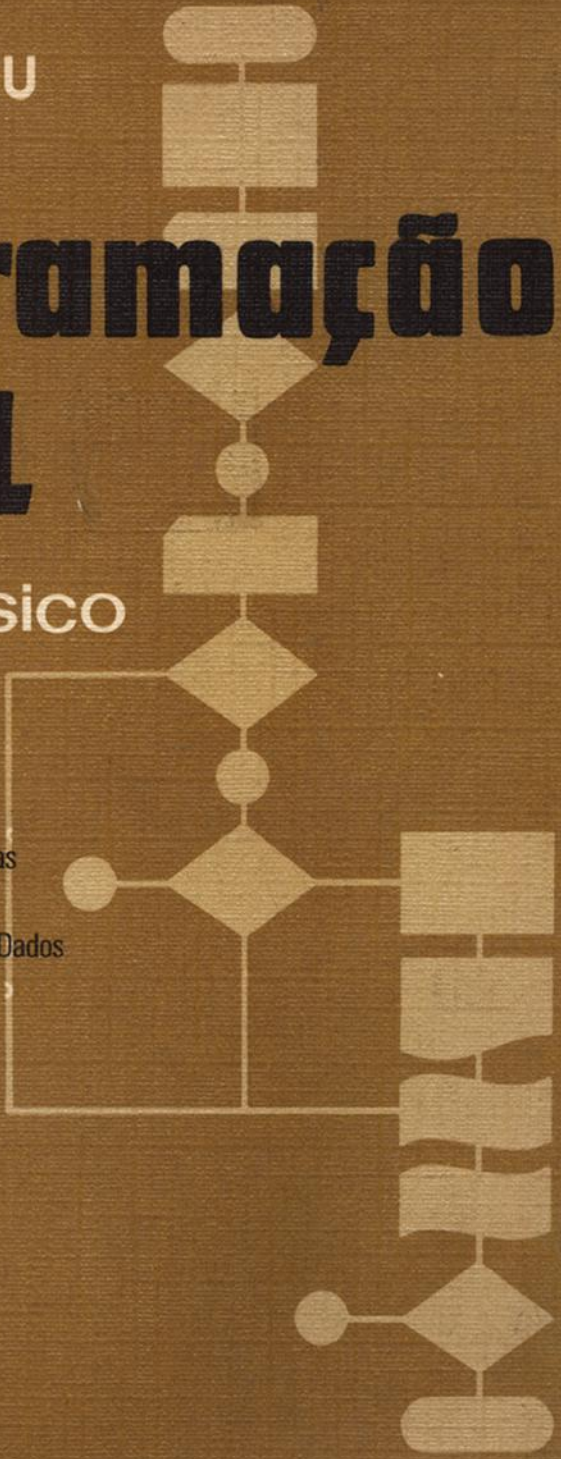
TAMIO SHIMIZU

Programação COBOL

Curso Básico

- COBOL Introdutório
- COBOL Estruturado
- COBOL Avançado
- Uso de Arquivos em Fitas e Disco Magnético
- Conceitos de Banco de Dados
- Exercícios e Testes

2ª edição



atlas

PROGRAMAÇÃO COBOL

CURSO BÁSICO

Tamio Shimizu

Escrito com a preocupação de difundir a linguagem COBOL em termos compreensíveis mesmo aos que mantêm seu primeiro contato com essa linguagem de programação, este texto se destina ao ensino profissionalizante. Considerando que a linguagem COBOL, para a programação comercial, pode ser equiparável à linguagem FORTRAN, para a programação científica e tecnológica, tornam-se necessários textos que a apresentem de forma didática e introdutória, facilitando a sua assimilação por participantes de cursos cuja preocupação fundamental seja a preparação de profissionais para militar no setor.

Procurando atender a esse objetivo, este texto traz inicialmente uma introdução ao computador e os principais conceitos relacionados ao uso dessa máquina. Essa introdução procura remover a idéia de que os computadores eletrônicos são máquinas de complexidade tal que só os mais dotados é que reúnem condições para operá-las, demonstrando ainda aos usuários os múltiplos serviços que elas podem prestar. Em seguida, trata dos princípios em que se fundamentam a programação e a construção de fluxogramas, a partir dos quais se desenvolvem. São também vistos os arquivos e os registros de dados típicos da programação comercial e, em especial, da linguagem COBOL. Toda essa parte introdutória destina-se a eliminar as dificuldades criadas pela falta de conhecimentos básicos, práticos e globais sobre o processamento eletrônico de dados com que em geral se defrontam todos os que, sem essa base, são colocados de imediato em contato com comandos e técnicas sofisticados. Destina-se também a programadores já experientes, que tenham necessidade de consultas esclarecedoras sobre as formas mais gerais de cada comando ou instrução COBOL.

Além desta parte introdutória, o texto é ainda constituído por duas outras partes: **estruturado e avançado.** Em COBOL estruturado são focalizados os seguintes

(Continua na orelha posterior)

(Continuação da orelha anterior)

aspectos técnicos dessa linguagem de programação: sentenças; comandos, expressões aritméticas e lógicas; **identification division** e **evironment division; data division**: definição de arquivos, definição de dados pela cláusula **picture** e outras cláusulas, **working-storage section** e **constant section**: comandos aritméticos, de controle de seqüência de programa, condicionais, de entrada e saída, **exit** e **note**, e, finalmente, programa completo em COBOL. Em COBOL avançado são tratados tópicos mais especializados, incorporados à programação comercial, tais como programação estruturada, uso de sub-rotinas externas em COBOL e conceito de base de dados.

Ao longo do texto, há farto material ilustrativo, exemplos, fluxogramas e figuras, cuja finalidade é facilitar a assimilação da matéria. Há ainda exercícios sobre os assuntos tratados em cada capítulo, que podem ser utilizados para estudos programados, com a mínima assistência de um orientador. A maior parte dos exercícios foi projetada no sentido de que o estudante tente, por seu próprio esforço, encontrar as soluções e preparar o seu próprio programa. A experiência e o treinamento individual, resultantes do esforço próprio, são tidos como indispensáveis nessa área profissional

NOTA SOBRE O AUTOR

TAMIO SHIMIZU é graduado em Matemática pela USP, Mestre em Ciências pelo ITA, de São José dos Campos, e Doutor em Ciências pela Escola Politécnica da USP. É professor livre-docente do Departamento de Engenharia da Produção da Escola Politécnica da USP e professor titular no ITA. Tem cursos a nível pós-doutoral nos Estados Unidos, no **Union College** e no **Rensselaer Polytechnic Institute.** É autor do livro **Simulação em Computador Digital** e de diversos trabalhos científicos e didáticos. É consultor para programação e análise de sistemas, construção de **software** de computadores e pesquisa operacional. É autor de **Processamento de dados — Conceitos básicos** e **Processamento de dados nas empresas,** do acervo da Atlas.

APLICAÇÃO

Livro-texto para cursos profissionalizantes de PROGRAMAÇÃO COBOL. Livro de apoio para as disciplinas PROGRAMAÇÃO e LINGUAGENS DE PROGRAMAÇÃO do curso de Computação

PROGRAMAÇÃO COBOL

curso básico

CIP-Brasil. Catalogação-na-Publicação
Câmara Brasileira do Livro, SP

Shimizu, Tamio, 1938-
S559p Programação COBOL : curso básico : COBOL introdu-
2.ed. tório, COBOL estruturado, COBOL avançado, uso de ar-
quivos em fitas e discos magnéticos, conceitos de
banco de dados / Tamio Shimizu. — 2. ed. — São Pau-
lo : Atlas, 1984.

1. COBOL (Linguagem de programação para computa-
dores) I. Título.

17. CDD-651.8
83-1547 18. -001.6424

Índices para catálogo sistemático:

1. COBOL : Linguagem de programação : Computadores :
Processamento de dados 651.8 (17.) 001.6424 (18.)
2. Programação COBOL : Processamento de dados 651.8
(17.) 001.6424 (18.)

EDITORA ATLAS S.A.
Rua Helvetia, 574/578 — CElis
Caixa Postal 7186 — Tel.: (011) 221-9144
01215 São Paulo (SP)
BRASIL

Tamio Shimizu

PROGRAMAÇÃO COBOL

curso básico

- COBOL Introdutório
- COBOL Estruturado
- COBOL Avançado
- Uso de Arquivos em Fitas e Discos Magnéticos
- Conceitos de Banco de Dados
- Exercícios e Testes

atlas

PROGRAMAÇÃO COBOL – Curso Básico
Tamio Shimizu

Diagramação de
PAVEL GERENCER

Capa de
NATANAEL F. DE MOURA

2.ª EDIÇÃO — 1984

Impresso no Brasil
Printed in Brazil

SUMÁRIO

2

3

4

INTRODUÇÃO À PROGRAMAÇÃO EM LINGUAGEM COBOL, 64

5

PROGRAMAÇÃO ESTRUTURADA EM COBOL, 83

6

7

8

9

10

11

15

16

17

24

25

26

ÍNDICE POR ASSUNTO

PREFÁCIO

O COBOL destaca-se como a principal linguagem para programação comercial, cuja importância e nível de difusão, neste campo, podem ser comparáveis à linguagem FORTRAN para programação científica e tecnológica.

Entretanto, o COBOL não é uma linguagem simples.

A maioria dos textos existentes sobre o COBOL, embora muito bem explicados e exemplificados, muitas vezes acarreta dificuldades aos programadores novatos, pois que os obriga ao contato quase que imediato e constante com exposições longas sobre comandos e técnicas bastante sofisticados, sem o necessário acompanhamento da visão prática e global do que está acontecendo.

Por outro lado, os programadores mais experientes necessitam de um texto de consulta permanente e bem organizado sobre o COBOL, a fim de elaborar e testar seus programas. Para estes, é essencial a consulta às formas mais gerais de cada comando ou instrução COBOL.

O próprio professor de um curso sobre COBOL pode sentir dificuldades em organizar a seqüência adequada dos assuntos, bem como de dosar o nível de profundidade de tais assuntos, na dependência do grau e da experiência dos seus alunos.

Este texto procura atender aos diferentes interesses dos programadores, estudantes e professores de processamento de dados, apresentando a linguagem COBOL de maneira simples e compacta, em três níveis:

- COBOL INTRODUTÓRIO: Destinado à leitura ou curso rápido de 1 semana ou 15 horas de aula sobre os fundamentos do COBOL, apresentando um primeiro contato e uma visão global da linguagem.* (Capítulos 1 a 4.)
- COBOL ESTRUTURADO: Destinado a um curso normal de programa comercial em COBOL, com duração de um semestre; pode também ser usado como *manual de consulta* para o programador. (Capítulos 5 a 21.)
- COBOL AVANÇADO: Estuda os tópicos mais especializados ou avançados da programação comercial, tais como uso de variáveis subscritas, programação com arquivos e conceito de banco de dados. Destina-se a programadores

* Usa, basicamente, alguns capítulos do livro *Processamento de Dados: Conceitos Básicos*, de T. Shimizu, publicado pela Atlas em 1980.

bastante experimentados, que necessitem de aplicações mais sofisticadas: pode ser usado como material de leitura pelos alunos do curso de COBOL Estruturado. (Capítulo 22 em diante)

Texto ilustrado, procura, sempre que possível, apresentar os assuntos de forma clara e compacta, trazendo exemplos, fluxogramas ou figuras, pois é fundamental que o programador ou o estudante tenham sempre uma visão concreta do processo executado pelo computador.

A distribuição dos assuntos e os exercícios de recapitulação e de aplicação permitem também que o texto seja utilizado como MANUAL DE ESTUDO PROGRAMADO, com a assistência mínima de um professor.

As ilustrações no final de cada capítulo, visualizadas através de figuras, devem ser encaradas como um resumo do assunto principal tratado.

O início de cada capítulo traz uma síntese da matéria nele contida, isto para facilitar a apresentação de seminários, de aulas de recapitulação e de recordação para exames com a utilização de *slides* ou de transparências projetadas.

Alguns assuntos especiais que versam sobre a elaboração de relatórios (REPORT SECTION), ou classificação de dados (SORT FEATURES), ou que dependem da especificação particular do computador usado, como o conceito e o uso de LIBRARY's, não são normalmente tratados em textos deste tipo. Para esses assuntos, ou outros não mencionados no texto, o programador deve sempre consultar o Manual de Programação COBOL do computador que está sendo utilizado.

Uma das dificuldades da linguagem COBOL é que, apesar de oferecer razoável liberdade e facilidade de uso, pode apresentar certo rigor no uso das pontuações, ou espaços em branco, bem como imprecisão na escolha de comandos e de cláusulas opcionais. Tais fatos, aliados a outras particularidades oferecidas pelo compilador COBOL em uso, podem acarretar dificuldades ao programador na formulação de um raciocínio lógico e mais preciso sobre o uso dos comandos, o que não acontece com outras linguagens, como BASIC, FORTRAN, e PL/1.

Procuramos seguir as recomendações contidas nos documentos oficiais da definição da linguagem COBOL, deixando de lado muitas variações ou opções que dependem do computador em uso.

Ainda, um lembrete fundamental: a experiência em programação só pode ser adquirida através da prática. Sendo assim, após ler e entender um texto didático ou uma aula, o estudante deve resolver *sozinho* os exercícios e preparar o seu próprio programa, pois só assim poderá enfrentar as dificuldades reais e adquirir experiência.

Nesta 2ª edição acrescentamos os seguintes assuntos:

- Programação Estruturada em Cobol (com exemplos).
- Uso de Arquivos em Disco Magnético.
- Glossário de Palavras-chaves.
- Testes de Múltipla Escolha.

NOTA DE RECONHECIMENTO SOLICITADA PELA AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)

"Qualquer organização interessada em reproduzir o relatório e especificações do COBOL total ou parcialmente, usando idéias do relatório como base para um manual de instrução ou para quaisquer outras finalidades, poderá fazê-lo. Entretanto, solicita-se de todas essas organizações a reprodução desta seção como parte da introdução ao documento. Quando se tratar de um texto curto, como, por exemplo, a resenha de um livro, solicita-se a menção de 'COBOL' em reconhecimento da fonte, mas não é necessário citar toda essa seção.

"O COBOL é uma linguagem de indústria e não é propriedade de nenhuma empresa ou grupo empresarial, nem de qualquer organização ou grupo de organizações.

"Nenhum dos contribuidores do Comitê de COBOL faz qualquer garantia, explícita ou implícita, sobre a exatidão do funcionamento do sistema de programação e da linguagem. Além disso, nenhum contribuidor, nem o comitê, assume responsabilidade alguma nesse sentido.

"Foram estabelecidos procedimentos para a manutenção do COBOL. Os pedidos de informações sobre os procedimentos para sugestão de alterações devem ser dirigidos ao Comitê Executivo da Conferência sobre Linguagem de Sistemas de Dados.

"Os autores e proprietários de material protegido pela legislação sobre direitos autorais aqui usado, FLOWMATIC (marca registrada da Sperry Rand Corporation), Programação ao Univac I e II, Sistema de Automação de Dados registrado em 1958, 1959, pela Sperry Rand Corporation; IBM Commercial Transistor Form Nº F28-8013, com Copyright de 1959, da IBM; FACT, DSI 27A52602760, com Copyright de 1960 da Minneapolis-Honeywell, autorizaram especificamente o uso deste material, total ou parcialmente, nas especificações de COBOL. Essa autorização aplica-se também ao uso de especificações de COBOL em manuais de programação ou publicações similares."

Parte I

COBOL INTRODUTÓRIO

1 COMO FUNCIONA UM COMPUTADOR

SÍNTESE DA MATÉRIA CONTIDA NO CAPÍTULO

AS OPERAÇÕES BÁSICAS DE UM COMPUTADOR
- Leitura de dados (Entrada de Dados).
- Armazenamento (Memória).
- Cálculos (Processamento).
- Fornecedor de resultados (Saída de Dados).

COMPONENTES BÁSICOS DE UM COMPUTADOR
- Unidade de Memória.
- Unidade Central de Processamento ou UCP.
- Unidade de Entrada de Dados.
- Unidade de Saída de Dados.
- Unidade de Comunicação: canais seletores e multiplexadores.

PRINCIPAIS UNIDADES DE ENTRADA E DE SAÍDA
- Leitora e Perfuradora de Cartões.
- Impressora de Linha.
- Leitora e Gravadora de Fita Magnética.
- Leitora e Gravadora de Disco Magnético etc.

COMO FUNCIONAM AS UNIDADES DE UM COMPUTADOR
- Carregar o Programa na Memória.
- Executar o Programa.

PALAVRAS, BYTES E BITS DA MEMÓRIA: A LINGUAGEM DE MÁQUINA
- As palavras de uma memória e seu endereço.
- Os bits e a notação binária.
- Palavra de oito bits: O BYTE.
- A linguagem de máquina.

O USO DIFÍCIL DA LINGUAGEM DE MÁQUINA: OUTRAS LINGUAGENS DE PROGRAMAÇÃO
- A Linguagem Simbólica ou Assembler.
- As Linguagens de alto nível: FORTRAN, COBOL, PL/1.

O PROCESSO DE TRADUÇÃO OU COMPILAÇÃO: O COMPILADOR OU TRADUTOR, O PROGRAMA-FONTE E O PROGRAMA-OBJETO.

1.1. OPERAÇÕES BÁSICAS

O computador funciona executando o programa ou seqüência de operações que está armazenada na sua memória. Esse programa é formado basicamente por combinações das seguintes operações:

- Ler dados a serem processados (Entrada de Dados).
- Armazenar os dados lidos (Memória).
- Efetuar os cálculos necessários (Processamento).
- Fornecer os resultados (Saída de Dados).

Essas quatro operações básicas constituem as operações básicas de qualquer tipo de computador independente do tamanho ou do serviço executado. O que pode ocorrer são formas variadas ou sofisticadas de executar cada uma das operações básicas. Por exemplo:

- a entrada ou a saída de dados pode ser feita de diferentes maneiras dependendo do equipamento usado; ou
- o processamento pode ser de vários tipos: operações aritméticas com dados isolados ou conjunto de dados, transferência ou comparação de dados de tabelas etc.

1.2. COMPONENTES BÁSICOS DO COMPUTADOR

Os equipamentos básicos (ou *hardware*) que compõem um computador ou sistema de computação podem ser divididos nas seguintes categorias:

- MEMÓRIA.
- UNIDADE CENTRAL DE PROCESSAMENTO OU UCP.
- UNIDADES DE ENTRADA DE DADOS.
- UNIDADES DE SAÍDA DE DADOS.
- CANAIS DE COMUNICAÇÃO: CANAIS SELETORES E MULTIPLEXADORES.

1.2.1. As unidades de entrada de dados

Servem para receber os dados que foram usados ou preparados pelo ser humano (como cartão perfurado, fita magnética, cheques com caracteres ótico ou magnético etc.) e colocá-los na memória do computador em forma de sinais eletromagnéticos de fácil manipulação pelo computador (esses sinais são chamados "bits").

1.2.2. As unidades de saída de dados

Servem para efetuar a transformação dos dados da memória que estão em forma de sinais eletromagnéticos ou "bits", em forma acessível ou manipulável ao ser humano tal como caráter impresso em folha de papel, cartão perfurado, fita magnética gravada etc.

As unidades de Entrada ou de Saída de dados são comumente denominadas de **Unidades Periféricas.**

As figuras do capítulo mostram os diversos tipos de equipamentos de entrada e de saída de dados. Descrição mais detalhada das características de tais equipamentos e veículos de entrada e de saída usados encontra-se em *Processamento de Dados: Conceitos Básicos.* *

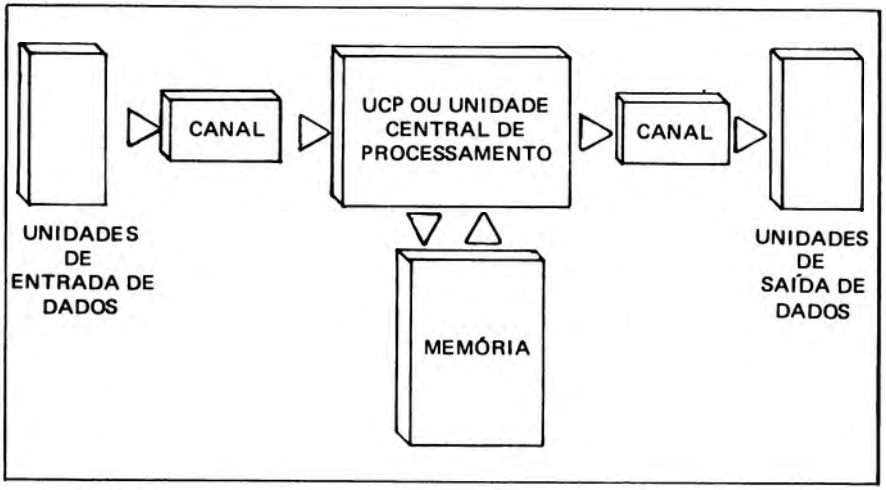


Figura 1.1. *Componentes básicos de um computador.*

1.2.3. A Unidade Central de Processamento (UCP)

Contém circuitos eletrônicos para efetuar as operações de adição, subtração, multiplicação e divisão, além de controlar a manipulação de dados da memória e a execução de um programa.

1.2.4. A memória

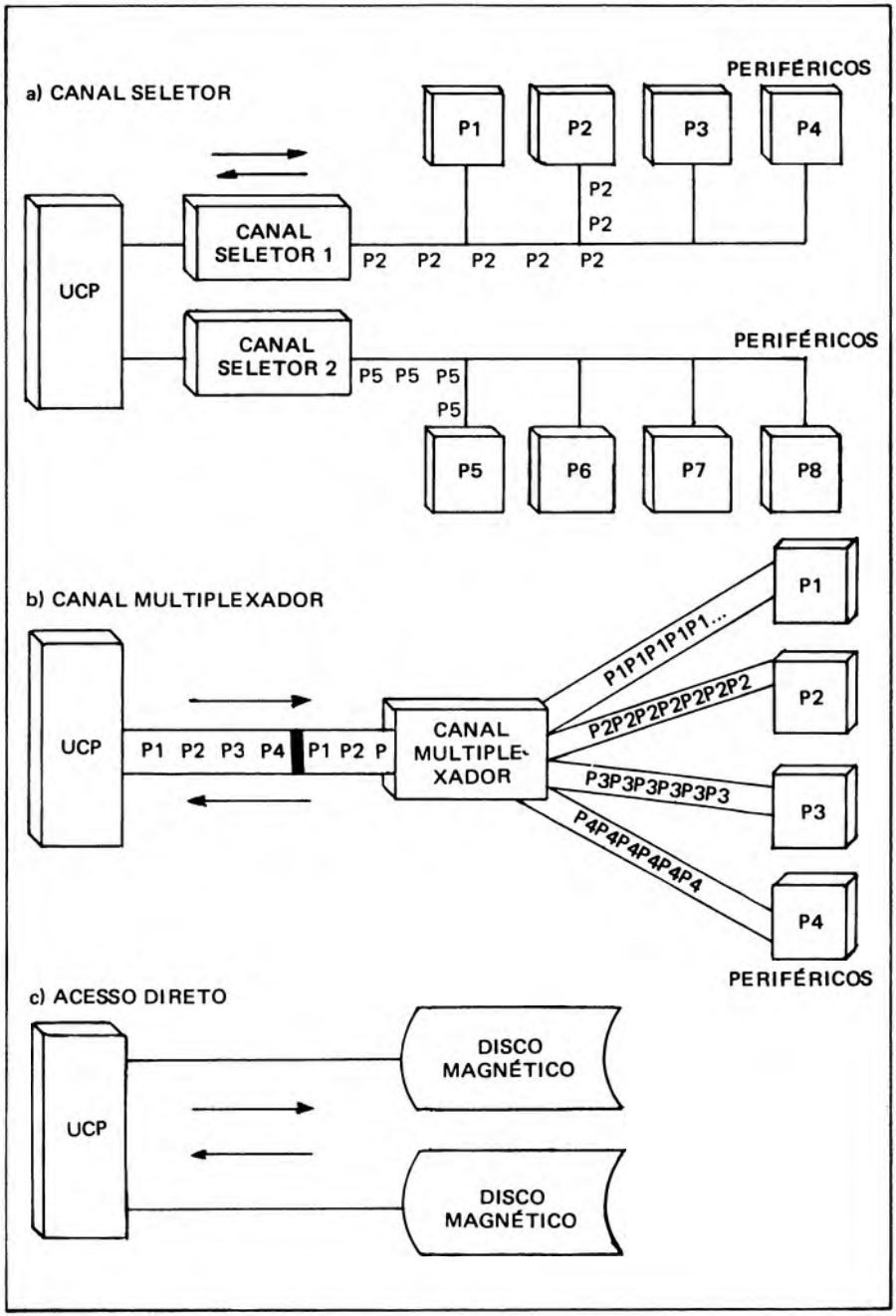
É usada para guardar os dados lidos, os resultados intermediários e finais obtidos e uma cópia do programa que o computador está executando.

1.2.5. Unidades de comunicação entre a UCP e as unidades de entrada e saída: os canais seletores e multiplexadores

Os **Canais** são unidades que ligam as diversas unidades de Entrada e de Saída com a UCP, controlando e otimizando a operação de transferência de dados entre as mesmas, compensando as diferenças de características físicas, de capacidade de guardar informação e de velocidade.

---

* T. Shimizu, *Processamento de Dados: Conceitos Básicos* (São Paulo, Atlas, 1980).



28 Figura 1.2. *Canais de comunicação.*

Existem dois tipos básicos de canais: os canais seletores e os canais multiplexadores, que podem ser visualizados na Figura 1.2.

a) *Canais seletores*

Um canal seletor transfere o dado da UCP para uma unidade periférica de cada vez, ou vice-versa, controlando (ou selecionando), uma a uma, as operações de Entrada ou Saída de inúmeras unidades periféricas a ele ligadas, conforme Figura 1.2a.

O canal seletor serve como intermediário para atender à operação de unidades periféricas de velocidades menores do que a da UCP, liberando a alta velocidade da UCP para sua atividade normal.

A seleção e o controle dos diversos dispositivos ligados a um canal são efetuados por atendimento a sinais denominados **Interrupções.**

b) *Canais multiplexadores*

Em um canal multiplexador, como se pode observar na Figura 1.2b, mais de uma unidade periférica pode operar ao mesmo tempo, pois cada unidade-padrão de dados (bytes ou palavras, ver seção 1.5) transferidos de unidades periféricas ou para elas está intercalada entre si, permitindo que todas as unidades periféricas sejam atendidas de modo contínuo, praticamente sem interrupção.

O canal multiplexador difere do canal seletor que atende (seleciona), durante certo intervalo de tempo, apenas uma das unidades periféricas, deixando as demais na espera.

Os canais multiplexadores são usados nos sistemas de computação denominados **Time-Sharing (Tempo Compartilhado)** e sistemas com processamento remoto **(Teleprocessamento).**

c) *Comunicação ou acesso direto*

Outra modalidade de comunicação entre UCP e Entrada/Saída é a ligação ou acesso direto, sem o uso do canal.

O acesso direto só se justifica quando o volume de dados e a velocidade de operação de Entrada ou Saída de dados forem bastante altos, como acontece com o Disco Magnético. Isto porque cada unidade ligada por acesso direto estaria substituindo a capacidade de um canal de comunicação, que poderia controlar várias unidades periféricas ao mesmo tempo.

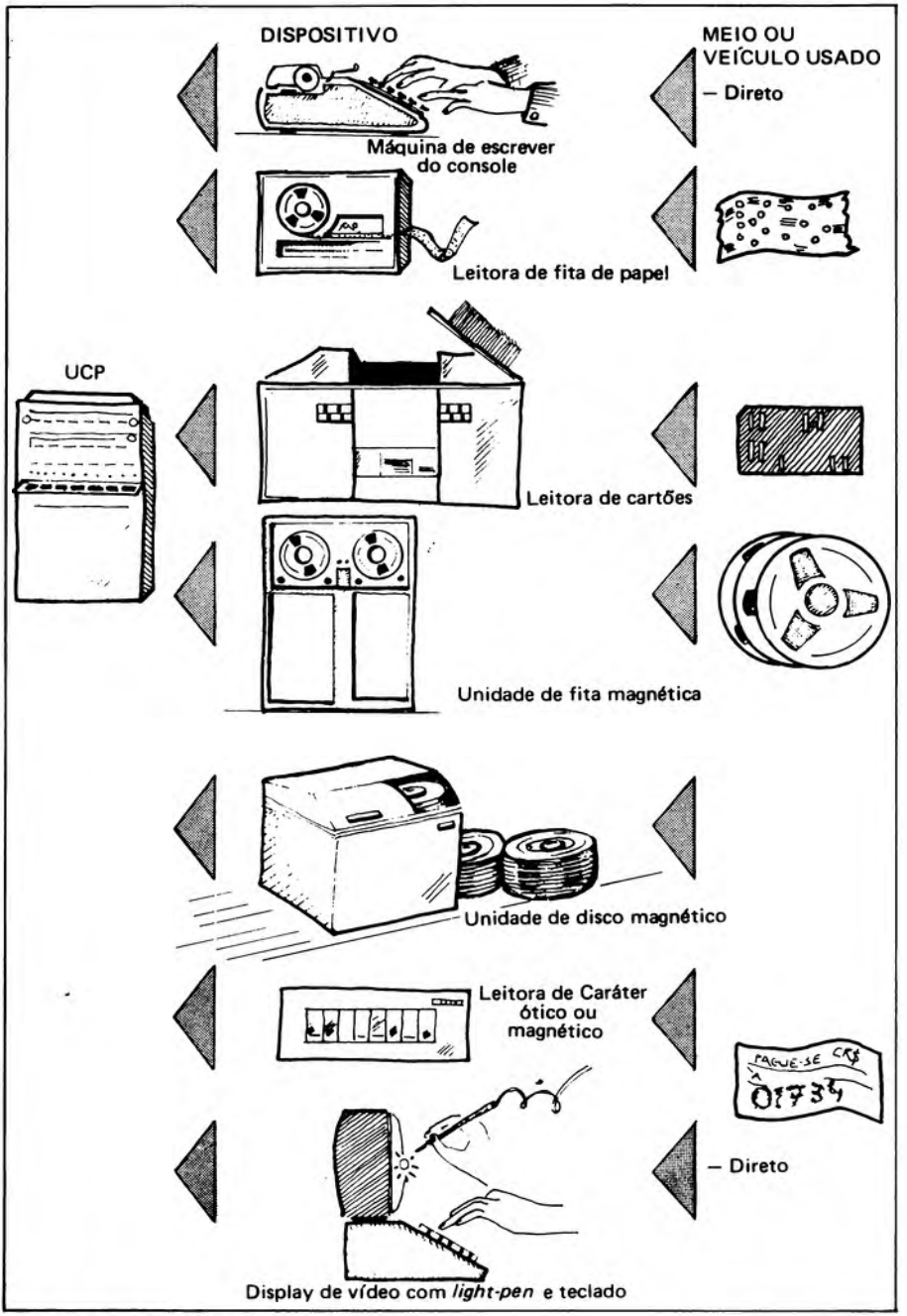


Figura 1.3. *Unidades de entrada.*

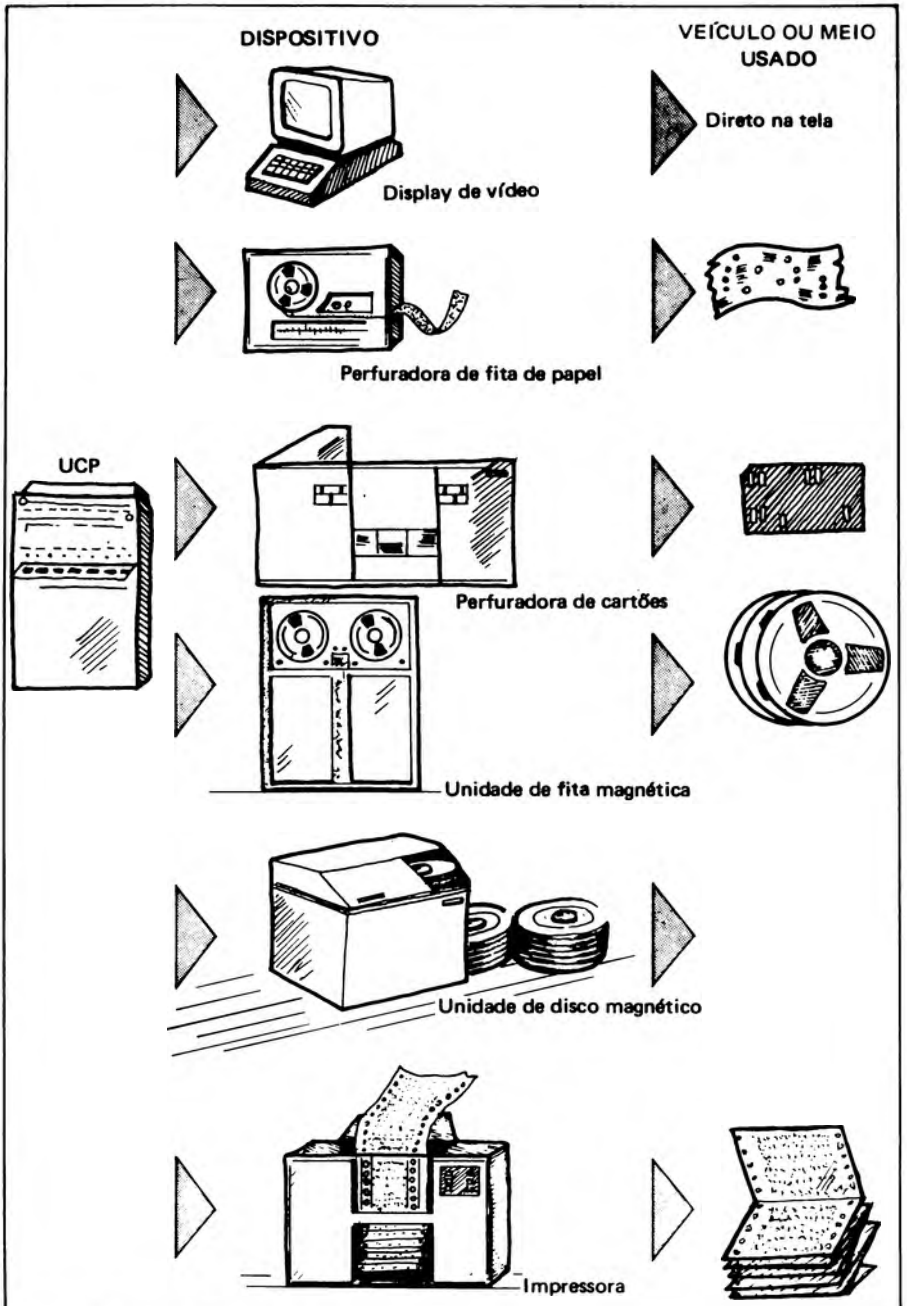


Figura 1.4. *Unidades de saída.*

1.3. AS PRINCIPAIS UNIDADES DE ENTRADA E DE SAÍDA

TIPO	Entrada/ Saída	Meio Usado	Velocidade
Leitora de Cartões	Entrada	Cartão Perfurado	Média
Perfuradora de Cartões	Saída	Cartão Perfurado	Média/Baixa
Impressora de Linha	Saída	Formulário Contínuo	Média/Alta
Unidade de Fita Magnética	Ambas	Fita Magnética	Alta
Unidade de Disco Magnético	Ambas	Disco Magnético	Altíssima
Máquina de Escrever	Ambas	Teclado e Formulário	Lenta
Leitora de Fita de Papel	Entrada	Fita de Papel	Média
Perfuradora de Fita de Papel	Saída	Fita de Papel	Média/Baixa
Leitora Ótica ou Magnética	Entrada	Documentos com caráter ótico ou magnético	Média/Baixa
Display de Vídeo TV	Ambas ou só saída	Direto na Tela ou entrada por "light-pen"	Alta

1.4. COMO FUNCIONAM AS UNIDADES DE UM COMPUTADOR

Vamos verificar, passo a passo, como o computador executa o seguinte problema simples de cálculo:

"SOMAR OS VALORES 5 e 10 e IMPRIMIR O RESULTADO."

FASE UM: LEITURA E CARREGAMENTO DO PROGRAMA

Inicialmente, o computador deve estar pronto para receber o programa. Supondo que já sabemos escrever um programa, tal programa será perfurado em cartão e colocado na seção alimentadora de cartões da Unidade Leitora de Cartões. O operador apertará a tecla "CARREGA ou LOAD" e os cartões serão lidos e o programa carregado na Memória. Os cartões lidos surgem na seção ou escaninho dos cartões lidos.

FASE DOIS: EXECUÇÃO DO PROGRAMA QUE ESTÁ NA MEMÓRIA

O operador aperta a tecla "PARTIDA ou START" da UCP e o computador começa a executar o programa que deve conter as seguintes operações:

- Ler um cartão de dados com os valores 5 e 10 perfurados e colocar tais valores na Memória.
- Calcular a soma 5 + 10 = 15 e colocar o resultado na Memória.
- Escrever o resultado 15 sobre uma folha de papel (formulário contínuo) que está na Impressora.

Após a execução, o computador está pronto para receber e executar o próximo programa.

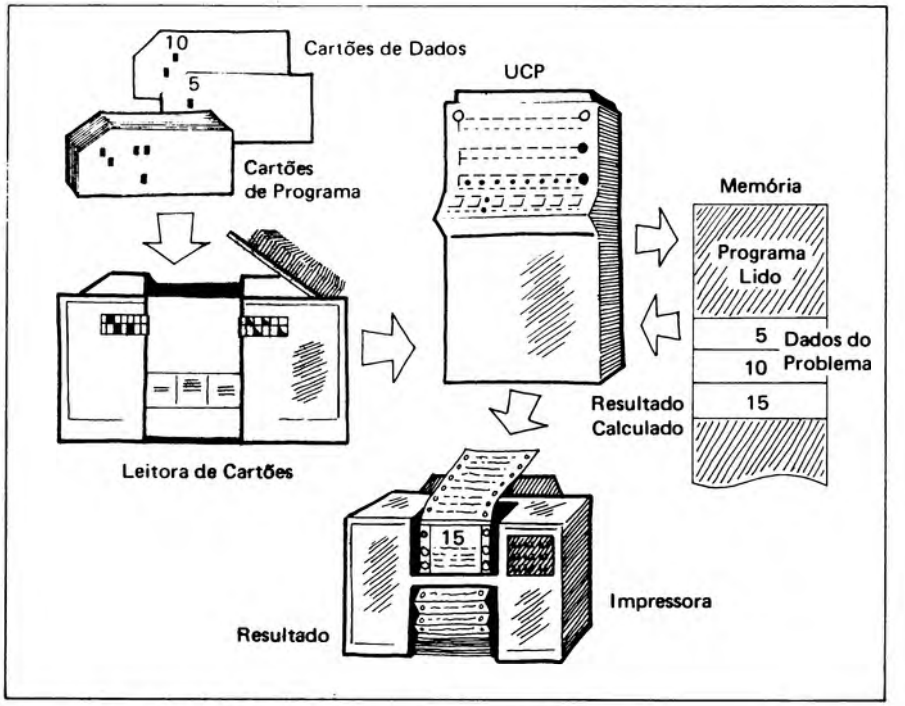


Figura 1.5. *Execução de um programa de computador.*

1.5. PALAVRAS, BYTES E BITS DA MEMÓRIA: A LINGUAGEM DE MÁQUINA

A memória de um computador está subdividida em unidades chamadas PALAVRAS.

