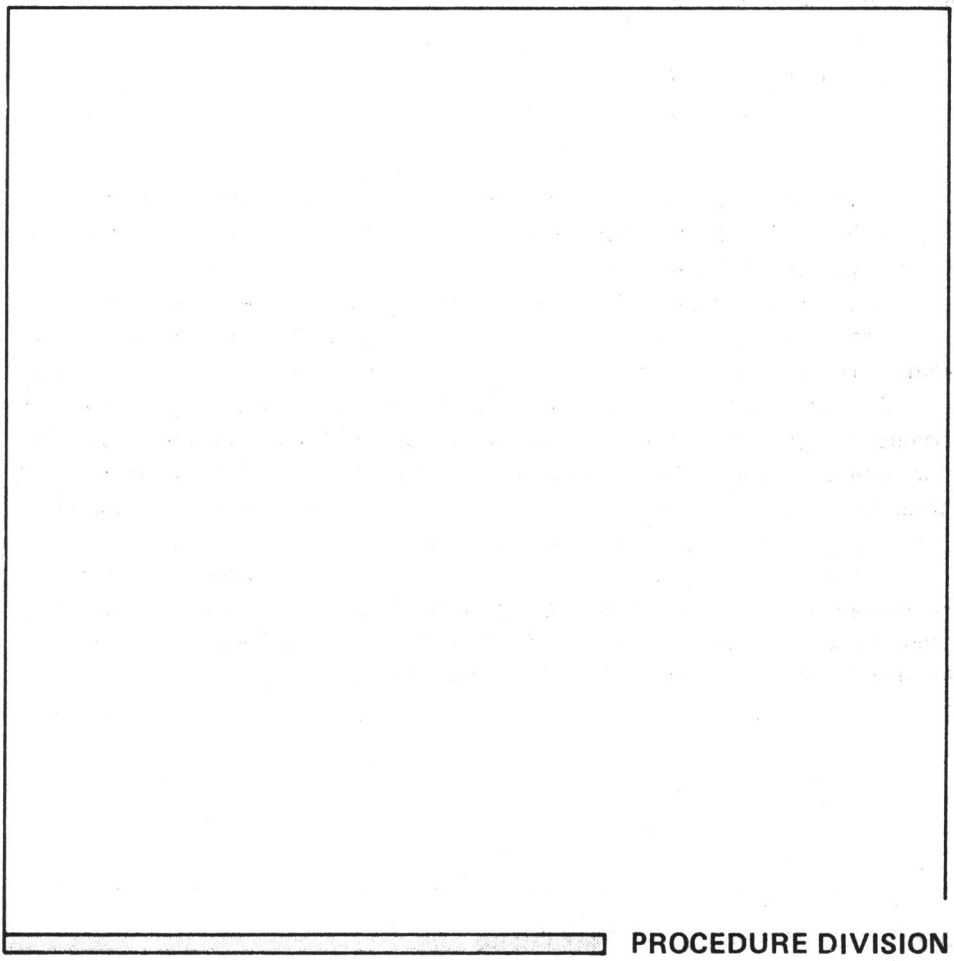
Esta seção contém descrições de registros que não fazem parte de arquivos e descrição de itens independentes.

Itens independentes são os dados que não possuem quaisquer relações com os outros não precisando, portanto, estarem agrupados em estruturas. A descrição de um item independente é feita por intermédio de uma entrada iniciada pelo número de nível 77.

Exemplo:

```
77 CONTADOR      PIC 99 VALUE ZERO.
```

5



PROCEDURE DIVISION

A PROCEDURE DIVISION é a quarta divisão de um programa COBOL. Sua função é descrever todos os procedimentos necessários para a resolução de um problema dado.

Os procedimentos necessários são escritos em "STATEMENTS", os quais podem ser combinados para formar as sentenças. Grupos de sentenças formam os parágrafos, os quais podem ser combinados para formar as seções.

5.1 ESTRUTURA DA DIVISÃO

PROCEDURE DIVISION.
{nome de secao

{nome de paragrafo
{sentenca} ...} ...} ...

Como vimos acima o "STATEMENT" é a unidade básica da PROCEDURE DIVISION. Sintaticamente um "STATEMENT" é formado por combinações válidas de palavras e símbolos, começando por um verbo.

Existem dois tipos de "STATEMENTS": condicionais e imperativos.

Uma sentença é composta por um ou mais "STATEMENTS", terminando sempre por um ponto seguido de um espaço.

As sentenças podem ser agrupadas para formar um parágrafo. Este começa sempre por um nome-de-parágrafo seguido por um ponto. Composto por uma ou mais sentenças, um parágrafo termina imediatamente antes do próximo nome-de-parágrafo ou pelo término do programa. Um nome-de-parágrafo aparece sempre na margem A (coluna 8).

Um ou mais parágrafos formam uma seção, que deve ser iniciada por um nome-de-seção seguido da palavra SECTION e por um ponto. Um nome-de-seção aparecendo sempre na margem A (coluna 8). Uma seção termina imediatamente antes do próximo nome-de-seção ou pelo término do programa.

5.2 EXPRESSÕES ARITMÉTICAS

5.2.1 — Operadores aritméticos

São símbolos usados com funções específicas. Quando usados devem estar precedidos e seguidos por espaços em branco.

Os operadores válidos em expressões aritméticas COBOL são:

+	Adição
–	Subtração
*	Multiplicação
/	Divisão
**	Potenciação

5.2.2 — Expressão aritmética

É um conjunto de constantes e/ou nomes dados pelo programador ligados por operadores aritméticos segundo regras prefixadas.

Exemplo:

```
BRUTO - 8 * BRUTO / 100
```

5.2.3 — Regras de formação de expressão aritmética

Dois operadores aritméticos não podem aparecer juntos. Para evitar o encontro usamos parênteses.

Exemplo:

```
CALC* - RESTO ⟶ não é correto,
```

mas sim

```
CALC * (- RESTO)
```

Operadores não podem ser deixados implícitos em uma expressão.

Exemplo:

A expressão COBOL correspondente a a(b + c) será:

```
A * (B + C)
```

O número de parênteses que se abrem em uma expressão aritmética deve ser igual ao número dos que se fecham. Parênteses a mais não alteram a expressão.

Exemplo:

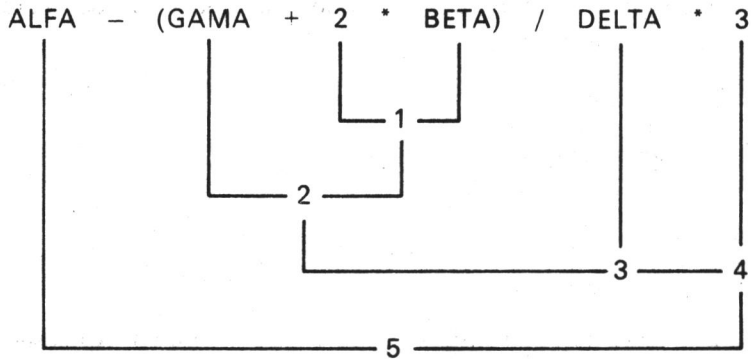
```
( ( (LIQ) + (BRUTO) ) / (FUNC ) )  é correspondente a:
( LIQ + BRUTO ) / FUNC
```

5.2.4 — Hierarquia das operações

Quando numa expressão aritmética, as operações são realizadas na seguinte ordem:

1) Operações dentro de parênteses (do mais para o menos interno)
2) Menos unário
3) Potenciação
4) Multiplicação e divisão
5) Adição e subtração
6) Em caso de igualdade de hierarquia a expressão é varrida da esquerda para a direita.

Exemplo:



5.3 COMANDOS ARITMÉTICOS

São formados por verbos aritméticos seguidos por seus operandos e têm por função calcular expressões aritméticas.

Os ⟨nome de dados⟩ utilizados nos comandos aritméticos devem estar declarados na DATA DIVISION como itens numéricos e não podem ser índices.

5.3.1 — COMPUTE

Usado para dar a um ou mais ⟨nome de dado⟩ o valor de um dado ou de uma expressão aritmética.

Forma geral:

COMPUTE ⟨nome de dado 1⟩ [**ROUNDED**]
[⟨nome de dado 2⟩ [**ROUNDED**]] ...
=
{ ⟨nome de dado 3⟩ } [**SIZE ERROR** ⟨statement-1⟩].
⟨literal numerico⟩
⟨expressao aritmetica⟩

Onde:

⟨nome de dado 1⟩

Item elementar numérico que irá receber o valor de ⟨nome de dado 3⟩, ⟨literal numérico⟩ ou de ⟨expressão aritmética⟩.

⟨nome de dado 2⟩

Idem ⟨nome de dado 1⟩.

⟨nome de dado 3⟩

Item elementar numérico cujo valor será atribuído a ⟨nome de dado 1⟩ e a ⟨nome de dado 2⟩.

Opção ROUNDED

Faz com que o valor a ser armazenado seja arredondado para o inteiro mais próximo. (O 5 é arredondado para cima.)

Opção SIZE ERROR

Caso o valor da ⟨expressão aritmética⟩ ou de ⟨nome de dado 3⟩ depois do alinhamento do ponto decimal não possa ser armazenado em ⟨nome de dado 1⟩ e/ou ⟨nome de dado 2⟩ o ⟨statement-1⟩ será executado. O ⟨statement-1⟩ deve ser um comando imperativo.