O tamanho da memória é expresso em número de palavras. Assim, temos memória de 1.000 palavras, 12.000 palavras etc.

Cada palavra está identificada por um valor numérico chamado ENDEREÇO da palavra, através do qual a mesma é usada nos programas.

A palavra contém informação ou dado representado através de elementos chamados BITS e que assumem o valor UM (ligado ou magnetizado) ou ZERO (desligado ou desmagnetizado). A informação representada pelos bits da palavra corresponde ao conteúdo da palavra em NOTAÇÃO BINÁRIA OU CÓDIGO DE MÁQUINA.

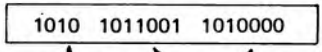
A palavra de um computador pode ser de 8 bits ou 12, 16, 24, 32, 48, 64 etc. Cada conjunto de 8 bits é denominado BYTES, sendo comum chamar palavras de 1, 2, 3 etc. bytes àquelas que contêm respectivamente 8 bits, 16, 24 etc.

1.5.1. A linguagem de máquina

A linguagem que o computador entende e trabalha é chamada LINGUAGEM DE MÁQUINA do computador e é formada por um conjunto de operações codificadas em notação binária.

O computador, para cada código, aciona determinadas peças ou circuitos eletrônicos e executa a operação correspondente. Cada uma das operações codificadas em notação binária e que ocupa uma ou mais palavras da memória é denominada INSTRUÇÃO DE MÁQUINA.

Por exemplo, a instrução de máquina para a operação de soma, em certo computador, poderia ter a forma:



Código que indica SOMA — Local ou endereço da primeira e da segunda parcela que entram na soma

e significa: "somar o valor da primeira parcela que está no endereço 1011001 da memória com a segunda parcela que está no endereço 1010000".

Codificando-se todas as operações para executar um problema sob a forma de instrução de máquina, teremos um PROGRAMA EM LINGUAGEM DE MÁQUINA para resolver esse problema.

1.6. O USO DIFÍCIL DA LINGUAGEM DE MÁQUINA: OUTRAS LINGUAGENS DE PROGRAMAÇÃO

A notação binária torna extremamente difícil o aprendizado e o entendimento de um programa em linguagem de máquina.

Para facilitar o usuário, outros tipos de linguagens mais acessíveis foram elaborados. Essas linguagens usam códigos simbólicos ou mnemônicos dos nomes das operações, ou fórmulas matemáticas, ou frases fáceis de serem entendidas, facilitando a tarefa de preparação, teste e documentação do programa. Apresentamos a seguir alguns comentários sobre outras linguagens de programação.

- *Linguagem Assembler:* é também chamada Linguagem Simbólica ou de Montador, pois indica o código das operações por códigos simbólicos ou mnemônicos e os endereços dos operandos envolvidos por nomes também simbólicos. É também uma representação ou montagem, instrução por instrução, em forma de símbolos, das instruções de máquina. Exemplo:

```
ADD     X,Y     Soma de X com Y
BRA     P1      Desvia (Branch) para o local P1 da Memória
```

- *Linguagem FORTRAN:* é uma linguagem com descrição bastante próxima da língua inglesa e notações matemáticas e por isso chamada de **Linguagem de Alto Nível** destinada a resolver problemas científicos.
  Exemplo:

```
A = X + Y
C = (B/D) * TOTAL     Dividir B por D e multiplicar o resultado por
                      TOTAL
```

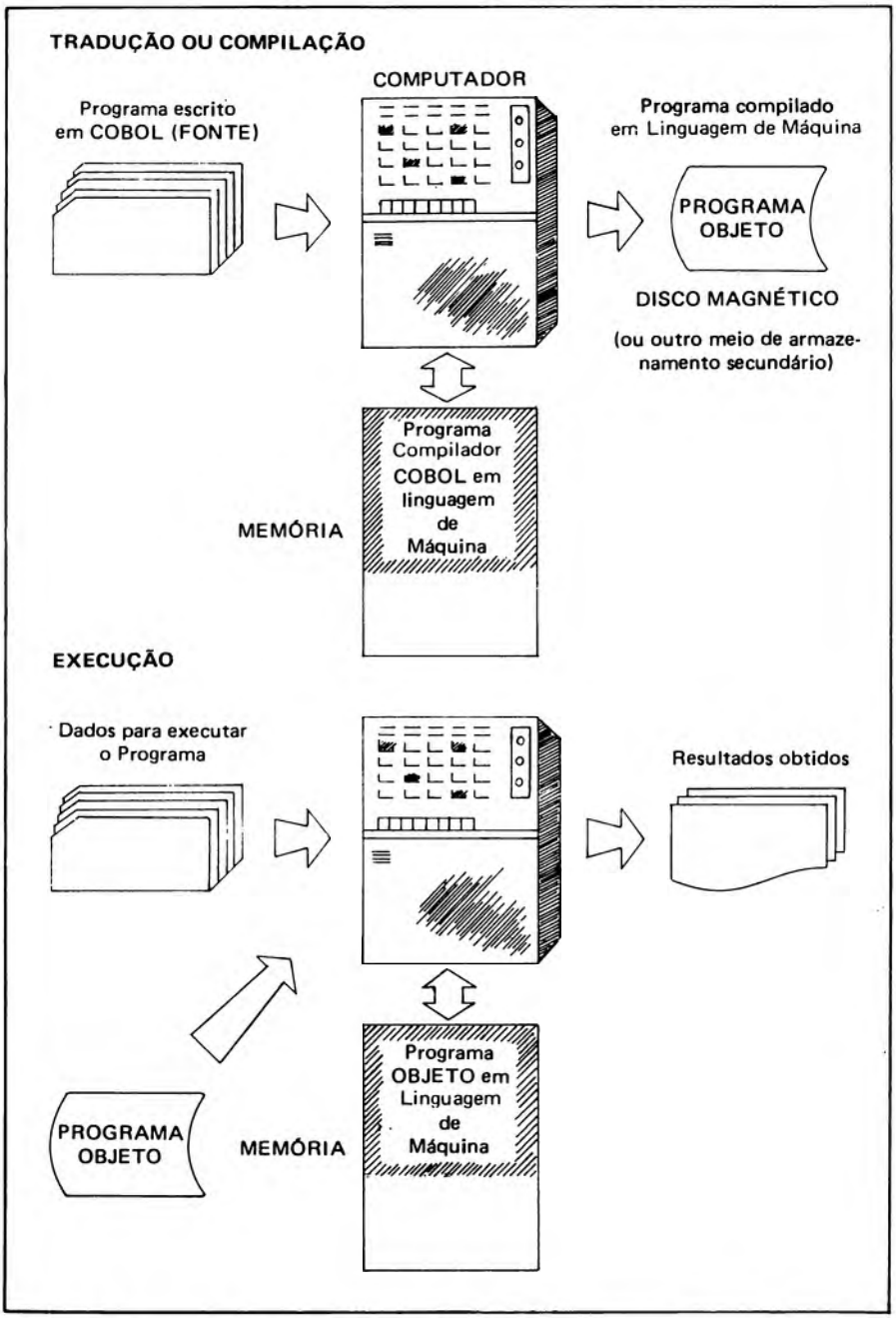


Figura 1.6. *Processo de compilação e execução de um programa.*

- *Linguagem COBOL:* é uma linguagem de alto nível, porém mais adequada para resolver problemas na área comercial e que envolvem valor monetário, emissão de documentos etc.
  Exemplo:

ADD TOTAL, ITEM GIVING SOMA-TOTAL	Somar TOTAL com ITEM e colocar o resultado em SOMA-TOTAL
MOVE AREA1 TO AREA2	Transferir o conteúdo de AREA1 para AREA2

1.7. O PROCESSO DE TRADUÇÃO OU COMPILAÇÃO: O COMPILADOR, O PROGRAMA-FONTE E O PROGRAMA-OBJETO

Somente programas escritos em linguagem de máquina podem ser executados diretamente pelo computador.

Os programas escritos em outras linguagens, como Assembler, FORTRAN etc., devem ser previamente traduzidos (também dizemos compilados) para a linguagem de máquina através de programas especiais chamados **Tradutores ou Compiladores.**

O processo de tradução ou compilação de um programa escrito em COBOL está ilustrado na Figura 1.6.

2 PRINCÍPIOS DE PROGRAMAÇÃO: A CONSTRUÇÃO DO RACIOCÍNIO ATRAVÉS DE FLUXOGRAMAS

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

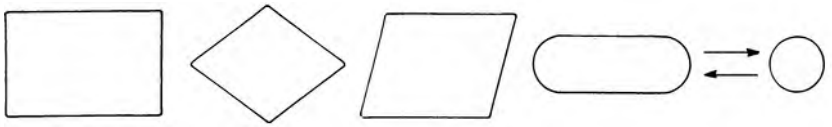
A PROGRAMAÇÃO COMO HABILIDADE DE USAR COMANDOS CORRETOS E ADEQUADOS DE UMA LINGUAGEM

ETAPAS PARA PREPARAR E EXECUTAR UM PROGRAMA

- Analisar o programa e projetar em forma de fluxogramas (Análise e Sistemas).
- Codificar o programa em uma linguagem de programação (Programação e Codificação).
- Preparar o programa codificado em forma adequada para leitura (Digitação).
- Teste do programa (Primeira Execução).
- Correção dos erros.
- Programa testado para processamento do serviço (Rotina Normal).
- Documentação (Elaboração de Manuais e Catalogação do Programa).

ANÁLISE DE SISTEMA POR FLUXOGRAMAS

- Principais símbolos usados no fluxograma:



EXEMPLOS DE FLUXOGRAMAS

- Exemplos de 1 a 9

Exercícios

2.1. A PROGRAMAÇÃO COMO HABILIDADE DE USAR COMANDOS CORRETOS E ADEQUADOS DE UMA LINGUAGEM

Para resolver um problema em computador é necessário escrever um programa em uma linguagem de programação, com comandos ou instruções organizadas de acordo com um raciocínio adequado para resolver o mesmo.

É necessário ter habilidade e treinamento para escolher as instruções ou comandos corretos de uma linguagem escolhida.

Não é essencial ter conhecimento de várias linguagens, sendo preferível possuir bom treinamento em uma única linguagem preferida (linguagem de máquina, Assembler, FORTRAN, COBOL, PL/1 etc.), após o que se deve partir para o uso de outras linguagens.

A tarefa de preparar e escrever um programa é denominada PROGRAMAÇÃO e a pessoa que escreve programas é chamada PROGRAMADOR.

É comum dois programadores escreverem programas diferentes para resolver um mesmo problema, pois a programação depende do gosto e da habilidade individual.

A programação é tarefa fascinante, que requer habilidade e, sobretudo, dedicação para a elaboração de um programa correto e eficiente.

2.2. ETAPAS PARA PREPARAR E EXECUTAR UM PROGRAMA

A preparação e a execução de um programa envolvem etapas, que podem ser vistas na Figura 2.1.

O **Analista de Sistemas** é uma pessoa que entende bem a resolução de problema proposto e geralmente conhece bem a linguagem de programação que está sendo usada.

O **Programador** conhece bem uma ou mais linguagens de programação e sabe testar o programa e corrigir os erros. Não entende necessariamente o mecanismo de resolução dos problemas, mas precisa saber dialogar e interagir com o analista de sistemas.

A apresentação do programa para teste e execução deve seguir as orientações ou exigências do centro de processamento de dados quanto à maneira de preencher as fichas de pedidos de execução, estimativa de tempo, tipo de serviço etc.

2.3. ORGANIZAÇÃO DO RACIOCÍNIO PARA RESOLVER O PROBLEMA: A ANÁLISE DE SISTEMAS

O estudo do problema e o estabelecimento do raciocínio para sua resolução são efetuados por uma pessoa que entenda bem o problema e que se chama **Analista de Sistemas.**

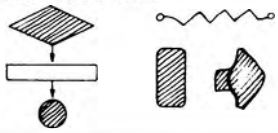
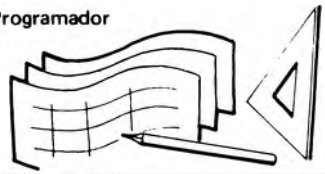
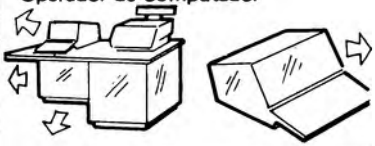
ETAPAS DE PROGRAMAÇÃO	PESSOA QUE EXECUTA
• Estudar o problema e estabelecer um esquema de raciocínio para resolvê-lo. Apresentar o esquema em forma de um gráfico chamado *Fluxograma.*	Analista de Sistemas 
• Transcrever a seqüência do raciocínio do fluxograma em forma de instrução ou comandos de uma linguagem de programação. O programa é escrito em uma *Folha de Programação.*	Programador 
• Cada comando ou instrução da folha de programação é perfurado em *cartão perfurado* (ou outro meio disponível), por meio de máquinas de digitação.	Perfuradora de Cartões ou digitadora 
• Entregar os cartões de programa para *Primeira Execução* no computador, juntamente com os cartões de dados para teste.	Operador de Computador 
• Receber os resultados do primeiro teste. Se houver erros indicados pelo computador, *corrigir os erros* e entregar o programa para nova execução. Se não houver erros nos testes, o programa estará pronto para execução e para resolver o problema proposto. Esta etapa é repetida até o programa não apresentar mais erros.	Programador e Analista de Sistemas 
• Entregar o programa pronto para que seja utilizado na *Rotina Normal de Serviço.*	Chefia do Centro ou da Operação
• Entregar uma cópia do programa e de toda documentação (fluxograma, listagem do programa e manual de uso) para o serviço de *Documentação e Arquivo.*	Bibliotecária de Programas 

Figura 2.1. *Etapas para a preparação e a execução de programas.*

O serviço efetuado pelo analista de sistemas chama-se **Análise de Sistemas,** que consiste em reconhecer o raciocínio ou fórmulas usadas para resolver o problema, estabelecer os dados de entrada e os resultados a serem obtidos e escrever o raciocínio completo da resolução, através de um gráfico chamado **Fluxograma.**

Sendo assim, com um fluxograma bem organizado e correto, o programador não terá dificuldades em codificar ou escrever as operações nele contidas, em forma de instrução ou comandos de uma linguagem.

2.4. A ANÁLISE DE SISTEMAS POR FLUXOGRAMAS

A organização do raciocínio de resolução de um problema (que é também chamado de **Algoritmo)** pode ser feita através de um esquema gráfico chamado **Fluxograma** ou **Diagrama de Blocos.**

O fluxograma usa um conjunto de símbolos gráficos que descreve as etapas de resolução do problema, e é sempre conveniente preparar o fluxograma correspondente antes de escrever o programa final.

O fluxograma fornece, portanto, um estudo prévio e visual das etapas de resolução de um problema.

2.4.1. Principais símbolos utilizados em fluxogramas

Os principais símbolos gráficos utilizados nos fluxogramas deste livro são apresentados na Figura 2.2.

Não existem regras rígidas para o uso dos símbolos em um fluxograma, podendo-se usar símbolos ou regras ligeiramente diferentes dos citados. Entretanto, o fluxograma deve indicar com clareza e precisão o raciocínio e as operações envolvidas, de modo que possa ser imediatamente transcrito em forma de programa, e sirva como meio eficiente de documentação do mesmo.

2.4.2. Os símbolos de fluxograma adotados pela ANSI – American National Standards Institute

Na Figura 2.3 apresentamos os principais símbolos gráficos usados em um **Fluxograma** e adotados pela ANSI.

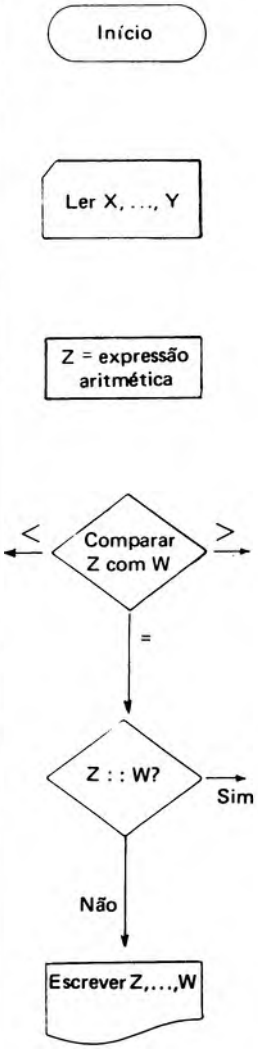


Figura 2.2. *Símbolos gráficos utilizados nos fluxogramas.*

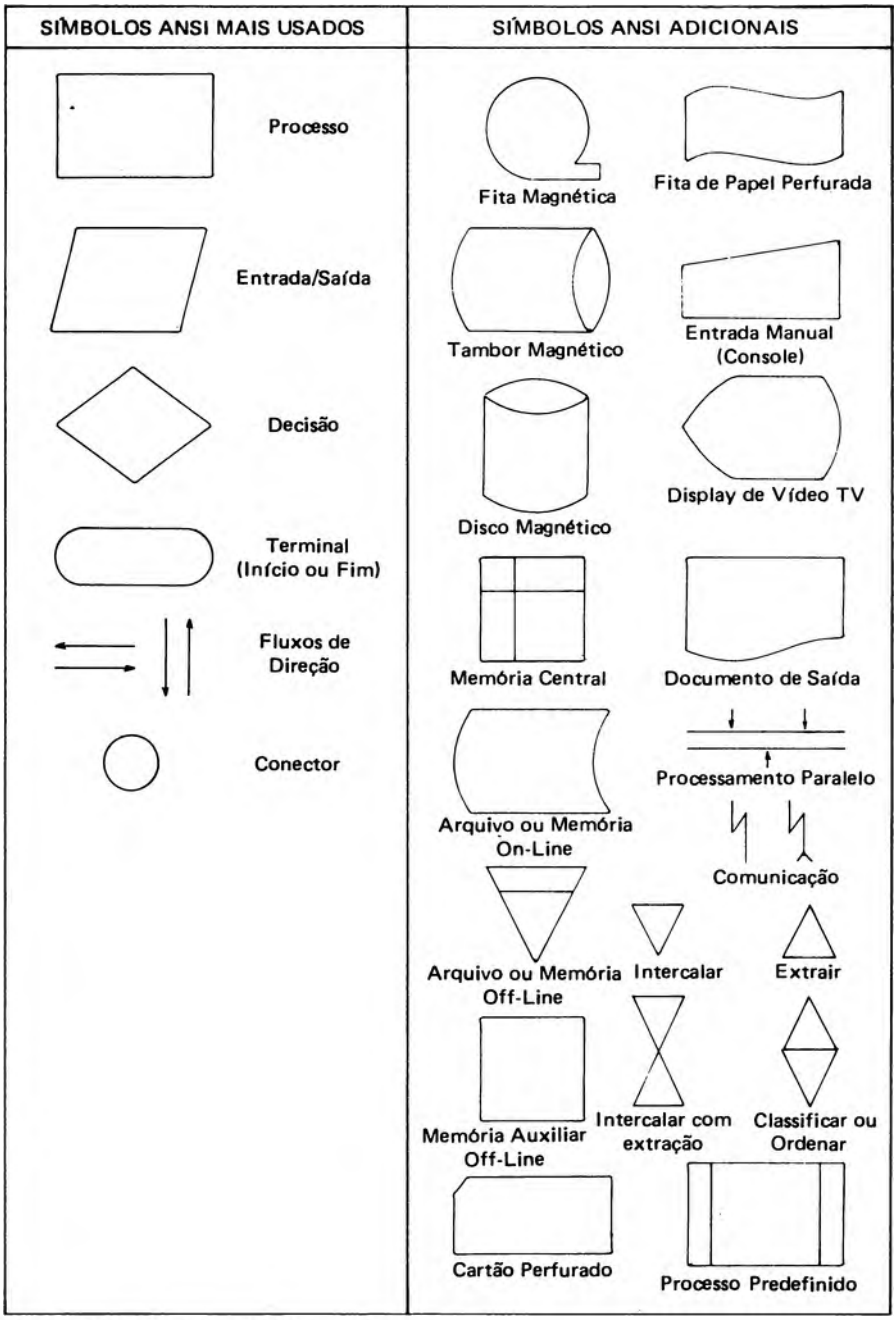
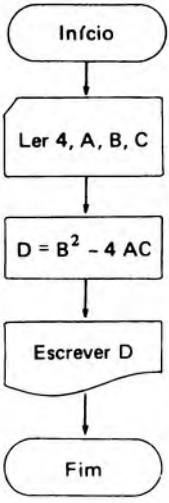


Figura 2.3. *Símbolos ANSI para fluxogramas.*

2.5. EXEMPLOS DE FLUXOGRAMAS

2.5.1. Exemplo 1

Fluxograma do problema: "Achar o valor da expressão $D = B^2 - 4\,A\,C$."

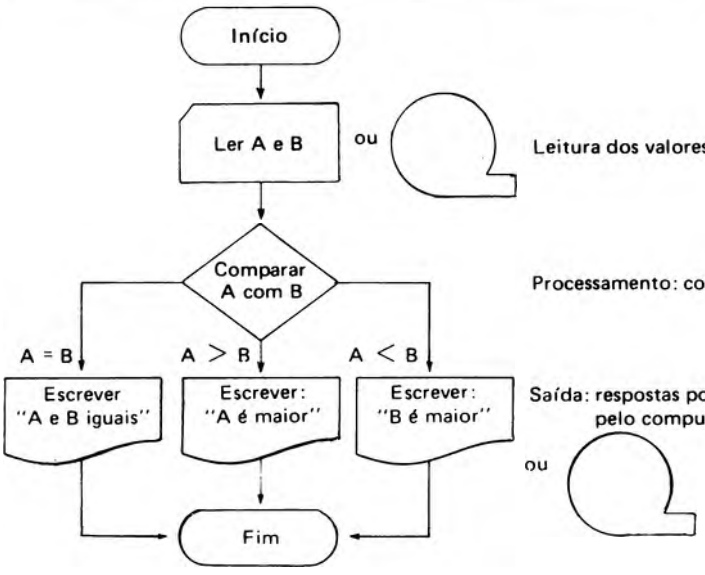


- Leitura dos valores dos elementos da fórmula (*) e que estão perfurados em um cartão.
- Processamento: cálculo da fórmula.
- Saída do resultado na impressora.

(*) O valor 4, por ser uma constante, não precisaria ser lido, pois as constantes podem ser usadas diretamente na expressão.

2.5.2. Exemplo 2

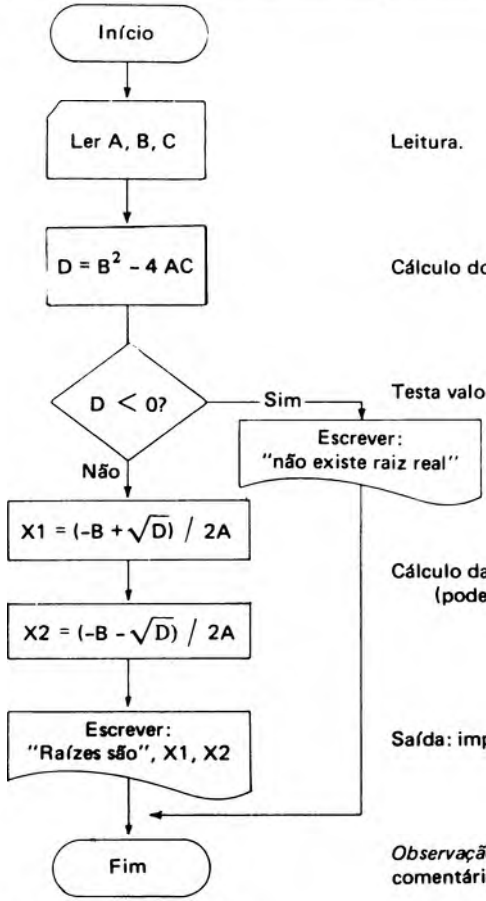
Fluxograma do problema: "Achar o maior de dois números A e B."



Observação: A leitura ou a saída de dados pode ser efetuada de diversos modos, através de unidades periféricas diferentes: leitora e perfuradora de cartão, de fita de papel, impressora, máquina de escrever etc. podendo, em cada caso, usar a simbologia correspondente. Usaremos, entretanto, o termo "Ler" e sinônimos para designar genericamente a Entrada de Dados e o termo "Escrever" e sinônimos para a Saída de Dados. No Capítulo seguinte, estudaremos a operação de Entrada e Saída de dados de um arquivo em programação comercial.

2.5.3. Exemplo 3

Fluxograma do Problema: "Achar as raízes da equação $ax^2 + bx + c = 0$."



Leitura.

Cálculo do valor do discriminante.

Testa valor de D.

Saída: em caso de raízes imaginárias.

Cálculo das duas raízes reais (podem ser iguais).

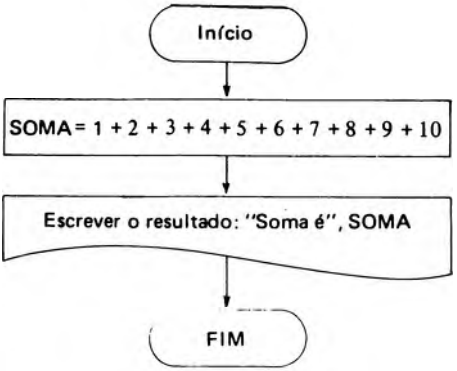
Saída: impressão dos resultados calculados.

Observação: os dizeres entre aspas (". . .") são comentários diretamente impressos pelo computador

2.5.4. Exemplo 4

Fluxograma do problema: "Calcular a soma dos 10 números inteiros que variam de 1 a 10, isto é, a soma dos números 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10."

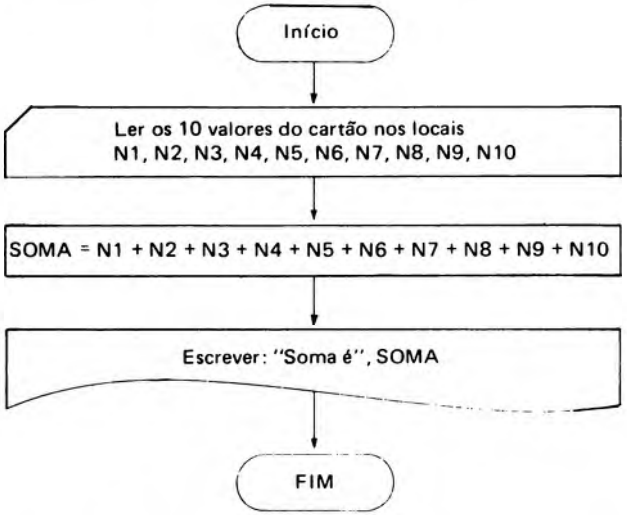
Como todos os valores são constantes e conhecidos, não é necessário efetuar a leitura dos mesmos. As constantes podem ser usadas diretamente no programa. O programa servirá apenas para achar a soma desses dez valores.



2.5.5. Exemplo 5

Fluxograma do Problema: "Calcular a soma de 10 números, todos perfurados em um único cartão."

Neste caso os valores devem ser lidos, pois não são conhecidos até serem lidos no cartão. Em compensação, o programa pode achar a soma de 10 valores quaisquer.

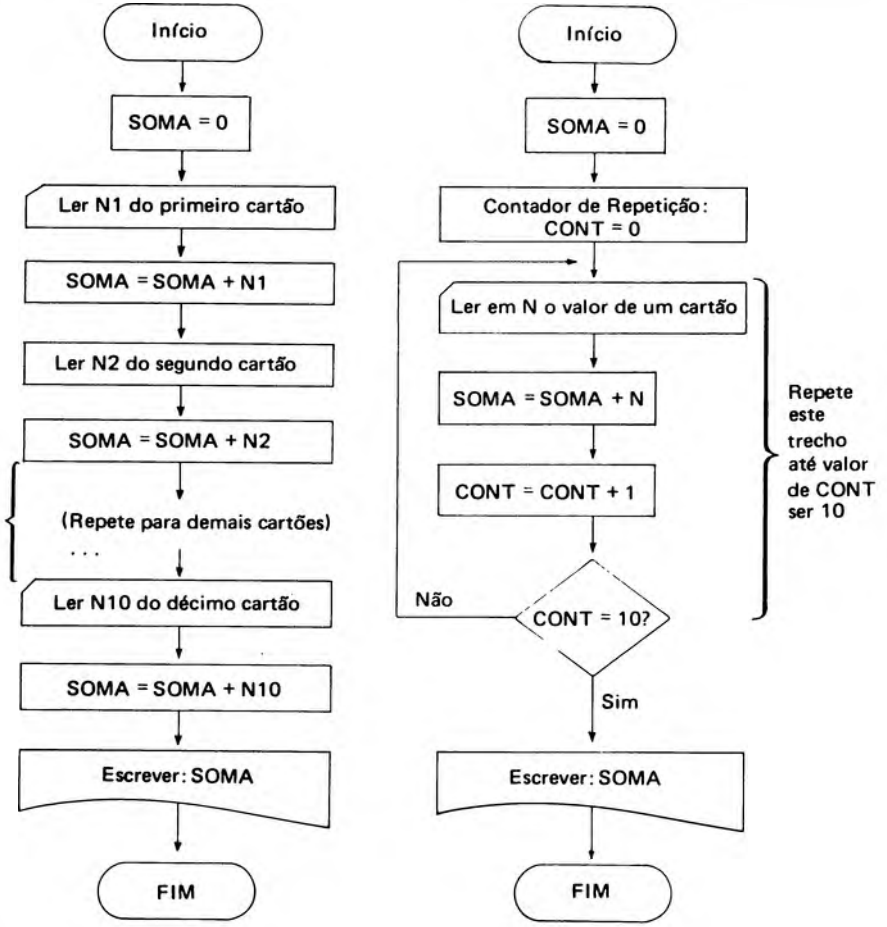


2.5.6. Exemplo 6

Fluxograma do problema: "Calcular a soma de dez números, cada qual perfurado em um cartão."

Neste caso, precisamos ler um cartão e somar o seu valor, repetindo o processo por dez vezes. Vamos examinar dois fluxogramas equivalentes: o pri-

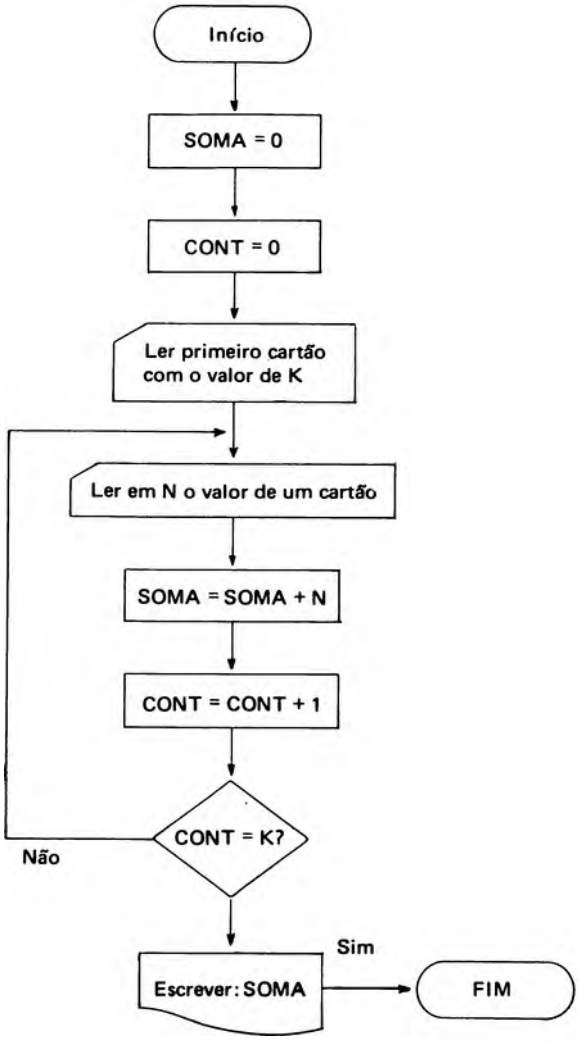
meiro, bem simples, é uma extensão ou repetição do exemplo anterior e o segundo, mais elegante, e mais compacto, repete o mesmo trecho do programa com o auxílio de um processo de contagem da repetição.



2.5.7. Exemplo 7

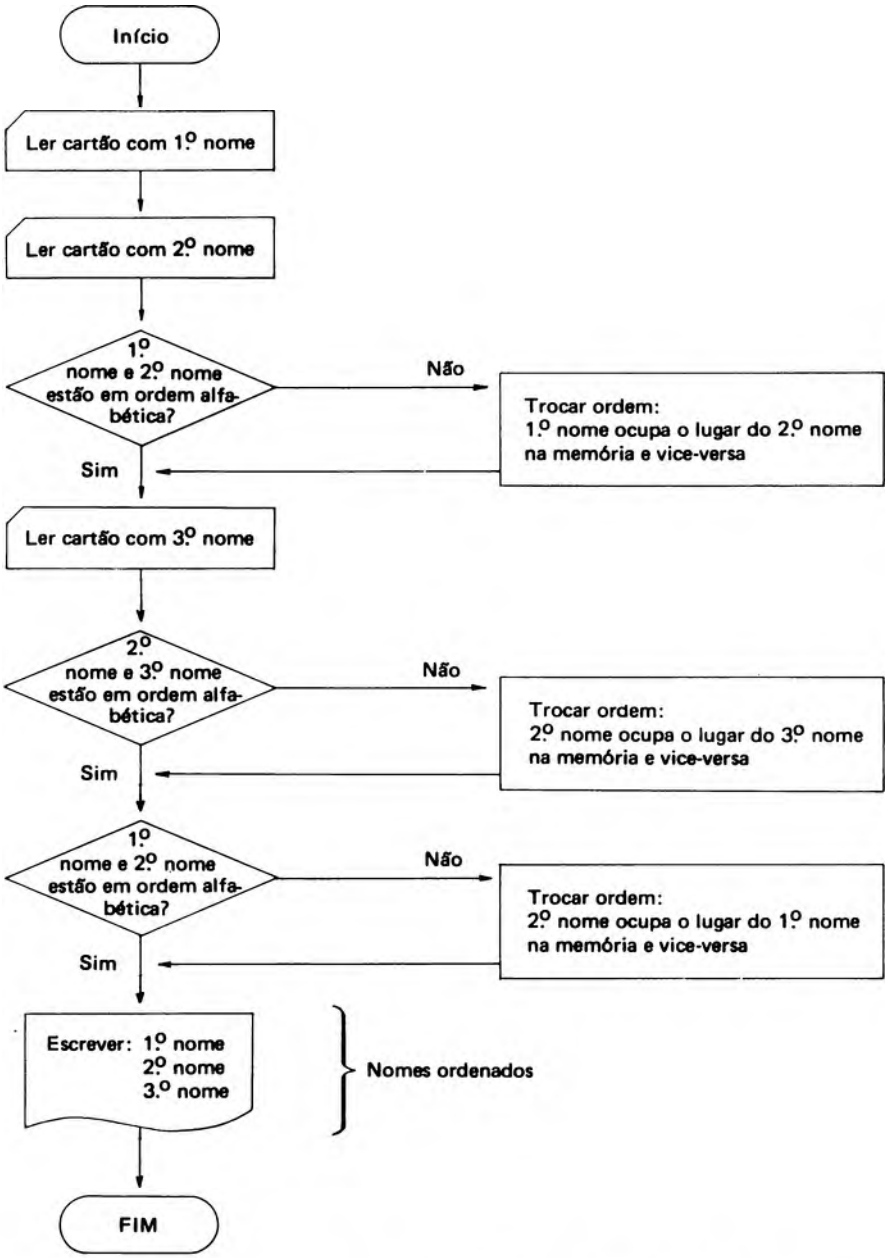
Fluxograma do problema: "Calcular a soma de K valores numéricos, cada qual perfurado em um cartão."

Neste caso, antes de ler os valores numéricos, precisamos saber o valor de K para saber se estamos lendo 10, 15, 50, 1.000 etc. cartões.



2.5.8. Exemplo 8

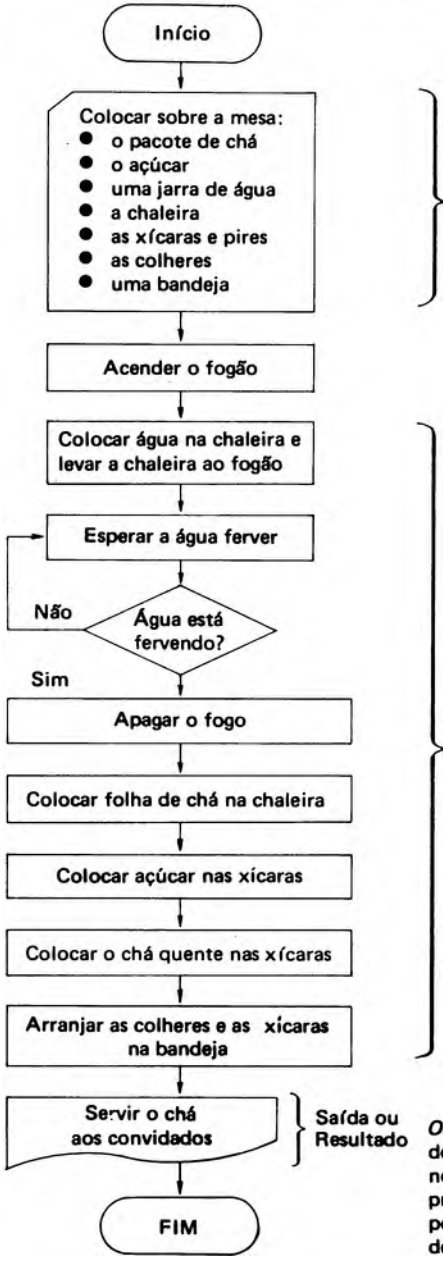
Fluxograma do programa: "Ler 3 cartões contendo, cada um, o nome de uma pessoa. Colocar os nomes em ordem alfabética e escrever na impressora."



Verificação: Testar o fluxograma com a seqüência de nomes *Sílvio*, *João* e *Antonio*.

2.5.9. Exemplo 9

Fluxograma do problema: "Preparar e servir chá aos convidados."



Observação: Este fluxograma ilustra a construção do raciocínio para executar uma atividade da nossa vida quotidiana. Entretanto, não pode ser programado e executado em um computador, pois este não possui a maioria das operações descritas.

EXERCÍCIOS DE RECAPITULAÇÃO

1. O que é programação?
2. O que é algoritmo?
3. O que é análise de sistemas?
4. O que é fluxograma?

EXERCÍCIOS DE APLICAÇÃO

Escrever um fluxograma para cada um dos problemas seguintes:

1. Calcular o perímetro de um triângulo sendo lidos os valores A,B e C dos seus lados.
2. Calcular a área do círculo e o comprimento da circunferência após ler o raio R. O valor π é uma constante igual a 3,1416.
3. Achar o maior valor dos três valores lidos A, B e C.
4. Ler 100 cartões cada um contendo um valor numérico. Calcular a soma dos números lidos que sejam maiores do que 25 e menores do que 125.
5. Uma pessoa vai conversar com o amigo pelo telefone.
6. Uma dona de casa leva sua lista de compras e vai ao supermercado.

Respostas: 1. Idêntico ao exemplo 1.
3. Idêntico ao exemplo 8.

3 PROGRAMAÇÃO COMERCIAL: OS ARQUIVOS, REGISTROS DE DADOS E FLUXOGRAMA COMERCIAL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

CARACTERÍSTICA DOS DADOS
- Variáveis Numéricas Reais ou Inteiras.
- Constantes
- Valores Alfanuméricos: Nome de pessoa, Data etc.

Diferença entre Valores Numéricos de Programa Científico e Comercial.

CAMPO OU ITEM DE UM DADO
- Nome do dado.
- Tipo: Numérico, alfabético e alfanumérico.
- Tamanho do dado.

O REGISTRO DE DADOS
- Conjunto determinado de campos ou itens de dado.

SUBCAMPOS OU SUBITENS DE UM REGISTRO: OS NÍVEIS HIERÁRQUICOS
- Exemplos

ARQUIVO DE DADOS
- *Exemplos:* Arquivo de Cartão, Arquivo em Fita Magnética, Disco Magnético etc.

ETIQUETAS OU REGISTROS DE INÍCIO E FIM DE ARQUIVO

OPERAÇÃO DE ENTRADA E SAÍDA COM OS DADOS DE UM ARQUIVO
- Operação READ e WRITE com valores (registros) de um arquivo.

FLUXOGRAMAS DE PROGRAMA COMERCIAL
- Exemplos.

Exercícios

3.1. CARACTERÍSTICAS DOS DADOS

Em problemas científicos que envolvem o uso de equações matemáticas, os dados envolvidos são as variáveis e constantes reais ou inteiras.

Em problemas comerciais, além dos valores reais ou inteiros, usam-se dados que se referem a:

- nome do cliente ou funcionário;
- endereço (nome da rua, número, nome do bairro, cidade, estado);
- código do departamento ou seção;
- valor do salário;
- código de estado civil;
- quantidade de mercadoria vendida ou estocada;
- número e valor do cheque;
- código de débito, crédito, câmbio, faturamento etc.

e outros tipos de dados de natureza comercial.

3.1.1. Valores numéricos em um problema científico e em um problema comercial

A diferença fundamental existente entre os dados manipulados em um problema científico e em um problema comercial é apresentada nos itens *a* e *b*.

a) *Em um problema científico*

Os dados podem variar praticamente de $-\infty$ a $+\infty$ (valores infinitos) e, além disso, são aceitos valores aproximados ou arredondados, para que os mesmos caibam em um tamanho fixo da palavra da memória, usando uma notação chamada PONTO FLUTUANTE. Por exemplo:

- Pode ocorrer valor da ordem de 3458×10^{34} ou -57899×10^{-39} etc.
- O valor da voltagem, 219,8 volts, pode ser aproximado para 220 volts.
- O valor 0,999999 pode ser aproximado para o valor 1,0 ou o valor 1,534579 para o valor 1,5346 etc., pois não acarretam alteração substancial no cálculo científico.

b) *Em um problema comercial*

Devem ser manipulados valores exatos que são, na maioria dos casos, valores monetários ou quantidades inteiras ou fracionárias com aproximação decimal exata. Por exemplo:

- o número do cheque, K-0357899, não pode ser confundido com o número K-0357900; ou
- a taxa de juros, 0,3457897, não pode ser aproximada para 0,346; ou
- o valor monetário, Cr$ 1.534.599,99, não pode ser aproximado para Cr$ 1.534.600,00 ou Cr$ 1.534.599,00 a não ser em casos legalmente permitidos. (Além disso, o arredondamento para Cr$ 1.534.599,00 não reduz o tamanho da memória ocupado pelo valor.)

Sendo assim, nos problemas comerciais, para cada tipo de dado ou valor, deve estar previsto e definido o tamanho de memória suficiente, de modo que não ocorra perda de algarismos ou aproximações que acarretem inexatidões nas listagens ou totalizações.

3.2. CAMPO OU ITEM DE UM DADO

Cada dado usado em um programa comercial forma um **Campo** ou **Item de Dado.** Um campo ou item de dado possui: nome, tipo, comprimento e valor ou conteúdo.

O **Nome** do campo é formado por letras, algarismos e hifens, formando, em geral, um significado alusivo ao seu conteúdo.

O **Tipo** do campo pode ser:

Numérico: quando contém algarismos e, eventualmente, um sinal + ou -.
Alfabético: quando contém somente letras do alfabeto e o caráter de espaço (branco).
Alfa-Numérico: quando contém letras, algarismos e sinais como §, &, % etc.

O **Comprimento** de um campo é o número máximo de elementos que cabem no mesmo. O espaço em branco também entra na contagem de comprimento e, geralmente, é indicado graficamente por "b".

Na memória de um computador, um campo numérico com 5 algarismos ocupa menos espaço do que um campo alfabético ou alfanumérico com 5 caracteres (mesmo que esses caracteres sejam algarismos), pois há diferença de representação binária entre caracteres numéricos e caracteres não numéricos.

EXEMPLOS:

NOME DO CAMPO	CONTEÚDO	TIPO E COMPRIMENTO
NUMERO-DE-CLIENTE	534001	Numérico com 6 algarismos
VALOR-DO-ITEM	-0015500	Numérico com comprimento 8
NOME-DO-FUNCIONARIO	JOSEbDAbSILVAbbbbbbbb	Alfabético, tamanho 20
VALOR-LIQUIDO	Cr$ 135.500,00	Alfanumérico, tamanho 13.

3.3. O REGISTRO DE DADOS

Um **Registro de Dados,** ou simplesmente **Registro,** é formado por diversos campos de dados que possuem certa finalidade para ficarem agrupados. Por exemplo:

a) Um **Registro de Funcionário** para folha de pagamento de uma firma poderia ser formado pelos seguintes campos:
   - número do funcionário;
   - nome do funcionário;
   - endereço;
   - código da seção ou departamento;
   - código da função ou cargo;
   - código do estado civil;
   - data de nascimento;
   - salário mensal ou salário-base.

b) Um **Registro de Controle de Estoques** de mercadorias poderia conter os campos:
   - código da mercadoria;
   - nome da mercadoria;
   - quantidade existente;
   - quantidade mínima que deve permanecer em estoque;
   - nome e endereço do fornecedor;
   - valor unitário da mercadoria.

Na prática, um registro corresponde a uma ficha completa ou a um cartão perfurado ou a uma linha da impressora etc. O registro também recebe um **Nome** através do qual podemos chamar ou referir ao seu conteúdo. O tamanho do registro é dado pela soma dos comprimentos dos campos que o formam. O registro possui TIPO somente se todos os campos forem do mesmo tipo.

3.4. SUBCAMPOS OU SUBITENS DE UM REGISTRO: OS NÍVEIS HIERÁRQUICOS

Para facilitar a manipulação, podemos considerar os campos de um registro subdivididos em subcampos ou subitens que, por sua vez, estão subdivididos em novos subcampos.

Dessa forma, desejando usar apenas uma parte de um campo, basta referir a essa parte através do nome atribuído ao subcampo. Na verdade, um registro é um supercampo subdividido em campos de dados.

É comum, em programação comercial, atribuir-se **Número de Nível** aos campos e subcampos, de um registro, a partir do valor 01, 02, 03 etc. Desse modo, podemos usar livremente qualquer subcampo, através do respectivo nome, sabendo em que posição hierárquica do registro o mesmo se encontra (ver Fig. 3.1).

3.5. ARQUIVOS DE DADOS

Em um serviço comercial, geralmente, existe grande quantidade de dados repetidos, sendo que cada dado está em forma de registro.

Em uma empresa, por exemplo, existem:

- um conjunto de fichas de registro de funcionários;
- um conjunto de fichas de mercadorias;
- um conjunto de fichas ou registro de clientes;
  e assim por diante.

O conjunto de certo tipo de registros forma um **Arquivo.**

Os registros que formam um arquivo de dados podem ser de nome, tamanho e tipo diferentes: por exemplo, no meio do arquivo de mercadorias, para cada 100 registros de mercadorias, pode existir um registro de subtotais desse grupo, e que é um registro diferente do registro de mercadorias.

Na prática, porém, para facilitar a manipulação, um arquivo é, normalmente, formado por repetição de um mesmo tipo de registro. Sendo assim:

- O conjunto de registro de funcionários forma um arquivo que pode ser chamado de **Arquivo-de-Funcionário.**
- O conjunto de registros de mercadorias pode formar um arquivo chamado de **Arquivo-de-Mercadorias,** e assim em diante.

Um arquivo pode estar em forma de fichas, cartão perfurado, fita magnética, disco magnético ou linhas de impressão em formulários.

Um arquivo é dito **Arquivo de Entrada** se for usado como entrada de dados em um programa, e é dito **Arquivo de Saída** se for formado por resultados obtidos por operação de um programa. Um arquivo de saída pode, eventualmente, tornar-se um arquivo de entrada para outro programa.

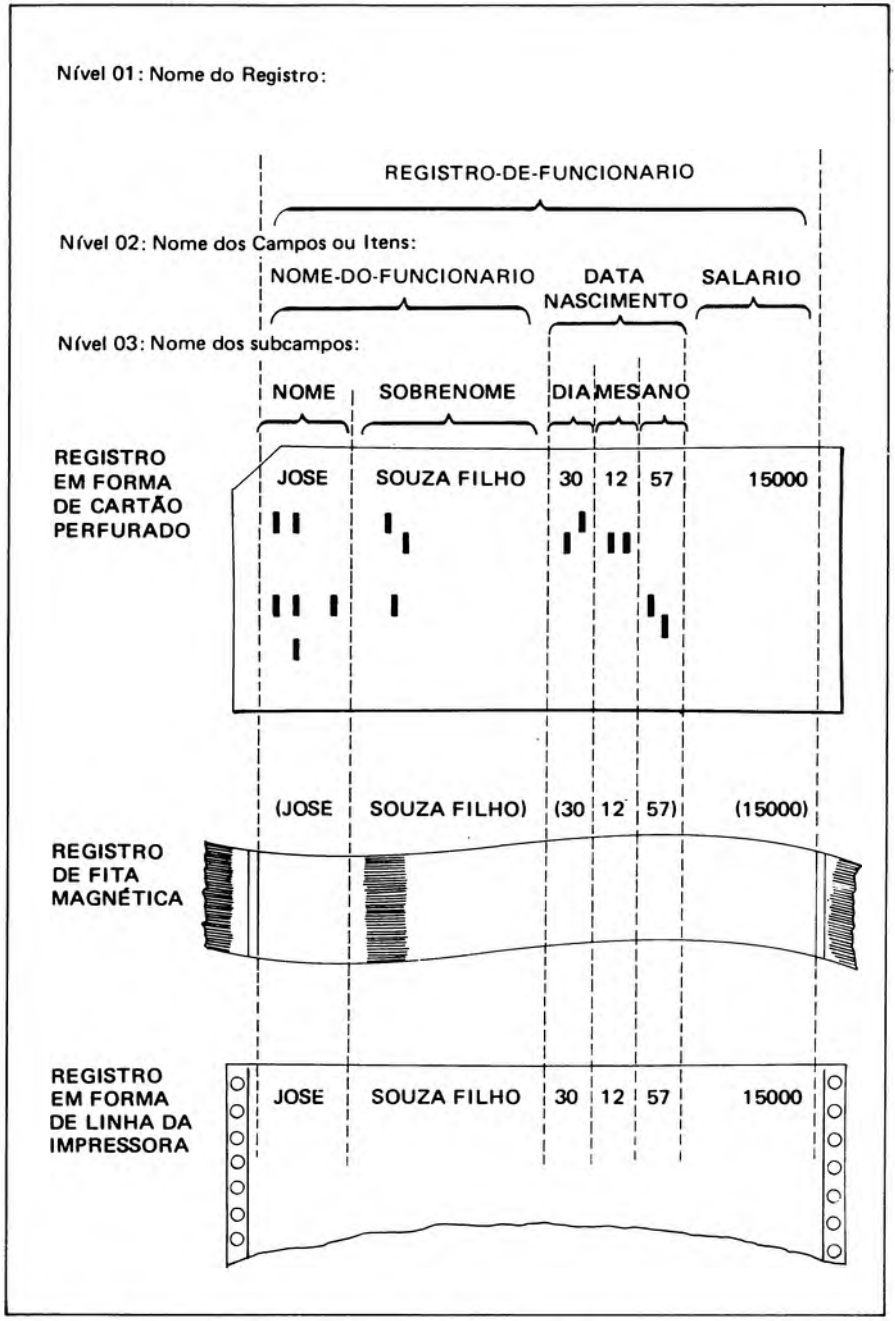


Figura 3.1. *Exemplos de registros e níveis hierárquicos.*

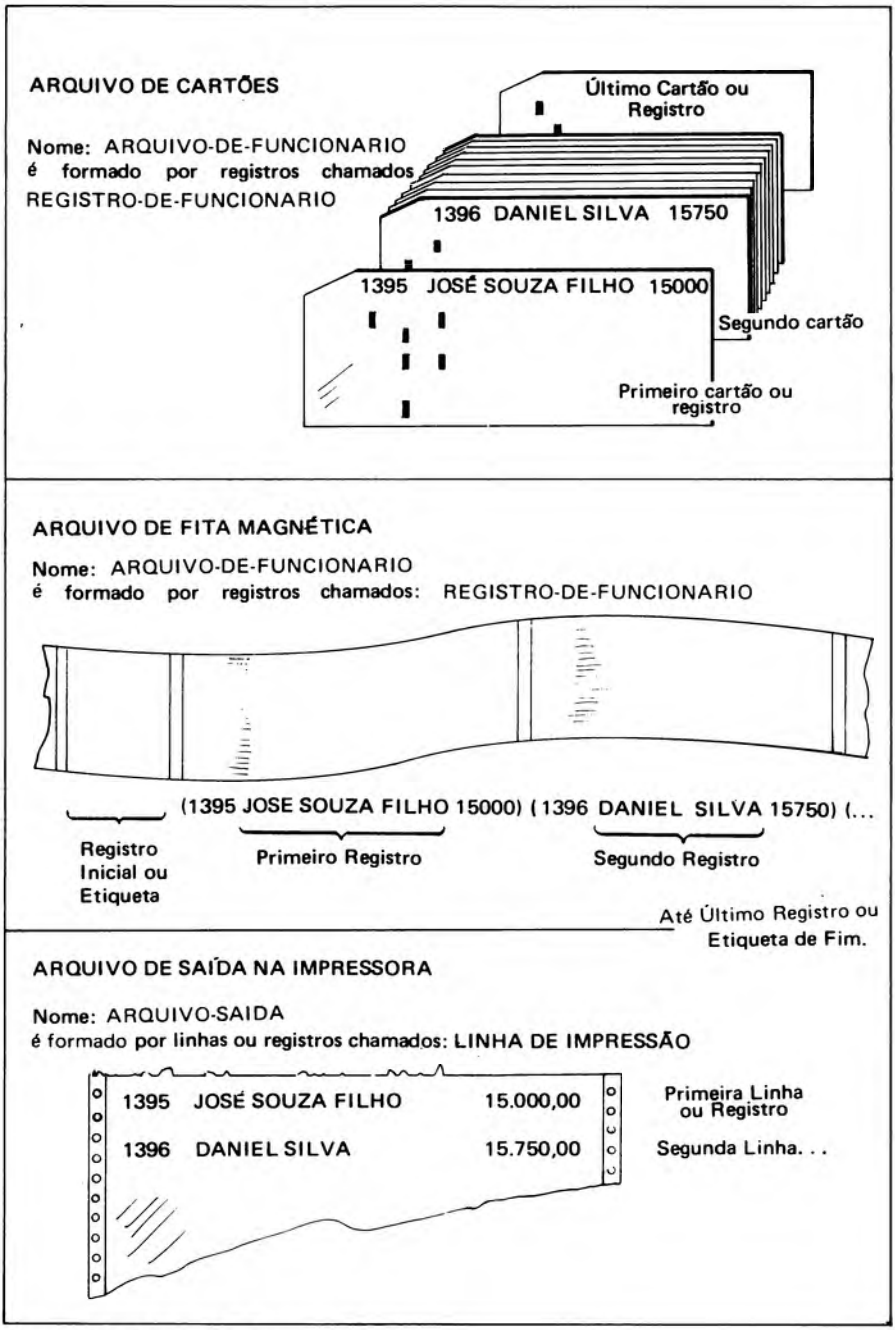


Figura 3.2. *Exemplos de arquivos.*

3.6. ETIQUETAS OU REGISTROS DE INÍCIO E DE FIM DE ARQUIVO

O primeiro e o último registro de um arquivo possuem forma especial e servem para indicar, respectivamente, o início e o fim desse arquivo. São chamados de **Labels** ou **Etiquetas** de início e de fim de arquivo.

Em arquivos de fita magnética ou disco magnético, essas etiquetas contêm dados indicadores do nome, tipo, data e uso do arquivo.

Nos arquivos formados por cartões perfurados, em geral, só existe a etiqueta de fim de arquivo, formada por um cartão chamado **Cartão-Sentinela** ou **Último Cartão** e que contém sinal indicativo de fim de arquivo.

A definição e o uso das etiquetas de início e fim de arquivo dependem das normas adotadas pelo centro de computação.

3.7. OPERAÇÃO DE ENTRADA E DE SAÍDA COM OS DADOS DE UM ARQUIVO

Na linguagem COBOL, que estudaremos a partir do capítulo seguinte, a operação de leitura de dados de um arquivo é efetuada pelo comando

READ- "nome do arquivo"

que colocará, cada vez que for executado, o conteúdo de um registro desse arquivo no local da memória, contendo o nome desse registro.

Na linguagem PL/I essa operação é efetuada pelo comando

READ FILE (nome do arquivo) INTO (nome do registro na memória)

A operação de saída, que consiste em levar o conteúdo de um registro que está na memória para o arquivo de saída, é executada pelos comandos:

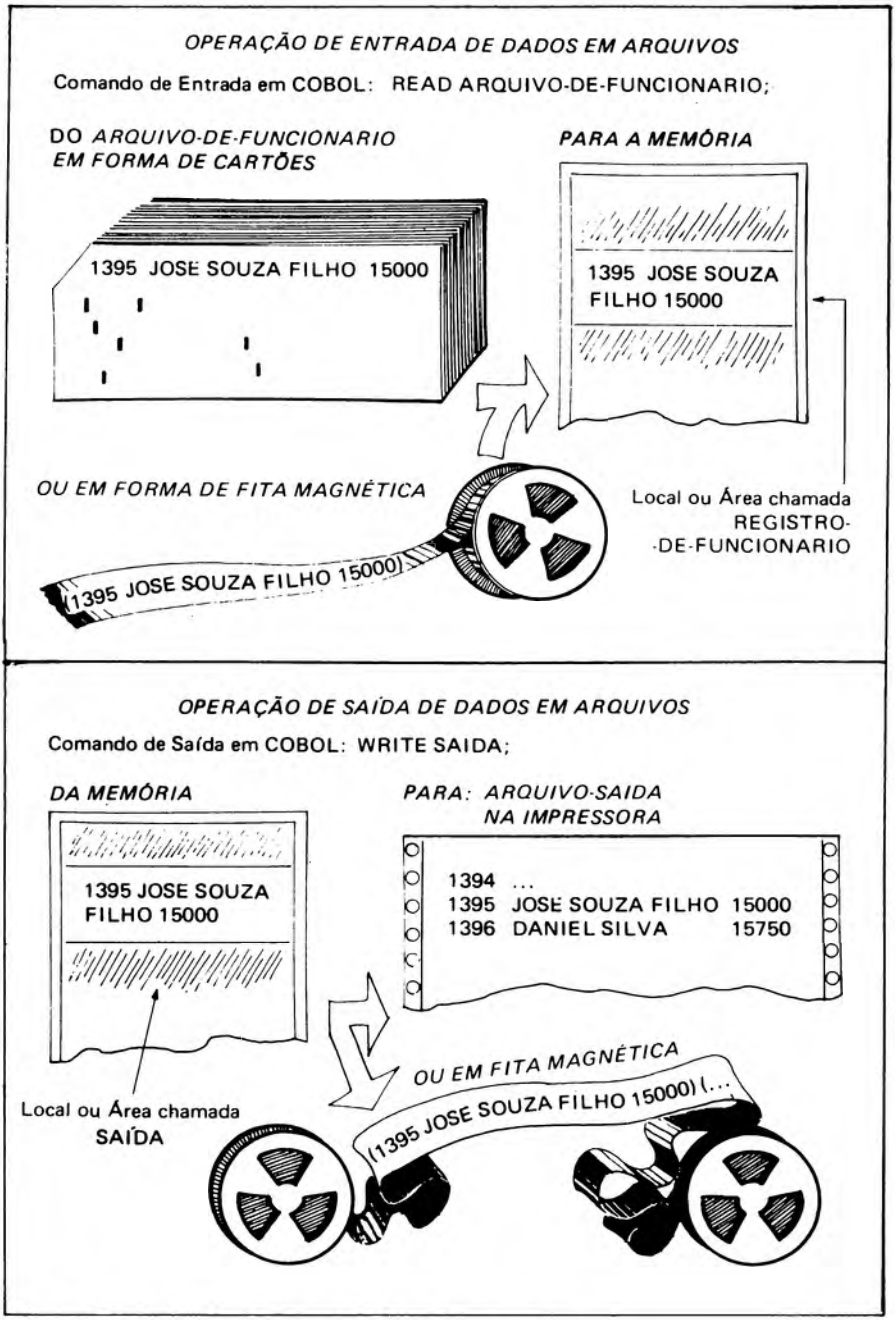
WRITE "nome do registro na memória" em linguagem COBOL e
WRITE FILE (nome do arquivo) FROM (nome do registro na memória) em PL/I.

Cada vez que for executado, o comando WRITE colocará, em um registro do arquivo de saída (linha de impressão ou fita magnética), o conteúdo da memória que está no local definido pelo "nome de registro na memória".

3.8. EXEMPLOS DE FLUXOGRAMA DE PROGRAMA COMERCIAL

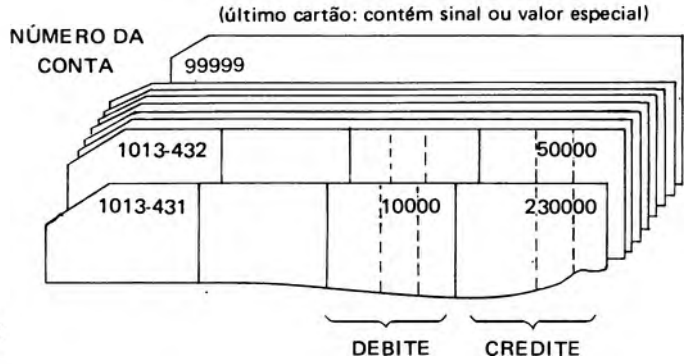
3.8.1. Exemplo 1

Seja um arquivo formado por um conjunto de cartões perfurados, cada cartão contendo os valores de NÚMERO DA CONTA, DEBITE e CREDITE, exceto o último cartão, que possui um valor identificador de fim de arquivo. Escrever um fluxograma que leia cada um dos cartões e imprima o seu conteúdo em uma linha de impressão.

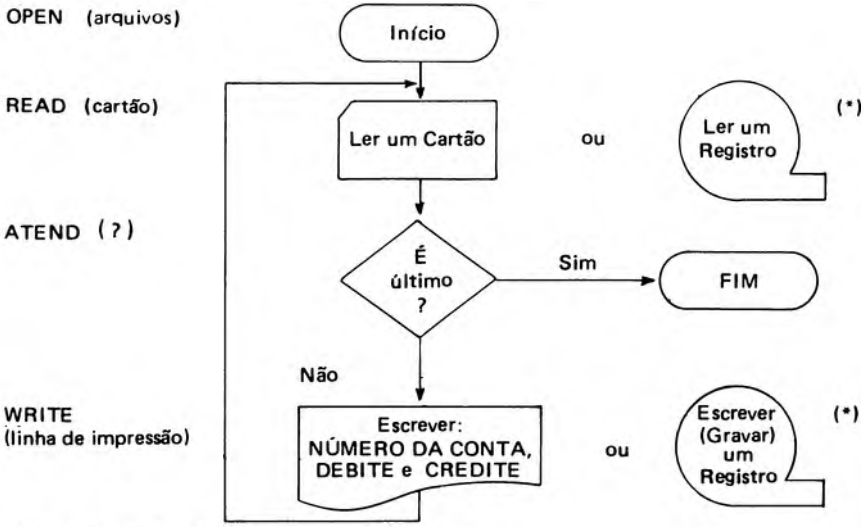


58 Figura 3.3. *Operação de entrada e saída com arquivos.*

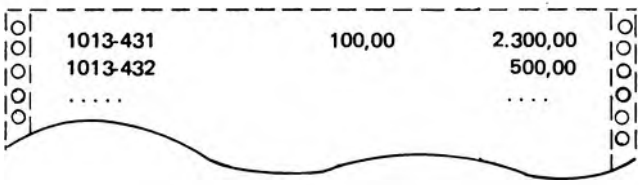
CARTÃO PERFURADO:



FLUXOGRAMA E COMANDOS



FOLHA DA IMPRESSORA:



Observação: A vírgula (,) e o ponto (.) dos valores de DEBITE e CREDITE e que não constavam nos cartões podem ser colocados através do programa COBOL. (Edição por PICTURE)

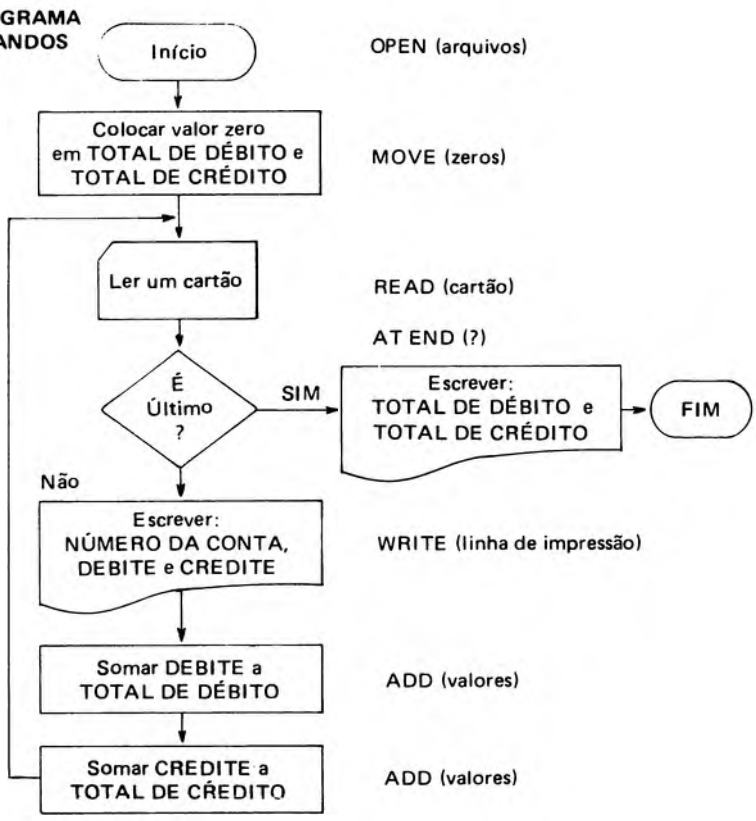
* Tanto o arquivo de entrada como o de saída poderiam ser formados por **Fita Magnética.** A observação é válida para os exemplos seguintes.
Os comandos são formas incompletas dos comandos COBOL e que estudaremos a partir dos capítulos seguintes.

3.8.2. Exemplo 2

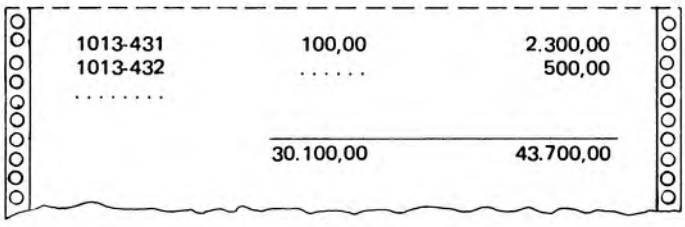
Seja o mesmo problema do exemplo um. Mas agora devemos calcular o total de todos os valores DEBITE e o total de todos os valores CREDITE, imprimindo-os após a leitura e impressão do último cartão do arquivo. Devemos então, guardar os valores totais em dois locais distintos da memória e chamados TOTAL DE DÉBITO e TOTAL DE CRÉDITO.

CARTÃO PERFURADO: (os mesmos do Exemplo Um)

FLUXOGRAMA E COMANDOS



FOLHA DA IMPRESSORA:



3.8.3. Exemplo 3

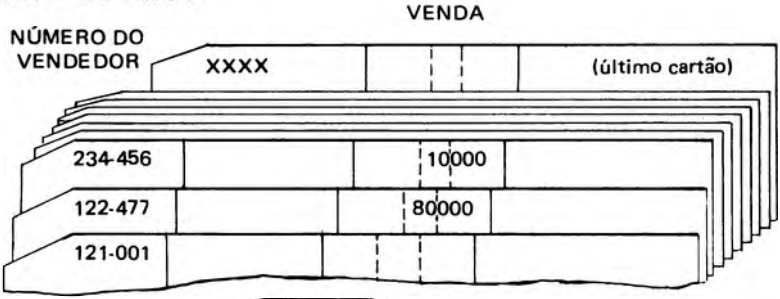
Seja um arquivo de cartões onde cada cartão contém o valor do NÚMERO DO VENDEDOR e da VENDA realizada.

A tabela da comissão a ser dada ao vendedor é:

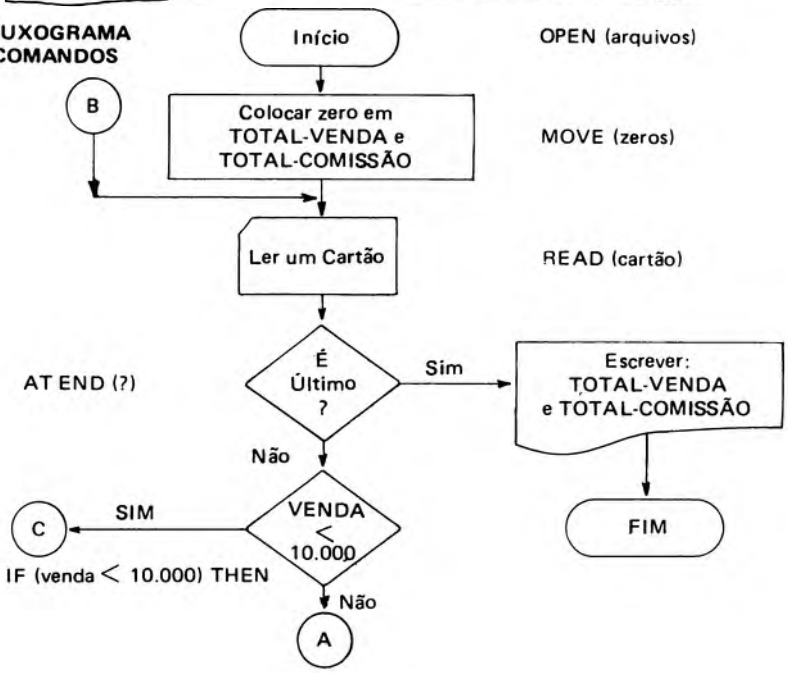
Vendas inferiores a $ 10.000	:	Comissão de 0%
Vendas entre $ 10.000 e $ 100.000	:	Comissão de 5%
Vendas superiores a $ 100.000	:	Comissão de 10%

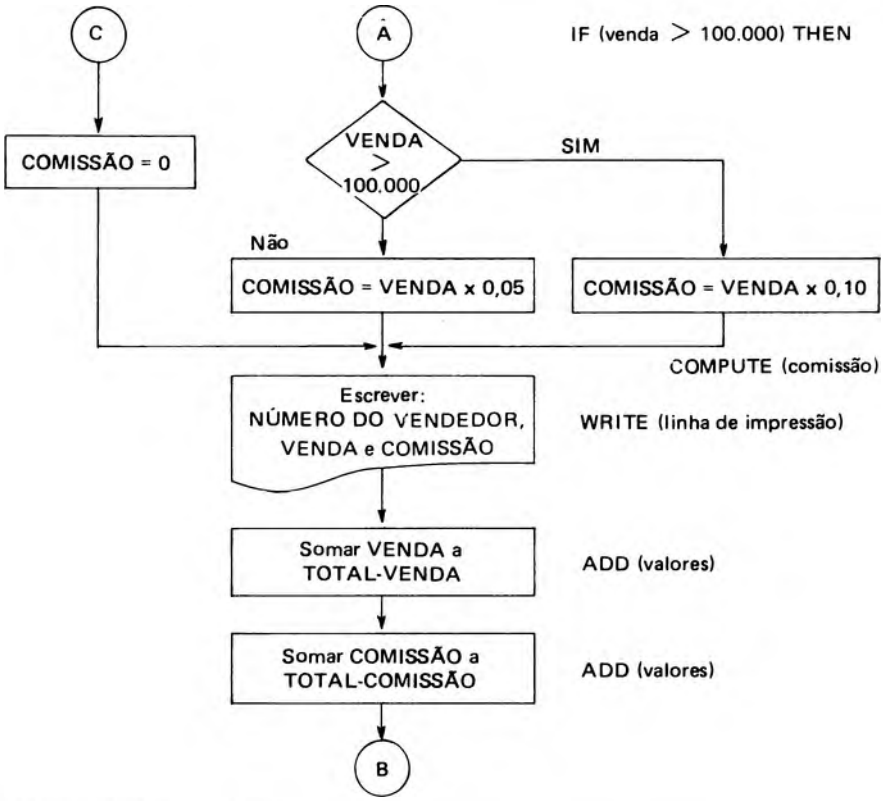
Escrever em cada linha da impressora o NÚMERO DO VENDEDOR, a VENDA e o valor da COMISSÃO calculada. Após leitura do último cartão, escrever os totais das VENDAS e das COMISSÕES.

CARTÃO PERFURADO:

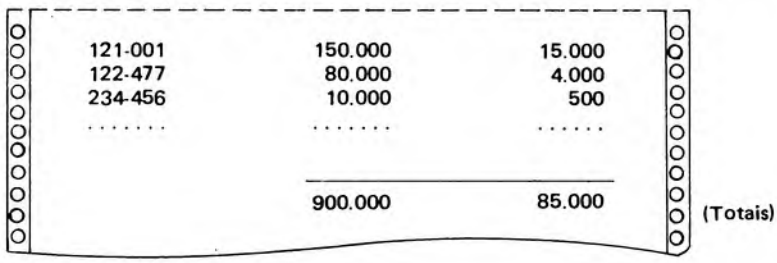


FLUXOGRAMA E COMANDOS





FOLHA DA IMPRESSORA:



EXERCÍCIOS DE RECAPITULAÇÃO

1. O que é um registro de dados? Dar exemplos.
2. O que é um arquivo de dados? Dar exemplos.
3. O que é um arquivo de entrada e um arquivo de saída e para que servem?
4. O que são Níveis e subcampos de um registro de dados? Dar exemplos.

EXERCÍCIOS DE APLICAÇÃO

Qual seria o registro, e quais seriam os campos e subcampos de cada registro que forma um arquivo de dados referente às informações seguintes? Dar nome, tamanho e tipo dos arquivos, registros e campos, tanto em forma de cartão perfurado como em fita magnética.

1. Arquivo do catálogo de livros de uma biblioteca.
2. Arquivo dos dados de registro de alunos de uma escola.
3. Arquivo contendo os dados dos Títulos Eleitorais de determinado distrito eleitoral.
4. Arquivo de veículos licenciados em uma Delegacia de Trânsito.

Sobre Fluxogramas

5. Em um arquivo de cartões perfurados cada cartão contém: NÚMERO DO FUNCIONÁRIO, HORAS-TRABALHADAS e SALÁRIO-HORA. Escrever um fluxograma que calcule: SALÁRIO = HORAS-TRABALHADAS x SALÁRIO-HORA e liste pela impressora: NÚMERO DO FUNCIONÁRIO, HORAS-TRABALHADAS, SALÁRIO-HORA e SALÁRIO. Totalizar o valor de SALÁRIO.
6. Escrever um fluxograma que lê e lista pela impressora os registros (dados) do arquivo definido pelo Exercício de Aplicação 1 deste capítulo.
7. Escrever um fluxograma que, usando o arquivo definido pelo Exercício 3 deste capítulo, forneça a lista do nome de eleitores com mais de 60 anos de idade.
8. Reescrever os fluxogramas dos Exercícios 5, 6 e 7 para arquivos em Fita Magnética. Desenhar os registros e subcampos de todos os arquivos.

Resposta: 1) Registro Nível

```
01: FICHA-DO-LIVRO
```

Subcampo Nível

```
02: TITULO
02: AUTOR
    03 NOME-DO-AUTOR
    03 SOBRENOME-DO-AUTOR
02: EDITORA
02: ANO-DE-EDIÇÃO
02: CODIGO-DE-CLASSIFICAÇÃO
02: OBSERVAÇÕES
```

4 INTRODUÇÃO À PROGRAMAÇÃO EM LINGUAGEM COBOL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

COBOL: LINGUAGEM PARA PROGRAMAS COMERCIAIS
(COBOL: Common Business Oriented Language)
As quatro divisões de um programa COBOL:

- Identification Division
- Environment Division
- Data Division
- Procedure Division

Folha de Codificação COBOL.

- Margem A: para nomes de divisões, parágrafos, arquivos etc.
- Margem B: para cláusulas de descrição e comandos.