Exemplos:

```
COMPUTE LIQUIDO = BRUTO – 8 * BRUTO / 100.
COMPUTE LIQUIDO ROUNDED = BRUTO – 8 * BRUTO / 100.
COMPUTE LIQUIDO = BRUTO – INPS SIZE ERROR DISPLAY "DEU
GALHO".
```

5.3.2 — ADD

Usado para somar dois ou mais itens numéricos.
Possui duas formas:

1. **ADD** { ⟨nome de dado 1⟩ ⟨literal numerico⟩ }
   [{ ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }] ...
   TO ⟨nome de dado n⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

2. **ADD** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   { ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }
   [{ ⟨nome de dado 3⟩ ⟨literal numerico 3⟩ }] ...
   GIVING ⟨nome de dado n⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

Onde:

⟨nome de dado 1⟩, ⟨nome de dado 2⟩, ⟨nome de dado 3⟩ e ⟨nome de dado n⟩

São itens numéricos elementares ou independentes.

Opções SIZE ERROR e ROUNDED

Idênticas ao comando COMPUTE.

Caso 1:

Os valores de todos os operandos que precedem a palavra TO são somados. Esta soma é então somada ao valor de ⟨nome de dado n⟩ e nele é armazenado.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 3.
COMPUTE C = 4.
ADD A B TO C.
```

Após este trecho teremos os seguintes valores:

A = 2, B = 3, C = 9

Caso 2:

Os valores de todos os operandos que precedem a palavra GIVING são somados e o resultado é armazenado no item que segue à palavra GIVING.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 3.
COMPUTE C = 4.
ADD A B GIVING C.
```

Após este trecho teremos os seguintes valores:

A = 2, B = 3, C = 5

5.3.3 — SUBTRACT

Usado para subtrair um item numérico, ou a soma de vários itens numéricos, de um ou mais itens numéricos.

Possui 2 formas:

1. **SUBTRACT** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   [{ ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }] ...
   FROM ⟨nome de dado n⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

2. **SUBTRACT** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   [{ ⟨nome de dado 2⟩ ⟨literal numerico 2 ⟩ }] ...
   FROM { ⟨nome de dado 3⟩ ⟨literal 3⟩ }
   GIVING ⟨nome de dado n⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

Onde:

⟨nome de dado 1⟩, ⟨nome de dado 2⟩, ⟨nome de dado 3⟩ e ⟨nome de dado n⟩

São itens numéricos elementares ou independentes.

Opções ROUNDED e SIZE ERROR

Idênticas ao comando COMPUTE.

Na opção 1 os valores de todos os operandos que precedem a palavra FROM são somados. Esta soma é, então, subtraída do valor de ⟨nome de dado n⟩ e o resultado é aí armazenado.

Exemplo:

```
COMPUTE A = 3.
COMPUTE B = 4.
SUBTRACT 1 A FROM B.
```

Após este trecho teríamos os seguintes valores:

A = 3, B = 0

Na opção 2 os valores de todos os operandos que precedem a palavra FROM são somados. Esta soma é subtraída do valor de ⟨nome de dado 3⟩ e o resultado é armazenado em ⟨nome de dado n⟩.

Exemplo:

```
COMPUTE A = 3.
COMPUTE B = 4.
COMPUTE C = 10.
COMPUTE D = 11.
SUBTRACT A B FROM C GIVING D.
```

Após este trecho teríamos os seguintes valores:

A = 3, B = 4, C = 10, D = 3

5.3.4 — MULTIPLY

Usado para multiplicar dois itens numéricos.
Possui duas formas:

1. **MULTIPLY** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   BY ⟨nome de dado 2⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

2. **MULTIPLY** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩
BY { ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }
GIVING ⟨nome de dado 3⟩ [**ROUNDED**]
[**SIZE ERROR** ⟨statement-1⟩].

Onde:

⟨nome de dado 1⟩, ⟨nome de dado 2⟩ e ⟨nome de dado 3⟩

São itens numéricos.

Opções ROUNDED e SIZE ERROR

Idênticas ao comando COMPUTE.

Na opção 1 o valor de ⟨nome de dado 1⟩ é multiplicado por ⟨nome de dado 2⟩, onde é armazenado o produto.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 3.
MULTIPLY A BY B.
```

Após este trecho teríamos os seguintes valores:

A = 2, B = 6

Na opção 2 o valor de ⟨nome de dado 1⟩ é multiplicado por ⟨nome de dado 2⟩ e o produto é armazenado em ⟨nome de dado 3⟩.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 3.
COMPUTE C = 4.
MULTIPLY A BY B GIVING C.
```

Após este trecho teríamos os seguintes valores:

A = 2, B = 3, C = 6

5.3.5 — DIVIDE

Usado para dividir itens numéricos.
Possui 4 formas:

1. **DIVIDE** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   INTO ⟨nome de dado 2⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

2. **DIVIDE** { ⟨nome de dado 1⟩ ⟨literal numerico 1⟩ }
   INTO { ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }
   GIVING ⟨nome de dado 3⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

3. **DIVIDE** { ⟨nome de dado1⟩ ⟨literal numerico 1⟩ }
   BY { ⟨nome de dado 2⟩ ⟨literal numerico 2⟩ }
   GIVING ⟨nome de dado 3⟩ [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

4. **DIVIDE** ⟨nome de dado 1⟩ **BY** { ⟨nome de dado 2⟩
   ⟨literal numerico 2⟩ } [**ROUNDED**]
   [**SIZE ERROR** ⟨statement-1⟩].

Onde:

⟨nome de dado 1⟩ ... ⟨nome de dado 3⟩

São itens numéricos.

Opções ROUNDED e SIZE ERROR

Idênticas ao comando COMPUTE.

Na opção 1 o valor de ⟨nome de dado 2⟩ é dividido pelo valor de ⟨nome de dado 1⟩. O quociente é armazenado em ⟨nome de dado 2⟩.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 6.
DIVIDE A INTO B.
```

Após o trecho teremos:

A = 2, B = 3

Na opção 2 o valor de ⟨nome de dado 2⟩ é dividido pelo valor de ⟨nome de dado 1⟩. O quociente é armazenado em ⟨nome de dado 3⟩.

Exemplo:

```
COMPUTE A = 2.
COMPUTE B = 6.
COMPUTE C = 8.
DIVIDE A INTO B GIVING C.
```

Após o trecho teremos os seguintes valores:

A = 2, B = 6, C = 3

Na opção 3 o valor de ⟨nome de dado 1⟩ é dividido pelo valor de ⟨nome de dado 2⟩. O quociente é armazenado em ⟨nome de dado 3⟩.

Exemplo:

```
COMPUTE A = 4.
COMPUTE B = 2.
COMPUTE C = 6.
DIVIDE A BY B GIVING C.
```

Após o trecho teremos os seguintes valores:

A = 4, B = 2, C = 2

Na opção 4 o valor de ⟨nome de dado 1⟩ é dividido pelo valor de ⟨nome de dado 2⟩. O quociente é armazenado em ⟨nome de dado 1⟩.

Exemplo:

```
COMPUTE A = 8.
COMPUTE B = 2.
DIVIDE A BY B.
```

Após o trecho teremos os seguintes valores:

A = 4, B = 2.

5.4 Comandos de manipulação de dados

São aqueles que examinam ou transferem dados de um lugar para outro da memória.

5.4.1 — MOVE

Usado para transferir dados, de acordo com as regras de edição, para uma ou mais áreas de dados.

Possui a forma:

MOVE { ⟨nome de dado 1⟩ ⟨literal 1⟩ } **TO** ⟨nome de dado 2⟩
[⟨nome de dado 3⟩] ...

Onde:

⟨nome de dado 1⟩

Representa o campo de origem.

⟨nome de dado 2⟩, ⟨nome de dado 3⟩

Representam os campos de destino.

O conteúdo de ⟨nome de dado 1⟩ é movido para ⟨nome de dado 2⟩, ⟨nome de dado 3⟩, etc., fazendo com que estes conteúdos tornem-se iguais.

Exemplo:

```
COMPUTE A = 5.
COMPUTE B = 3.
COMPUTE C = 1.
MOVE A TO B C.
```

Após o trecho teríamos:

A = B = C = 5

Um campo cuja USAGE seja INDEX não pode ser utilizado no comando MOVE.

Existem certas transferências de dados que não são válidas em virtude da categoria dos campos de origem e destino. Observe na tabela a seguir as transferências possíveis (SIM) e as inválidas (NAO).

Campo origem \ Campo destino		Alfabético	Alfanumérico e Alfanumérico (edição)	Numérico e Numérico (edição)
Alfabético		SIM	SIM	NÃO
Alfanumérico		SIM	SIM	SIM
Alfanum. edição		SIM	SIM	NÃO
Numérico	Inteiro	NÃO	SIM	SIM
	Real	NÃO	NÃO	SIM
Numérico edição		NÃO	SIM	NÃO

Quando o campo de destino (receptor) é um item alfanumérico, alfanumérico de edição ou alfabético, a transferência é feita realizando-se o alinhamento pela esquerda do campo. Caso o item receptor seja maior, o restante do campo é preenchido com espaços em branco. Caso seja menor ocorrerá truncamento dos caracteres mais à esquerda do item de origem.

Quando o campo de destino (receptor) for um item numérico ou numérico de edição, o alinhamento é feito pelo ponto decimal. Será usado o valor absoluto do campo de origem se o receptor não tiver sinal. Caso o campo receptor seja menor, tanto à esquerda quanto à direita do ponto decimal, estes dígitos serão truncados.

Exemplos:

Campo origem		Campo destino	
PICTURE	VALOR	PICTURE	VALOR DEPOIS DO MOVE
99V99	1234	S99V99	1234+
99V99	1234	99V9	123
S9V9	12–	99V999	01200
XXX	A2B	X(5)	A2Bbb
9V99	123	99.99	01.23

5.4.2 — INSPECT

Usado para contar o número de vezes que um determinado caractere aparece em um dado e/ou trocar certos caracteres de um item de dado por outros.

Possui a forma:

INSPECT ⟨nome de dado 1⟩ [**TALLYING** ⟨nome de dado 2⟩ FOR
{ **CHARACTERS** / { **ALL** / **LEADING** } ⟨operando 1⟩ }
[{ **BEFORE** } { **AFTER** } **INITIAL** ⟨operando 2⟩]]
[**REPLACING** { **CHARACTERS** / { **ALL** / **LEADING** / **FIRST** } ⟨operando 3⟩ } **BY** ⟨operando 4⟩
[{ **BEFORE** } { **AFTER** } INITIAL ⟨operando 5⟩]].

Onde:

〈nome de dado 1〉

Deverá ter sido declarado com USAGE DISPLAY.

〈nome de dado 2〉

Deverá ser um item numérico.

〈operando〉

Deverá ser um literal não-numérico, uma constante figurativa ou um nome de dado. Em qualquer caso, só poderá ser composto de 1 caractere.

Embora sejam opcionais, as cláusulas TALLYING e REPLACING não podem ser omitidas simultaneamente. Se ambas estão presentes, a ordem deve ser respeitada.

A cláusula TALLYING causa a comparação caractere a caractere, da esquerda para a direita, de 〈nome de dado 1〉 com 〈operando 1〉 ou verifica os caracteres de 〈nome de dado 1〉, incrementando 〈nome de dado 2〉 a cada comparação verdadeira.

A cláusula REPLACING causa a substituição dos caracteres de 〈nome de dado 1〉 por 〈operando 4〉 cada vez que a comparação dos caracteres de 〈nome de dado 1〉 com 〈operando 3〉 ou a cada ocorrência de um caractere em 〈nome de dado 1〉.

Vejamos agora a função de cada uma das outras opções:

. BEFORE INITIAL – a contagem (ou substituição) termina ao encontrar em 〈nome de dado 1〉 um caractere igual a 〈operando 2〉 (ou 〈operando 5〉).

. AFTER INITIAL – a contagem (ou substituição) só se inicia após a detecção de 〈operando 2〉 (ou 〈operando 5〉) em 〈nome de dado 1〉.

. CHARACTERS – implica que todo caractere de 〈nome de dado 1〉 é contado ou substituído.

. ALL – implica na contagem (ou substituição por 〈operando 4〉) de todo caractere de 〈nome de dado 1〉 igual a 〈operando 1〉.

. LEADING – especifica que apenas caracteres de 〈nome de dado 1〉, são trocados por 〈operando 1〉.

FIRST – especifica que apenas o primeiro caractere de ⟨nome de dado 1⟩ igual a ⟨operando 3⟩ é substituído. (Esta cláusula só existe na cláusula REPLACING.)

Quando as cláusulas TALLYING e REPLACING são usadas, o INSPECT funciona como se 2 INSPECTS estivessem escritos: o primeiro com a cláusula TALLYING e o segundo com a cláusula REPLACING.

Exemplos:

1. Supondo que:

```
DADO      = "MONIKETE"
CONTADOR  = 0
```

Após o comando

```
INSPECT DADO TALLYING CONTADOR FOR ALL "E" REPLACING
ALL "E" BY "I" AFTER INITIAL "T".
```

teríamos:

```
DADO      = "MONIKETI"
CONTADOR  = 2
```

2. Supondo que:

```
FRASE = "APRENDER ASSIM E MOLEZA ."
ESPACO = " "
PONTO = "."
```

Após o comando:

```
INSPECT FRASE REPLACING ALL ESPACO BY PONTO.
```

Teríamos:

```
FRASE "APRENDER.ASSIM.E.MOLEZA.."
ESPACO = " "
PONTO = "."
```

5.5 COMANDOS DE DESVIO

São aqueles que permitem alterar a execução seqüencial do programa.

5.5.1 — GO TO

Usado para transferir a execução do programa de uma parte para outra. Possui duas formas:

1. **GO** TO ⟨nome de parágrafo 1⟩.

2. **GO** TO ⟨nome de parágrafo 1⟩ [⟨nome de parágrafo 2⟩] ...
   DEPENDING ON ⟨nome de dado⟩.

Na opção 1 o fluxo de execução é transferido para o primeiro "statement" do ⟨parágrafo 1⟩.

Na opção 2 o fluxo é desviado dependente do valor de ⟨nome de dado⟩. Caso este valor seja 1 o desvio é feito para ⟨nome de parágrafo 1⟩, caso seja 2 para ⟨nome de parágrafo 2⟩, etc. ⟨nome de dado⟩ deve ter valor inteiro e caso o valor não possua parágrafo correspondente, seja zero ou negativo, o controle passa para o próximo comando.

Exemplo: GO TO TESTE LEITURA FIM DEPENDING ON VALOR.

Se VALOR = 1 o controle passará para TESTE,
Se VALOR = 2 o controle passará para LEITURA,
Se VALOR = 3 o controle passará para FIM,
Se VALOR for diferente dos anteriores passará para o próximo comando.

5.5.2 — PERFORM

Usado para desviar o processamento para a execução de um determinado procedimento até que uma determinada condição seja satisfeita. Após a execução do PERFORM o controle passa ao comando imediatamente seguinte.

Possui 4 formas:

1. **PERFORM** ⟨nome do proc. 1⟩
   { **THRU** / **THROUGH** } ⟨nome do proc. 2⟩.

2. **PERFORM** ⟨nome do proc. 1⟩
   { **THRU** / **THROUGH** } ⟨nome do proc. 2⟩
   { ⟨nome de dado 1⟩ / ⟨inteiro⟩ } **TIMES.**

3. **PERFORM** ⟨nome de proc. 1⟩
   { **THRU** / **THROUGH** } ⟨nome de proc. 2⟩
   UNTIL ⟨condicao 1⟩.

4. **PERFORM** ⟨nome do proc. 1⟩
   { **THRU** / **THROUGH** } ⟨nome do proc. 2⟩
   VARYING { ⟨nome de dado 1⟩ / ⟨nome de indice 1⟩ }
   FROM { ⟨nome de dado 2⟩ / ⟨literal numerico 2⟩ / ⟨nome de indice 2⟩ }
   BY { ⟨nome de dado 3⟩ / ⟨literal numerico 3⟩ }
   UNTIL ⟨condicao⟩.

Onde:

⟨nome de procedimento 1⟩, ⟨nome de procedimento 2⟩

Nome de parágrafo ou seção.

Quando o comando PERFORM é executado, o controle é transferido para o primeiro "statement" do parágrafo ou seção ⟨nome de procedimento 1⟩. O retorno automático para o "statement" seguinte ao PERFORM é realizado como se segue:

1. Se ⟨nome de procedimento 1⟩ é um parágrafo e a opção THRU não é usada, o retorno ocorre após a execução do último "statement" do parágrafo.

2. Se ⟨nome de procedimento 1⟩ é uma seção e a opção THRU não é usada, o retorno ocorre após a execução do último parágrafo da seção.

3. Se a opção THRU é especificada, o retorno ocorre após o término da execução do último "statement" do parágrafo (ou do último "statement" do último parágrafo da seção) de nome ⟨nome de procedimento 2⟩.

Devemos observar que, no programa, ⟨nome de procedimento 2⟩ deve vir após ⟨nome de procedimento 1⟩, isto é, ⟨nome de procedimento 1⟩ deve preceder, em execução seqüencial, ⟨nome de procedimento 2⟩.

Na opção 1 temos o PERFORM básico. O trecho ⟨nome de procedimento 1⟩ é executado uma vez e o controle retorna ao comando seguinte ao PERFORM.

A opção 2 faz com que ⟨nome de procedimento 1⟩ seja executado um certo número de vezes. Este número é dado pelo valor de ⟨nome de dado 1⟩ ou de ⟨inteiro⟩. Caso este valor seja negativo ou nulo o controle passa ao comando seguinte.

A opção 3 faz com que ⟨nome de procedimento 1⟩ seja executado até que ⟨condição 1⟩ seja verdadeira. Caso este já o seja quando o PERFORM for executado, o comando é ignorado passando o controle para o comando seguinte.

A opção 4 faz com que ⟨nome de procedimento 1⟩ seja executado até que ⟨condicao⟩ seja verdadeira. O valor de ⟨nome de dado 1⟩ (ou ⟨nome de índice 1⟩) inicial é dado por ⟨nome de dado 2⟩ (ou ⟨literal numerico 2⟩ ou ⟨nome de indice 2⟩). Este valor é incrementado pelo valor de ⟨nome de dado 3⟩ (ou ⟨literal numerico 3⟩) cada vez que ⟨nome de procedimento 1⟩ é executado. Caso ⟨condicao⟩ seja verdadeira ao ser executado o comando, este é ignorado, passando o controle para o comando seguinte.

Exemplo:

```
1. PRINCIPAL SECTION.
   PR-INICIO.
     PERFORM CABECALHO.
             - - -
             - - -
             - - -
             - - -
             - - -
             - - -
     PERFORM PR-MOVE-DADOS VARYING I FROM 1 BY 1 UNTIL I >10.
             - - -
             - - -
             - - -
             - - -
     PERFORM PR-TESTA-IMPRESSAO 5 TIMES.
             - - -
             - - -
             - - -
     PR-MOVE-DADOS.
          MOVE VOTO(I) TO VOTO-IMP(I).
          MOVE PONTOS(I) TO PONTOS-IMP(I).
     PR-TESTA-IMPRESSAO.
          WRITE LINHA FROM LINHA-1 BEFORE ADVANCING 2 LINES.
     CABECALHO SECTION.
     CABEC-INICIO.
             - - -
             - - -
             - - -
     CABEC-FIM.
             - - -
             - - -
             - - -
             - - -
```

O primeiro comando PERFORM (PERFORM CABECALHO) ocasiona o desvio da execução para a seção CABECALHO, implicando na execução dos comandos que compõem os parágrafos CABEC-INICIO e CABEC-FIM. Após a execução do último comando de CABEC-FIM o controle volta ao comando seguinte ao PERFORM.

O segundo comando PERFORM (PERFORM PR-MOVE-DADOS ...) provoca a execução dos comandos MOVE VOTO(I) ... e MOVE PONTOS(I)

10 vezes (I = 1, 2, 3, 4, ..., 10). Ao término da execução do PERFORM I estará valendo 11 e o próximo comando será executado.

O terceiro comando PERFORM (PERFORM PR-TESTA-IMPRESSAO...) causa a execução do comando WRITE LINHA... (será estudado mais adiante) 5 vezes. Isto corresponde a colocar-se 5 vezes o comando WRITE no lugar do PERFORM.

5.5.3 — EXIT

Usado para finalizar a execução de um PERFORM. Deve ser o único comando do parágrafo.

Sua forma geral é EXIT.

O uso do EXIT fora do controle do PERFORM faz com que ocorra um desvio para o parágrafo seguinte.

5.5.4 — STOP

Usado para terminar ou atrasar a execução do programa.

Possui a seguinte forma geral:

STOP { **RUN** / ⟨literal⟩ }

STOP RUN termina a execução do programa.

STOP ⟨literal⟩ escreve o literal no vídeo e suspende a execução do programa até que o operador intervenha.

5.6 COMANDO CONDICIONAL

Usado para selecionar procedimentos a serem executados caso uma condição seja verdadeira ou falsa.

5.6.1 — IF

Usado para avaliar uma determinada condição e, baseado nesta avaliação, selecionar os procedimentos a serem executados.

Sua forma geral é:

IF ⟨condicao⟩ {⟨statement 1⟩ **NEXT SENTENCE** }
ELSE {⟨statement 2⟩ **NEXT SENTENCE** }.

Onde:

⟨statement 1⟩ e ⟨statement 2⟩ podem ser imperativas e/ou condicionais.

Quando o comando é executado, é realizado o seguinte procedimento:

1. Se a ⟨condicao⟩ é verdadeira, ⟨statement 1⟩ é executado e o controle passa ao próximo comando.
2. Se a ⟨condicao⟩ é falsa, ⟨statement 2⟩ é executado e o controle passa ao próximo comando. Se a cláusula ELSE não é usada o controle passa ao próximo comando.

Se NEXT SENTENCE é usado, o controle passa ao próximo comando.

É permitido "ninho" de IFs. Cada opção ELSE encontrada será relacionada com o primeiro IF imediatamente anterior que não tenha sido relacionado a outro ELSE.

Exemplos:

1. IF A = B ADD 1 TO C.

2.
```
IF A = C
    IF A = D
        ADD 1 TO D
    ELSE
        ADD 1 TO A
ELSE
    ADD 1 TO B.
```

3.
```
IF A = E
    IF A = F
        DISPLAY "A=E=F"
    ELSE
        DISPLAY "=E NAO =F".
```

Um IF pode realizar 5 tipos de teste:

5.6.1.1 — Teste de classe

Usado para testar se um determinado dado é numérico ou alfabético.

IF ⟨nome do dado⟩ **IS** [**NOT**] { **NUMERIC** / **ALPHABETIC** }

Exemplo:

```
IF NOME-DO-ALUNO IS NOT ALPHABETIC
    DISPLAY "NOME DO ALUNO ERRADO".
```

5.6.1.2 — Teste de nome-de-condição

Usado para testar se o valor de uma variável é igual aos valores associados aos nomes condicionais.

IF ⟨nome de condicao⟩

Exemplo:

```
05 RESULTADO PIC (9.)
   88 VENCEDOR   VALUE 1.
   88 EMPATE     VALUE 2.
   88 PERDEDOR   VALUE 3.
IF VENCEDOR será verdadeiro se o valor do RESULTADO for 1.
```

5.6.1.3 — Teste de sinal

Usado para testar se um determinado dado é maior, menor ou igual a zero.

IF ⟨nome de dado⟩ **IS** [**NOT**] { **POSITIVE** / **NEGATIVE** / **ZERO** }

5.6.1.4 — Teste de relação

Usado para estabelecer uma comparação entre duas variáveis, literais ou constantes figurativas.

IF {⟨nome do dado 1⟩ ⟨literal 1⟩ ⟨constante figurativa 1⟩}
⟨operador de relacao⟩
{⟨nome de dado 2⟩ ⟨literal 2⟩ ⟨constante figurativa 2⟩}

Onde

⟨operador de relação⟩ pode ser:

EQUAL TO ou =	igual
LESS THAN ou ⟨	menor
GREATER THAN ou ⟩	maior
NOT EQUAL TO ou NOT =	diferente
NOT LESS THAN ou NOT ⟨	maior ou igual
NOT GREATER THAN ou NOT ⟩	menor ou igual

5.6.1.5 — Teste de condição composta

Duas ou mais condições simples podem ser combinadas pelos operadores lógicos OR e AND para formar uma condição composta.

Os operadores lógicos são: NOT AND OR (e são avaliados nesta ordem de hierarquia).

Para uma melhor compreensão do funcionamento destes operadores observemos a tabela abaixo, onde A e B são condições que podem assumir os valores Verdadeiro e Falso.

A	B	A OR B	A AND B	NOT A	NOT A AND B	NOT (A AND B)
V	V	V	V	F	F	F
F	V	V	F	V	V	V
V	F	V	F	F	F	V
F	F	F	F	V	F	V

Exemplos:

```
1. IF A > B AND A > C
       DISPLAY "A E O MAIOR"
   ELSE
       IF B > A AND B > C
           DISPLAY "B E O MAIOR"
       ELSE
           DISPLAY "C E O MAIOR".

2. IF N(I) > N(J) OR MED(I) > MED(J)
       DISPLAY "ENCONTREI O ALUNO"
       DISPLAY "NUMERO DO ALUNO = " NUMERO.
```

5.7 COMANDOS DE ENTRADA E SAÍDA

São usados para estabelecer o fluxo de dados entre o programa e os arquivos.

5.7.1 — ACCEPT

Usado para permitir a entrada de dados via console do operador (vídeo). Sua forma geral é:

ACCEPT ⟨nome-de-dado⟩

Uma linha é lida e seu valor é movido para ⟨nome-de-dado⟩ que, se possuir um tamanho maior que o lido, é completado com brancos.

Existe um tipo especial de ACCEPT que é utilizado para obtenção da data do sistema (data do dia).

Sua forma geral é:

ACCEPT ⟨nome-de-dado⟩ **FROM** { **DATE** / **DAY** / **TIME** }

O uso da cláusula DATE fornece a data na forma AAMMDD (ano, mês e dia). Neste caso ⟨nome-de-dado⟩ deve ter sido declarado com PIC 9(6).

O uso da cláusula DAY fornece a "data Juliana" na forma AANNN (ano, número do dia). Nesse caso ⟨nome-de-dado⟩ deve ter sido declarado com PIC 9(5).

O uso da cláusula TIME fornece a hora da execução do comando, na forma HHMMSSCC (hora, minuto, segundo e centésimo de segundo). Neste caso ⟨nome-de-dado⟩ deve ter sido declarado como PIC 9(8).

Exemplo:

Suponha que as variáveis sejam declaradas corretamente e que o programa esteja sendo executado no dia 7 de fevereiro de 1983, às 22 horas, 26 minutos e 47 segundos. Após os comandos:

```
ACCEPT    DATA-AMERICANA FROM DATE.
ACCEPT    DATA-JULIANA FROM DAY.
ACCEPT    HORARIO FROM TIME.
```

Teríamos:

DATA-AMERICANA = 830207
DATA-JULIANA = 83038 (31 dia de janeiro + 7 de fevereiro).
HORARIO = 22264700

5.7.2 — DISPLAY

Usado para dar saída aos dados sem a necessidade de criação de arquivos. Sua forma geral é:

DISPLAY {⟨nome de dado 1⟩ ⟨literal⟩ }
[{ ⟨nome de dado 2⟩ ⟨literal 2⟩ }] ... [**UPON** ⟨mnemonico⟩]

Quando UPON não é utilizado, os dados são impressos na console do operador (vídeo).

O uso de UPON significa que a saída dos dados realizar-se-á via impressora. Assim, ⟨mnemonico⟩ deverá estar associado a PRINTER no parágrafo SPECIAL-NAMES da ENVIRONMENT DIVISION.

Cada DISPLAY permite a escrita de, no máximo, 132 caracteres.

5.7.3 — OPEN

Usado para inicializar o processo de entrada e saída em arquivos. Não se pode ler ou escrever em um arquivo antes de "abri-lo".

OPEN {INPUT} {OUTPUT} {I-O} {EXTEND} ⟨arq 1⟩ [⟨arq 2⟩...]

A cláusula INPUT é usada para especificar que o arquivo é de entrada, isto é, será lido pelo programa. A cláusula OUTPUT especifica que o arquivo é de saída, isto é, será gerado pelo programa. O uso da cláusula I-O especifica que o arquivo é de entrada e saída, não sendo permitido seu uso para arquivos a serem criados pelo programa. O uso da cláusula EXTEND faz com que o arquivo seja posicionado depois do último registro, possibilitando assim uma "extensão" do arquivo. A cláusula I-O é válida apenas para arquivos em disco. Seu uso permite que o arquivo seja modificado (através do comando REWRITE). Q comando WRITE não pode ser usado para arquivos seqüenciais abertos como I-O. A cláusula EXTEND só pode ser usada para arquivos seqüenciais.

Todos os arquivos a serem manipulados e/ou gerados pelo programa devem ser abertos pelo comando OPEN. Um mesmo arquivo poderá ser aberto mais de uma vez pelo programa, desde que antes da nova abertura ele tenha sido fechado (o que é feito pelo comando CLOSE). Arquivos fechados com CLOSE WITH LOCK não podem ser reabertos pelo programa.

5.7.4 — CLOSE

Usado para indicar que terminou o processamento de um arquivo. Só pode ser executado após a execução de um OPEN para o mesmo arquivo.

Forma geral:

CLOSE ⟨arq 1⟩ [**WITH LOCK**] ...

Se a cláusula LOCK é usada o arquivo não pode ser reaberto pelo programa.

5.7.5 — READ

Usado para acusar o próximo registro de um arquivo seqüencial ou algum registro específico de um arquivo randômico.

A cada execução de um READ o valor de ⟨nome de dado⟩ associado à cláusula FILE STATUS do SELECT do arquivo é atualizado.

Possui duas formas:

1. **READ** ⟨arq⟩ RECORD [**INTO** ⟨nome de dado⟩] [AT **END** ⟨statement⟩.

2. **READ** ⟨arq⟩ RECORD [**INTO** ⟨nome de dado 1⟩] [**KEY IS** ⟨nome de dado 2⟩ **INVALID KEY** ⟨statement 1⟩].

A opção 1 é usada para leitura de arquivos seqüenciais.

O uso da cláusula INTO corresponde a um READ simples acompanhado logo após por um MOVE do registro do arquivo para ⟨nome de dado 1⟩.

Após a leitura do último registro, a execução do comando READ faz com que seja executado o ⟨statement 1⟩ que segue a cláusula AT END. Caso a cláusula não tenha sido utilizada o procedimento descrito no DECLARATIVES será executado; se nenhum DECLARATIVES é utilizado, um erro de execução ocorre.

A opção 2 é usada para arquivos indexados ou relativos que estão sendo acessados randômica ou dinamicamente e não será estudada nesta publicação.

5.7.6 — WRITE

Usado para gravar registros em arquivos de saída.

Possui duas formas:

1. **WRITE** ⟨nome reg⟩ [**FROM** ⟨nome de dado 1⟩]
   {**BEFORE**} {**AFTER**} ADVANCING { ⟨inteiro⟩ LINES / ⟨nome de dado 2⟩ LINES / {**PAGE**} }

2. **WRITE** ⟨nome reg⟩ [**FROM** ⟨nome de dado 1⟩]
   [**INVALID** KEY ⟨statement 1⟩].

O ⟨nome reg⟩ deverá estar definido na entrada FD da DATA DIVISION no nível 01 e o arquivo deverá ter sido aberto com OPEN OUTPUT.

A cada execução de um WRITE o valor de ⟨nome de dado⟩ associado à cláusula FILE STATUS do SELECT do arquivo é atualizado.

O uso da cláusula FROM corresponde à execução de um "MOVE ⟨nome de dado 1⟩ TO ⟨nome reg⟩ " antes da execução do WRITE.

Na opção 1 a cláusula ADVANCING só pode ser usada para arquivos associados à impressora. A opção PAGE movimenta a impressora para o início de uma nova página. A opção LINES indica o avanço de tantas linhas quantas forem especificadas (por ⟨inteiro⟩ ou pelo valor de ⟨nome de dado 2⟩) antes ou depois da impressão, dependendo da cláusula utilizada (BEFORE – escreve e avança – ou AFTER – avança e escreve).

⟨inteiro⟩ ou ⟨nome de dado 2⟩ devem variar de 0 a 60.

Se a cláusula ADVANCING não for utilizada fica assumido AFTER ADVANCING 1 LINE.

A opção 2 é usada somente para arquivos indexados e relativos, e não será estudada nesta publicação.

5.7.7 — REWRITE

Utilizado para atualizar registros já pertencentes a um arquivo. Este deverá ter sido aberto pelo comando OPEN I-O.

Possui duas formas:

1. **REWRITE** ⟨nome reg⟩ [**FROM** ⟨nome de dado⟩].

2. **REWRITE** ⟨nome reg⟩ [**FROM** ⟨nome de dado⟩] [**INVALID** KEY ⟨statement 1⟩.

O uso da cláusula FROM corresponde à execução de um "MOVE ⟨nome de dado⟩ TO ⟨nome reg⟩ " antes do REWRITE.

A opção 1 é utilizada em arquivos seqüenciais. O registro atualizado é aquele acessado pelo último READ realizado no arquivo.

A opção 2 é utilizada em arquivos indexados e relativos e não será estudada nesta publicação.

A execução de um REWRITE atualiza o valor de ⟨nome de dado⟩ associado à cláusula FILE STATUS do SELECT do arquivo.

5.9 COMANDOS PARA MANIPULAÇÃO DE TABELAS

5.9.1 — SEARCH

Usado para localizar em uma tabela um elemento que satisfaça a um conjunto de determinadas condições.

Possui 2 formas:

1. **SEARCH** ⟨nome de dado 1⟩ **VARYING** {⟨nome de dado 1⟩ ⟨indice⟩}
   [**AT END** ⟨statement 1⟩] [**WHEN** ⟨condicao 1⟩
   {⟨statement 1⟩ **NEXT SENTENCE** }]...

2. **SEARCH ALL** ⟨nome de dado 3⟩ [AT **END** ⟨statement 1⟩]
   WHEN ⟨condicao 1⟩ { ⟨statement 2⟩ **NEXT SENTENCE**}.

Onde:

⟨nome de dado 1⟩ não pode ser subscrito ou indexado, mas sua descrição na DATA DIVISION deve conter as cláusulas OCCURS e INDEXED BY. ⟨nome de dado 2⟩, quando usado, deve ser descrito como USAGE INDEX ou como um item elementar numérico e inteiro. ⟨nome de dado 2⟩, assim como o número da ocorrência (posição da tabela) é incrementado a cada procura. Se em lugar de ⟨nome de dado 2⟩ usamos índice, então:

– Se ele aparece na cláusula INDEXED BY de ⟨nome 1⟩ então é usado para a procura.

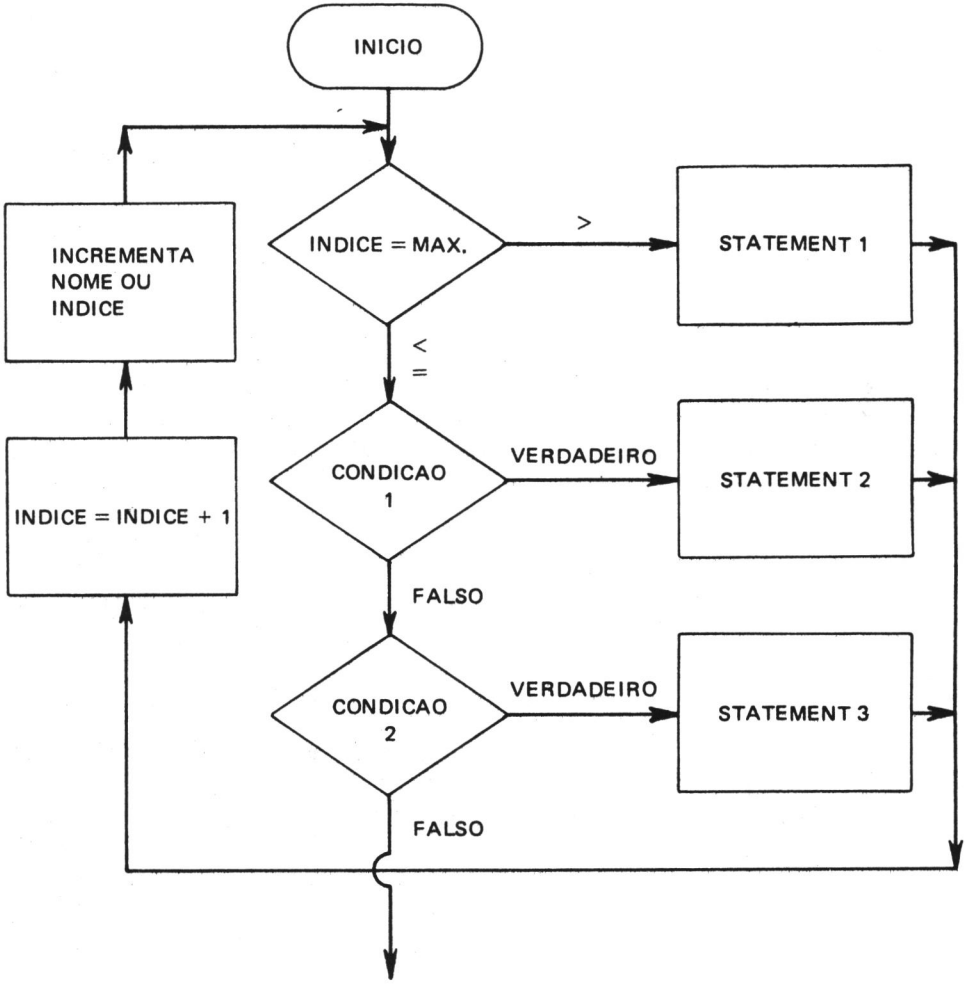
– Se ele aparece na cláusula INDEXED BY de outra tabela ele é incrementado.

⟨nome de dado 3⟩ – descrito como ⟨nome de dado 1⟩, porém sua cláusula OCCURS deve ter a opção {ASCENDING} {DESCENDING} KEY, isto é, a tabela descrita por ⟨nome de dado 3⟩ deve ter seus elementos ordenados.

A primeira opção é usada para a procura linear dos elementos.

Se, ao iniciar-se a procura na tabela, o valor do índice de ⟨nome de dado 1⟩ é maior que o permitido (aponta para fora da tabela), a procura é terminada por AT END, que se não for usado causa a execução da próxima sentença.

Porém, se o valor do índice é válido, a procura começa a partir deste ponto. Assim sendo, é uma boa prática preceder-se todo comando SEARCH com um comando SET.



A segunda opção realiza busca binária na tabela (veja Apêndice C). Esta busca, na maioria dos casos, é muito mais eficiente.

A tabela, quando do uso de SEARCH ALL, deverá estar ordenada e seu índice não precisa ser colocado na primeira posição da tabela antes do início da busca (ou seja, não é necessário preceder-se o SEARCH ALL de um SET).

5.8.2 — SET

Usado para inicializar, incrementar e/ou decrementar índices de tabelas. Possui duas formas:

1. **SET** { ⟨nome de dado 1⟩ ⟨indice 1⟩ }
   [⟨nome de dado 2⟩ ⟨indice 2⟩] ... **TO**
   { ⟨nome de dado 3⟩ ⟨indice 3⟩ ⟨literal numerico 1⟩ }

2. **SET** { ⟨indice 4⟩ [⟨indice 5⟩] } ... { **UP BY** / **DOWN BY** }
   {⟨nome de dado 4⟩}
   {⟨literal numerico 2⟩ }

Onde:

⟨nome de dado⟩ deve ser INDEX ou item elementar numérico e inteiro, com exceção apenas de ⟨nome de dado 4⟩ que não pode ser INDEX. Os índices são considerados relacionados a uma dada tabela e são definidos especificamente pela cláusula INDEXED BY do OCCURS.

A primeira opção é usada para dar valor ao índice e a segunda opção é usada para incrementar (UP BY) ou decrementar (DOWN BY) o índice.

Exemplo:

```
SET INDICE TO 5.              (INDICE = 5)
SET INDICE UP BY 1.           (INDICE = 6)
SET INDICE DOWN BY 3.         (INDICE = 3)
```

Vejamos agora alguns exemplos:

1. Suponha que o campo ESTADO-CIVIL-REG de um registro lido possa ter um dos códigos:

   A – solteiro C – desquitado E – casado G – outros
   B – separado D – viúvo F – divorciado

Quero transformar este código em seu estado civil correspondente para imprimi-lo através do campo ECIVIL.

Para isso poderíamos ter:

– Na DATA DIVISION (WORKING-STORAGE).

```
01 CODIGO-EST-CIVIL          PIC X(007) VALUE "ABCDEFG".
01 TAB-COD-EST-CIVIL REDEFINES CODIGO-EST-CIVIL.
   03 COD-EST-CIVIL          PIC X(001) OCCURS 7 TIMES
                             INDEXED BY CECV.
01 TABECV.
   03 FILLER                 PIC X(010) VALUE "SOLTEIRO".
   03 FILLER                 PIC X(010) VALUE "SEPARADO".
   03 FILLER                 PIC X(010) VALUE "DESQUITADO".
   03 FILLER                 PIC X(010) VALUE "VIUVO".
   03 FILLER                 PIC X(010) VALUE "CASADO".
   03 FILLER                 PIC X(010) VALUE "DIVORCIADO".
   03 FILLER                 PIC X(010) VALUE "OUTROS".
01 TABELA-ESTADO-CIVIL REDEFINES TABECV.
   03 TAB-EST-CIVIL          PIC X(010) OCCURS 7 TIMES
                              INDEXED BY TECV.
```

– Na PROCEDURE DIVISION.

```
SET CECV TO 1.
SEARCH COD-EST-CIVIL AT END MOVE "INVALIDO" TO ECIVIL
        WHEN COD-EST-CIVIL (CECV) = ESTADO-CIVIL-REG
             SET TECV TO CECV
             MOVE TAB-EST-CIVIL (TECV) TO ECIVIL.
```

2. Suponha que o campo NATURAL-REG de um registro lido contenha a sigla de um estado. Queremos colocar no campo NATUR-TI o nome do estado correspondente a esta sigla.

Poderíamos fazer:

– Na DATA DIVISION (WORKING-STORAGE).

```
01  TAB.
    03 FILLER          PIC X(002) VALUE "AC".
    03 FILLER          PIC X(019) VALUE "ACRE".
    03 FILLER          PIC X(002) VALUE "AL".
    03 FILLER          PIC X(019) VALUE "ALAGOAS".
01  TABELA-NATURALIDADE REDEFINES TAB.
    03 TABNAT          OCCURS 28 TIMES
                       INDEXED BY NAT.
       05 SIGLA        PIC X(002).
       05 NOME         PIC X(019).
```

– Na PROCEDURE DIVISION.