COBOL VS. BASIC.
EXEMPLO DO PROGRAMA SIMPLES
EXEMPLO DA IDENTIFICATION DIVISION E ENVIRONMENT DIVISION

- Identificação: dar nome e observações do programa
- Environment Division: definir os equipamentos e as condições especiais do sistema usado.

EXEMPLO DO PROCEDURE DIVISION – PROGRAMA COBOL-1

- Contém os comandos de execução.

  Nome variáveis e Palavras-chave.
  Constantes Numéricas e Constantes Figurativas.
  Principais comandos do COBOL:

  MOVE, ADD, SUBTRACT, MULTIPLY, DIVIDE, GOTO, IF, READ, WRITE, OPEN e CLOSE

PARA QUE SERVE O DATA DIVISION

- Definir: Arquivos, Registros e itens de dado, Áreas de trabalhos e tabela constantes.

Exemplo de DATA DIVISION – PROGRAMA COBOL - 1
Algumas especificações do DATA DIVISION

- As cláusulas PICTURE, FILLER e VALUE.

EXEMPLO DE PROGRAMA COBOL-2
RESUMO DE UM PROGRAMA COBOL

Exercícios

4.1. COBOL: LINGUAGEM PARA PROBLEMAS COMERCIAIS

O COBOL (iniciais de Common Business Oriented Language) é uma linguagem de programação destinada para processamento de serviços comerciais. Por essa razão possui maiores facilidades para definir e manipular nome de pessoas, códigos, formação de arquivos, edição de valores monetários, elaboração de relatórios, – emissão de cheques, notas fiscais etc

Para entender o conceito de arquivo e registros de dados de serviço comercial, recomenda-se a leitura do capítulo anterior.

Neste capítulo procuraremos apenas *exemplificar* os conceitos e as instruções sobre o COBOL, sem preocupação em analisar detalhes.

4.1.1. As quatro divisões de um programa COBOL

Um programa escrito em COBOL é obrigatoriamente formado por quatro partes denominadas DIVISÕES, e que aparecem na seguinte ordem:

```
Margem
A    B
-------------------------------------------------------------------
IDENTIFICATION DIVISION.
     (Contém frases ou cláusulas de identificação do programa)
ENVIRONMENT DIVISION.
     (Contém frases ou cláusulas de definição dos equipamentos usados)
DATA DIVISION.
     (Contém frases ou cláusulas de definição e especificação dos arquivos de dados, registros de dados e áreas de trabalho)
PROCEDURE DIVISION.
     (Contém os parágrafos, sentenças e comandos de execução)
-------------------------------------------------------------------
```

4.1.2. A folha de codificação COBOL

A folha de codificação COBOL oferece uma forma relativamente livre de ser usada e preenchida, bastando observar as seguintes restrições:

COLUNAS 1 a 6 – NUMERAÇÃO DA SEQÜÊNCIA DA LINHA
Pode ser preenchida apenas por dígitos ou por espaços em branco: esta numeração não interfere no programa.

COLUNA 7 – INDICAÇÃO DE CONTINUAÇÃO DE UM VALOR CONSTANTE NÃO NUMÉRICO
Coloca-se um hífen quando um valor constante ou palavra não coube na linha anterior. Não se trata de continuação de um comando ou cláusula (como no FORTRAN), pois podemos continuar por várias linhas só terminando com um PONTO FINAL.

COLUNAS 8 a 72 – INSTRUÇÕES DA LINGUAGEM COBOL

COLUNAS 73 a 80 – CÓDIGO DE IDENTIFICAÇÃO DO PROGRAMA
Uso opcional e que pode aparecer no topo da página de listagem do programa fonte.

MARGEM A – COLUNAS 8 a 11 – NOME DE DIVISÕES, SECÇÕES, PARÁGRAFOS
Os nomes acima e os números de níveis 01, 77 e 88, FD devem começar nesta margem.

MARGEM B – COLUNAS 12 a 72 – COMANDOS E CLÁUSULAS
Os comandos, cláusulas e os números de níveis em geral, devem começar a partir da coluna 12 de cada linha.

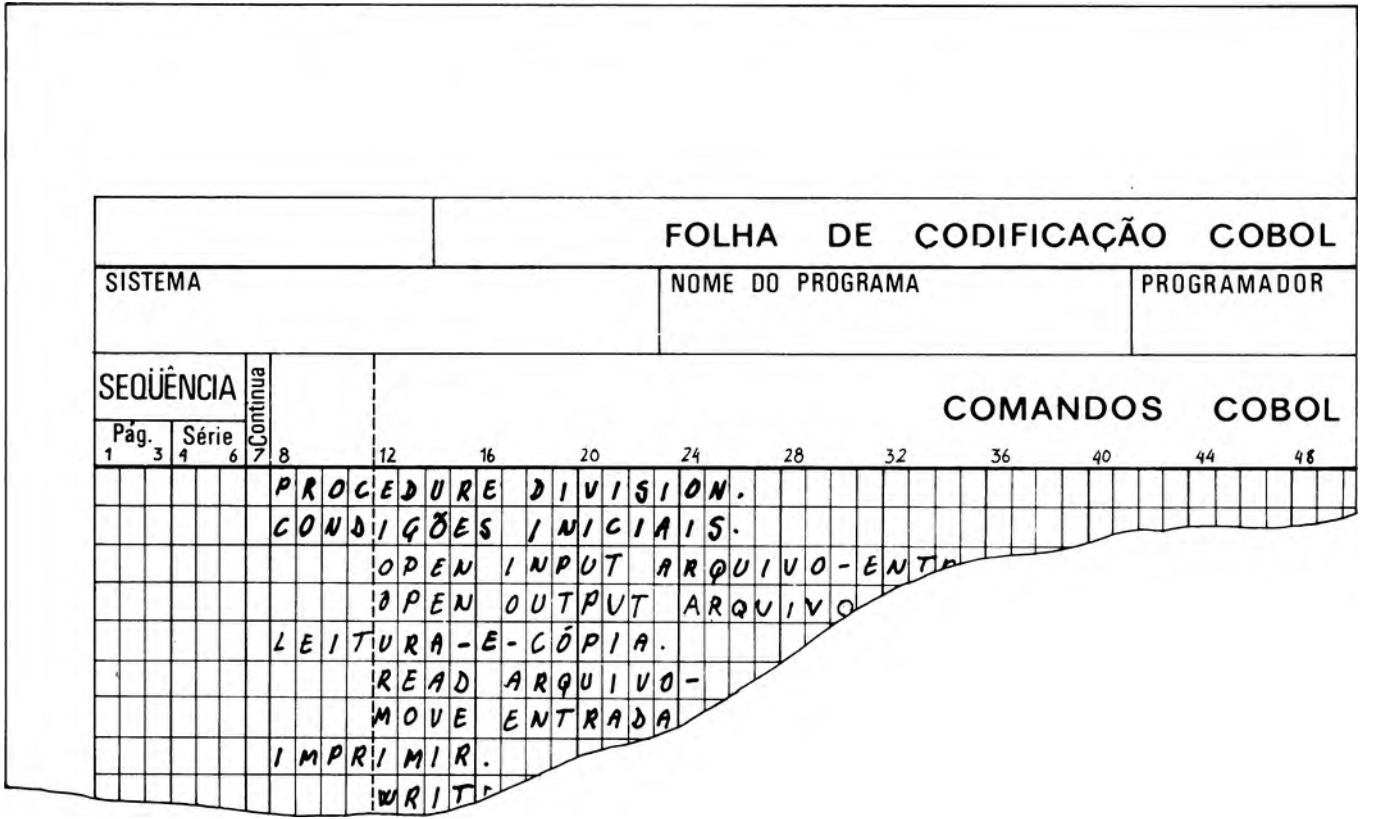


Figura 4.1. *Folha de codificação COBOL.*

4.2. COBOL VS. BASIC: EXEMPLO DE PROGRAMA SIMPLES

Um dos maiores argumentos em favor da linguagem como o BASIC ou FORTRAN (e recentemente o VISICALC) reside na sua simplicidade de programação, sem necessidade de treinamento prolongado. Praticamente ao mesmo tempo que recebe aulas sobre os primeiros comandos Basic ou Fortran, o programador consegue escrever e processar o seu próprio programa.

Com o COBOL já não acontece o mesmo, pois o programador deve sempre estudar e levar em conta alternativas sobre os inúmeros comandos, cláusulas e divisões disponíveis. O COBOL é uma linguagem poderosa e deve ser usada com cuidado e planejamento.

Entretanto, para quebrar esse preconceito, vamos mostrar de modo comparativo com o Basic que a linguagem COBOL também pode ser usada em programas simples, como na Figura 4.2.

```
PROGRAMA COBOL SIMPLES: ESCREVER PELA IMPRESSORA O NOME E
ENDEREÇO DE UMA PESSOA
POR EXEMPLO:          JOSE J. SILVA
                      RUA DOS PARDAIS, 34XX
                      CEP 0000 – S. PAULO – SP
```

4.2.1. Versão 1: Programa em BASIC

O programa em Basic seria simplesmente o seguinte:

```
NEW
05 REM PROGRAMA SIMPLES
10 PRINT "JOSE J. SILVA"
20 PRINT "RUA DOS PARDAIS, 34XX"
30 PRINT "CEP 0000 – S.PAULO – SP"
40 END
50 RUN
```

4.2.2. Versão 2: Programa em COBOL usando a impressora

```
Margem  A     B
        IDENTIFICATION DIVISION.
        PROGRAM-ID. PROGRAMA SIMPLES.              (nome do programa)

        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
           SELECT ARQUIVO-NOME ASSIGN TO PRINTER   (define o uso
                                                    da Impressora)
```

continua

continuação

A B

```
DATA DIVISION.
FILE SECTION.
FD ARQUIVO-NOME  LABEL RECORDS ARE OMITTED.
01   NOME-ENDEREÇO PICTURE X (25).        (define linha a ser
                                              usada na impressão)

PROCEDURE DIVISION.
ESCREVER-NOME.                            (define nome do pará-
                                              grafo)
    OPEN OUTPUT ARQUIVO-NOME.             (abre arquivo de saída)
    MOVE 'JOSE J. SILVA' TO NOME-ENDEREÇO.
    WRITE NOME-ENDEREÇO.                  (escreve nome)
    MOVE 'RUA DOS PARDAIS, 34XX' TO NOME-ENDEREÇO.
    WRITE NOME-ENDEREÇO.                  (escreve nome da rua)
    MOVE 'CEP 0000–S.PAULO –SP' TO NOME-ENDEREÇO.
    WRITE NOME-ENDEREÇO.                  (escreve CEP e cidade)
    CLOSE ARQUIVO-NOME.                   (fecha arquivo)
    STOP RUN.
```

a) PROGRAMA EM BASIC

```
NEW
05 REM PROGRAMA SIMPLES
10 PRINT "JOSE J. SILVA"
20 PRINT "RUA DOS PARDAIS, 34XX"
30 PRINT "CEP 0000 – S.PAULO -SP"
40 END
RUN
```

b) PROGRAMA EM COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROGRAMA SIMPLES
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL
SELECT ARQUIVO-NOME ASSIGN TO
        PRINTER.

DATA DIVISION.

FILE SECTION.

FD ARQUIVO-NOME LABEL RECORDS
   ARE OMITTED.

01 NOME-ENDEREÇO PICTURE X(25).

PROCEDURE DIVISION.

ESCREVER-NOME.

OPEN OUTPUT ARQUIVO-NOME.

MOVE 'JOSE J. SILVA'
   TO NOME-ENDEREÇO.
WRITE NOME-ENDEREÇO.
MOVE 'RUA DOS PARDAIS, 34XX'
   TO NOME-ENDEREÇO.
WRITE NOME-ENDEREÇO.
MOVE 'CEP 0000 – S. PAULO - SP'
   TO NOME-ENDEREÇO.
WRITE NOME-ENDEREÇO.
CLOSE ARQUIVO-NOME.
STOP RUN.
```

Figura 4.2. *Programa simples COBOL versus BASIC para escrever nome e endereço.*

O programa tornou-se mais longo, pois requer o uso de comandos adicionais como OPEN, MOVE, e as demais divisões. Entretanto, é um programa simples, claro e perfeitamente executável.

4.2.3. Versão 3: Programa COBOL usando máquina do console

Uma versão mais parecida com o Basic seria utilizar-se o terminal ou máquina de escrever do console e assim teríamos:

```
MARGEM   A      B
-------------------------------------------------------------
    IDENTIFICATION DIVISION.
    PROGRAMA-ID. PROGRAMA SIMPLES.
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
       (não é usada pelo programa, mas deve-se colocar alguma descrição mínima exigi-
       da pelo Manual COBOL em uso)
    DATA DIVISION.
    FILE SECTION.
       (mesma observação do ENVIRONMENT DIVISION)

    PROCEDURE DIVISION.
    ESCREVER-NOME.
       DISPLAY 'JOSE J. SILVA' UPON CONSOLE.
       DISPLAY 'RUA DOS PARDAIS, 34XX' UPON CONSOLE.
       DISPLAY 'CEP 0000 – S. PAULO – SP' UPON CONSOLE.
       STOP RUN.
-------------------------------------------------------------
```

Esta versão do programa em COBOL é *desaconselhada* devido às seguintes razões:

- o uso do terminal do CONSOLE (Máquina de escrever) através do comando DISPLAY é em geral reservado para fornecer mensagens de erros de processamento e é proibido para fornecer resultado de processamento normal;
- a ausência de uso de arquivo de entrada/saída normal (através da Leitora de Cartões, Impressora, Fita Magnética e Disco) deixa o preenchimento da ENVIRONMENT DIVISION e DATA DIVISION dependente da versão da linguagem COBOL que está sendo usada;
- alguns sistemas ou versões da linguagem COBOL podem simplesmente deixar de processar esta versão do programa.

4.3. EXEMPLO DO IDENTIFICATION DIVISION E DO ENVIRONMENT DIVISION

As duas primeiras divisões de um Programa COBOL contêm dados padronizados exigidos pelo centro de programação e não oferecem dificuldades ao programador.

Por exemplo, podemos ter:

Margem	A	B	
	IDEN	TIFICATION DIVISON.	
	PROG	RAM-ID.	(nome do programa)
	AUTH	OR. SEBASTIAO-JOSE-SILVA.	(nome do autor)
	INST	ALLATION. FIRMA-ABC.	(nome da empresa)
	DATE	WRITTEN. 31/DEZ/19XX.	(Data)
	REMA	RKS. PROGRAMA EXEMPLO PARA LER UM CARTAO E IMPRIMIR PELA IMPRESSORA.	(Observações)
	ENVI	RONMENT DIVISION.	
	CONF	IGURATION SECTION.	
	SOUR	CE COMPUTER.	
		COMPUTADOR-LMN/MOD.007.	(código do sistema usado na compilação e execução)
	OBJE	CT COMPUTER.	
		COMPUTADOR-LMN/MOD.007.	
	INPUT	– OUTPUT SECTION.	
	FILE	– CONTROL.	(relaciona arquivos aos equipamentos de entrada e saída).
		SELECT ARQUIVO-ENTRADA	
		ASSIGN TO CARD-READER.	
		SELECT ARQUIVO-SAIDA	
		ASSIGN TO PRINTER.	

4.4. EXEMPLO DE PROCEDURE DIVISION – PROGRAMA COBOL - 1

O programador deverá preocupar-se somente com a organização e preenchimento do DATA DIVISION e do PROCEDURE DIVISION, uma vez que as duas primeiras divisões contêm somente dados de rotina.

Por razões didáticas, embora o DATA DIVISON sempre apareça antes do PROCEDURE DIVISION em um programa, estudaremos esta última divisão antes.

Os comandos de execução ou operação devem aparecer nesta divisão. Os comandos são agrupados em **Parágrafos** que possuem os respectivos nomes.

Vamos estudar um programa COBOL mais elaborado, visualizado na Figura 4.3.

PROGRAMA COBOL -1: "Ler um lote de cartões perfurados e escrever o conteúdo de cada cartão em uma linha da impressora."

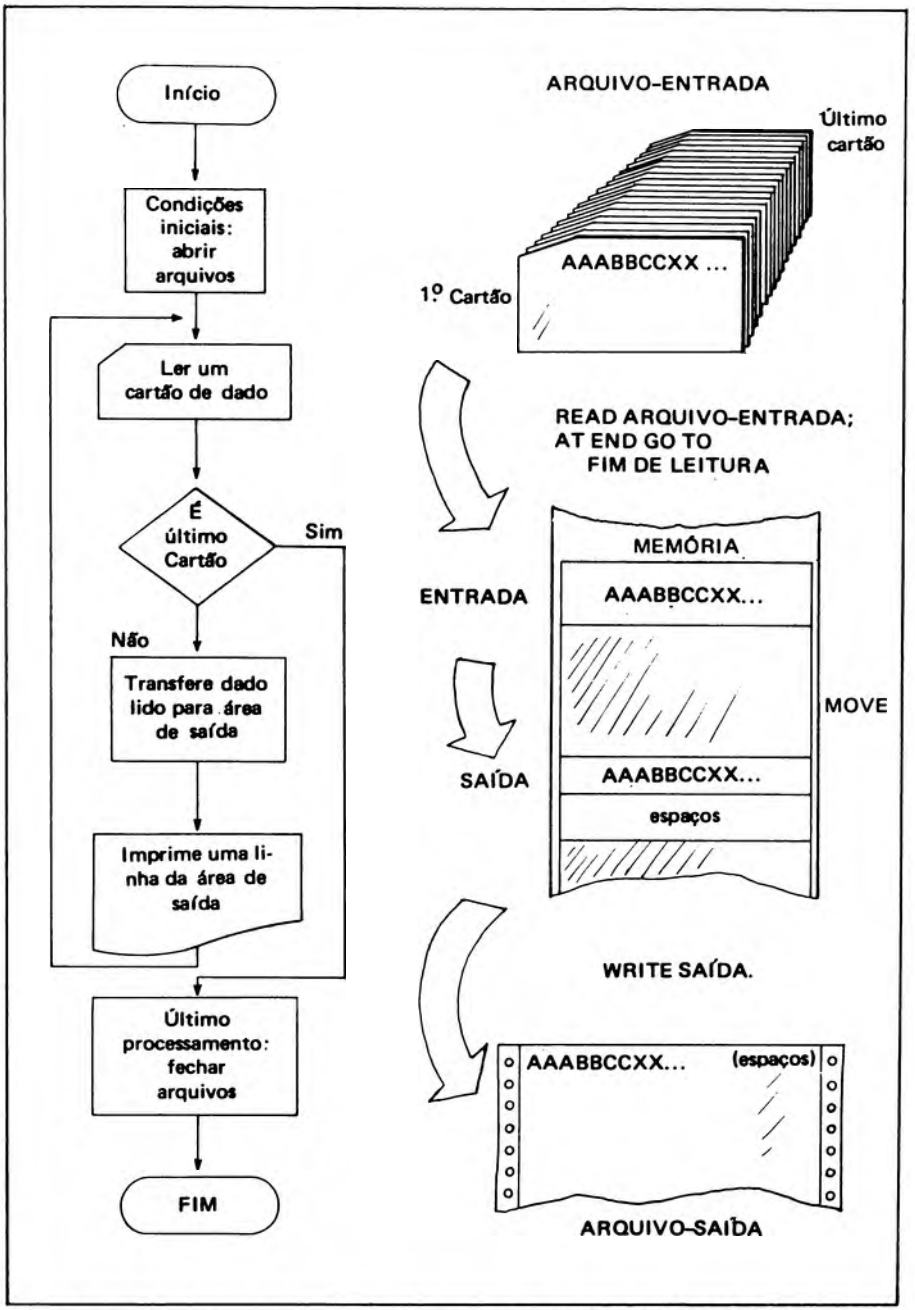


Figura 4.3. *Fluxograma e exemplo de operação do PROCEDURE DIVISION do programa que lê e imprime cartões.*

COMANDO COBOL		EXPLICAÇÃO
Margem A	Margem B	(Comparar com o fluxograma da Figura 4.3)
PROC	EDURE DIVISION.	• Título da divisão.
COND	ICOES-INICIAIS.	• Nome do *primeiro parágrafo.*
	OPEN INPUT ARQUIVO-ENTRADA.	• Preparar um arquivo chamado ARQUIVO-ENTRADA e que será usado pelo comando READ.
	OPEN OUTPUT ARQUIVO-SAIDA.	• Preparar um arquivo chamado ARQUIVO-SAIDA a ser usado pelo comando WRITE.
LEIT	URA-E-COPIA	• Nome do *segundo parágrafo.*
	READ ARQUIVO-ENTRADA AT END GO TO FIM-DE-LEITURA.	• Ler um cartão do ARQUIVO-ENTRADA; se for último cartão desviar para o parágrafo FIM-DE-LEITURA; se for cartão comum colocar o conteúdo no local - chamado ENTRADA na memória e definido pelo DATA DIVISION.
	MOVE ENTRADA TO COPIA-SAIDA.	• Transfere o conteúdo do local ENTRADA para o local chamado COPIA-SAIDA.
IMPR	IMIR.	• Nome do *terceiro parágrafo.*
	WRITE SAIDA.	• Imprime o conteúdo do local chamado SAIDA (definida no DATA DIVISION) em uma linha do ARQUIVO-SAIDA que está associado à SAIDA.
	GO TO LEITURA-E-COPIA.	• Volta para ler e copiar novo cartão.
FIM-	DE-LEITURA.	• Nome do último parágrafo.
	CLOSE ARQUIVO-ENTRADA.	• Operação para fechar arquivo usado.
	CLOSE ARQUIVO-SAIDA.	• Idem.
	STOP RUN.	• Fim da execução.

Observação: O ponto final deve ser sempre colocado. O uso de vírgula e ponto e vírgula é opcional.

4.4.1. Nomes de variáveis e as palavras-chave

Palavras-chave são palavras usadas para identificar os comandos — tais como OPEN, MOVE, READ etc. ou completá-los tais como AT END, TO etc. Não se deve usar variáveis, ou parágrafos com os nomes das palavras-chave.

Os nomes de **Variáveis** são usados para identificar os arquivos, registros, os campos, os subcampos e os parágrafos, e são formados por até 30 caracteres que podem ser letras, algarismos, ou sinais especiais.

Exemplos: ARQUIVO-ENTRADA
NOME-DO-FUNCIONÁRIO
ITEM-X345

Todos os nomes, exceto os nomes de parágrafos, devem estar previamente definidos no DATA DIVISION.

4.4.2. Constantes numéricas e constantes figurativas

Os valores constantes tais como 100, 1.55, -54 etc. podem ser usados diretamente nos comandos sem necessidade de definição no DATA DIVISION.

Somente valores constantes aos quais se quer atribuir nomes devem ser definidos no DATA DIVISION.

Os valores numéricos ou sinais especiais de uso freqüente já estão predefinidos na linguagem e podem ser diretamente usados por seus nomes, sem necessidade de definição no DATA DIVISION.

Exemplos: MOVE ZEROS TO AREA-UM. - a AREA-UM será preenchida com valores zeros.
MOVE SPACES TO AREA-DOIS. - idem com espaços em branco.

4.4.3. Principais comandos em COBOL

MOVE Transfere o valor da primeira variável ou constante para a segunda variável.

Exemplos:

MOVE DADO-1 TO DADO-2.

ANTES DA EXECUÇÃO		DEPOIS
DADO-1	5	5
DADO-2	-3	5

MOVE 100 TO VALOR.
MOVE ZERO TO PARCELA.

VALOR	54	100
PARCELA	33	0

ADD Soma valor da primeira variável ou constante com o valor da segunda variável e o resultado fica na segunda variável. É possível colocar o resultado em uma terceira variável usando-se o termo GIVING.

Exemplos:

ADD A TO B.

ANTES		DEPOIS
A	11	11
B	6	17

ADD A, B GIVING C.

A	11	11
B	6	6
C	5	17

SUBTRACT Efetua subtração de modo análogo à ADD.

Exemplos:

SUBTRACT A FROM B

ANTES		DEPOIS
A	5	5
B	8	3

SUBTRACT A FROM B GIVING C.

A	5	5
B	8	8
C	19	3

MULTIPLY Exemplos:

MULTIPLY X BY Y.

ANTES		DEPOIS
X	2	2
Y	5	10

MULTIPLY X BY 5 GIVING W.

X	3	3
W	50	15

DIVIDE Exemplos: (Divide o valor de TOTAL por X e coloca resultado em MÉDIA)

DIVIDE X INTO TOTAL GIVING MEDIA.

X	5	5
TOTAL	20	20
MEDIA	10	4

GO TO — Desvia o programa para o início do parágrafo citado.
Exemplo: GO TO PARAGRAFO-UM.

IF — Compara dois valores: se a comparação for verdadeira então executa o comando que vem junto com o comando IF; caso a comparação for falsa o programa ignora este segundo comando e prossegue normalmente com o comando seguinte.

Exemplos:

IF PRICE IS LESS THAN CUSTO MOVE PRICE TO AREA-1.

(AREA-1 receberá o valor de PRICE só se PRICE for menor do que valor de CUSTO)

IF CODIGO IS EQUAL TO 3 GO TO PARAGRAFO-CODIGO-3

(o programa desvia para PARAGRAFO-CODIGO-3 se o valor de CODIGO for igual a 3).

Os termos de comparação que são usados no comando IF são:

IS GREATER THAN	IS NOT GREATER THAN
IS LESS THAN	IS NOT LESS THAN
IS EQUAL TO	IS NOT EQUAL TO
IS UNEQUAL TO	

READ — Serve para ler o conteúdo de um registro completo que está no arquivo de entrada cujo nome foi citado no comando e colocá-lo em uma área da memória com nome associado a esse arquivo pelo DATA DIVISION.

Exemplos:

READ ARQUIVO-DE-ENTRADA AT END GO TO PARAGRAFO-FINAL.
READ ARQUIVO-DOIS.

WRITE — Serve para escrever os dados contidos em uma área de memória cujo nome foi citado no comando, para o arquivo de saída associado a essa área.

Exemplos:

WRITE ENDEREÇO.
WRITE VALOR-TOTAL AFTER ADVANCING 4 LINES (Escrever o conteúdo de VALOR-TOTAL após avançar 4 linhas a posição do formulário da Impressora)

Observação: Ilustrações das operações dos comandos READ e WRITE podem ser encontradas no capítulo anterior.

OPEN e CLOSE — Estes comandos servem para abrir (OPEN) ou fechar (CLOSE) qualquer arquivo usado em um comando de entrada (READ) ou de saída (WRITE). Nos arquivos de entrada, o ato de abrir um arquivo significa a verificação automática dos códigos de identificação e de segurança escritos na etiqueta inicial do arquivo; e o ato de fechar significa verificar os códigos do último cartão ou etiqueta de fim de arquivo, além de colocar a unidade de leitura na posição inicial. Nos arquivos de saída, abrir arquivo significa gerar o cabeçalho ou etiqueta de início e fechar o arquivo significa gerar a etiqueta de fim de arquivo.

COMPUTE — Usado para calcular fórmulas ou expressões aritméticas mais complexas através das operações (+, -, *, /) para soma, subtração, multiplicação e divisão, respectivamente.
Exemplos:
COMPUTE X = 55.0.
COMPUTE VALOR = JUROS * SALDO – DESCONTO.

4.5. PARA QUE SERVE O DATA DIVISION?

O DATA DIVISION tem as seguintes utilidades:

- Define o nome, tamanho e tipo de todos os *arquivos de entrada ou de saída* usados no programa pelos comandos READ ou WRITE. Cada arquivo é definido com nível FD.
- Define o nome, tamanho, tipo e demais características dos locais de memória onde cada *registro dos arquivos de entrada ou de saída* irão ser lidos ou formados. Os registros possuem um número de nível 01.
- Define o nome, tipo, tamanho e demais características dos *campos e subcampos* de cada registro de um arquivo, atribuindo número de nível 02, 03, 04 etc... A definição dos arquivos, local dos registros, campos e subcampos na memória é feita em uma seção chamada FILE SECTION.
- Define o nome e as características de todas as *demais áreas* e *variáveis* usadas na memória pelos comandos do PROCEDURE DIVISION. Estas áreas e variáveis formam uma área de trabalho do programa e são definidas na parte denominada WORKING-STORAGE SECTION do DATA DIVISION.

4.5.1. Exemplo de preenchimento do DATA DIVISION — Programa COBOL - 1

Os comandos de definição de dados do DATA DIVISION equivalem, em parte, às especificações usadas no comando FORMAT da linguagem FORTRAN.

EXEMPLO: "DATA DIVISION do programa para ler um cartão e escrever o seu conteúdo em uma linha da impressora." (O mesmo usado para exemplificar o PROCEDURE DIVISION.)

Margem A	Margem B — COMANDOS COBOL	EXPLICAÇÃO (FIG. 4.4)
DATA	DIVISION.	• Título da divisão.
FILE	SECTION.	• Título da seção de definição de arquivos de entrada e de saída.
FD	ARQUIVO-ENTRADA; LABEL RECORDS ARE OMITTED; DATA RECORD IS ENTRADA.	• Nome do arquivo de entrada, especificando que cada registro ou cartão lido será colocado na área de memória chamada ENTRADA.
01	ENTRADA PICTURE IS X (80).	• O campo chamado ENTRADA consegue guardar até 80 caracteres alfanuméricos, isto é, o conteúdo de um cartão.
FD	ARQUIVO-SAIDA; LABEL RECORDS ARE OMITTED; DATA RECORD IS SAIDA.	• Nome do arquivo de saída, especificando que cada registro ou conteúdo de uma linha de impressão será formado no local chamado SAIDA.
01	SAIDA.	• SAIDA está subidividido em dois subcampos.
	02 COPIA-SAIDA PICTURE IS X (80).	• Subcampo de 80 caracteres para receber a cópia do cartão lido.
	02 FILLER PICTURE IS X (40).	• Subcampo composto de 40 espaços em branco para completar as 120 posições de uma linha de impressão.
WORK	ING-STORAGE SECTION	• Título da seção de definição de campos de trabalho que não foram usados neste programa, mas é necessário escrever o título da seção.

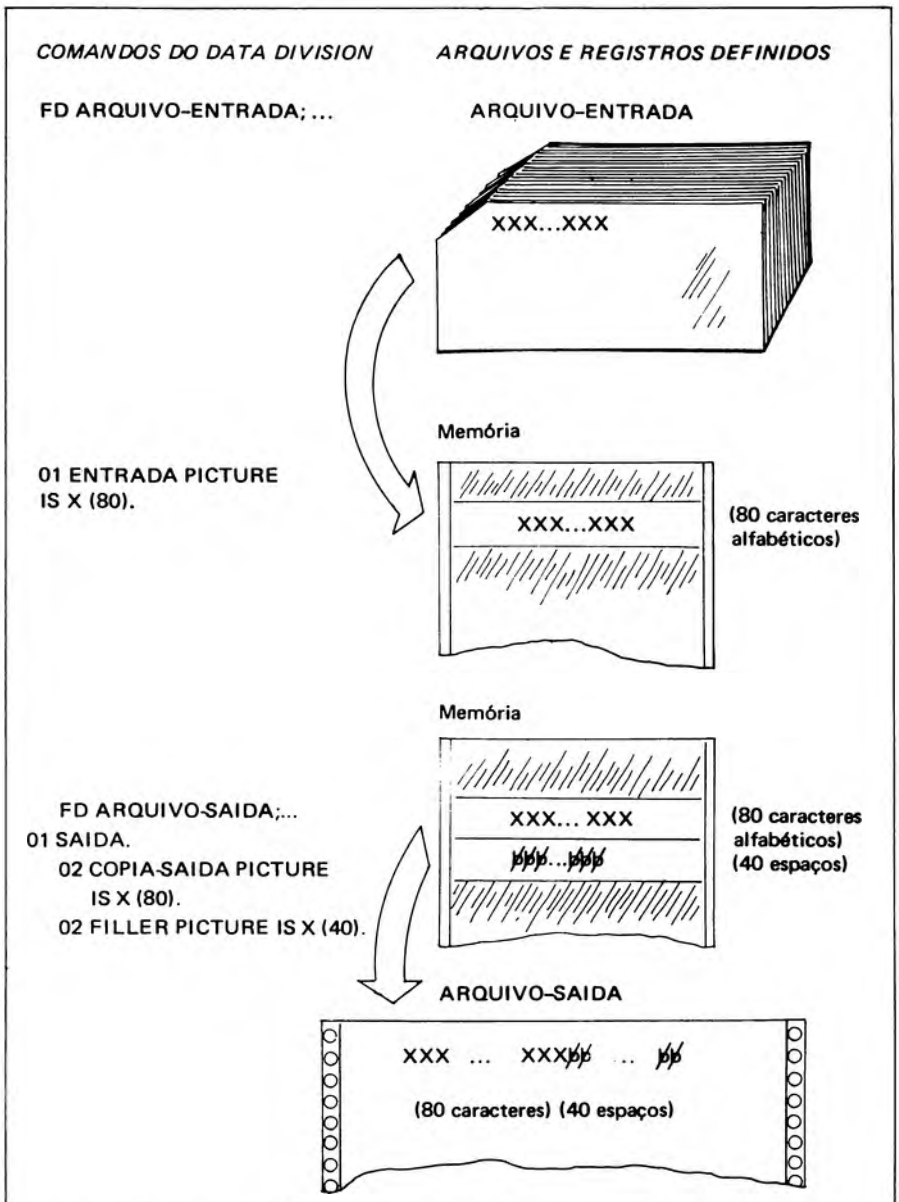


Figura 4.4. *Operação do DATA DIVISION do programa que lê e imprime cartões.*

OBSERVAÇÕES

O FILE SECTION contém a definição de todos os arquivos de entrada e de saída usados no programa. O número de nível 01 refere-se à definição de um registro de dados, e os números 02, 03, 04 são os números de nível dos subcampos de um registro, se for o caso. Cada arquivo é definido pela sigla FD.

PICTURE's são especificações do tipo, tamanho dos campos ou subcampos e só devem ser usadas em subcampos de nível hierárquico com número mais elevado, isto é, só quando os campos ou subcampos não forem mais subdivididos.

FILLER é uma especificação que fornece automaticamente espaços em branco ou zeros, conforme o campo seja alfanumérico ou numérico.

LABEL RECORDS ARE OMITTED significa que não há definição especial das etiquetas ou rótulos de registros.

O número total de caracteres em uma linha da Impressora foi padronizado em 120, mas varia de Impressora para Impressora.

4.5.2. Algumas especificações do DATA DIVISION

PICTURE IS KKK – Descreve e especifica o tipo e o tamanho de um campo ou subcampo de um registro ou de uma variável usada no programa.

A parte KKK pode ser do seguinte tipo:

X(k) ou X...X para indicar que o campo vai receber k caracteres alfabéticos ou numéricos;

9(k) ou 9...9 para indicar que o campo vai receber k caracteres exclusivamente numéricos. Para campo numérico o valor máximo de k é 18 dígitos.

Exemplos:

```
01 TOTAL PICTURE IS 9(4). (ou 9999).      • O campo TOTAL vai receber até 4 algarismos.

01 CAMPO-UM
   02 CAMPO-NOME PICTURE IS X(15).        • Subcampo com capacidade para 15 caracteres.

   02 CAMPO-DATA.
      03 DIA  PICTURE IS 99.              • DIA contém 2 algarismos.
      03 MES  PICTURE IS 99.              • MES contém 2 algarismos.
      03 ANO  PICTURE IS 99.              • ANO contém 2 algarismos.
```

FILLER: Define campo formado por espaços em branco ou zeros.

VALUE IS KKK: Pode ser usado após o PICTURE na WORKING-STORAGE SECTION e serve para colocar um valor inicial ao campo.

Exemplos:

```
01 CODIGO PICTURE IS 99; VALUE IS 15.     • CODIGO tem valor inicial igual a 15.
01 TAXA PICTURE IS 9; VALUE IS ZERO.       • TAXA com 1 algarismo possui valor inicial 0.
```

4.6. EXEMPLO DE PROGRAMA COBOL-2

Consideremos um lote de cartões que formam um arquivo chamado ARQUIVO-DE-PRODUTO, cada cartão contendo os seguintes dados:

- Nome do Produto em 30 colunas (campo alfanumérico)
- Quantidade em 5 colunas (campo numérico)
- Valor Unitário em 10 colunas (campo numérico)
- Demais colunas em branco.

Para cada cartão lido, calcular a fórmula:

Custo do produto = quantidade x valor unitário.

Imprimir uma linha na impressora com os seguintes dados:

- Nome do produto ocupando 30 espaços (caracteres alfanuméricos)
- A quantidade ocupando 5 espaços, (campo numérico)
- Valor Unitário em 5 espaços, (campo numérico)
- Custo do Produto em 15 espaços, (campo numérico)

colocando 5 espaços em branco entre cada item. Essas linhas de impressão formam o arquivo de saída chamado LISTAGEM-DE-SAÍDA.

O DATA DIVISION e o PROCEDURE DIVISION deste programa são:

```
A     B
DATA  DIVISION.
FILE  SECTION.
FD    ARQUIVO-DE-PRODUTO  LABEL RECORDS ARE OMITTED;
      DATA RECORD IS CARTAO-LIDO.
01    CARTAO-LIDO.                              (Definição dos dados do
      02 NOME-DO-PRODUTO PICTURE IS X(30).         Arquivo de Entrada)
      02 QUANTIDADE PICTURE IS 9(5).
      02 VALOR-UNITARIO PICTURE IS 9 (10).
      02 FILLER PICTURE IS X(35).
FD    LISTAGEM-DE-SAIDA  LABEL RECORDS ARE OMITTED;
      DATA RECORD IS LINHA-DE-LISTAGEM.
01    LINHA-DE-LISTAGEM.                        (Definição dos dados do
      02 SAIDA-NOME PICTURE IS X(30).               Arquivo de Saída)
      02 FILLER PICTURE IS X(5).
      02 SAIDA-QUANTIDADE PICTURE IS 9(5).
      02 FILLER PICTURE IS X(5).
      02 SAIDA-VALOR-UNITARIO PICTURE IS 9(10).
      02 FILLER PICTURE IS X(5).
      02 CUSTO-DO-PRODUTO PICTURE IS 9(15).
      02 FILLER PICTURE IS X(45).
WORK  ING-STORAGE SECTION.                       (Definição da área de
      77 CUSTO-CALCULADO PICTURE IS 9(15).          trabalho usada)
```

```
A     B
PROC  EDURE DIVISION.
INIC  IO.
      OPEN INPUT ARQUIVO-DE-PRODUTO.
      OPEN OUTPUT LISTAGEM-DE-SAIDA.
LEIT  URA.
      READ ARQUIVO-DE-PRODUTO  AT END GO TO ULTIMO-CARTAO.
TRAN  SFERENCIA-DADOS-LIDOS.
      MOVE NOME-DO-PRODUTO TO SAIDA-NOME.
      MOVE QUANTIDADE TO SAIDA-QUANTIDADE.
      MOVE VALOR-UNITARIO TO SAIDA-VALOR-UNITARIO.
CALC  ULO-DO-CUSTO.
      MULTIPLY QUANTIDADE BY VALOR-UNITARIO GIVING CUSTO-CALCULADO.
      MOVE CUSTO-CALCULADO TO CUSTO-DO-PRODUTO.
IMPR  ESSAO.
      WRITE LINHA-DE-LISTAGEM.
      GO TO LEITURA.
ULTI  MO-CARTAO.
      CLOSE ARQUIVO-DE-PRODUTO.
      CLOSE LISTAGEM-DE-SAIDA.
      STOP RUN.
```

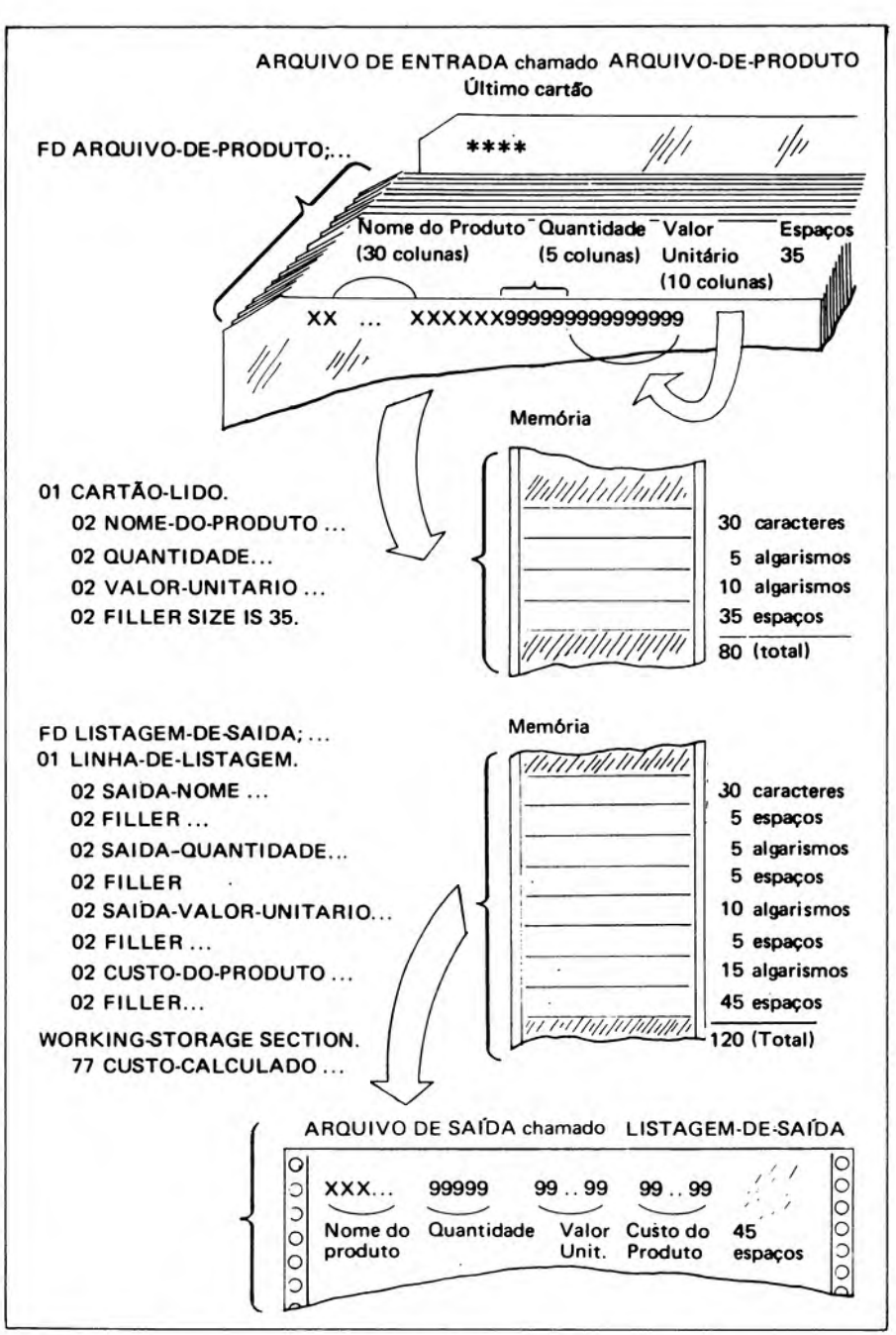


Figura 4.5. *Operação do DATA DIVISION do programa COBOL-2.*

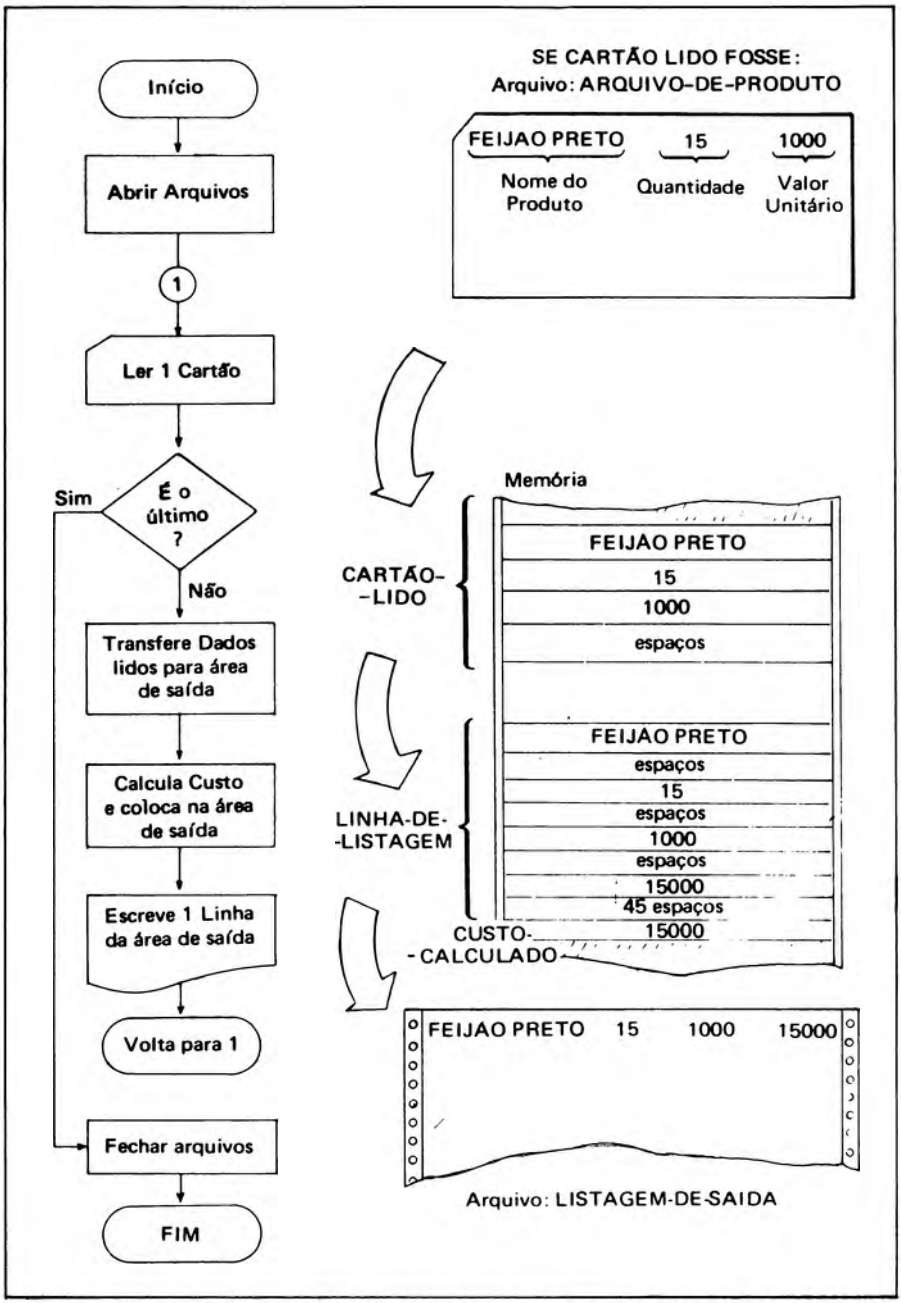


Figura 4.6. *Fluxograma e operação do PROCEDURE DIVISION do programa COBOL-2.*

4.7. RESUMO DE UM PROGRAMA COBOL

a) Necessita de uma identificação (nome do programa) através do IDENTIFICATION DIVISION.
b) É preciso atribuir cada arquivo de entrada ou de saída do programa a uma Unidade de Entrada ou Saída de Dados, como Impressora, Leitora, Perfuradora, Unidade de Disco e Fita. Esta ligação Arquivo-Unidades Periféricas é efetuada na ENVIRONMENT DIVISION.
c) É necessário definir o nome de cada arquivo e definir na memória o local onde cada registro desse arquivo é lido ou escrito para operação de Entrada/Saída. Essa ligação MEMÓRIA-ARQUIVO é feita pela DATA DIVISION. Outros dados de trabalho ou valores constantes são também definidos pelo DATA DIVISION.
d) Todo tipo de processamento é efetuado pelo PROCEDURE DIVISION através dos seus comandos que manipulam os dados definidos pelo DATA DIVISION.

EXERCÍCIOS DE RECAPITULAÇÃO

1. Explicar o que faz cada uma das divisões de um programa em COBOL.
2. Explicar o que faz cada um dos seguintes comandos COBOL, levando-se em conta que inicialmente o campo A contém valor 5, o campo B contém 11, X contém 21 e Y contém 0. Dizer quais são os valores finais desses campos após a execução em seqüência de todos os comandos.

```
ADD A TO B.
ADD A, B GIVING X.
SUBTRACT X FROM Y.
MOVE B TO X.
MULTIPLY X BY Y GIVING A.
```

3. Explicar o que faz os seguintes comandos de definição:

```
01 NAME PICTURE IS X(20).
01 IMPORTANCIA PICTURE IS 9(10).
   02 TOTAL PICTURE IS 9999.
   02 ITEM PICTURE IS XXX.
```

EXERCÍCIOS DE APLICAÇÃO

1. Explicar, comando por comando, o que faz cada um dos seguintes trechos de programa:

a)
```
OPEN INPUT ARQ.
READ ARQ AT END GO TO FINAL.
ADD X, Y GIVING Z.
MULTIPLY Z BY W.
IF Z GREATER THAN W GO TO COMEÇO.
```

b)
```
OPEN INPUT ARQ.
OPEN OUTPUT SAIDA.
READ ARQ.
IF X IS NOT GREATER THAN W WRITE REGISTRO-SAIDA.
MOVE X TO W.
GO TO INICIO.
```

2. Seja um lote de cartões, cada qual contendo 5 valores numéricos perfurados em 4 colunas cada um. Para cada cartão lido calcular a soma desses 5 valores numéricos e perfurar uma linha na impressora contendo o valor da soma. Escrever o DATA DIVISION e o PROCEDURE DIVISION.

3. Em um lote de cartões, cada cartão contém os seguintes dados:
   - Nome do funcionário (20 colunas)
   - Código do estado civil (colunas 21 e 22) onde
     00 = solteiro
     01 = casado
     02 = viúvo
     03 = desquitado ou divorciado
   - Código de sexo (coluna 23) onde
     0 = sexo masculino
     1 = sexo feminino

   Escrever o DATA DIVISION e PROCEDURE DIVISION de um programa que liste o nome de todos os funcionários casados do sexo masculino.
4. No último programa-exemplo em COBOL deste capítulo, explicar comando por comando, o DATA DIVISION e o PROCEDURE DIVISION.

Resposta: 1. Abre arquivo de entrada ARQ.
Lê um registro desse arquivo. Soma X e Y colocando o resultado em Z. Se Z for maior do que W, continuar no parágrafo COMEÇO.

Parte II

COBOL ESTRUTURADO

5 PROGRAMAÇÃO ESTRUTURADA EM COBOL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

O QUE É PROGRAMAÇÃO ESTRUTURADA?

- Uso de Estruturas Básicas:
  - Estrutura de Seqüência Simples.
  - Estrutura de Seleção IF-THEN-ELSE.
  - Estrutura de Controle de Repetição (Loops).

O PAPEL DO COMANDO "GO TO"

A ESTRUTURAÇÃO NA LINGUAGEM COBOL

EXEMPLO DE PROGRAMA COBOL SEM ESTRUTURAÇÃO E COM ESTRUTURAÇÃO

Programa não estruturado.

Programa estruturado.

Fluxogramas dos exemplos.

Comparação dos dois programas.

EXEMPLOS DE ESTRUTURA BÁSICA DE SELEÇÃO: IF

EXEMPLOS DE ESTRUTURA BÁSICA DE REPETIÇÃO: PERFORM . . . UNTIL.

TRANSFORMAÇÃO DO PROGRAMA COBOL-2 EM PROGRAMA ESTRUTURADO.

- Versão não estruturada e estruturada do programa COBOL-1
- Uma palavra de advertência.

Exercícios

5.1. O QUE É PROGRAMAÇÃO ESTRUTURADA?

A filosofia principal da Programação Estruturada consiste em construir programas cada vez mais claros e legíveis para o usuário desses programas e sobretudo para a documentação dos mesmos.

Por outro lado, a eficiência e o tamanho do programa também não devem ser desprezados. Para isso, é necessário que os programas usem *Estruturas Padronizadas ou Básicas* que facilitem o entendimento dos mesmos por parte de qualquer pessoa, sem contudo perder a eficiência na sua execução.

Foi mostrado, pelos pesquisadores em Ciência da Computação, que é possível escrever *qualquer* programa usando somente combinações de 3 tipos de estruturas básicas, visualizadas na Figura 5.1.

- *Estrutura de SEQÜÊNCIA SIMPLES:* onde os comandos são executados na ordem em que estão escritos.
- *Estrutura de SELEÇÃO IF-THEN-ELSE:* onde os grupos de comandos podem ser selecionados pelas cláusulas THEN ou ELSE.
- *Estrutura de CONTROLE DE REPETIÇÃO OU LOOP:* através de comandos do tipo DO (FORTRAN), FOR (BASIC e PASCAL), PERFORM (COBOL), DO-WHILE, DO-UNTIL (PL/1) etc.

A Programação Estruturada não é somente uma maior estruturação de programas através da clareza no uso de blocos de comandos e modularização de trechos de programas, como muitos dos programadores habilidosos julgam já estarem fazendo.

A Programação Estruturada consiste em impor uma disciplina rígida de uso das 3 estruturas básicas no programa, o que certamente faz com que os programas sejam melhorados mais ainda quanto a clareza e eficiência.

5.2. O PAPEL DO COMANDO "GO TO"

O comando GO TO é considerado indesejável no processo de estruturação de um programa. Considera-se que o uso indiscriminado desse comando dificulta a compreensão e a clareza do programa, pois o texto escrito (Documentação) do programa usando GO TO não traz uma indicação visual e prática da seqüência de comandos executados pelo computador.

Entretanto, não é correto dizer que apenas a eliminação ou a minimização do uso do comando GO TO produzem um programa estruturado.

A situação real é que, quando as 3 estruturas básicas citadas forem usadas corretamente, não há necessidade do uso do comando GO TO, que, por sinal, é um comando poderoso demais (desvia para qualquer parte do programa!) para ser usado sem critério.

Também não existe a necessidade imperiosa de eliminação total do comando GO TO, pois em certos casos o seu uso é mais simples e eficiente do que procurar uma alternativa usando as estruturas básicas.

É conveniente, porém, tentar limitar o uso de GO TO somente para casos extremos, pois o uso de GO TO, além de certos limites, é que traz a complexidade e a falta de clareza nos programas.

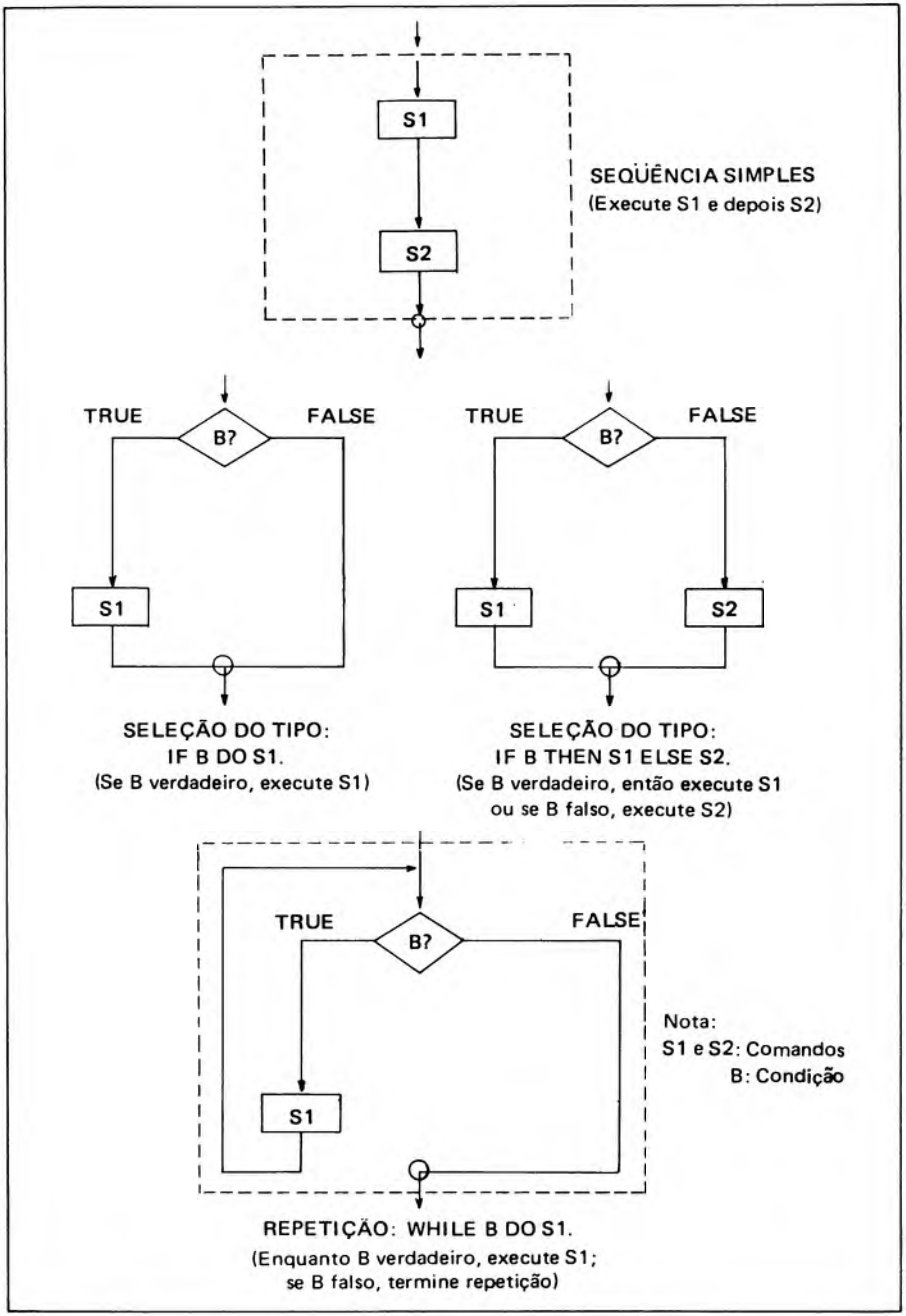


Figura 5.1. *Estruturas básicas de programação estruturada.*

5.3. A ESTRUTURAÇÃO NA LINGUAGEM COBOL

Nos programas escritos em COBOL, a estruturação pode ser conseguida pelo uso combinado dos seguintes comandos:

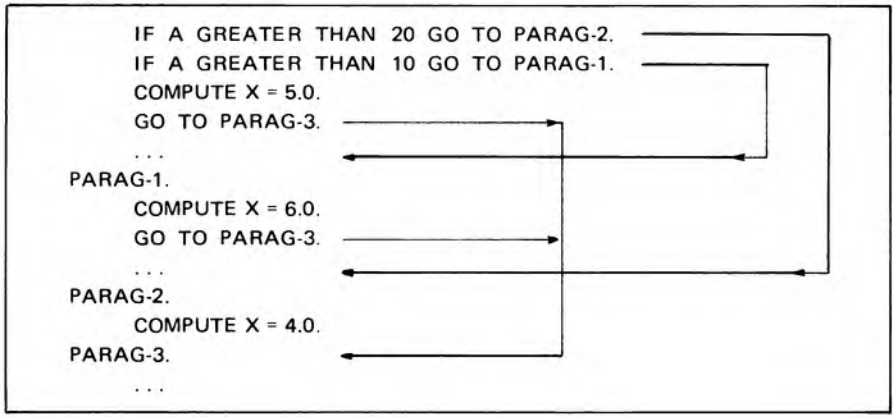
```
IF[THEN]ELSE para a seleção e
PERFORM. . . UNTIL para o controle da repetição.
```

Além disso, o fato de o comando GO TO em COBOL desviar somente para o início do parágrafo de um conjunto de comando é um fator atenuante ao uso desse comando quando necessário.

5.4. EXEMPLO DE PROGRAMA COBOL SEM ESTRUTURAÇÃO E COM ESTRUTURAÇÃO

Os dois programas seguintes executam exatamente as mesmas operações.

5.4.1. Programa não-estruturado



5.4.2. Programa estruturado

```
IF A GREATER THAN 20 COMPUTE X = 4.0;
   ELSE IF A GREATER THAN 10 COMPUTE X = 6.0;
        ELSE COMPUTE X = 5.0.
PARAG-3.
   . . .
```

5.4.3. Fluxogramas dos exemplos

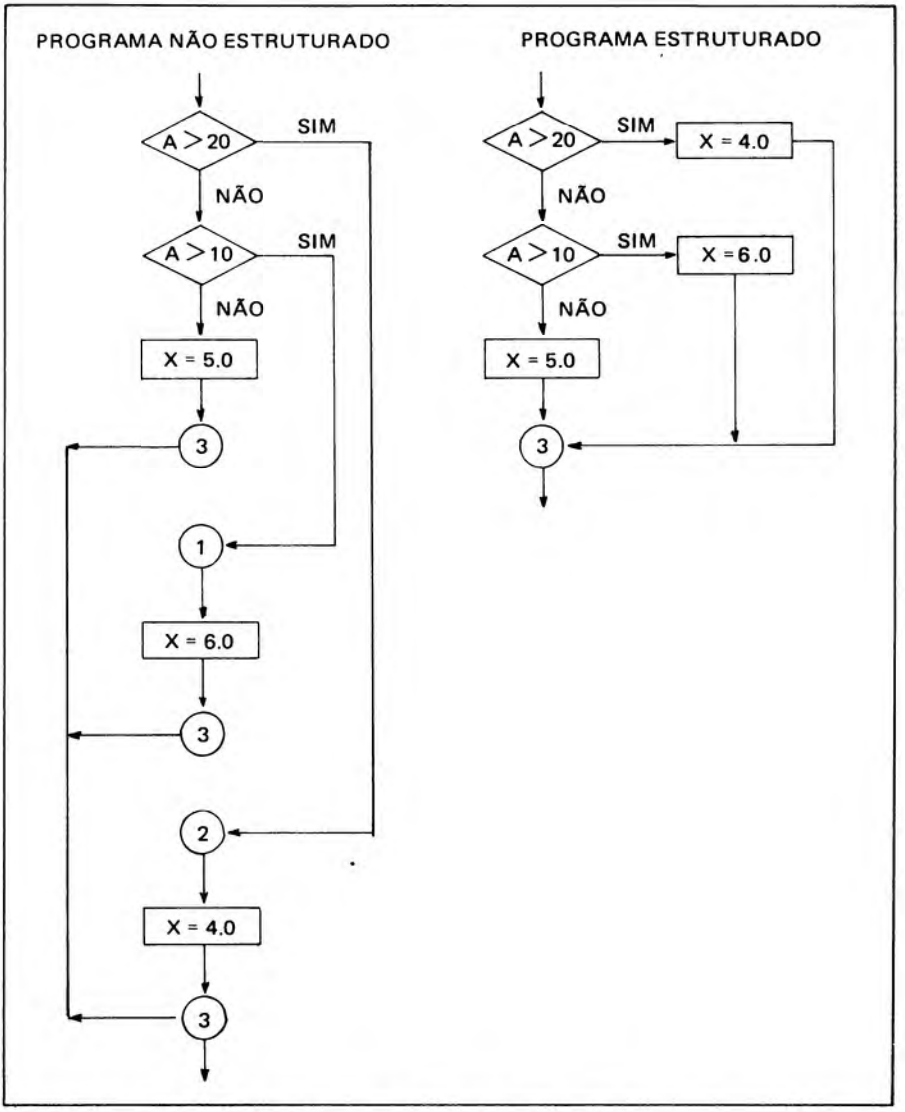


Figura 5.2. *Fluxogramas decorrentes dos exemplos dados.*

5.4.4. Comparação dos dois programas

No programa não estruturado verificaram-se diversos pontos de interrupção (saltos com GO TO), que prejudicam a seqüência natural de apresentação dos comandos. Isso poderia trazer sérias dificuldades ao entendimento do progra-

ma, pois os pontos de saltos ou desvios (PARAG-1, PARAG-2, PARAG-3) poderiam estar situados em páginas ou folhas diferentes do comando IF, prejudicando a compreensão do raciocínio lógico do programa. Se em cada parágrafo do desvio tivéssemos trechos maiores de cálculo do que o comando COMPUTE, a dificuldade seria maior.

No programa estruturado, ao contrário, sempre teremos controle da seqüência natural de execução, pois tudo está dentro do controle de uma das estruturas básicas já citadas (IF-ELSE).

É importante notar que, no programa estruturado, sempre temos um ÚNICO ponto de ENTRADA e um ÚNICO ponto de SAÍDA do programa ou da estrutura básica usada, ao passo que, no programa não estruturado, o uso do GO TO permite abrir qualquer número de SAÍDAS através de desvios incondicionais a QUALQUER PARTE do programa. Sem dúvida alguma, esse é um RECURSO PODEROSO DEMAIS para ser usado sem critério no programa.

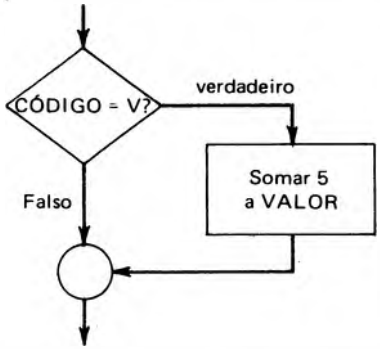
5.5. EXEMPLOS DE ESTRUTURA BÁSICA DE SELEÇÃO: IF

Vamos dar exemplos de uso de blocos de seleção do tipo:

IF B DO S1 (Se B verdadeiro, execute S1) e
IF B THEN S1 ELSE S2 (Se B verdadeiro, execute S1; caso contrário, execute S2).

Os comandos de execução S1 ou S2 podem ser substituídos por qualquer comando simples do COBOL (inclusive outro comando IF) ou por um conjunto de comandos agrupados em parágrafos e executados pelo comando chamado PERFORM como se fosse sub-rotina. Em COBOL a seleção IF B DO S1 torna-se IF B S1 ou IF B THEN S1.

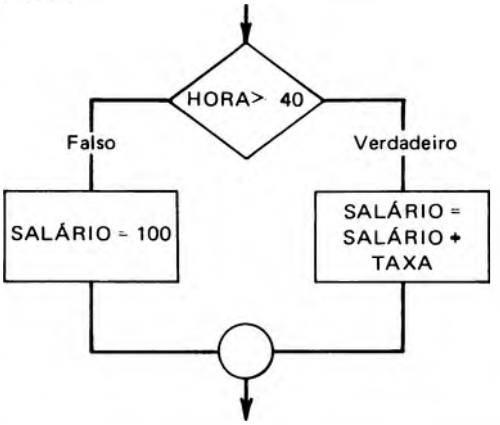
EXEMPLO 1:



Comando:

IF CÓDIGO = V
ADD 5 TO VALOR.

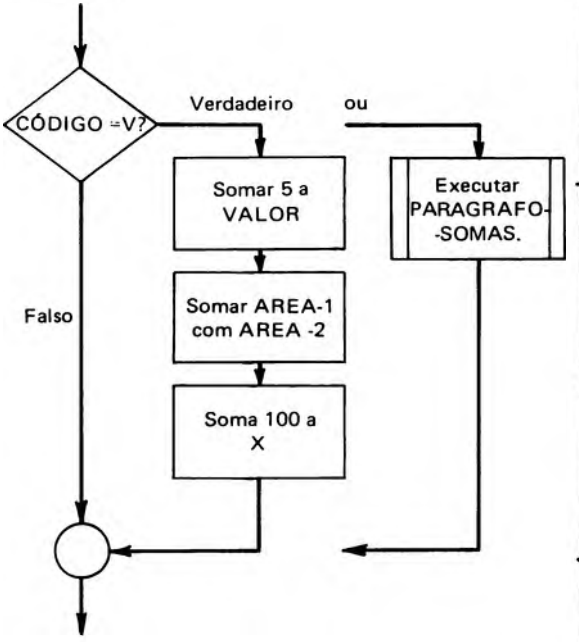
EXEMPLO 2:



Comando:

```
IF HORA IS GREATER
   THAN 40 THEN ADD
   TAXA TO SALARIO
   ELSE MOVE 100 TO
   SALARIO.
```

EXEMPLO 3:



Comandos:

```
IF CODIGO = V  ADD 5 TO
   VALOR

        ADD  area  -1 TO
        AREA - 2
        ADD 100 TO X.
```

Obs. por ser um único comando IF, colocar um **único ponto** após o X.

ou

```
IF CODIGO = V PERFORM
   PARAGRAFO-SOMAS.
. . . .
. . . .
PARAGRAFO-SOMAS.
        ADD 5 TO VALOR.
        ADD  AREA - 1 TO
        AREA - 2.
        ADD 100 TO X.
```

(Parágrafo executado como se fosse sub-rotina)

EXEMPLO 4:

```
IF HORAS-TRABALHADAS IS GREATER THAN 200
      THEN PERFORM PARAGRAFO-HORAS-EXTRAS
      ELSE PERFORM PARAG-HORAS-NORMAIS.
. . . .
PARAGRAFO-HORAS-EXTRAS.
      (cálculo das horas extras)

PARAG-HORAS-NORMAIS.
      (cálculo das horas normais)
```

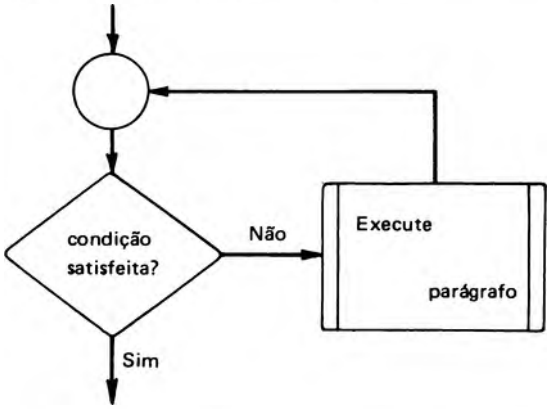
5.6. EXEMPLOS DE ESTRUTURA BÁSICA DE REPETIÇÃO: PERFORM. . . UNTIL.

Em COBOL a estrutura básica WHILE B DO S1 (enquanto B verdadeiro), execute S1) será executada pela forma equivalente PERFORM S1 UNTIL B1 (execute S1 até que condição B1 seja satisfeita).

A estrutura básica terá a seguinte forma:

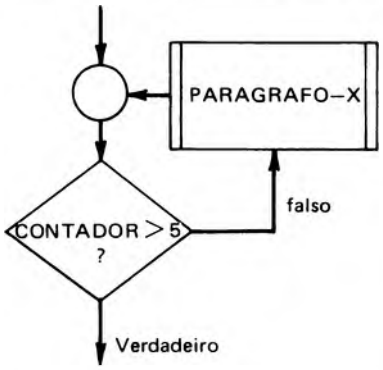
PERFORM nome-de-parágrafo UNTIL condição

e que tem a seguinte representação gráfica:



Neste caso o parágrafo (ou parágrafos), se a opção PERFORM parágrafo-1 THRU parágrafo-x for usada, é executado várias vezes até que a condição seja satisfeita.

EXEMPLO 1: Repetir o PARAGRAFO-X por cinco vezes.



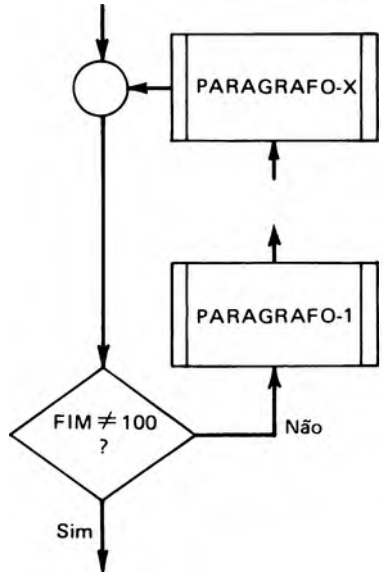
Comandos:

a) PERFORM PARAGRAFO-X 5 TIMES.
(Repetição é automática)

b) PERFORM PARAGRAFO-X UNTIL CONTADOR > 5.
(o CONTADOR deve ser incrementado de 1 em 1 dentro do PARAGRAFO-X).

c) PERFORM PARAGRAFO-X VARYING CONTADOR FROM 1 BY 1 UNTIL CONTADOR > 5.
(CONTADOR é incrementado pelo próprio comando PERFORM)

EXEMPLO 2: Repetir a execução do PARAGRAFO-1 ao PARAGRAFO-X até que a condição FIM ≠ 100 seja atingida.



Comando:

PERFORM PARAGRAFO-1 THRU PARAGRAFO-X
UNTIL FIM NOT EQUAL TO 100.

. . .
PARAGRAFO-1.
. . . .
PARAGRAFO-X.
. . . .

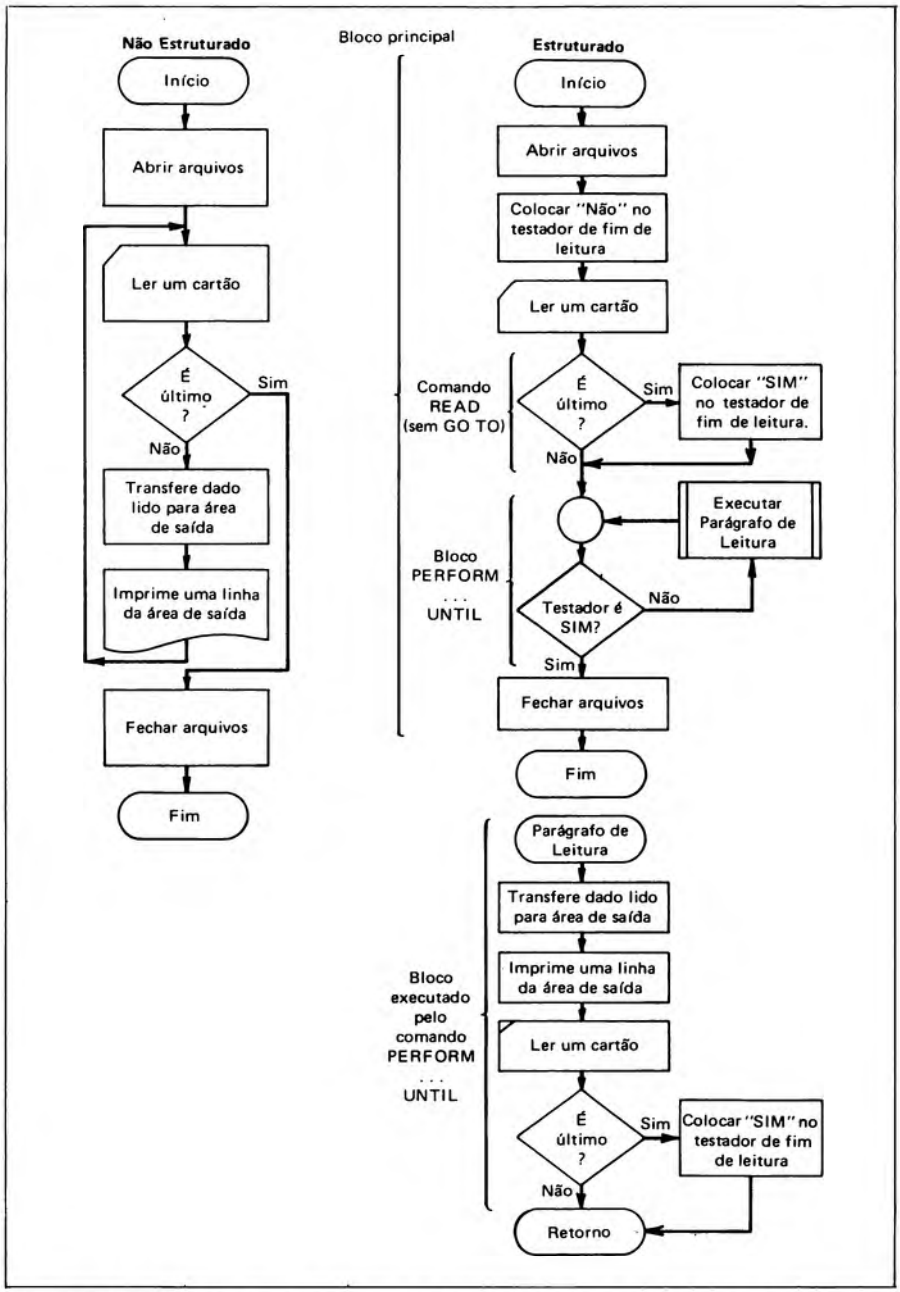


Figura 5.3. *Fluxograma do programa não estruturado e estruturado do Programa COBOL -1.*

5.7. TRANSFORMAÇÃO DO PROGRAMA COBOL-1 EM PROGRAMA ESTRUTURADO

Nos programas exemplos mencionados no Capítulo 4, e designados por PROGRAMA COBOL-1 e COBOL-2, não houve a preocupação de aplicar as técnicas de estruturação, pois são programas bastante simples e claros e o uso do comando GO TO não prejudica essa clareza do programa.

Entretanto, devemos iniciar o treinamento da estruturação de programas através desses tipos de exemplos, para podermos esclarecer bem a técnica de estruturação.

A ESTRUTURAÇÃO deve ser usada com o OBJETIVO FIRME de: "tornar um programa de tamanho médio ou grande em programa fácil de entender e de ser modificado, utilizando-se as estruturas básicas citadas".

Na estruturação somente o PROCEDURE DIVISION é afetado. Apresentaremos a versão NÃO ESTRUTURADA E ESTRUTURADA do Programa COBOL-1 já mencionado.

O Fluxograma das duas versões está apresentando na Figura 5.3.

5.7.1. Versão não estruturada e estruturada do programa COBOL-1

NÃO ESTRUTURADA

A B