```
SET NAT TO 1.
SEARCH TABNAT AT END SET NAT TO 28
        WHEN SIGLA (NAT) = NATURAL-REG NEXT SENTENCE.
MOVE NOME (NAT) TO NATUR-TI.
```

Obs.:

A posição 28 de TAB está preenchida com SPACES.

APÊNDICE A

PALAVRAS RESERVADAS NO COBOL-80

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
AND
AREA
AREAS
ASCENDING
ASCII
ASSIGN
AT
AUTHOR

BEEP
BEFORE
BLANK
BLOCK
BY

CALL
CHARACTER
CHARACTERS
CLOSE
CODE-SET

COLLATING
COMMA
COMP
COMPUTATIONAL
COMP-3
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
COPY
COUNT
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DISK
DISPLAY
DIVIDE
DIVISION
DOWN
DYNAMIC

ELSE
END
ENVIRONMENT
EQUAL
ERROR
EXCEPTION
EXHIBIT
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILE-ID
FILLER
FIRST
FOR
FROM

GIVING
GO
GREATER

HIGH-VALUE
HIGH-VALUES

IDENTIFICATION
IF
IN
INDEX
INDEXED
INITIAL
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
I-O
I-O-CONTROL

JUST
JUSTFIED

KEY

LABEL
LEADING
LEFT
LESS
LINE
LINES
LINKAGE
LOCK

LOW-VALUE
LOW-VALUES

MEMORY
MODE
MODULES
MOVE
MULTIPLY

NAMED
NATIVE
NEGATIVE
NEXT
NOT
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PERFORM
PIC
PICTURE
POINTER
POSITION
POSITIVE
PRINTER
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROMPT

QUOTE
QUOTES

RANDOM
READ
READY
RECORD
RECORDS
REDEFINES
RELATIVE
REPLACING
RESERVE
RESET
RIGHT
ROUNDED
RUN

SAME
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUBTRACT
SYNC
SYNCHRONIZED

TALLYING
THEN
THROUGH
THRU
TIME
TIMES
TO
TRACE
TRAILING
UNSTRING
UNTIL
UP
UPON
USAGE
USE
USING

APÊNDICE B

ORGANIZAÇÃO DE ARQUIVOS

B.1 CONCEITOS BÁSICOS

Suponhamos que uma bibliotecária possui, cadastrados em fichas, todos os usuários de sua biblioteca. Estas fichas contêm informações do tipo: nome do usuário, número de cadastramento na biblioteca, endereço, etc. ...

Podemos então dizer que esta bibliotecária possui um ARQUIVO de usuários. Cada uma dessas fichas seria um REGISTRO deste ARQUIVO e cada uma das informações constantes de cada um destes REGISTROS (fichas) seria um CAMPO ou ATRIBUTO. Note que cada um dos campos possui características bem definidas quanto ao tipo e tamanho.

Formalizando então os conceitos apresentados, temos que:

– Um ARQUIVO é um conjunto de registros lógicos.

– Um REGISTRO, também chamado de registro lógico, é formado por uma seqüência de itens, denominados CAMPOS ou ATRIBUTOS.

– Cada CAMPO possui um nome (para poder ser referenciado), um tipo (de acordo com o seu conteúdo) e um comprimento (tamanho). Este tamanho (para um mesmo campo) poderá ser fixo ou variável.

Exemplos:

Se nesta ficha a bibliotecária anotasse os títulos dos livros que estão em poder do usuário certamente o campo LIVROS EM EMPRÉSTIMO teria tamanho variável, pois cada usuário teria em seu poder um determinado número de livros (considere que todos os títulos possuem o mesmo comprimento).

De modo similar, os registros podem ser de tamanho fixo (todos os registros do arquivo possuem o mesmo tamanho) ou de tamanho variável (os registros do arquivo podem assumir comprimentos compreendidos entre um limite mínimo e um limite máximo).

Geralmente, o armazenamento dos registros no arquivo é feito por BLOCOS DE REGISTROS (registros físicos). Assim sendo, a cada leitura ou gravação todo um registro físico (conjunto de registros lógicos) é lido ou gravado. Cada bloco armazena um número inteiro de registros.

Suponhamos agora que a bibliotecária deseja encontrar uma determinada ficha em seu lote. Ela deverá procurar esta ficha de acordo com o valor de um determinado campo (ou por mais de um, caso apenas um não sirva para determinar exatamente o usuário). Ela procurará a ficha de acordo com o valor da CHAVE de seu arquivo.

Uma CHAVE é, então, uma seqüência de um ou mais campos em um arquivo. Quando a chave apresenta um valor diferente para cada registro do arquivo, de tal forma que dado um valor à CHAVE apenas um único registro é identificado, a chave é dita primária. Geralmente a chave primária é formada por um único campo. Uma chave secundária é aquela que identifica um conjunto de registros.

Quando a chave é utilizada para uma operação de acesso ao arquivo (como no caso da bibliotecária) ela é chamada de chave de acesso. O valor da chave de acesso em uma operação é denominado argumento de pesquisa.

Note agora que o trabalho de busca a uma determinada ficha pela bibliotecária é muito mais rápido quando o arquivo possui uma determinada ordenação. Chave de ordenação é a chave primária usada para estabelecer a seqüência dos registros em um arquivo.

B.2 ORGANIZAÇÕES BÁSICAS DE UM ARQUIVO

Achamos ser bastante intuitiva a idéia de que, à medida que cresce o volume de dados de um arquivo e/ou a freqüência de consultas a este arquivo, crescem também os problemas de eficiência em sua utilização. Ou seja, a eficiência na utilização de um arquivo está diretamente ligada à adequação de sua organização.

Veremos agora alguns tipos de organização de arquivos, pois apenas o conhecimento das várias organizações existentes permite-nos fazer uma escolha adequada para a solução de um problema específico.

B.2.1 — Arquivo seqüencial

Um arquivo seqüencial é aquele em que os registros são dispostos seqüencialmente no arquivo, ou seja, o primeiro registro a ser gravado será, obrigatoriamente, o primeiro a ser lido em um outro processamento. Além disso, para gravar-se (ou ler-se) o n-ésimo registro do arquivo teremos que gravar (ou ler) os $(n - 1)$ registros que o antecedem.

É usual que a chave de acesso ao registro coincida com a chave de ordenação, o que, se dificulta a inserção de registros no arquivo (por quê?), otimiza em muito a procura de registros (por quê?).

B.2.2.1 — Operações

. Acesso a um registro

Podemos acessar um registro de forma seqüencial ou aleatória. O processo seqüencial consiste em acessar o registro que segue ao último acessado. Para esta forma de acesso, arquivos seqüenciais são extremamente eficientes.

No acesso aleatório devemos considerar se a chave de acesso coincide ou não com a chave de ordenação. Se elas não coincidem devemos fazer a pesquisa de forma seqüencial até que se ache o registro procurado ou então o fim do arquivo, o que significa que o registro procurado não se encontra no arquivo. Caso elas coincidam, a pesquisa seqüencial pode ser interrompida quando se encontra o registro procurado ou quando a chave de um registro é maior que o argumento de pesquisa, o que leva a concluir que o registro não se encontra no arquivo.

Se o arquivo, além de ordenado pela chave de acesso, está armazenado em um dispositivo de acesso direto (disco, tambor, etc. ...) podemos utilizar um método de pesquisa mais eficiente denominado pesquisa binária, no qual o primeiro registro a ser consultado é aquele que ocupa a posição média do arquivo. Se a chave do registro for igual ao argumento de pesquisa, a pesquisa termina com sucesso; caso contrário:

a. Se a chave do registro for maior que o argumento de pesquisa, o processo de busca é repetido para a metade superior (de cima) do arquivo.
b. Se a chave do registro for menor, o processo de busca é repetido para a metade inferior (de baixo) do arquivo.

A busca é encerrada sem sucesso quando a área de pesquisa, que a cada comparação é reduzida à metade, assumir o comprimento zero.

. Inserção de um registro

A única forma de inserir-se um registro em um arquivo seqüencial é criar-se um novo arquivo. Assim sendo, é necessário que o número de inserções justifique esta operação.

. Exclusão de um registro

A exclusão pode ser feita de 2 formas: exclusão física e exclusão lógica. Para efetuarmos a exclusão física (retirar, realmente, o registro do arquivo) temos que proceder como no caso da inserção. Para efetuarmos a exclusão lógica é necessário que o registro possua um campo indicando o seu estado e, na exclusão, basta alterarmos o valor deste campo para o código correspondente a "EXCLUIDO". Durante o processamento do arquivo basta desconsiderarmos os registros marcados como "excluídos".

. Alteração de um registro

Consiste na modificação do valor de um ou mais campos. O registro deve ser localizado, lido e seus campos alterados e então ser gravado novamente. Esta operação pode ser feita normalmente, desde que não ocorra uma das seguintes situações:

a. A alteração faça com que o registro mude de tamanho;
b. A alteração modifique o valor da chave de ordenação.

Devido a esses 2 problemas, geralmente esta operação é feita como no caso da inserção de um registro.

B.2.2 — Arquivo seqüencial indexado

Quando o número de acessos aleatórios a um arquivo seqüencial torna-se muito grande, surge a necessidade da utilização de uma estrutura de acesso mais eficiente. Podemos, então, utilizar um arquivo seqüencial indexado, que nada mais é do que um arquivo seqüencial acrescido de um índice.

Um índice é formado por uma coleção de pares, cada um deles associando um valor da chave de acesso a um endereço do arquivo. Sua finalidade é permitir a rápida determinação do endereço de um registro do arquivo, dado um argumento de pesquisa. O endereço identifica a posição onde está armazenado o registro na memória secundária.

Com o objetivo de tornar mais eficiente o processo de pesquisa, um índice pode ser estruturado em vários níveis, sendo o número de níveis proporcional ao número de entradas do índice.

A figura a seguir mostra um arquivo seqüencial indexado, no qual o índice é estruturado em dois níveis. O índice é associado à chave de acesso NUMERO, que coincide com a chave de ordenação dos dados. Cada entrada deste índice identifica um bloco de registros.

No.	END.
148	1
202	4
*	7

INDICES

No.	END.
105	1
135	4
148	7
180	10
195	13
202	16
208	19
270	22
*	25

Obs.:

* → maior valor que a chave pode assumir.

	No.	NOME	SEXO	NOTA
1	100	ALEX	M	95
2	103	BRUNO	M	83
3	105	CLAUDIA	F	44
4	110	DANIEL	M	22
5	130	ELIANA	F	100
6	135	FRED	M	44
7	140	GIL	M	80
8	144	HELIO	M	22
9	148	JOAO	M	45
10	160	KATIA	F	100
11	170	LUCIANNA	F	95
12	180	MARLI	F	40
13	130	NORMA	F	95
14	192	OTAVIO	M	80
15	195	PAULO	M	70
16	197	RAUL	M	85
17	200	RUI	M	10
18	202	SAUL	M	45
19	204	VANIA	F	33
20	206	VERA	F	22
21	208	VILMA	F	40
22	250	YARA	F	5
23	260	ZELIA	F	20
24	270	ZILDA	F	75
25	290	ZOZIMO	M	62
26	310	ZULMIRA	F	65
27	320	ZULMIRO	M	70

EXTENSAO

1	
2	
3	

O índice associado à chave de ordenação é chamado INDICE PRIMARIO; os outros são os INDICES SECUNDARIOS.

A área de extensão (também chamada área de overflow) destina-se a conter os registros inseridos após a criação do arquivo. Essas áreas de extensão são necessárias porque em arquivos seqüenciais indexados não é viável a inserção de registros no meio do arquivo, pois isto acarreta uma completa mudança de endereço dos registros.

Existem duas alternativas principais para a implementação das áreas de extensão: a primeira consiste em destinar a cada registro um campo de elo para conter o endereço da lista encadeada de seus sucessores na área de extensão; a segunda é a utilização de um campo de elo a cada bloco de registros.

B.2.2.1 — Operações

. Acesso a um registro

O acesso seqüencial pode ser feito diretamente sobre a área de dados, sem a utilização do índice, havendo apenas cuidados adicionais pela existência de áreas de extensão.

O acesso aleatório é feito através do índice. O argumento de pesquisa define o caminho sobre o índice que determina o endereço do registro desejado ou do bloco que o contém; nesse caso é necessária uma busca no bloco para a localização do registro, o que pode requerer acesso às áreas de extensão.

. Inserção de um registro

O processo de inserção requer a utilização do índice para determinação do local onde ele deve ser inserido. Uma vez determinada a posição, o registro é inserido na área de extensão e os elos são atualizados (caso de um elo por registro). A inserção, no caso de elos por blocos, requer uma reorganização do bloco, podendo acarretar deslocamentos de registros para a área de extensão e a inserção do novo registro na "área principal".

. Exclusão de um registro

É implementada pela colocação do estado "EXCLUIDO" em um campo adicional do registro que indica o seu estado.

. Alteração de um registro

Localizado o registro (através do índice), se a alteração não envolver a chave de ordenação nem seu comprimento, ele é alterado. Caso contrário, o registro deverá ser excluído e só então incluído.

. Reorganização do arquivo

À medida que o arquivo sofre inserções e exclusões, a performance de acessos vai sendo degradada pela necessidade de acessos cada vez mais freqüentes às áreas de extensão e pela sobrecarga representada pela necessidade de desconsideração dos registros excluídos. É necessário, então, que periodicamente o arquivo seja reorganizado. A reorganização consiste na transferência dos registros para uma nova área, o que acarreta a colocação de todos os registros na área principal e a liberação da área de extensão, que fica totalmente desocupada. Neste processo os registros marcados como "EXCLUIDOS" são retirados fisicamente do arquivo.

B.2.3 — Arquivo direto

Sua idéia básica consiste na instalação dos registros em endereços determinados a partir do valor de uma chave primária, de modo que se tenha rápido acesso aos registros desejados, sem que haja necessidade de utilização de índices.

Em um arquivo direto, o cálculo do endereço é feito por uma função que utiliza para o cálculo o argumento de pesquisa.

O principal problema associado a arquivos diretos é a determinação da função F, que transforma o valor C da chave de um registro no endereço E que lhe corresponde no arquivo. A escolha de uma função inadequada pode ocasionar a geração de mesmo endereço para várias chaves diferentes (colisão). Existem várias formas para tratamento de colisão, mas isto não será visto nesta publicação.

B.2.3.1 — Operações

. Acesso a um registro

O acesso seqüencial só pode ser feito se a função de cálculo de endereço preservar a ordem dos registros pelo valor da chave de acesso sendo, neste caso, feito diretamente sobre a área de dados.

O acesso aleatório é feito pela aplicação da função de cálculo de endereço ao argumento de pesquisa.

. Inserção de um registro

A função de cálculo de endereço é aplicada à chave do registro a ser inserido, resultando o endereço E. Caso este endereço esteja livre, o registro é aí inserido; caso contrário ele é inserido à primeira área livre encontrada pelo tratamento de colisões.

. Exclusão de um registro

O registro é acessado e seu campo de estado é alterado para "EXCLUIDO".

. Alteração de um registro

Se a alteração não modificar o valor da chave nem o comprimento do registro, este será simplesmente lido, alterado e gravado no mesmo endereço. Caso contrário, o registro será excluído, alterado e novamente inserido (conforme o tópico "Inserção de um Registro" explica).

. Reorganização do arquivo

De um modo geral, os arquivos diretos, assim como os arquivos seqüenciais indexados, necessitam de reorganizações periódicas visando manter a eficiência de acesso.

Além dos tipos de organização aqui descritos, ainda podemos citar:

. Arquivos indexados
. Arquivos relativos
. Arquivos invertidos.

O presente texto foi elaborado baseando-se totalmente no livro:

Organização de Banco de Dados
de
A. L. Furtado e C. S. dos Santos

APÊNDICE C

BUSCA BINÁRIA

A busca binária consiste em dividir-se a tabela ao meio, verificar-se, pelo valor da chave, em que parte da tabela encontra-se o elemento procurado, considerando-se esta parte como a nova tabela, e repetir o processo.

Exemplo:

Seja a tabela de códigos de endereçamento postal:

CEP	BAIRRO
20081	SAUDE
20230	SANTA TERESA
20240	CENTRO
20560	GRAJAU

CEP	BAIRRO
21350	MADUREIRA
21520	PAVUNA
21931	ILHA DO GOVERNADOR
22210	FLAMENGO
22290	BOTAFOGO
22700	JACAREPAGUA
23000	CAMPO GRANDE

(repare que a tabela está ordenada pelo número do CEP)

Suponha que estamos procurando o bairro correspondente ao CEP 20560. Assim:

1. Dividimos a tabela ao meio.
   (11 elementos : 2 = 5,5 = 6)
2. Verificamos se o CEP procurado encontra-se na parte superior ou inferior dá tabela.