```
PROCEDURE DIVISION
CONDIÇÕES-INICIAIS
    OPEN INPUT ARQUIVO-ENTRADA.
    OPEN OUTPUT ARQUIVO-SAIDA.
LEITURA-E-COPIA.
    READ ARQUIVO-ENTRADA AT END
        GO TO FIM-DE-LEITURA.
    MOVE ENTRADA TO COPIA-SAIDA.
IMPRIMIR.
    WRITE SAIDA.
    GO TO LEITURA-E-COPIA
FIM-DE-LEITURA.
    CLOSE ARQUIVO-ENTRADA.
    CLOSE ARQUIVO-SAIDA.
    STOP RUN.
```

ESTRUTURADA

A B

```
. . . . . .
WORKING-STORAGE SECTION.
01 TESTADOR-CARTAO PICTURE XXX.
PROCEDURE DIVISION.
BLOCO-PRINCIPAL.
    OPEN INPUT ARQUIVO-ENTRADA.
    OPEN OUTPUT ARQUIVO-SAIDA.
    MOVE 'NAO' TO TESTADOR-CARTAO.
    READ ARQUIVO-ENTRADA AT END
      MOVE 'SIM' TO TESTADOR-CARTAO
    PERFORM PARAG-LEITURA UNTIL
        TESTADOR-CARTAO = 'SIM'.
    CLOSE ARQUIVO-ENTRADA.
    CLOSE ARQUIVO-SAIDA.
    STOP RUN.          (Fim da execução)
PARAG-LEITURA.
    MOVE ENTRADA TO COPIA-SAIDA.
    WRITE SAIDA.
    READ ARQUIVO-ENTRADA AT END
      MOVE 'SIM' TO TESTADOR-CARTAO.
```

5.7.2. Comentários sobre a versão estruturada

a) O bloco principal do programa está composto por SEQÜÊNCIA DE ESTRUTURAS BÁSICAS SEM DESVIOS, eliminando-se o GO TO:
   OPEN
   OPEN
   MOVE
   READ
   PERFORM . . . UNTIL
   CLOSE
   CLOSE
   STOP RUN

b) O parágrafo PARAG-LEITURA é um bloco independente executado pelo comando PERFORM...UNTIL dentro do bloco principal. Em todo programa COBOL estruturado, além do bloco principal (que é formado por seqüências de estruturas básicas sem desvios) existem parágrafos independentes que são executados exclusivamente pelos comandos PERFORM do bloco principal.

c) Foi necessário usar uma nova variável chamada TESTADOR-CARTAO para eliminar o GO TO usando no comando READ. . . AT END.

d) O uso de dois comandos READ (dentro do bloco principal e também no bloco executado pelo PERFORM. . . UNTIL) foi necessário, pois, caso o arquivo de entrada esteja vazio (isto é, o programa foi executado sem colocar cartões na Leitora), é necessário parar imediatamente o programa antes de executar qualquer outro comando contido no PERFORM...UNTIL. Sem o primeiro READ, haveria, nesse caso, a execução dos comandos MOVE e WRITE **antes** de percebermos que o arquivo de entrada está vazio.

e) A versão estruturada ficou mais longa e sobrecarregada pelo uso de comandos adicionais MOVE e PERFORM. Isto mostra que em **programas simples** e em certos casos especiais o uso do comando GOTO simplifica o programa. Entretanto, a sobrecarga adicional do programa é um preço a ser pago em favor da clareza e da facilidade de manutenção e documentação do programa. Em programas mais longos, o esforço adicional de programação (que será minimizada com a prática) e a sobrecarga do programa devido a comandos adicionais ficam plenamente compensados devido a essas vantagens. Mais adiante veremos um programa completo em COBOL um pouco mais complexo, com e sem estruturação.

5.7.3. Uma palavra de advertência

A técnica de estruturação de programa ajuda consideravelmente a tornar os programas mais claros e em casos complexos até mais eficientes. Entretanto, é errado dizer que **PROGRAMA SEM GO TO É BOM** e **PROGRAMA COM GO TO É RUIM.**

Deve-se sempre evitar atitudes extremadas, lembrando sempre as frases das pessoas que estudaram e propuseram a técnica de estruturação de programas, tais como:

E. Dijkstra: "Por favor não julguem que eu sou terrivelmente dogmático sob o uso do comando GO TO. Eu estou tendo um sentimento terrível que alguns estão tomando o fato como se fosse uma RELIGIÃO. . ."

D. Knuth: "Eu sou a favor da eliminação dos GO TO em certos casos, e pela sua introdução em outros casos."

Fonte: Structured Programming with go to statements -
D. Knuth, Computing Surveys, vol. 6, n. 4, Dezembro, 1974.

EXERCÍCIOS DE RECAPITULAÇÃO

1. O que é Programação Estruturada?
2. Citar e descrever as operações das estruturas básicas usadas na Programação Estruturada.
3. Dar exemplos da estrutura básica IF em COBOL.
4. Dar exemplos da estrutura básica de repetição PERFORM-UNTIL.
5. Comentar, comando a comando, a versão não estruturada e estruturada do Programa COBOL-1 deste capítulo. Quais são as principais diferenças entre as duas versões?

EXERCÍCIOS DE APLICAÇÃO

1. Transformar o EXEMPLO DE PROGRAMA COBOL-2 do Capítulo 4, em versão estruturada, fornecendo o fluxograma e o novo programa COBOL.
2. Escrever um relatório sucinto, de 20 a 30 linhas, comentando o uso do comando GO TO em programação estruturada, analisando os exemplos de não estruturação e estruturação e o PROGRAMA COBOL-1 deste capítulo.
3. Escrever os fluxogramas estruturados de cada exemplo de fluxogramas do Capítulo 3.

6 TIPOS DE PALAVRAS EM COBOL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

PALAVRAS RESERVADAS

- Palavras-chave (Key-words). Ex.: MOVE, FILE, ADD etc.
- Palavras Opcionais. Ex.: IS, ON, THEN.
- Palavras Conectivas. Ex.: OF e IN.

IDENTIFICADORES OU NOMES (NAMES)

- Nome de Dados. Ex.: X, CODIGO-MERCADORIA
- Nome de Condição.
- Nome de Procedimento, Seção e Parágrafo. Ex.: PARAGRAFO-UM.
- Nomes Especiais. Ex.: CARD, PRINTER, SYSOUT etc.
- Regras para formação de nomes.
- Exemplos.

CONSTANTES OU LITERAIS

- Constantes ou Literais Numéricas. Ex.: 0, 3.1416.
- Constantes ou Literais Não Numéricas. Ex.: 'ZERO', 'JOSE', '100' etc.
- Exemplos.
- Constantes Figurativas. Ex.: ZERO, ZEROS, SPACES, QUOTE.

Exercícios

As sentenças e as frases usadas em COBOL são formadas por PALAVRAS, classificadas conforme veremos a seguir.

6.1. PALAVRAS RESERVADAS

O COBOL possui uma lista considerável (vide apêndice B) de palavras que não podem ser usadas como nomes de variáveis por serem palavras usadas para

determinada finalidade. Por exemplo: DATA, FILE, PROCEDURE etc. são palavras reservadas.

As palavras reservadas podem ser de três tipos:

6.1.1. Palavras-chave (key-words)

São palavras reservadas que representam alguma ação e são os "verbos" da linguagem COBOL. São de uso obrigatório nas frases e comandos e aparecem sublinhadas nos formatos gerais. Por exemplo, são palavras-chave: *MOVE, ADD, MULTIPLY* etc.

6.1.2. Palavras opcionais

São palavras reservadas que melhoram a redação das sentenças podendo ser dispensadas e por essa razão não aparecem sublinhadas. Exemplos: IS e ON usadas em *VALUE* IS ..., ON *SIZE ERROR...* .

6.1.3. Palavras conectivas

São as palavras OF ou IN usadas para associar um nome a um outro que o qualifica.

Exemplos:

```
ADD VALUE OF ITEM TO SOMA.
MOVE DIA IN DATA-DE-ENTREGA TO PRAZO-FINAL.
```

6.2. IDENTIFICADORES OU NOMES (NAMES)

6.2.1. Nomes de dados

São nomes ou identificações atribuídas aos dados usados no programa, tais como HORAS-TRABALHADAS, SALARIO-LIQUIDO etc. Todos os dados usados no programa devem ser identificados ou definidos no DATA DIVISION.

Para evitar o uso de PALAVRAS RESERVADAS, como nome de dados, recomendam-se nomes compostos, formados por mais de uma palavra em PORTUGUÊS e ligados por hífen. Algumas palavras válidas em Português, como FINAL, ÁREA, DATA são palavras reservadas.

6.2.2. Nomes de condição

São nomes usados com associação a várias condições ou valores numéricos. Por exemplo, se existir um grupo de seis distritos diferentes, cada distrito pode ser designado por um nome. O nome DISTRITO então pode assumir seis valores diferentes, evitando-se o uso de nomes dos distritos individuais, o que pode economizar espaço em um programa. Um exemplo será visto na descrição do DATA DIVISION e também do comando IF.

6.2.3. Nomes de procedimento: nome de parágrafo e nome de seção

São nomes designando conjunto de comandos do PROCEDURE DIVISION e que formam um parágrafo que pode ser usado como subprograma ou simplesmente para ponto de desvio do programa. A seção, por sua vez, é um conjunto de parágrafos e que pode receber um nome.

6.2.4. Nomes especiais

São nomes designados a componentes do computador usado, tais como CARD-PUNCH, PRINTER etc. Tais nomes são designados aos componentes através do ENVIRONMENT DIVISION.

6.2.5. Regras para formação de nomes

O programador pode usar nomes que representem de modo significativo a natureza dos elementos ou dados por ele designados. As regras básicas para formação de nomes são:

1. Usar letras do alfabeto (A a Z), algarismos (0 a 9) ou hífens (—).
2. Um nome deve conter de 1 a 30 caracteres no máximo.
3. Espaços não são permitidos no meio dos nomes.
4. Um nome não deve começar nem terminar com hífen, embora seu uso seja permitido no meio do nome.
5. Nome de dados, nome de condição e nomes especiais devem conter pelo menos um caráter alfabético. Nomes de parágrafos podem conter apenas algarismos.
6. Os nomes podem ser "qualificados" ou "atribuídos" a outros nomes pelo uso da preposição OF ou IN. Nesse caso, um mesmo nome pode ser atribuído a elementos distintos. Por exemplo:
   Se existir o valor chamado CODIGO tanto no CARTAO-MESTRE como no CARTAO-CLIENTE, podemos qualificar cada CODIGO como sendo:
   CODIGO OF CARTAO-MESTRE (ou CODIGO IN CARTAO-MESTRE) e
   CODIGO OF CARTAO-CLIENTE (ou CODIGO IN CARTAO-CLIENTE).

6.2.6. Exemplos de nomes

SALARIO-FAMILIA
8XYZ
ZONA-1
69345 (só pode ser nome de parágrafo)
%-TOTAL (O caráter % não é permitido)
DATA (não é permitido, pois é palavra reservada)
VALOR-DO-ITEM-ESTOCADO-EM-DEPOSITO (mais de 30 caracteres)
051 e 51 (nome de parágrafos distintos)
NOME DO ALUNO (espaços não são permitidos)
NOME OF ALUNO (nome qualificado por OF)

6.3. CONSTANTES OU LITERAIS

A constante é um valor que não se altera durante a execução do programa. A constante pode ser numérica, alfabética (como itens de um cabeçalho) ou alfanumérica (combinação de letras, números e sinais).

Uma constante pode ser usada no PROCEDURE DIVISION através de um nome ou diretamente através do seu valor. No primeiro caso deve ser definida no WORKING-STORAGE SECTION do DATA DIVISION onde recebe um nome e um valor.

EXEMPLO:

```
02  PI  PICTURE  9V9(4)  VALUE  IS  3.1416 .
```

Se o valor de uma constante for diretamente usado no programa sem ser definido no DATA DIVISION, isto é, sem receber um "nome", ele é chamado de LITERAL.

As constantes ou literais podem ser:

6.3.1. Constantes ou literais numéricas

São usadas para operação aritmética ou comparação, não podendo em geral ter mais de 18 dígitos, dependendo do computador usado. Podem possuir:

- *sinal menos ou mais* (este sendo opcional) colocado na posição mais à esquerda;
- *um ponto decimal* (a vírgula pode ser usada se a especificação "DECIMAL-POINT IS COMMA" for usada no SPECIAL NAMES do ENVIRONMENT DIVISION) que pode ser usado em qualquer posição exceto na posição mais à direita.

6.3.2. Constantes ou literais não numéricas

Devem aparecer cercadas por dois apóstrofos ('), e os espaços em branco são tratados como caracteres ocupando espaço na memória. Podem ser de qualquer tamanho, a menos que seja limitado pelo computador em uso. (Em geral, limitado em 120 caracteres.)

6.3.3. Exemplos de constantes ou literais

.26	literal numérica
'.26'	literal não numérica
–026	literal numérica
3560.	idem, incorreta, o certo seria 3560.0
'ZONA–X15'	literal não numérica
ADD 0.15 TO VALOR	literal numérica usada no comando ADD
MOVE '0.15' TO VALOR	literal não numérica usada no comando MOVE

6.3.4. Constantes figurativas

Algumas constantes de uso comum já estão definidas no COBOL, bastando usar o seu nome diretamente no programa. Tais constantes chamadas *Figurativas* são:

ZERO, ZEROS ou ZEROES (fornece seqüência de valores zero);
SPACE ou SPACES (fornece seqüência de espaços em branco);
HIGH-VALUE (fornece maior valor numérico possível assumido pelo COBOL);
LOW-VALUE (idem menor valor possível);
QUOTE (indica ocorrência de apóstrofo);
ALL "literal" (fornece seqüência formada pelo literal ou constante indicada).

Exemplos:

MOVE ZEROS TO ZONA (o local ou área ZONA conterá zeros).
IF CODIGO IS EQUAL TO SPACES (compara CÓDIGO com espaços em branco)
MOVE '3' TO CODIGO (preenche CODIGO com seqüência de dígito 3).
MOVE 'AB' TO CODIGO (idem com caracteres AB).
MOVE '*' TO CODIGO (idem com caráter *).
DISPLAY QUOTE 'OBA' QUOTE (escreve 'OBA', permitindo o uso de apóstrofos dentro do literal).

```
IDENTIFICATION DIVISION. ( ↑ )
...
ENVIRONMENT DIVISION. ( ↑ )
... SELECT ( ↑ ) TRANSC-FILE (*) ASSIGN ( ↑ ) TO ( ↑↑ ) TAPE-UM. (***)

DATA DIVISION ( ↑ )
FD TRANSC-FILE (*)
   BLOCK ( ↑ ) CONTAINS ( ↑↑ ) 5 RECORDS ( ↑ )
   LABEL RECORDS ( ↑ ) ARE ( ↑↑ ) STANDARD. ( ↑ )
01 TRANSA. (*)
   02 IDENTIDADE (*) PICTURE ( ↑ ) IS ( ↑↑ ) A (10).
WORKING-STORAGE SECTION. ( ↑ )
77 CINCO ( + ) PICTURE ( ↑ ) 99 VALUE ( ↑ ) IS ( ↑↑ ) 5. ( ++ )
...

PROCEDURE DIVISION. ( ↑ )
START. ( **)
   OPEN INPUT ( ↑ ) TRANSC-FILE. (*)
   MULTIPLY ( ↑ ) BASE (*) BY ( ↑ ) 35. ( ++ )
   MOVE ( ↑ ) IDENTIDADE ( *) TO ( ↑ ) NOME (*) OF ( ↑↑↑ )
   TABELA. (*)
   GO TO ( ↑ ) PARAGRAFO-FINAL. (**)
.....
PARAGRAFO-FINAL. (**)
   MOVE ( ↑ ) SPACES ( +++ ) TO ( ↑ )
   AREA-UM. (*)
.....
```

LEGENDA

(↑)	– Palavra-chave
(↑↑)	– Palavra opcional
(↑↑↑)	– Palavra conectiva
(*)	– Nome ou identificador de variável
(**)	– Nome de parágrafo
(***)	– Nome especial
(+)	– Constante com nome
(++)	– Constante literal
(+++)	– Constante figurativa

 Figura 6.1. *Atribuição de nomes em COBOL.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. O que são palavras reservadas, palavras-chave e palavras opcionais em COBOL?
2. O que são identificadores e quais são as regras básicas de formação?
3. Quais são os tipos de nomes ou identificadores? Dar exemplos.
4. Quais são os tipos de constantes em COBOL? Dar exemplos.

EXERCÍCIOS DE APLICAÇÃO

1. Apontar os erros cometidos na formação dos seguintes nomes:
   XPL NO1
   ––21X
   WORK-AREA-
   WORKING-STORAGE
   NUMERO-DE-ITEM-DO-OBJETO-COMPRADO
   AREA-DE TRABALHO 1
2. Dizer o tipo de literal (numérica, não numérica) e se estão corretas ou incorretas:
   '25.78'
   2578
   25.78
   'VALOR 2.00'
   AXZ50
   $2.00
   $2.00
   ZERO
3. Nos programas-exemplos do Capítulo 4, dizer quais são as palavras-chave, palavras opcionais, nomes de variável, constantes literais e outros elementos de atribuição de nomes.

7 SENTENÇAS: COMANDOS, EXPRESSÕES ARITMÉTICAS E LÓGICAS, CLÁUSULAS

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

ELEMENTOS DE UMA SENTENÇA
- Expressões Aritméticas e Lógicas. Ex.: (X − Y) * Z, X IS EQUAL TO Z etc.
- Comandos. Ex.: MOVE X TO Y.
- Cláusulas ou frases de Descrição. Ex.: PICTURE IS X(5).

EXPRESSÃO ARITMÉTICA
- Operações Aritméticas: Adição e Subtração: + e −
  Multiplicação: *
  Divisão: /
  Potenciação: **
- Regras de Precedência das Operações Aritméticas.

EXPRESSÕES LÓGICAS OU CONDICIONAIS
- Operações Relacionais:
  IS GREATER THAN ou >
  IS EQUAL TO ou =
  IS LESS THAN ou <
- Operações Lógicas:
  AND, OR e NOT
- Exemplos de Expressão Lógica ou Condicional.
- Regras de Precedência das Operações Lógicas.

COMANDOS
- Comandos Imperativos: ADD, GO TO, READ etc.
- Comandos Condicionais: IF

CLÁUSULAS OU FRASES DE DESCRIÇÃO
Ex.: SIZE, VALUE, PICTURE, SELECT etc.

REGRAS DE PONTUAÇÃO

NOTAÇÃO DO FORMATO GERAL DOS COMANDOS E CLÁUSULAS
- Sinais [ ] , { } , ... etc.

EXEMPLO ILUSTRADO

Exercícios

7.1. ELEMENTOS DE UMA SENTENÇA

A linguagem COBOL, de modo análogo a qualquer outra linguagem natural como o Português ou Inglês, utiliza grupos de sentenças combinadas em seções ou parágrafos para descrever um programa.

Existem três elementos principais em uma sentença:

- As Expressões Aritméticas e Lógicas.
- Os Comandos.
- As Cláusulas ou Frases de Descrição.

As sentenças são formadas por um ou mais comandos ou cláusulas e as expressões fazem parte de certos comandos.

7.2. EXPRESSÃO ARITMÉTICA

Representam as expressões ou fórmulas matemáticas e são formadas por combinação de nomes de dados, literais, operadores aritméticos e parênteses. As expressões aritméticas assumem um único resultado numérico.

As operações aritméticas permitidas são:

- Adição e Subtração: + e −
- Multiplicação e Divisão: * e /
- Potenciação: **

Exemplos:

```
(A + B) / (C − D) * 0.25
(SALARIO − HORAS-EXTRAS) * TAXA − DESCONTO
```

7.2.1. Espaçamento dos operadores aritméticos

Cada operador aritmético (+ , − , * , / , **) e o sinal de igualdade (=) que aparecem nas fórmulas e expressões aritméticas devem ser precedidos e seguidos pelo menos por um espaço. O sinal "abrir parênteses" não deve ser imediatamente seguido por um espaço e o sinal "fechar parênteses" não deve ser imediatamente precedido por um espaço.

As expressões aritméticas são usadas nos comandos COMPUTE, IF e PERFORM.

Exemplos:

```
COMPUTE X = (A + B) * 13.5
```

7.2.2. Regras de precedência das operações aritméticas

As operações aritméticas são executadas, dentro de uma mesma expressão, obedecendo às seguintes regras de precedência ou de hierarquia:

- As operações que aparecem "entre parênteses" são executadas primeiro, sendo os parênteses mais internos executados antes.
- Em um mesmo nível de hierarquia, a precedência segue esta ordem:
  - potenciação
  - multiplicação e divisão
  - adição e subtração
- A prioridade para operação de mesmo nível é sempre da esquerda para a direita.

Exemplos:

A – B * C	Equivale a A – (B * C).
A / B * C – D	Equivale a ((A / B) * C) – D.
(A / B) * C – (D + E)	Executa (A / B) e (D + E) primeiro, a seguir o produto (*) e finalmente a subtração (–).

7.3. EXPRESSÕES LÓGICAS OU CONDICIONAIS

Representam valores ou expressões que podem assumir apenas o resultado TRUE ou FALSE. São formadas por combinação de variáveis e sinais que envolvem operações lógicas ou relacionais da teoria de conjuntos ou da lógica simbólica.

7.3.1. Operações relacionais

São operações que efetuam a comparação de dois valores:

IS GREATER THAN	ou sinal	>
IS EQUAL TO	ou sinal	=
IS LESS THAN	ou sinal	<

7.3.2. Operações lógicas

São operações que ligam uma ou mais expressões lógicas formando uma nova expressão lógica:

AND	(indica a ocorrência simultânea das duas condições ou relações indicadas)
OR	(indica a ocorrência de ambas ou apenas uma das condições ou relações indicadas)
NOT	(usado para negar ou inverter o sentido da condição ou relação que aparece logo em seguida).

7.3.3. Exemplos de expressão lógica ou condicional

CAMPO-A IS EQUAL TO CAMPO-B. (se o conteúdo de CAMPO-A for igual ao conteúdo de CAMPO-B a expressão toda assume o valor TRUE, caso contrário, o valor FALSE).

C IS NOT EQUAL TO B. (análogo a anterior, mas se valores forem diferentes).

(CAMPO-A IS NOT EQUAL TO CAMPO-B) OR (CAMPO-A IS LESS THAN 350).
(para que a expressão tenha resultado TRUE, CAMPO-A não deve ser igual a CAMPO-B *ou* deve ser menor que 350 ou deve satisfazer a ambas as condições).

As expressões lógicas são usadas nos comandos condicionais IF-THEN, IF-THEN-ELSE ou PERFORM.

Exemplos:

```
IF TAXA IS EQUAL TO 150 ADD Y TO TAXA.
PERFORM PARAGRAFO-VENDA UNTIL VENDA >PREVISTO.
```

7.3.4. Regras de precedência das operações lógicas

- A operação AND sempre é executada antes da operação OR.
- As operações "entre parênteses" são executadas antes.
- A precedência da operação do mesmo nível é sempre da esquerda para a direita.

Exemplo:

(X = 1) OR (Z = 3) AND (W < 2)

A expressão será verdadeira se ocorrer X = 1 OU se ocorrer (Z = 3 e W < 2).

7.4. COMANDOS

Um comando simples consiste de um VERBO (que indica ação, operação ou função) e de OPERANDOS usados na execução da ação. Os comandos formam as sentenças usadas no PROCEDURE DIVISION.

Os comandos podem ser:

7.4.1. Comandos imperativos

São os comandos diretamente executados pelo computador. Por exemplo:

```
SUBTRACT X FROM TOTAL.
GO TO FIM-DE-OPERACAO.
```

7.4.2. Comandos condicionais

Permitem ao computador a escolha de ações alternativas dependendo do valor TRUE ou FALSE da expressão condicional que aparece nos mesmos. Por exemplo:

```
IF SOMA IS GREATER THAN 1500 GO TO FIM-DE-SOMA.
```

7.5. CLÁUSULAS OU FRASES DE DESCRIÇÃO

Descrevem as características de um dado, de um arquivo ou de um dispositivo e são usadas nas sentenças do DATA DIVISION, ENVIRONMENT DIVISION e IDENTIFICATION DIVISION. Por exemplo:

```
SIZE IS 40 CHARACTERS.
VALUE IS ZERO
SELECT FILE-UM ASSIGN TO CARD-READER.
```

7.6. REGRAS DE PONTUAÇÃO

Uma sentença em COBOL deve terminar com um ponto seguido de pelo menos um espaço. Quando a sentença contém mais de um comando ou cláusula, estas podem estar separadas por um dos seguintes separadores:

```
  (ponto e vírgula)
espaço
THEN
AND
  (vírgula)
```

Além disso, deve existir pelo menos um espaço entre dois nomes. Títulos de divisões (DATA DIVISION, PROCEDURE DIVISION etc.), seções (FILE SECTION) e nome de parágrafos devem terminar com um ponto seguido de pelo menos um espaço.

Exemplos:

```
SUBTRACT A, B AND C FROM D.
MOVE X TO Y; MOVE Z TO W. GO TO PAR-UM.
```

O ponto final deve ser usado imediatamente após o último elemento da sentença e também deve estar seguido por pelo menos um espaço. A mesma observação é válida para os separadores (,) e (;).

7.7. REGRAS DE NOTAÇÃO DO FORMATO GERAL DOS COMANDOS E CLÁUSULAS

Para a descrição dos comandos e cláusulas da linguagem COBOL será usada a seguinte simbologia que facilita o entendimento do **Formato Geral** ou **Descrição Genérica** desses elementos (Ver Apêndice A).

7.7.1. Palavras-chave

Aparecem em **Letras Maiúsculas Sublinhadas** e indicam determinada ação ou função do comando ou sentença, sendo o seu uso obrigatório onde foi indicado.

Exemplos:

ADD, PICTURE, WRITE, VALUE etc.

7.7.2. Palavras reservadas opcionais

Aparecem em **Letras Maiúsculas Não Sublinhadas** e são usadas para melhorar o entendimento do comando.

Exemplos:

IS, ON, ARE, CLASS IS.

7.7.3. Nomes de variáveis e constantes

Aparecem em **Letras Minúsculas** e representam os valores de informação que devem ser definidos pelo programador.

Exemplos:

Constante, nome-de-variável, nome-de-parágrafo, etc.

7.7.4. Par de colchetes "[ ]"

É usado para indicar que o item nele contido é de uso opcional, dependendo da necessidade do programador.

Exemplo:

IS [*NOT*] pode significar "IS" ou "IS NOT".

7.7.5. Par de chaves "{ }"

Contém itens dispostos verticalmente, podendo ocorrer apenas um deles. Exemplo:

$$\left\{ \begin{array}{l} \underline{POSITIVE} \\ \underline{NEGATIVE} \\ \underline{ZERO} \end{array} \right\}$$ significa "POSITIVE" ou "NEGATIVE" ou "ZERO".

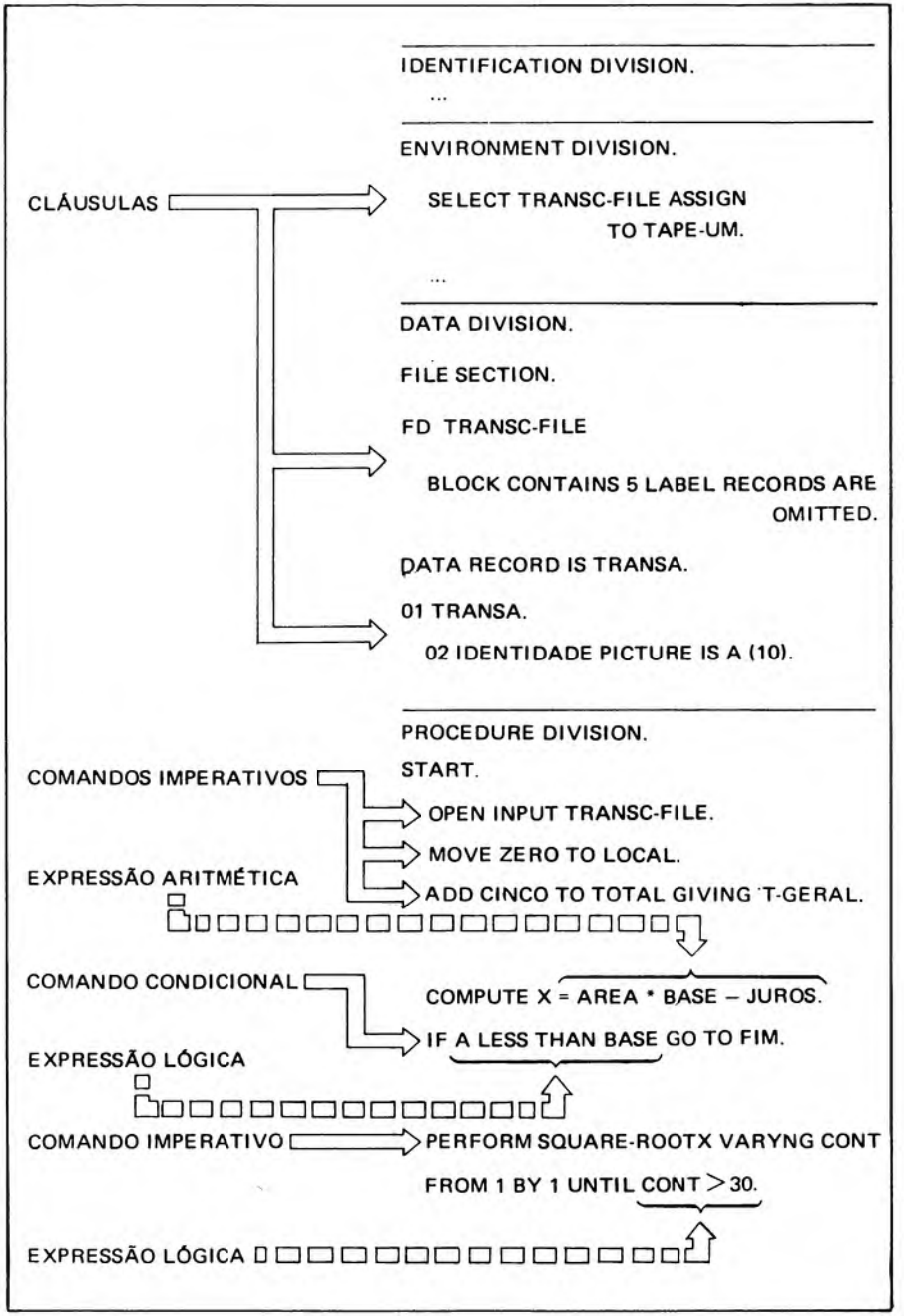


Figura 7.1. *Sentenças: comandos, cláusulas e expressões em COBOL.*

7.7.6. Seqüência de três pontos " ... "

Indica a repetição do item imediatamente anterior e que pode ser um sinal, nome de dado ou trecho de sentença. A repetição é de um número qualquer de vezes.

Exemplo:

nome [, nome ...] — significa "nome" e repetição opcional de "nome" separado por vírgula.

7.7.7. Exemplo de um formato geral

```
ADD { literal-1      } [ [ { , literal-2      } ... ]
    { nome-de-dado-1 }     { , nome-de-dado-2 }

    { TO    } nome-de-dado-3  [ ROUNDED ] [ ; ON SIZE ERROR comando]
    { GIVING }
```

O exemplo possibilita, entre outras formas possíveis, o uso de:

```
ADD A TO B.
ADD A B C GIVING D ROUNDED ON SIZE ERROR GO TO PARAGRAFO-UM.
```

Notar que é obrigatório usar TO ou GIVING mas é proibido usar ambos.

EXERCÍCIOS DE RECAPITULAÇÃO

1. Quais são os elementos que formam uma sentença em COBOL?
2. Dar exemplos de operação relacional e operação lógica.
3. Dar exemplos de expressão condicional e expressão matemática.
4. Dar exemplos de comandos imperativos e comandos condicionais.
5. O que são cláusulas de descrição? Dar exemplos.

EXERCÍCIOS DE APLICAÇÃO

1. Supondo que os nomes X, Y, Z, K tenham valores 10, 2, 20.5 e 2 respectivamente, qual é o valor final das expressões:
   (X + Y + Z) *K
   (X – Y) / (Z * K) ** 2
2. Com os mesmos valores de X, Y, Z e K dizer qual o valor das seguintes expressões condicionais (a resposta deve ser VERDADEIRO ou FALSO).
   X + Y IS GREATER THAN Z
   (X + Y) * K IS NOT GREATER THAN Z
   (X IS GREATER THAN Y) AND (Z IS LESS THAN 50)
   (X IS EQUAL TO 10) OR (Z IS NOT EQUAL TO 10) OR (K IS LESS THAN X)
3. Corrigir os erros de pontuação e de margem no programa a seguir:

```
Margem
A    B
     PROCEDURE DIVISION. PARAGRAFO-UM
     ADD X TO Y. SUBTRACT XFROM Z IF SOMA IS GREATER
THEN TOTAL GO TO PARAGRAFO-FINAL;
     PARAGRAFO-DOIS. ADD X, Y, Z AND W TO TOTAL.
PARAGRAFO-FINAL. WRITE X.
     IF COMENT IS EQUAL TO 'XYZ' GO TO Z.
```

8 IDENTIFICATION DIVISION E ENVIRONMENT DIVISION

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

IDENTIFICATION DIVISION
- Formato Geral

ENVIRONMENT DIVISION
- Formato Geral
- Seções do ENVIRONMENT DIVISION:
  - a) CONFIGURATION SECTION:
    SPECIAL-NAMES
  - b) INPUT-OUTPUT SECTION:
    FILE-CONTROL: A cláusula SELECT ... ASSIGN
    I-O CONTROL: A cláusula APPLY

EXEMPLO ILUSTRADO

Exercícios

8.1. IDENTIFICATION DIVISION

8.1.1. Formato geral

```
      A          B
  { IDENTIFICATION } DIVISION.
  { ID             }
  PROGRAM-ID. nome-do-programador.
  [ AUTHOR. comentários. ]
  [ INSTALLATION. comentários. ]
  [ DATE-WRITTEN. comentários. ]
  [ DATE-COMPILED. comentários. ]
  [ SECURITY. comentários.]
  [ REMARKS. comentários. ]
```

Esta divisão serve como identificação e informação sobre o programa e não tem efeito sobre o *programa-objeto* a ser processado. Entretanto, todo programa em COBOL deve possuir o IDENTIFICATION DIVISION que é sempre a primeira divisão do programa.

Além do título de divisão, o único elemento de uso obrigatório é o PROGRAM-ID, que deve estar seguido pelo nome do programa. Os demais elementos são de uso opcional e servem apenas como orientação para o usuário e ao programador.

Exemplo:

O caso mais simples desta divisão seria:

```
A    B
IDENTIFICATION DIVISION
PROGRAM-ID.  'PROGTEST'.
```

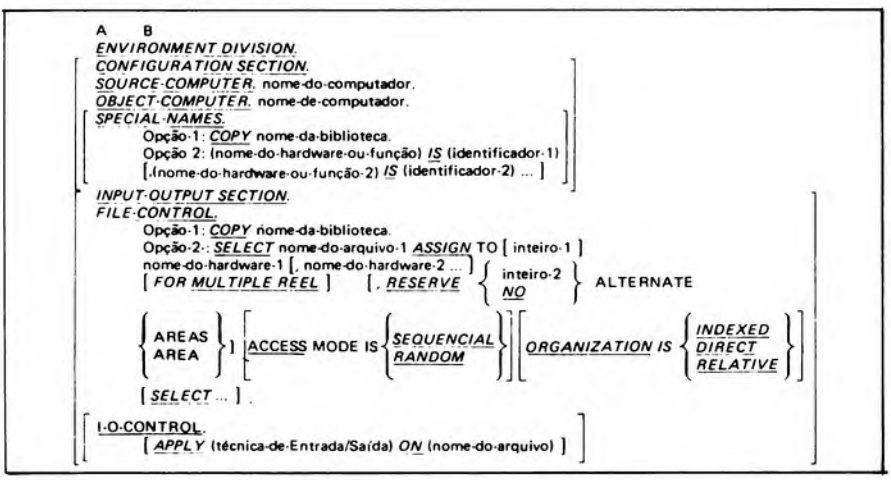
Outro exemplo de preenchimento seria:

```
A    B
IDENTIFICATION DIVISION.
PROGRAM-ID.  'F-PAGAMENTO'.
AUTHOR.  ANALISTA-FRANCISCO.
INSTALLATION.  BRASUS/E/CIA.
DATE-WRITTEN.  JAN 23. 78.
DATE-COMPILED.  JAN 23. 78.
REMARKS.
        ENTRADAS DO ARQUIVO-MESTRE E ARQUIVO-ATUALIZAÇÃO.
        SAIDA PARA ARQUIVO-ATUALIZADO E ARQUIVO-SAIDA.
```

Alguns computadores usam para identificação do programa somente os oito primeiros caracteres do nome colocado no PROGRAM-ID, e, além disso, convertem o hífen usado em caráter zero. Esta identificação aparecerá no topo das páginas de listagem do programa.

8.2. ENVIRONMENT DIVISION

8.2.1. Formato geral



8.2.2. Seções de preenchimento do ENVIRONMENT DIVISION

Por ser a divisão que descreve o computador onde será processado o programa, o preenchimento desta divisão depende bastante do computador usado, havendo a necessidade de consultar o manual COBOL desse computador. O uso do ENVIRONMENT DIVISION pode ser completamente opcional, deixando ao próprio computador o preenchimento automático das especificações.

Entretanto, se algum item dessa divisão for usado, devemos incluir os seguintes itens:

A	B
ENVIRONMENT DIVISION.	
CONFIGURATION SECTION.	
SOURCE-COMPUTER.	nome do computador
OBJECT-COMPUTER.	nome do computador
INPUT-OUTPUT SECTION.	
FILE-CONTROL.	
I-0-CONTROL.	

a) Configuration section

Tanto a seção toda como qualquer parágrafo que a compõe são opcionais.

Alguns itens mais gerais desta seção são:

- *SOURCE-COMPUTER:* é o item que recebe o nome ou o código do computador no qual o programa-fonte (SOURCE PROGRAM) em Cobol é compilado. O formato do nome ou código é livre, uma vez que este parágrafo é de uso opcional.
- *OBJECT-COMPUTER:* é o item que recebe o nome ou a descrição do computador no qual o programa-objeto deve ser processado. É também de uso opcional.
- *COPY (nome da biblioteca):* o compilador COBOL de um computador possui geralmente uma biblioteca interna (lista de informações e programas catalogados) onde guarda diversos itens padronizados ou úteis para o programa, tais como nome ou descrição do computador, nomes especiais associados aos equipamentos etc.

  Para evitar que o programador tenha o trabalho de escrever tais nomes, existe a opção COPY, onde basta citar o nome de biblioteca que possui o nome ou lista de nomes que se deseja.
  Por exemplo: se na biblioteca chamada XOBJETO já existir a descrição do computador-objeto, basta escrever:

  OBJECT-COMPUTER. COPY XOBJETO. em vez de

  OBJECT-COMPUTER. nome-do-computador.

- *SPECIAL-NAMES:* são nomes designados a diversos equipamentos do computador ou às chaves (SWITCHES) e técnicas usadas pelo programa. Se no PROCEDURE DIVISION nenhum nome que designa especificamente algum equipamento ou chave do computador for usado, este parágrafo pode ser omitido. Se a biblioteca contém uma lista de nomes especiais para equipamentos ou chaves arquivados sob o nome XSPECIAL, então basta escrever:

  SPECIAL-NAMES. *COPY* XSPECIAL.

  Caso contrário, o nome do dispositivo ou técnica especial usada no programa deve ser definido pelo próprio programador. Por exemplo:

  SPECIAL-NAMES. (nome do dispositivo) IS (nome dado pelo programador)
  SPECIAL-NAMES. (nome da chave-1) IS (nome dado pelo programador)
  SPECIAL-NAMES. DECIMAL POINT IS COMMA. (Usado para converter o ponto decimal em vírgula).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "CALC-JUROS".
AUTHOR. SR. XAVIER.
INSTALLATION. CPD/ABC.
DATE WRITTEN. 31/DEZ/19XX.
REMARKS. ESTE PROGRAMA CONTEM ...
    E CALCULA OS SEGUINTES VALORES, etc., etc.
ENVIRONMENT DIVISON.
CONFIGURATION SECTION.
SOURCE COMPUTER.    SYSTEMA LMN/007.
OBJECT COMPUTER.    SYSTEMA LMN/007.
INPUT-OUTPUT SECTION.
FILE-CONTROL
    SELECT ARQUIVO-ENTRADA-A ASSIGN
        TO CARD-READER.
    SELECT ARQUIVO-ENTRADA-B ASSIGN
        TO TAPE-1.
    SELECT ARQUIVO-SAIDA-NOVO ASSIGN
        TO TAPE-2.
    SELECT ARQUIVO-RELATORIO ASSIGN
        TO PRINTER.
```

Figura 8.1. *As duas primeiras divisões de um programa COBOL.*

b) Input-output section

É a seção onde os arquivos de entrada e de saída, bem como os métodos de controle de entrada e de saída, são especificados. Divide-se em dois parágrafos: FILE-CONTROL e I-O-CONTROL.

- *FILE-CONTROL:* é o parágrafo usado para associar cada arquivo, definido e usado pelo programa, a um dispositivo de Entrada ou Saída específico. É o único ponto do programa que associa, por exemplo, o ARQUIVO-MESTRE do programa à Leitora de Cartões ou o ARQUIVO-SAÍDA à Impressora do computador.
  Se a associação de arquivos aos dispositivos externos puder ser feita de modo automático e predefinido, basta usar o parágrafo

FILE-CONTROL. COPY (nome-da-biblioteca).

  - *SELECT ... ASSIGN:* Se não for usada a opção COPY acima, cada arquivo usado deve estar designado a um dispositivo do *hardware* através desta cláusula.

FILE-CONTROL. SELECT (nome do arquivo 1) *ASSIGN* TO (nome do dispositivo). [*SELECT* (nome do arquivo 2) *ASSIGN* TO (código do dispositivo)...]

  O "código do dispositivo" deve ser fornecido pelo Manual COBOL do computador em uso. Por exemplo, o sistema IBM usa código do tipo SYS007-UT-24005, etc.
  As cláusulas *ACCESS MODE* e *ORGANIZATION:* serão discutidas no capítulo sobre métodos de acesso a dados.

- *I-O-CONTROL:* é o parágrafo usado para escolher técnicas especiais de Entrada/Saída. Em geral estas técnicas estão predefinidas em biblioteca e basta usar:

I-O-CONTROL. APPLY (técnica de Entrada/Saída) *ON* (nome de biblioteca)

Observação:

As formas corretas de "nome do computador", "nome-da-biblioteca", "código do dispositivo" etc. devem ser consultadas no manual COBOL do computador que está sendo usado.

EXERCÍCIOS DE RECAPITULAÇÃO

1. Descrever as seções do ENVIRONMENT DIVISION.
2. Descrever os itens: SPECIAL NAMES, FILE-CONTROL e I-O-CONTROL.

EXERCÍCIOS DE APLICAÇÃO

1. Dar um exemplo de IDENTIFICATION DIVISION e ENVIRONMENT DIVISION, usando somente as especificações obrigatórias.
2. Dar um exemplo de IDENTIFICATION DIVISION e ENVIRONMENT DIVISION, usando todas as especificações possíveis.

9 DATA DIVISION: AS SEÇÕES E OS NÍVEIS DE DADOS

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

DATA DIVISION
- Finalidades

AS SEÇÕES DO DATA DIVISION
- FILE SECTION: especifica os arquivos de Entrada e Saída
- WORKING-STORAGE SECTION: especifica as áreas de trabalhos e constantes
- LINKAGE SECTION: especifica os parâmetros de sub-rotinas
- REPORT SECTION: especifica relatórios padronizados

DESCRIÇÃO DE ARQUIVO, REGISTRO OU DADOS

Os Níveis dos Dados
- Nível FD: para cada arquivo usado.
- Nível 01: para cada registro de um arquivo.
- Nível 02 a 49: para subcampos de registro.
- Nível 77: para área de trabalho do WORKING-STORAGE SECTION.
- Nível 88: para nome de condição.

QUALIFICAÇÃO DE NOMES POR "OF" E "IN"

Exemplos: NOME OF FIRMA-UM
NOME OF CODIGO-FICHA

FORMATO GERAL

EXEMPLO ILUSTRADO

Exercícios

9.1. DATA DIVISION

É uma das partes vitais de um programa COBOL. É bastante fácil aprender a preencher o PROCEDURE DIVISION, que é a parte operativa do programa e contém os comandos de operação, tais como somar, subtrair, mover dados etc. Entretanto, cada dado, cada registro ou cada arquivo usado pelos comandos do PROCEDURE DIVISION precisa ser especificado pelo DATA DIVISION que reserva espaço na memória, fornecendo o nome, o tipo, o tamanho, a localização e eventualmente o valor de cada um desses itens. Sem essa descrição o programa simplesmente não pode funcionar.

Recomenda-se voltar ao Capítulo 4, que apresenta EXEMPLOS de programa COBOL INTRODUTÓRIO, para visualizar bem a função do DATA DIVISION.

9.2. AS SEÇÕES DO DATA DIVISION

O DATA DIVISION é formado pelas seções seguintes:

- *FILE SECTION* onde são especificados todos os arquivos usados no programa. O FILE SECTION até pode ser omitido, se não for usado arquivo de Entrada ou Saída, o que é quase impossível de ocorrer na prática, pois limitaria a operação de Entrada/Saída só pelos comandos ACCEPT e DISPLAY.
- *WORKING-STORAGE SECTION* onde são especificados todos os elementos de uso temporário do programa, e que não irão aparecer nem no arquivo de entrada nem no de saída, ou seja, descreve os elementos da área de trabalho.
- *CONSTANT-SECTION* onde são especificadas todas as constantes usadas no programa (exceto as constantes literais e figurativas). Entretanto esta seção foi abolida na maioria das linguagens COBOL, pois sua função pode ser suprida pelo WORKING-STORAGE SECTION.
- *LINKAGE SECTION* que só aparece em programa usado como sub-rotina e serve para descrever os parâmetros transferidos pelo programa principal.
- *REPORT SECTION* que serve para descrever relatórios padronizados e só aparece em sistemas especiais (REPORT SECTION não será discutido neste trabalho).

9.3. DESCRIÇÃO DE ARQUIVO, REGISTRO OU DADOS

Cada seção do DATA DIVISION é formada pela descrição dos arquivos e respectivos registros ou simplesmente pela descrição direta dos dados como acontece no WORKING-STORAGE SECTION. A descrição propriamente dita de arquivos, registros e dados será explicada nos capítulos seguintes. Neste capítulo discutiremos os Níveis de dados.

9.3.1. Os níveis de dados

Como já vimos no Capítulo 3, os dados estão relacionados e categorizados entre si em hierarquias diferentes chamadas níveis, conforme Figura 9.1

1. Em COBOL, o nível mais elevado é atribuído ao ARQUIVO (FILE) de dados que possui código FD, e pode ser um arquivo de disco magnético ou fita, arquivo de cartões perfurados, ou arquivo de saída em formulários contínuos de impressão. Deve-se notar que, para cada arquivo *designado* pelo SELECT no ENVIRONMENT DIVISION, há a definição de um e só um arquivo *descrito* pelo código FD no DATA DIVISION.
2. A seguir vem o REGISTRO (RECORD) de dados que possui código 01 ou 1 e pode ser um registro de um disco magnético ou fita magnética, ou um cartão perfurado ou uma linha impressa em um formulário de papel.
3. Cada registro pode ser subdividido em diversos subcampos de nível 02 ou 2, estes podem ser subdivididos em subcampos de nível 03 e assim por diante.
4. O maior valor de nível que pode ser usado é 49. Além disso existem códigos especiais (77 e 88), sendo que o nível 77 é usado no WORKING-STORAGE SECTION e o 88 é usado para nome de condições.
5. Os códigos FD, 01 e 77 são sempre escritos na margem A do formulário. Os demais códigos podem ser escritos em qualquer posição após a margem B.
6. O nível 88 é usado para indicar os nomes das condições (vide seção sobre nomes e identificadores) de um dado item. Isto é, os nomes de nível 88 indicam todas as condições possíveis que um dado item pode assumir.

Exemplos:

```
03    COR-DE-CABELO PICTURE 9.
      88 CASTANHA VALUE IS 1.
      88 NEGRA VALUE IS 2.
      88 LOURA VALUE IS 3.
      88 RUIVA VALUE IS 4.
```

Neste exemplo o nome do dado é COR-DE-CABELO e as condições que assume são: CASTANHA, NEGRA, LOURA e RUIVA respectivamente com valores 1, 2, 3 e 4.

O uso de nome de condição pode abreviar a descrição de condição de teste nos comandos IF. Por exemplo, se não usar o nome de condição, seria necessário usar o comando

```
IF COR-DE-CABELO IS EQUAL TO 1 GO TO PARAG-CASTANHA
```

mas definindo-se o nome de condição, basta usar o comando

```
IF CASTANHA GO TO PARAG-CASTANHA
```

O uso de nomes de condições será melhor explicado no comando de condição IF e com variáveis subscritas.

9.4. QUALIFICAÇÃO DE NOMES POR "OF" E "IN"

Dois arquivos ou registros podem ter subcampos de mesmo nome, os quais são diferenciados por OF ou IN seguido pelo nome do registro ou campo de ordem superior. O arquivo pode qualificar registros de nível 01.

Exemplo:

No arquivo seguinte:

```
FD ARQUIVO-UM.
01 FICHA-1
   02 NOME
   02 CODIGO
01 FICHA-2
   02 NOME
   02 CODIGO
```

é possível usar os seguintes nomes qualificados:

```
NOME OF FICHA-1, NOME IN FICHA-2, CODIGO IN FICHA-1,
FICHA-1 OF  ARQUIVO-UM, etc.
```

9.5. FORMATO GERAL

```
    A    B
    DATA DIVISION.
    FILE SECTION.
[   FD  { nome-do-arquivo  COPY nome-da-biblioteca }                    ]
        { nome-do-arquivo e sua descrição          }
[   01 ou 1 nome-do-registro e sua descrição                            ]
    [       [ 02 a 49  nome-dos-subcampos e suas descrições ]          ]
    [       [ 88       nome-de-condição e suas descrições   ]          ]
[   [ 01      ... ]                                                    ]
    [ FD      ... ]
```

WORKING-STORAGE SECTION.
[77 nome-da-área-de-trabalho e sua descrição]
[01 ou 1 (análogo ao registro do FILE SECTION)]
...
[
LINKAGE SECTION. (só usado em sub-rotinas)
[77 nome-do-parâmetro e sua descrição]
[01 ou 1 nome-de-registro-parâmetro (análogo ao registro do FILE SECTION)]
]

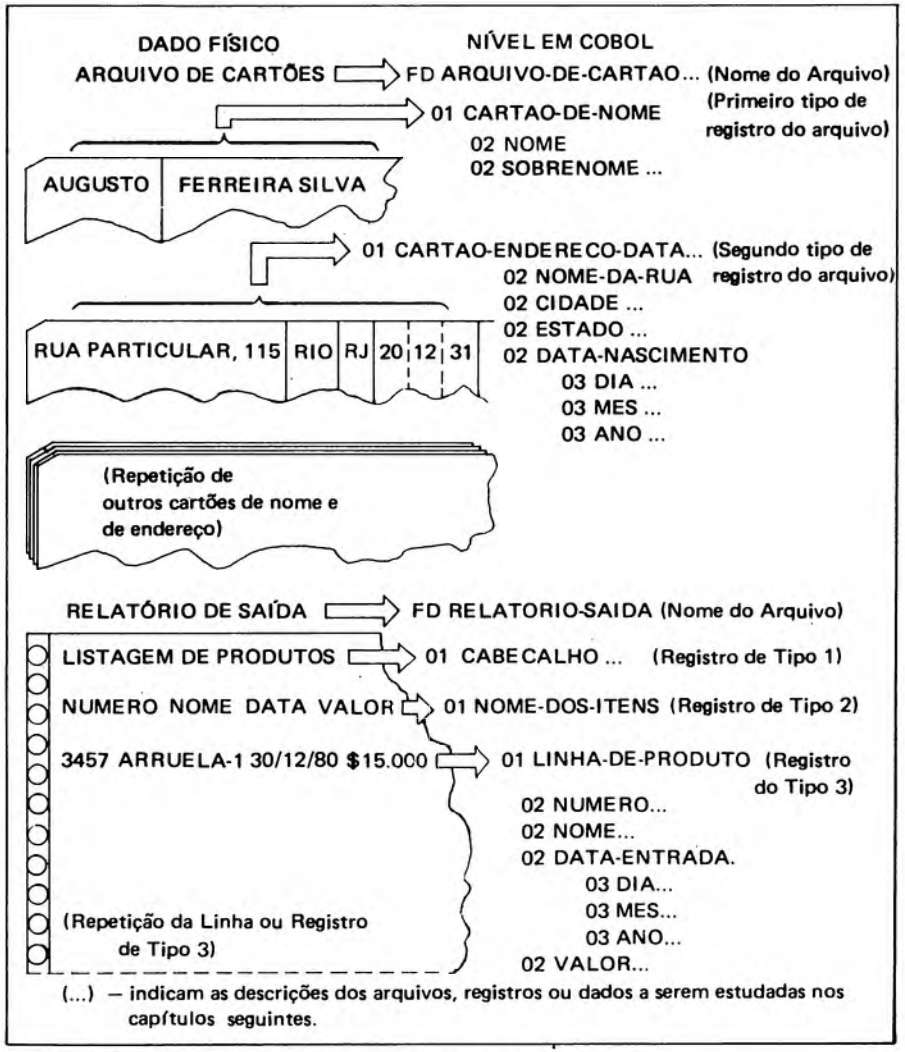


Figura 9.1. *Associação dos nomes e níveis de dados.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Quais são as seções do DATA DIVISION e para que servem?
2. Quais são os principais níveis de descrição de arquivos e dados? Dar exemplos.
3. O que é "qualificação de nomes" e como se faz?

EXERCÍCIOS DE APLICAÇÃO

1. Em um arquivo-mestre existem registros de estoques que contêm, cada um, as seguintes partes:
   Número do item, Unidade, quantidade existente, custo e aplicação. O campo de custo contém custo por unidade e o custo total da quantidade existente e o campo de aplicação possui um dos quatro valores seguintes: tipo 1, 2, 3 e 4. O campo de Unidade pode ser: KG, DÚZIA ou METRO.
   Estruturar esse arquivo em níveis no DATA DIVISION.
2. Um arquivo técnico de bibliografias deve conter os seguintes itens, para cada bibliografia guardada:
   - título do artigo;
   - nome do autor;
   - n.º do volume e nome da revista a que pertence;
   - mês e ano da publicação;
   - n.º de página inicial e final do trabalho;
   - código da prateleira ou estante;
   - código da área (Engenharia, Economia, Contabilidade etc.);
   - código da biblioteca.

   Estruturar esse arquivo em níveis, no DATA DIVISION.

10 DATA DIVISION: A DEFINIÇÃO DE ARQUIVOS

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

FORMATO GERAL
AS OPÇÕES
- Opção 1: FD (nome de arquivo) COPY (nome da biblioteca)
- Opção 2: com cláusulas mínimas mais usadas
- Opção 3: contendo todas as cláusulas possíveis

A Cláusula LABEL RECORDS (obrigatória).
A Cláusula DATA RECORDS (opcional).
A Cláusula RECORDING MODE (pode ser omitida).
A Cláusula BLOCK CONTAINS (omitida quando fator de bloco é 1).
A Cláusula RECORD CONTAINS (pode ser omitida).
A Cláusula VALUE de descrição de arquivo (em geral, desnecessária).
EXEMPLOS

Exercícios

10.1. FORMATO GERAL

```
A    B
FILE SECTION
Opção 1:
FD   nome-do-arquivo COPY nome-da-biblioteca.

Opção 2:
FD   nome-do-arquivo:
     LABEL  { RECORDS ARE }  { STANDARD                    }
            { RECORD  IS  }  { OMITTED                     }
                             { nome-do-dado                }
                             { nome-da-biblioteca IN LIBRARY }
     DATA [ { RECORD IS  }  nome-do-registro-1 [ , nome-do-registro-2,... ] ]
            { RECORDS ARE}
```

Opção 3:

```
FD   nome-do-arquivo [ ; RECORDING MODE IS modo ]
     [ ; BLOCK CONTAINS [ inteiro-1 TO ] inteiro-2 { RECORDS    }
                                                   { CHARACTERS }
     ; LABEL (Repete LABEL da Opção 2)
     [ ; VALUE OF nome-do-dado-1 IS { nome-do-dado-2 }
                                    { literal        }
       [ , nome-do-dado-3 IS ... ] ]

     [ ; DATA RECORD IS (Repete DATA RECORD da Opção 2) ].
```

10.2. AS OPÇÕES

Um arquivo é definido e descrito por uma das opções apresentadas na forma geral, sendo que o código de nível FD na margem A indica o início da descrição de um arquivo. A descrição de um arquivo começa com o item FD e termina com um ponto (.) colocado após a última cláusula de descrição.

- OPÇÃO 1: *FD (Nome do arquivo) COPY (nome da biblioteca).* Indica que a descrição do arquivo está em forma padronizada, de acordo com a descrição guardada em uma biblioteca do COBOL. Basta o usuário consultar o manual COBOL do computador em uso e colocar o (nome de biblioteca) lá indicado. O (nome de arquivo) deve ser o mesmo usado na cláusula SELECT, do ENVIRONMENT DIVISION.
- OPÇÃO 2: É a opção que deve ser preenchida pelo usuário e contém somente as cláusulas mínimas mais usadas.
- OPÇÃO 3: É a opção completa, pois contém todas as cláusulas possíveis.

10.3. A CLÁUSULA LABEL RECORDS

Os LABELS ou os rótulos (ou etiquetas) de um arquivo são registros especiais de identificação colocados no começo e no fim do arquivo. O "rótulo de início" informa o número e código da fita, data, código do JOB etc., ao passo que o "rótulo de fim", chamado também de "sentinela" no caso de arquivo de cartão, contém os mesmos códigos além do código especial de fim de arquivo. Os rótulos são escritos por ocasião da gravação do arquivo e seu preenchimento é padronizado ou deve-se consultar o manual COBOL do computador.

O uso de LABEL RECORDS é *sempre obrigatório* em um arquivo, mas depende da política adotada pela instalação. Os casos mais usados são os seguintes:

- Para arquivo de Entrada: LABEL RECORDS { IS / ARE } { STANDARD / OMITTED }.
- Para arquivo de Saída: LABEL RECORDS ARE STANDARD (se o programador quiser os rótulos gravados no arquivo).
- Para Arquivo de Saída: LABEL RECORDS ARE OMITTED (se o programador não quiser ter rótulos no arquivo).

10.4. A CLÁUSULA DATA RECORDS

Esta é uma descrição que era de *uso obrigatório* na descrição de um arquivo. Ela cita o nome do registro ou registros que compõe o arquivo. Somente devem ser citados os nomes de registros com nível 01. Os arquivos são em geral formados

pela repetição de um mesmo tipo de registro, bastando citar, então um só nome de registro. Arquivos de saída podem ter mais de um registro de tipo diferente. Por exemplo: um registro de cabeçalho ou título, um registro de itens que se repetem e um registro de totalização. Neste caso, existem 3 registros distintos que devem ser citados no DATA RECORDS e descritos um a um no nível 01.

Entretanto, como o nome de registro deve aparecer em primeiro lugar, com o nível 01, o compilador sempre pode associar um registro ao arquivo. Isso tornou esta cláusula de uso *opcional,* sendo tratado pelo compilador, como comentário.

Exemplos:

```
DATA RECORD IS ITEM-DE-ESTOQUE.
DATA RECORDS ARE TITULO ITEM TOTAL.
```

10.5. A CLÁUSULA RECORDING MODE

Alguns computadores possuem várias maneiras de gravação de um arquivo que podem variar em velocidade ou tamanho do registro. Deve-se escolher então o modo correto de gravação, que é fixado pelo fabricante do sistema. Se esta descrição não for usada, assume-se que o modo padrão de gravação está em uso. Na maioria dos casos esta descrição é omitida.

10.6. A CLÁUSULA BLOCK CONTAINS

Para efeito de leitura ou gravação os registros de uma fita magnética são agrupados em blocos e a leitura ou gravação é feita de bloco em bloco. O número de registros em um bloco, no caso da fita magnética, pode ser determinado pelo programador para aumentar a eficiência da leitura ou da gravação.

A descrição de bloco deve citar o número de registros em um bloco se todos os registros forem de mesmo tamanho, caso contrário, deve-se citar o número total de caracteres do bloco. O cartão perfurado e a linha da impressora são considerados como bloco com único registro, e dizemos que o fator de bloco é 1.

Na fita magnética e no disco magnético, o fator de bloco pode também ser 1, isto é, cada bloco contém apenas um registro. Nos arquivos com o fator de bloco 1, a descrição BLOCK CONTAINS pode ser omitida.

Exemplos:

```
BLOCK CONTAINS 10 RECORDS
BLOCK CONTAINS 400 CHARACTERS
BLOCK CONTAINS 80 CHARACTERS
BLOCK CONTAINS 100
```

No último caso, subentende-se que o bloco contém 100 caracteres e não 100 registros.

10.7. A CLÁUSULA RECORD CONTAINS

Descreve o número de caracteres em um registro que compõe o bloco. Mas, como tal número sempre é fornecido na descrição de registros, esta descrição pode ser omitida.

10.8. A CLÁUSULA VALUE

Esta descrição pode ser usada pelo programador que quiser colocar valores ou códigos específicos no rótulo ou LABEL. Se for arquivo de saída o valor ou código será escrito no rótulo. Se for arquivo de entrada, o VALUE verificará se existe o valor ou código especificado no rótulo, acusando erro se for o caso. Entretanto, na maioria dos compiladores, esta cláusula é tratada como simples comentário.

Exemplo:

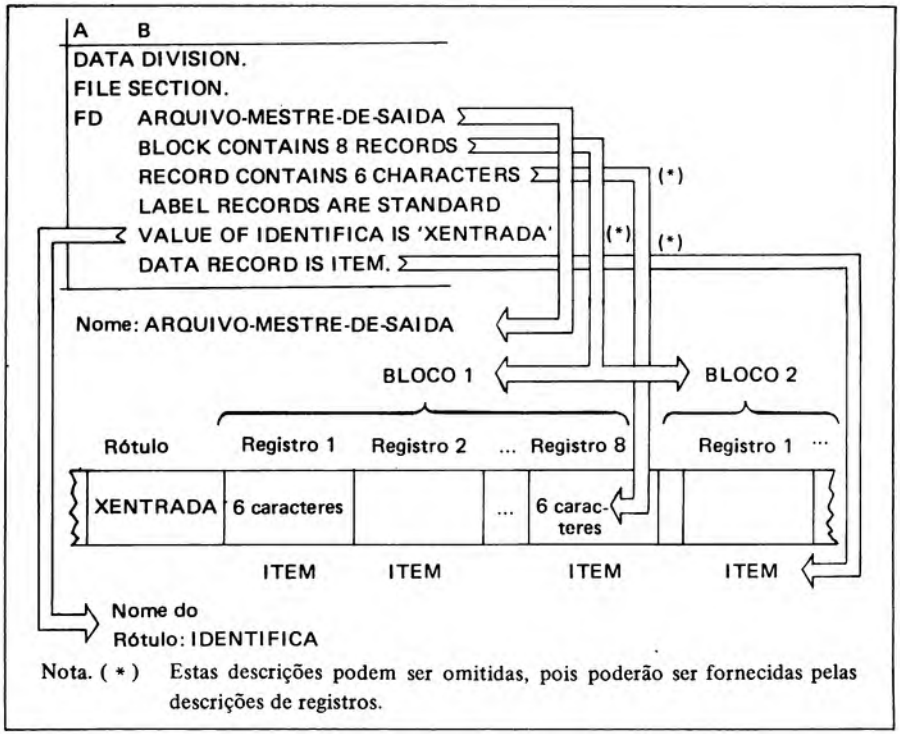
```
VALUE OF CODIGO IS 'XY124'
```

Arquivo de Saída: XY124 será escrito no rótulo como sendo o valor do CÓDIGO.

Arquivo de Entrada: o rótulo será testado para verificar se existe o valor XY124 no CÓDIGO, acusando erro (deixando de ler o arquivo) se não existir.

10.9. EXEMPLO DE DEFINIÇÃO DE ARQUIVO

Um arquivo é chamado ARQUIVO-MESTRE-DE-SAIDA e os registros chamados de ITEM são de tamanho fixo (6 caracteres) e blocados de 8 em 8 registros. O rótulo possui o nome IDENTIFICA, que deve conter o código XENTRADA. A sua descrição ilustrada é:



Nota. (*) Estas descrições podem ser omitidas, pois poderão ser fornecidas pelas descrições de registros.

Figura 10.1. *Ilustração da definição de um arquivo.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Dizer o que faz a OPÇÃO 1 da definição de Arquivos.
2. Dar exemplos de uso das seguintes cláusulas:
   LABEL RECORDS
   DATA RECORDS
   RECORDING MODE
   BLOCK CONTAINS
   VALUE
3. Explicar cada um dos exemplos acima citados.
4. Explicar cada uma das cláusulas de definição do arquivo do EXEMPLO DE DEFINIÇÃO DE ARQUIVO da seção 10.9.

EXERCÍCIOS DE APLICAÇÃO

1. Escrever as sentenças para descrição completa do seguinte arquivo:
   O arquivo é chamado CADASTRO-DE-ITENS, e contém registros com tamanho fixo de 120 caracteres, e chamados de ITEM. O fator de bloco é 1.
2. Explicar o que significa a seguinte descrição:
   FD ARQUIVO-X BLOCK CONTAINS 10 RECORDS LABEL RECORDS ARE STANDARD VALUE OF IDENTIFICA IS 'X100' DATA RECORDS ARE NOME, CUSTO.

11 DATA DIVISION: DEFINIÇÃO DE UM DADO PELA CLÁUSULA PICTURE

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

ITEM DE GRUPO E ITEM DE DADO
- Item de grupo: pode ser subdividido
- Item de dado: não pode ser subdividido

A CLÁUSULA PICTURE OU PIC

Caracteres usados na descrição pelo PICTURE.
- A: representa caráter alfabético.
- X: representa caráter alfanumérico.
- 9: representa dígito.
- V: indica posição do ponto implícito.
- S: indica ocorrência do sinal.
- P: representa escala do valor numérico.

Repetição do A, X e 9.
Ex.: A(10), X(5), 9(2)

Número máximo de caracteres usados na descrição.
Limite de Valores Numéricos.
Edição de dados pelo PICTURE (Capítulo 22).

EXEMPLO ILUSTRADO DO USO DO PICTURE

Exercícios

11.1. ITEM DE GRUPO E ITEM DE DADO

Cada arquivo descrito por uma cláusula de definição de arquivo (nível FD) deve estar formado por registros. (RECORDS)

Cada registro, por sua vez, pode estar subdividido em subcampos que se dividem novamente até que chegue a um item final indivisível e que passa a ser chamado de *dado* ou *item de dado.*

Se o registro (Nível 01) não for subdividido ele será o próprio dado.

Qualquer registro ou subcampo que admite subdivisão é chamado de *item de grupo,* pois contém um grupo de dados.

A seguir veremos a descrição de item de dado e item de grupo, através da cláusula PICTURE.

11.2. A CLÁUSULA PICTURE OU PIC

11.2.1. Formato geral

```
{ PICTURE } IS (descrição)
{ PIC     }
```

É a cláusula mais importante DATA DIVISION, dada a maneira compacta e flexível de descrever um item de dado, atribuindo o tipo de classe, o comprimento e outras informações para a edição.

O PICTURE só pode ser usado em um item de dado (dado ou campo que não pode ser subdividido). Substitui muitas outras cláusulas como o SIZE, CLASS etc. e que foram abolidas em vários compiladores.

11.2.2. Caracteres usados na descrição do PICTURE

Os seguintes caracteres são usados para formar a descrição básica (pois existem outros caracteres de edição a serem vistos no Capítulo 22) de um PICTURE:

A	Representa um caráter alfabético ou um espaço e é usado para descrever nome de pessoas, cabeçalhos etc.
X	Representa um caráter alfanumérico, isto é, dígitos, letras ou qualquer sinal aceitável pelo computador. Pode sempre ser usado no lugar do caráter "A", pois mesmo em nome de pessoas podem ocorrer caracteres que não são alfabéticos. Exemplo: D'AVILA, JR. etc.
9	Representa um dígito decimal, e é usado para descrever valores numéricos.
V	Representa a posição de um ponto decimal (vírgula no nosso caso) implícito dos números puros usados na operação aritmética. Não ocupa espaço na memória, pois é apenas uma indicação da posição do ponto decimal.
S	Indica que o item tem sinal, isto é, pode assumir valor positivo ou negativo. Se omitíssemos o S, o item seria sempre considerado positivo, mesmo que o resultado de uma operação aritmética colocasse no local descrito um valor negativo. Não ocupa espaço na memória e deve ser o primeiro caráter usado na descrição.
P	Representa variação na escala decimal, preenchendo com zero cada local contendo este caráter e deslocando o ponto decimal para a direita ou para a esquerda do último P. Equivale a multiplicar (ou dividir) o valor original por dez para cada caráter P colocado à esquerda (ou direita) da posição do ponto decimal.

Observações:

- Somente os valores numéricos descritos por caracteres 9 (podendo aparecer V, S ou P) podem ser usados em operações aritméticas. Por exemplo, é possível descrever e ler os valores de DIA, MÊS e ANO usando as descrições XX, XX e XX para cada valor, mas não podemos usá-los em operações aritméticas, pois são itens alfanuméricos. Se quisermos calcular o valor de (DIA + 1), isto é, o DIA SEGUINTE, devemos descrever DIA através de 99.

- É possível misturar A, X ou 9 para descrever um mesmo item.
- V e S não ocupam espaço na memória, pois a representação interna do valor numérico em COBOL permite distinguir automaticamente o valor positivo ou negativo (através da notação binária denominada "complemento de dois") e a posição do ponto decimal (através da notação binária denominada "ponto flutuante").
- O uso do sinal operacional S não implica a sua impressão na operação de saída, a menos que o sinal seja editado antes. (Ver Capítulo 22).

Exemplos: (A, X e 9).

PICTURE IS AA ou PIC AA	Indica um item com 2 caracteres alfabéticos.
PICTURE IS XXX	Idem 3 caracteres alfanuméricos.
PICTURE IS 99999	Idem 5 algarismos.

Exemplo: (S e V)

PICTURE IS S99V99	Indica um item positivo ou negativo com 4 algarismos e ponto decimal após a segunda casa.

Exemplos: (P)

PICTURE IS 99PPV	Se o valor do item for 22 na memória, esta cláusula o transforma no valor 2200.
PICTURE IS VP99	O valor 22 da memória passa a ser .022.

Nota-se que P não ocupa espaço na memória, pois o tamanho do item é 2 algarismos (99) e se transforma no novo valor de 4 ou 3 dígitos só quando for solicitado para alguma operação.

11.2.3. Repetições de A, X ou 9

Repetições de mesmo caráter podem ser substituídas pelo caráter seguido pelo número de repetições cercado por parênteses:

Exemplo:

Um item com 3 caracteres alfabéticos e 5 numéricos pode ser descrito por:
PICTURE IS AAAXXXXX ou
PICTURE IS A(3)X(5).

11.2.4. Número máximo de caracteres usados na descrição

Cada descrição não deve exceder 30 caracteres. *Atenção:* o tamanho do item descrito pode exceder a 30.

Exemplos:

PICTURE IS X(50) a descrição usa 5 caracteres e representa um item de 50 caracteres alfanuméricos.
PICTURE IS 9999...999 a descrição usou 50 caracteres 9 (não é permitido).
50 vezes

11.2.5. Limites dos valores numéricos

Na notação binária do computador, um valor numérico negativo não possui um sinal negativo "–", mas assume outra forma binária diferente dos valores positivos, chamada de "complementos de 2". A descrição com o caráter S, indica, então, estes 2 tipos de valores.

Exemplos:

999	representa	valores de 000 a 999
S999	"	valores de -999 a +999
SV99	"	valores de -,99 a +,99
9V99	"	valores de 0,00 a +9,99

11.2.6. Edição de dados pela cláusula PICTURE

Será vista com detalhes e exemplos, no Capítulo 22.

11.3. EXEMPLO ILUSTRADO DO USO DO PICTURE

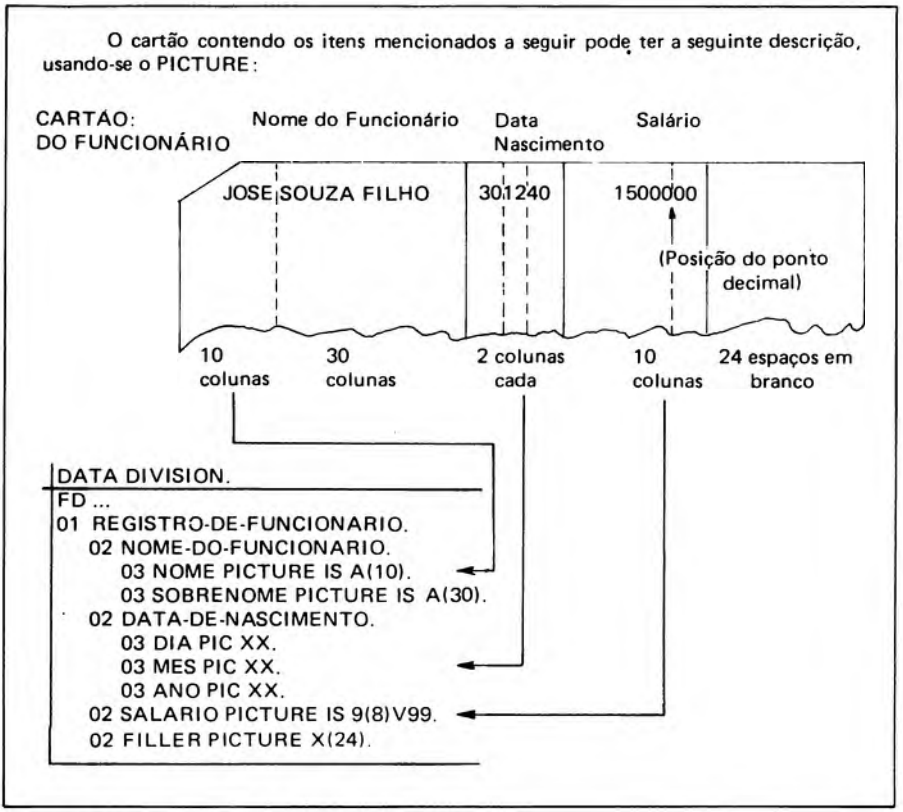


Figura 11.1. *Ilustração do uso do PICTURE.*

EXERCÍCIO DE RECAPITULAÇÃO

1. Dizer o que representa cada uma das seguintes descrições do PICTURE:

999V99	VPPP999	XXXX
9(6)	AAAA	X(9)9(2)
9(6)V	A(7)	

EXERCÍCIOS DE APLICAÇÃO

1. Supondo que exista na memória o valor 1257, dizer qual seria o valor ou representação que cada uma das descrições do Exercício de Recapitulação acima daria a esse valor para ser usado nas operações do programa (por exemplo, a descrição 999V99 trataria 1257 como se fosse o valor numérico 12.57). Indicar o caso de descrição errada.
2. Repita o exercício anterior, supondo que na memória exista o valor XYZ9 e depois 1.
3. Fornecer as descrições adequadas do PICTURE para os seguintes dados de um cartão perfurado:

NUMERO-DO-PRODUTO	Colunas 1 a 10
NOME-DO-PRODUTO	Colunas 11 a 30
PREÇO	Colunas 31 a 40
Taxa de Depreciação Anual	Colunas 41 a 45 (valor decimal que varia de 0.000 a 0.1000%).

12 DATA DIVISION: OUTRAS CLÁUSULAS PARA DEFINIR DADOS

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

A CLÁUSULA FILLER
- Preenche área com espaços ou zeros, sem atribuir nome.
  Exemplo: FILLER PICTURE X(5).

A CLÁUSULA USAGE
- Define o tipo de representação de dados.
  Exemplo: USAGE IS DISPLAY.

A CLÁUSULA VALUE
- Atribui valor inicial ao dado.
  Exemplo: VALUE IS 100. VALUE IS SPACES.

A CLÁUSULA OCCURS
- Define repetição de dados.
  Exemplo: 03 XZ-UM PICTURE X(10) OCCURS 5 TIMES.

A CLÁUSULA JUSTIFIED
- Encosta o dado à direita ou à esquerda do campo.
  Exemplo: JUSTIFIED RIGHT.

A CLÁUSULA SYNCHRONIZED
- Separa os dados em cada palavra da memória.

A CLÁUSULA REDEFINES
- Redefine área equivalente de dados.