```
20560 : tab (6)
          ↓
        21520
```

Como é menor, o bairro procurado encontra-se na parte superior. Assim sendo, despreza-se a parte inferior e considera-se a parte superior como a nova tabela e reinicia-se o processo:

CEP	BAIRRO
20081	SAUDE
20230	SANTA TERESA
20240	CENTRO
20560	GRAJAU
21350	MADUREIRA
21520	PAVUNA

1. 6 : 2 = 3
2. 20560 : TAB (3)

```
          ↓
        20240
```

Como é maior, considera-se a porção inferior:

CEP	BAIRRO
20560	GRAJAU
21350	MADUREIRA
21520	PAVUNA

1. 3 : 2 = 1,5 = 2

2. 20560 : TAB (2)

21350

Como é menor, considera-se a porção superior:

CEP	BAIRRO
20560	GRAJAU
21350	MADUREIRA

1. 2 : 2 = 1

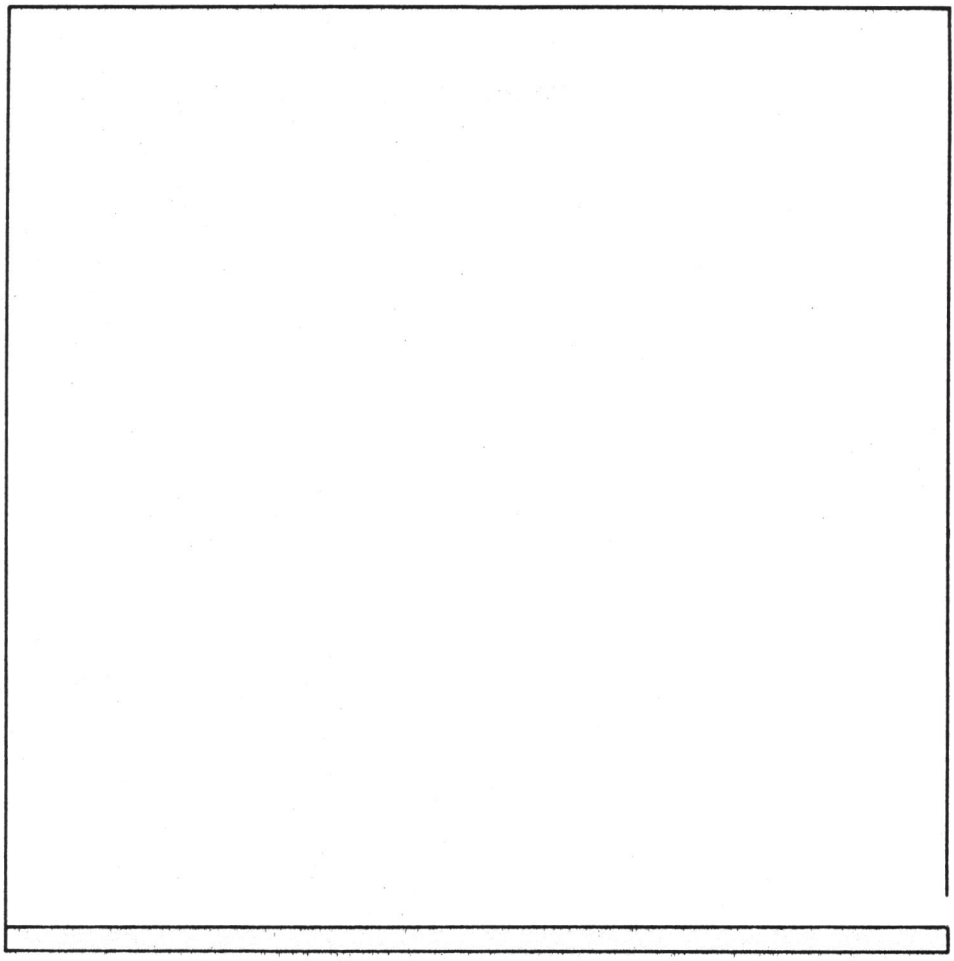
2. 20560 : TAB (1)

Como é igual, encontrou-se o elemento procurado.

A busca termina quando se encontra o elemento procurado ou quando se chega à conclusão de que ele não pertence à tabela.

Em muitas implementações, visando aumentar a eficiência, considera-se sempre a nova tabela até (ou a partir) o elemento contíguo ao elemento comparado, e não até (ou a partir) ele, pois ele já foi comparado ao elemento procurado. Assim, na primeira divisão da tabela, a nova tabela teria apenas 5 elementos (iria até 21350 MADUREIRA).

BIBLIOGRAFIA



_____, COBOL-80 Reference Manual – Microsoft

PRAÇA, Eliana – Apostila de COBOL – NCE/UFRJ

FURTADO, A. L. e SANTOS, C. S. dos – Organização de Banco de Dados

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

ANOTAÇÕES

Impresso na
ERCA Editora e Gráfica Ltda.
Rua Silva Vale, 870 - Cavalcante
Rio de Janeiro - RJ

COBOL PARA MICROS

ELIANA PRAÇA

A idéia deste trabalho surgiu da inexistência, em português, de textos didáticos relativos ao uso da linguagem COBOL em microcomputadores.

O presente texto refere-se ao COBOL-80, utilizado por micros da família INTEL 8080 que "rodam" CP/M.

A obra abrange, com objetividade, os assuntos:

- O COBOL
- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION
- Apêndices: Palavras reservadas no COBOL-80
  Organização de Arquivos
  Busca Binária

MAIS UM LANÇAMENTO
DA
LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

ISBN: 85-216-0405-X