FORMATO GERAL

EXEMPLO ILUSTRADO

Exercícios

12.1. A CLÁUSULA FILLER

Representa uma área à qual o programa não fará referência alguma, dispensando a atribuição de um nome. Representa também a ocorrência de espaços em branco ou zeros, conforme a área seja alfanumérica ou numérica. Após o FILLER somente a cláusula PICTURE deveria ser usada, mas alguns compiladores permitem o uso de VALUE ou SIZE.

Exemplos:

```
02 FILLER PIC X(5).  ←— Indica a ocorrência de 5 espaços.
02 FILLER PIC 9(5).  ←— Idem, de 5 zeros.
03 FILLER SIZE IS 5. ←— Ocorrência de 5 espaços em algum compilador.
```

12.2. A CLÁUSULA USAGE

Serve para distinguir se o dado será usado para o fim *computacional* ou fim de *apresentação* em forma visual escrita (na impressora, console ou vídeo de TV).

Exemplos:

```
USAGE IS COMPUTATIONAL.
USAGE IS DISPLAY.
```

O USAGE pode ser usado em qualquer nível. O dado COMPUTATIONAL será sempre um dado numérico puro, ao passo que DISPLAY indica um dado na forma apropriada para saída. Se o USAGE for omitido, o computador assume que o dado esteja na forma DISPLAY, mesmo que o valor seja exclusivamente para cálculo interno no computador.

É importante notar que cada computador possui diversos modos de definir o USAGE que está intimamente ligado à maneira pela qual o computador manipula os dados internos da máquina. A escolha do USAGE adequado é importante para o uso correto e econômico do computador. O sistema IBM possui, por exemplo, os seguintes tipos de USAGE:

- Representação do dado:
  DISPLAY: Um caráter por byte.
  COMPUTATIONAL ou COMP: Dado em forma binária.
  COMPUTATIONAL-1 ou COMP-1: Dado em ponto decimal flutuante de curta precisão.
  COMPUTATIONAL-2 ou COMP-2: idem, de longa precisão.
  COMPUTATIONAL-3 ou COMP-3: Dado em forma decimal interna ou condensada.

Se o USAGE correto não for usado (por exemplo dado NUMÉRICO com USAGE DISPLAY) o computador não altera o resultado, mas simplesmente desperdiça tempo na conversão de uma forma para outra, o que é inconveniente. Recomenda-se consultar o Manual COBOL do sistema utilizado para conhecer o USAGE adequado.

12.3. A CLÁUSULA VALUE

Define o valor inicial de um item ou o valor de uma constante na WORKING-STORAGE SECTION. Na FILE SECTION só pode ser usada com os nomes de condição, usando o nível 88.

O VALUE não pode ser usado junto com o OCCURS. Não confundir esta cláusula VALUE, com o VALUE da definição de arquivos no nível FD e que é tratado como comentário.

Exemplos:

```
VALUE IS 100.00.
VALUE IS ZERO.
```

12.4. A CLÁUSULA OCCURS

É usada para eliminar a repetição da descrição de mesmo tipo. É mais usada com Tabelas e valores com índices ou subscritos (veja Capítulo 24). Não deve ser usada nos níveis 01, 77 ou 88.

Exemplo:

03 XZ-1 PICTURE IS X(10) OCCURS 50 TIMES (o item de 10 caracteres ocorre 50 vezes).

12.5. A CLÁUSULA JUSTIFIED

Quando um item de dado é movido dentro da memória, para um local de tamanho maior, é necessário dizer a partir de que lado o dado é colocado.

Exemplo:

$$\left\{\begin{matrix}JUSTIFIED\\JUST\end{matrix}\right\} \qquad \left\{\begin{matrix}LEFT\\RIGHT\end{matrix}\right\}$$

Dados alfabéticos ou alfanuméricos serão automaticamente colocados a partir da posição mais à esquerda no novo campo ou local, a menos que o JUSTIFIED RIGHT seja usado.

Dados numéricos são automaticamente encostados à direita do campo, a menos que o JUSTIFIED LEFT seja usado.

12.6. A CLÁUSULA SYNCHRONIZED

É usada para posicionar o dado corretamente dentro de cada palavra da memória do computador, de modo que cause menor esforço ao mesmo. Por exemplo, se cada palavra do computador for capaz de guardar 4 caracteres e temos os dados XXX, YYY e ZZZZ, estes seriam colocados, na operação de leitura, em palavras subseqüentes do seguinte modo:

Palavra	1	2	3
	XXXY	YYZZ	ZZb b

o que acarreta enorme trabalho para separar os dados corretos novamente. Se usarmos o SYNCHRONIZED LEFT, teríamos os dados separados em cada palavra:

Palavra	1	2	3
	XXXb	YYYb	ZZZZ

A outra opção SYNCHRONIZED RIGHT daria:

Palavra	1	2	3
	bXXX	bYYY	ZZZZ

12.7. A CLÁUSULA REDEFINES

Será explicada no capítulo que trata sobre variáveis subscritas.

12.8. FORMATO GERAL

Observação: No APÊNDICE estão listadas outras cláusulas como CLASS e SIZE, que na maioria dos sistemas são substituídas pela cláusula PICTURE.

Item do grupo (quando pode ser subdividido em subcampos)

A B

{ n.º nível } { nome-do-registro / FILLER } [; OCCURS...] [; USAGE ...] [...] [; REDEFINES.]

Item de dado (não pode ser subdividido)

{ 01 / 77 } ou { 02 a 49 / 88 } { nome-de-dado / FILLER } [; OCCURS...] [; REDEFINES...]

[; SYNCHRONIZED ...] [{ ; PICTURE / ; PIC } IS ...] [; USAGE IS ...]

[; ...] [; VALUE IS literal] [; JUSTIFIED ...]

CARTÃO:

Col. 7 22 26 30 35 45

CODIGO NOME QTD. VALOR

X-320␣␣ARREBITE TIPO ␣␣32 15000

DESCRIÇÃO COBOL

```
01 CARTAO-DE-PECA.
02 CODIG PICTURE X(7).
02 NOME PICTURE X(15).
02 FILLER PICTURE X(3).
02 QUANTIDADE PICTURE 9(4).
02 FILLER PICTURE X(5).
02 VALOR PICTURE 9(8)V99.
02 FILLER PICTURE X(35).
```

(␣) representa coluna em branco.

SE CARTÃO FOSSE:

␣␣X-320ARREBITE TIPO 32␣␣ 15000

DESCRIÇÃO SERIA:

```
01 CARTAO-DE-PECA.
02 CODIG PIC X(7) JUSTIFIED RIGHT.
02 NOME PICTURE X(15).
02 FILLER PIC X(3).
02 QUANTIDADE PIC 9(4) JUSTIFIED LEFT.
02 ... (demais não se alteram)
...
```

Figura 12.1. *Uso da PICTURE e de outras cláusulas.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Dizer o que faz cada uma das cláusulas seguintes:
   FILLER, USAGE, JUSTIFIED, SYNCHRONIZED.
2. Dar exemplos de uso dessas cláusulas.

EXERCÍCIOS DE APLICAÇÃO

1. Descrever completamente os seguintes registros de dados, atribuindo a cada campo o número conveniente de espaços e caracteres:

a)

Nível						
01	ITEM EM ESTOQUE					
02	PEÇA		QUANTIDADE	ESPAÇO	VALOR UNITARIO	ESPAÇO
03	CODIGO	NOME				

b)

Nível													
01	FICHA DE FUNCIONARIO												
02	FUNCIONARIO				SEXO	DATA NASCIM			DATA ADMIS			SALARIO	
03	DEPTO	SECAO	NUMERO	NOME		DIA	MES	ANO	DIA	MES	ANO		

13 DATA DIVISION: WORKING-STORAGE SECTION

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

WORKING-STORAGE SECTION

- Nível 77: para definir item indivisível de dado.
- Nível 01, 02, ... , 49: para definir itens de dados divisíveis
- Nível 88: para definir nome de condição.
- Exemplos.
- Formato Geral.

AS CONSTANTES

TABELAS DE CONSTANTES

- Tabelas Longas.

Exercícios.

13.1. WORKING-STORAGE SECTION

É a seção do DATA DIVISION que define, na memória do computador, as variáveis e áreas de trabalho e constantes usadas durante o processamento do programa. Por esta razão ela não possui arquivos, mas somente uma série de descrição de registros ou dados. Todos os dados ou tabelas intermediárias que não são diretamente usados como entrada e saída devem ser definidos nesta seção.

Mesmo que o programa não use dados intermediários, o título WORKING-STORAGE SECTION deve aparecer no programa.

13.1.1. Nível 77

Se o item definido no WORKING-STORAGE SECTION é independente dos demais e não é subdividido em outros itens, usa-se o número de nível 77 na margem A.

Estes itens de dados independentes com nível 77, se existirem, devem ser os primeiros a aparecer na seção. A descrição total do item pode ser normalmente completada pelo PICTURE, VALUE etc.

13.1.2. Níveis 01, 02, 03, ... , 49

Todos os nomes de registros que podem ser subdivididos assumem o nível 01, 02 ... até 49 de modo idêntico aos nomes de registros do FILE-SECTION usando, portanto, as mesmas cláusulas de descrição.

13.1.3. Nível 88

Os nomes de condições também podem ser usados de modo idêntico ao FILE-SECTION.

Exemplos:

```
WORKING-STORAGE SECTION
77 AREA-DE-SOMA-1 PICTURE IS 9(6) VALUE IS ZEROES.
77 AREA-DE-CODIGO  PICTURE X(5).
01 CABECALHO.
   02 FILLER PIC X(10).
   02 CODIGO PICTURE X(5).
   02 FILLER PIC X(5).
   02 NOME PICTURE A(20).
```

O primeiro item 77 reserva uma área numérica de 6 dígitos cujo valor inicial é zero; o segundo reserva uma área alfa-numérica de 5 posições para formar algum código intermediário. O item de nível 01 é a área onde será formado um cabeçalho e tem o formato:

10 espaços	5 posições para CODIGO	5 espaços	20 posições para NOME

Se a cláusula VALUE não for usada, o valor inicial das posições reservadas é imprevisível, exceto no FILLER que preenche com espaços. Entretanto, se quisermos imprimir esse cabeçalho, devemos transferi-lo para uma área pertencente a um arquivo de saída, usando o comando MOVE.

13.1.4. Formato geral

```
A    B
WORKING-STORAGE SECTION.
[ 77      nome-de-dado-1 e sua descrição. ]
[ 77      nome-de-dado-2 e sua descrição. ]
          ...
[ 01      nome-de-registro e sua descrição.
          02 a 49 nome-de-subcampo e sua descrição. ]
          ...
```

13.2. AS CONSTANTES

As constantes usadas nos comandos do PROCEDURE DIVISION através de um nome são definidas no nível 77, de modo análogo a outros itens do WORKING-STORAGE SECTION.

Exemplo:

```
77 TAXA-JURO PICTURE 9V99 USAGE COMPUTATIONAL VALUE 0.12.
```

13.3. TABELA DE CONSTANTES

As constantes tabeladas podem ser diretamente definidas usando-se o item de grupo com nível 01.

Exemplo:

```
01 TABELA-DE-RAIZ-QUADRADA.
   02 DOIS PICTURE 9V9999 USAGE COMPUTATIONAL VALUE 1.4142.
   02 TRES PICTURE 9V9999 USAGE COMPUTATIONAL VALUE 1.7320.
   02 QUATRO PICTURE 9V9999 USAGE COMPUTATIONAL VALUE 2.0000.
   02 CINCO PICTURE 9V9999 USAGE COMPUTATIONAL VALUE 2.2360.
```

13.3.1. Tabelas longas

Se a tabela for demasiadamente longa, o método de descrição direta das constantes torna-se trabalhoso, pois é comum a manipulação de tabelas com mais de 50, 100 valores. Nesse caso é conveniente usar outra técnica que receba a tabela de constantes de um arquivo externo, usando variáveis subscritas ou com índices. Essa técnica será apresentada no capítulo sobre **Variáveis Subscritas.**

EXERCÍCIOS DE APLICAÇÃO

1. Escrever a WORKING-STORAGE SECTION que contém os índices de conversão da Tabela Price (12% a.a.) para 6, 12, 18, 24 e 36 meses.
2. Definir a WORKING-STORAGE SECTION para os 10 dados de uma tabela, que serão recalculados durante o processamento e possuem valores iniciais respectivamente iguais a: 1 , 3 , 3.5 , 4 , 7 , 1.1 , 2.0 , 2.0 , 2.2 , 3.0 .
3. Sejam os valores A, B, C e D lidos de um arquivo. Deseja-se efetuar o seguinte cálculo
   (A * TAXA-1) + (B * TAXA-2) + (C * TAXA-3) + (D * TAXA-1)
   e guardar o resultado.
   TAXA-1, TAXA-2, TAXA-3 são valores fixos que não se alteram e têm valores 0.5, 0.2 e 0.3, respectivamente.
   Definir a WORKING-STORAGE SECTION para esse cálculo.

14 DATA DIVISION (EXEMPLO DE PREENCHIMENTO)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

O PROBLEMA

RESPOSTA

Exercícios

14.1. O PROBLEMA

Escrever o DATA DIVISION para o seguinte problema:

Existe um arquivo de entrada chamado VALORES-DE-ENTRADA e um arquivo de saída chamado RESULTADO-DE-CÁLCULO. Ambos os arquivos contêm blocos de 6 registros cada. Os valores chamados ÁREA, PERÍMETRO e ALTURA contêm 8 dígitos cada um e estão no arquivo VALORES-DE-ENTRADA, cujo rótulo (LABEL) de identificação é XX-10.

A fórmula a ser calculada é:

$$CÁLCULO = (5 * (ÁREA * ALTURA)) / (PERÍMETRO)^2$$

O arquivo RESULTADO-DE-CÁLCULO contém somente registros de 10 dígitos e serve para guardar o valor do CÁLCULO. Reservar duas áreas de trabalho: uma para o cálculo de fórmula e outra para guardar o resultado.

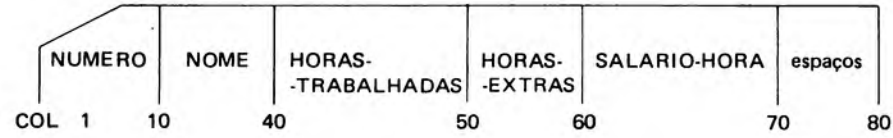
14.2. RESPOSTA

```
A     B
DATA  DIVISION.
FILE  SECTION.
FD    VALORES-DE-ENTRADA BLOCK CONTAINS 6 RECORDS
      LABEL RECORDS ARE STANDARD
      VALUE OF IDENTIFICA IS 'XX-10'  10'
      DATA RECORD IS VALORES.
01    VALORES.
      02 AREA PICTURE IS 9(8).
      02 PERIMETRO PICTURE IS 9(8).
      02 ALTURA PICTURE IS 9(8).
FD    RESULTADO-DE-CALCULO
      BLOCK CONTAINS 6 RECORDS
      LABEL RECORDS ARE STANDARD DATA RECORD IS RESULTADO.
01    RESULTADO PICTURE 9(10).
WORKING-STORAGE SECTION.
77    AREA-1     PICTURE 9(10).
77    AREA-2     PICTURE 9(10).
```

EXERCÍCIOS

1. Escrever o DATA DIVISION de um programa que lê cartões de um arquivo chamado ENTRADA-CARTÃO e efetua o cálculo de SALÁRIO, que é formado pelo SALÁRIO--HORA multiplicado por HORAS-TRABALHADAS mais SALÁRIO-HORA-EXTRA, que é formado por HORAS-EXTRAS vezes SALÁRIO-HORA, com 35% de acréscimo no valor calculado.

   Há desconto de 8% sobre o SALÁRIO para obter o SALÁRIO-LÍQUIDO e para cada cartão lido será impresso uma linha na impressora.
   O formato do cartão e da linha de impressão é:



2. Ilustrar, de modo análogo à Figura 10.1, os arquivos de entrada e de saída (em fitas magnéticas), bem como as áreas da memória utilizadas pelo DATA DIVISION do problema resolvido neste capítulo.

15 PROCEDURE DIVISION (COMANDOS ARITMÉTICOS ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE, ADD E SUBTRACT CORRESPONDING)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

COMANDO ADD
- Formato Geral.
  Exemplos: ADD A TO B. ADD A, B GIVING C.
- As opções TO e GIVING.
- A opção ROUNDED.
- A opção ON SIZE ERROR.

COMANDO SUBTRACT
- Formato Geral
  Exemplo: SUBTRACT 15 FROM Y GIVING Z.

COMANDO MULTIPLY
- Formato Geral
  Exemplos: MULTIPLY X BY Y GIVING Z.

COMANDO DIVIDE
- Formato Geral
  Exemplo: DIVIDE A INTO B GIVING C

COMANDO COMPUTE
- Formato Geral
  Exemplo: COMPUTE SALARIO-LIQUIDO =SALARIO-HORA *
  HORAS – DESCONTO.
  COMPUTE X = (Z – A / B) * 1.24.

POSICIONAMENTO DO PONTO DECIMAL
- Exemplos

COMANDO ADD CORRESPONDING E SUBTRACT CORRESPONDING
- Formato Geral e Exemplos

EXEMPLO ILUSTRADO

Exercícios

15.1. COMANDO ADD

15.1.1. Formato geral

```
ADD { literal-1       } [ , { literal-2       } ... ] { TO    } nome-de-dado-n
    { nome-de-dado-1  }     { nome-de-dado-2  }       { GIVING }

[ ROUNDED ] [ ; ON SIZE ERROR outro comando imperativo ]
```

Lembramos que, em todos os comandos do PROCEDURE DIVISION, os nomes de dados (variáveis, constantes) devem estar previamente definidos na DATA DIVISION, com exceção das constantes literais que são usadas diretamente no comando. Todos os elementos envolvidos (literal, nome-de-dado) devem ser numéricos.

Algumas formas possíveis deste comando, são:

```
ADD A TO B. (Soma A com B e guarda resultado em B)
ADD A, B TO C.  Soma de A, B e C fica em C que perde o valor anterior)
ADD A, B GIVING C. (Soma de A e B é colocada em C)
ADD A, B, C GIVING D. (Soma de A, B e C é colocada em D)
ADD 30 TO A.  Valor de A fica acrescido de 30)
```

15.1.2. As opções TO e GIVING

Notar que é obrigatório o uso de TO ou GIVING mas é proibido o uso de ambos. A opção TO soma os elementos guardando o resultado no último, ao passo que a opção GIVING soma dois ou mais elementos, guardando o resultado no elemento colocado após o GIVING.

Sendo assim:

```
ADD X, Y.  (é incorreto).
ADD X TO Y, GIVING Z.  (é incorreto)
```

15.1.3. A opção ROUNDED

Serve para arredondar os dígitos decimais que excedem o campo do resultado de uma operação aritmética.

Por exemplo, se o campo de C tiver XXX.X posições e se A + B = 689.68, então

```
ADD  A, B   GIVING C (sem ROUNDED) terá C com valor 689.6 e
ADD  A, B   GIVING C ROUNDED terá C com valor 689.7.
```

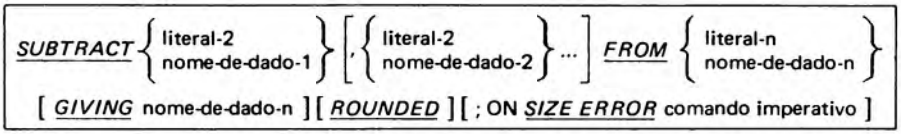
15.1.4. A opção ON SIZE ERROR

Quando há "estouro" ou "transbordo" de um valor numérico (*over-flow*) em uma operação aritmética não se sabe o que acontece. Portanto, é sempre recomendável usar a opção ON SIZE ERROR seguida de um comando de desvio do tipo GO TO ROTINA-DE-ERRO para avisar ou corrigir esta situação.

Exemplo:

ADD SOMA-1 TO SOMA-2 ROUNDED; ON SIZE ERROR GO TO ROTINA-E.

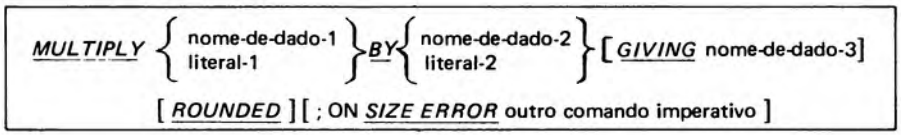
15.2. COMANDO SUBTRACT



Exemplos:

SUBTRACT X FROM Y. (O valor de (Y −X) é colocado em Y)
SUBTRACT X, Y FROM Z GIVING W ROUNDED; ON SIZE ERROR GO TO ROTINA-ERRO-2. (O valor de Z − (X + Y) é colocado em W, sem alterar os valores originais de X, Y e Z).
SUBTRACT X, Y, Z FROM W ROUNDED; ON SIZE ERROR GO TO ROTINA--ERRO-2. (O valor de W − (X + Y + Z) é colocado em W que perde o seu valor original.)
SUBTRACT 15 FROM Y GIVING Z. (O valor de Y − 15 é colocado em Z.)

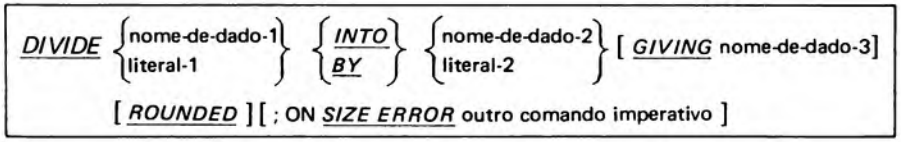
15.3. COMANDO MULTIPLY



Exemplos:

MULTIPLY X BY Y. (Valor de X * Y é colocado em Y.)
MULTIPLY X BY Y GIVING Z ROUNDED; ON SIZE ERROR GO TO ROTINA--MULT. (Coloca o resultado de X * Y em Z.)

15.4. COMANDO DIVIDE



A opção DIVIDE X BY Y corresponde à divisão normal que conhecemos, isto é, divide X por Y.

A opção DIVIDE X INTO Y significa dividir Y ENTRE X. Então, X igual a 10 crianças e Y igual a 5 doces teríamos: dividir 5 doces entre 10 crianças e o resultado seria 0,5 doces cada (na opção INTO) e dividir 10 crianças para (BY) 5 doces resultaria em 2 crianças (para cada doce).

Exemplo:

```
DIVIDE A INTO B. (Valor de (B ÷ A) é colocado em B.)
DIVIDE 10.5 INTO VALOR GIVING VALOR-CORRIGIDO
ROUNDED; ON SIZE ERROR GO TO ROTINA-ERRO. (Valor de VALOR dividido por 10.5 é colocado em VALOR-CORRIGIDO.)
```

15.5. COMANDO COMPUTE

É usado para calcular fórmulas matemáticas em operações mais complicadas, evitando o uso de comandos ADD, SUBTRACT, MULTIPLY e DIVIDE.

```
COMPUTE  nome-de-dado-1 [ ROUNDED ] = { nome-de-dado-2
                                      { literal-1
                                      { expressão aritmética }

[ ; ON SIZE ERROR outro comando imperativo ]
```

Exemplos:

```
COMPUTE Z = Y + ( B / C ) * 5.
COMPUTE Z ROUNDED = Y + ( B / C ) * 5; ON SIZE ERROR GO TO ROTINA-DE-ERRO.
COMPUTE SALARIO-LIQUIDO = SALARIO * HORAS – DESCONTOS
```

O uso do ROUNDED e ON SIZE ERROR é sempre recomendável. Entre os sinais operadores deve existir pelo menos 1 (um) espaço.

15.6. POSICIONAMENTO DO PONTO DECIMAL

A posição do ponto decimal do resultado de uma operação aritmética ou de uma fórmula irá depender da posição do ponto decimal implícito, definido no campo reservado para esse resultado. Isto é, dependerá da posição do caráter V definido pelo PICTURE de cada operando, no DATA DIVISION. Cabe também lembrar que, em COBOL, o tamanho máximo de um operando que entra na operação aritmética é 18 (dezoito) dígitos.

Exemplos:

		PICTURE	VALOR
ADD A, B, GIVING C.	A	99V99	35.37 (o ponto é implícito)
	B	9V9	4.1
	C	99V99	39.47 (operação normal correta)
	A	99V99	35.37
	B	9V99	4.15
	C	999	039. (perde parte decimal)

DIVIDE 2 INTO B
GIVING COTA.

B	99V9999	0.1416
COTA	9V9999	0.0708 (resultado correto)
B	99V999	0.1416
COTA	99V999	0.070 (perde último algarismo)

DIVIDE 2 INTO B
GIVING COTA ROUNDED.

B	99V999	0.1416
C	99V999	0.071 (arredonda última casa decimal).

15.7. COMANDO ADD CORRESPONDING E SUBTRACT CORRESPONDING

15.7.1. Formato geral

ADD { *CORRESPONDING* / *CORR* } nome-de-dado-1 *TO* nome-de-dado-2
[*ROUNDED*] [; ON *SIZE ERROR* comando imperativo]

SUBTRACT { *CORRESPONDING* / *CORR* } nome-de-campo-1 *FROM* nome-do-campo-2
[*ROUNDED*] [; ON *SIZE ERROR* comando imperativo]

Soma ou subtrai subcampos que possuem nomes idênticos dentro de campos distintos. Por exemplo:

ADD CORRESPONDING CAMPO-1 TO CAMPO-2 ROUNDED.

onde CAMPO-1 é

M	N	O	P	Q	R

e CAMPO-2 é

N	P	R	T	W	Q	S

resulta em somar: subcampo N de CAMPO-1 em subcampo N de CAMPO-2
" P " " " " P " "
" R " " " " R " "
e " Q " " " " Q " "

Para SUBTRACT CORRESPONDING o mecanismo é o mesmo e em ambos os casos o resultado permanece no CAMPO-2.

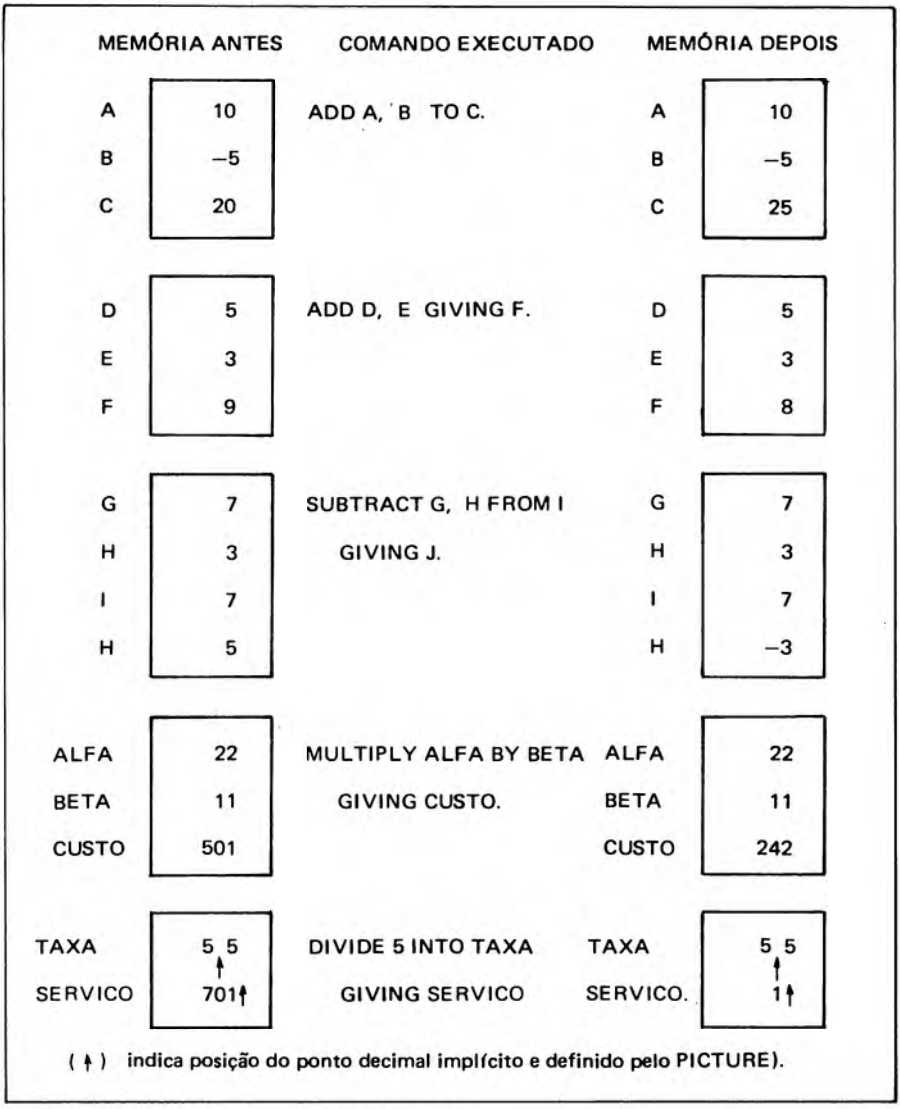


Figura 15.1. *Uso dos comandos aritméticos.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Dizer o que faz e citar as formas possíveis dos comandos ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE, ADD CORRESPONDING e SUBTRACT CORRESPONDING de acordo com o formato geral de cada comando.

EXERCÍCIOS DE APLICAÇÃO

1. Os itens chamados X, Y, Z, W e VALOR têm respectivamente os valores 1, 5, 10, 3, e 0. Indicar o valor de cada item após a execução de cada um dos comandos seguintes:

```
ADD X, Y, Z, W TO VALOR.
ADD X, Y, Z, W GIVING VALOR.
SUBTRACT VALOR FROM Y GIVING Z.
SUBTRACT VALOR FROM Z.
DIVIDE VALOR INTO Y; ON SIZE ERROR COMPUTE X = Y * Z * 2.
```

2. Seja o DATA DIVISION contendo os campos:

```
01 CAMPO-A
     02   NUMERO          PICTURE 9(5)  VALUE IS 15100.
     02   QUANTIDADE      PICTURE 9(5)  VALUE IS  2500.
     02   VALOR           PICTURE 9(3) V9(2)
                                        VALUE IS 453.30.
01 CAMPO-B
     02   NUMERO          PICTURE 9(5)  VALUE IS  9500.
     02   CODIGO          PICTURE 9(4)  VALUE IS ZERO.
     02   VALOR           PICTURE 9(3)V9(2)
                                        VALUE IS 300.00.
     02 QUANTIDADE    PICTURE 9(5)  VALUE IS 555.
```

Queremos somar os valores de nomes iguais nos 2 campos e manter o resultado no CAMPO-B. Escrever o comando ADD CORRESPONDING correto e também o conjunto de comando ADD simples equivalentes.

16 PROCEDURE DIVISION (COMANDOS QUE ALTERAM A SEQÜÊNCIA DE EXECUÇÃO: GO TO, ALTER, STOP E PERFORM)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO.

ALTERAÇÃO DA SEQÜÊNCIA DE EXECUÇÃO DO PROGRAMA

COMANDO GO TO
- Exemplo: GO TO nome-de-parágrafo.

COMANDO CONDICIONAL GO TO ... DEPENDING ON
- Exemplo: GO TO PARAGRAFO-UN, PAR-2 DEPENDING ON X.

COMANDO STOP
- Exemplos: STOP RUN. STOP 002.

COMANDO ALTER
- Exemplo: ALTER PARAGRAFO-UM TO PROCEED TO ROTINA-200.

COMANDO PERFORM
- Execução de parágrafos como se fossem sub-rotinas.
  Exemplos: PERFORM PARAGRAFO-DOIS.
  PERFORM PARAGRAFO-DOIS THRU PARAGRAFO-CINCO.
  etc.

EXECUÇÃO RECURSIVA DO PERFORM
- Exemplos

FORMATO GERAL DO COMANDO PERFORM

EXEMPLO ILUSTRADO DO USO DO COMANDO PERFORM

Exercícios

16.1. ALTERAÇÃO DA SEQÜÊNCIA DE EXECUÇÃO DO PROGRAMA

Os comandos são executados sempre em sua seqüência natural de acordo com a ordem de apresentação. Entretanto, como resultado de uma condição ou

fim de um parágrafo, o programador pode alterar essa seqüência normal de execução do programa, desviando-o para um ponto desejado (início de um parágrafo qualquer). A seguir, examinaremos os comandos que permitem tais desvios ou alterações. O leitor deve tomar cuidado extremo ao usar tais comandos que podem afetar a boa estruturação de um programa. Ver Capítulo 5, sobre Programação Estruturada em COBOL.

16.2. COMANDO GO TO

Desvia o programa para o início do parágrafo indicado. Não pode ser usado no meio de uma sentença imperativa, pois, acarretando um desvio obrigatório, faz com que os comandos do resto da sentença nunca sejam executados. Já com o uso dos comandos de condição IF tal fato pode ser evitado.

Exemplos:

```
GO TO PARAGRAFO-UM.
GO TO ROTINA-DE-ERRO.
```

16.3. COMANDO CONDICIONAL GO TO ... DEPENDING ON

Usa-se a palavra-chave DEPENDING ON e desvia para um dos parágrafos indicados dependendo do valor da variável indicada.

Exemplo:

```
GO TO PARAGRAFO-X, PARAGRAFO-Y, PARAGRAFO-Z
DEPENDING ON VALOR-CODIGO.
```

No Exemplo acima, VALOR-CODIGO é uma variável definida no DATA DIVISION e assume valor 1, 2 ou 3; se VALOR-CODIGO for 1 o programa desvia para PARAGRAFO-X, se for 2 para o PARAGRAFO-Y etc. Se VALOR-CODIGO estiver fora do limite permitido (no caso, valores 1, 2 e 3), será executado o comando seguinte ao GO TO... DEPENDING ON.

16.4. COMANDO STOP

Serve para acarretar a parada temporária ou final de um programa.

Exemplos:

```
STOP RUN      (parada final do programa)
STOP 002
STOP 'PARTE-1'
```

Os dois últimos exemplos param temporariamente o programa, que pode ser reiniciado pelo operador, a partir do ponto de parada, apertando a tecla START; 002 e 'PARTE-1' são nomes ou códigos dados pelo programador, e que podem ser escritos no console do sistema por ocasião da parada.

16.5. COMANDO ALTER

Altera o desvio incondicional de um comando GO TO que foi usado em algum parágrafo especial do programa. Esse parágrafo só pode conter o comando

GO TO e nada mais. O uso do comando ALTER pode otimizar o tempo de execução do programa.

Exemplo:

```
PARAGRAFO-1. GO TO ROTINA-2.
...
ALTER PARAGRAFO-1 TO PROCEED TO ROTINA-200.
```

O comando ALTER faz com que o comando GO TO ROTINA-2 seja transformado em GO TO ROTINA-200.

16.6. COMANDO PERFORM

Serve para executar determinados parágrafos do programa como se fossem sub-rotinas ou subprogramas. O parágrafo ou parágrafos chamados pelo PERFORM são executados como se fossem inseridos no local do comando PERFORM, após o que o programa prossegue normalmente. Um mesmo parágrafo pode ser chamado várias vezes por diversos comandos PERFORM. O último comando do último parágrafo chamado pelo PERFORM é o ponto de retorno automático ao ponto de chamada e por essa razão não deve ser um comando GO TO. Para ponto de retorno múltiplo deve-se usar o comando "EXIT" (Capítulo 20).

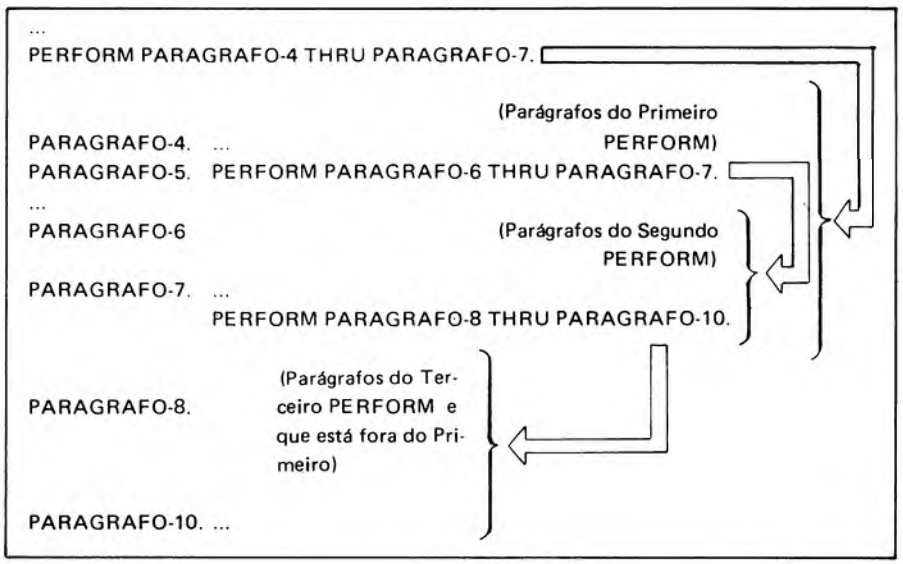
Exemplos:

```
PERFORM PARAGRAFO-DE-CONTAS. (executa o parágrafo completo)

PERFORM PARAGRAFO-X THRU PARAGRAFO-Y. (executa o PARAGRAFO-
-X e todos os seguintes até encontrar o PARAGRAFO-Y executando-o inclusive).

PERFORM PARAGRAFO-X THRU PARAGRAFO-Y 10 TIMES. (repete a execução do comando por 10 vezes).

PERFORM PARAGRAFO-X THRU PARAGRAFO-Y UNTIL CONTADOR IS
EQUAL TO 100. (repete a execução, até que a variável CONTADOR assuma o
valor 100).

PERFORM PARAGRAFO-X THRU PARAGRAFO-Y VARYING CONTADOR
FROM 1 BY 1 UNTIL CONTADOR = 100 (neste exemplo repete-se a execução
100 vezes, pois CONTADOR varia a partir de 1, até 100, de 1 em 1, ao passo que
no exemplo anterior o CONTADOR pode assumir a qualquer hora (ou nunca) o
valor 100 dado por um comando do programa).
```

16.7. EXECUÇÃO RECURSIVA DO PERFORM

Se, dentro de um parágrafo executado por um comando PERFORM, existir outro comando PERFORM, todos os parágrafos a serem chamados por esse segundo PERFORM devem estar incluídos entre os parágrafos chamados pelo primeiro PERFORM ou então todos deverão estar completamente excluídos dos parágrafos do primeiro PERFORM. Não é permitido executar, pelo segundo PERFORM, alguns parágrafos do primeiro PERFORM e outros alheios ao mesmo.

Exemplos:



Não é permitido executar pelo comando PERFORM o próprio parágrafo onde está situado o comando.

No exemplo anterior, não seria permitido usar o comando:

```
PARAGRAFO-7.
    PERFORM PARAGRAFO-5 THRU PARAGRAFO-9.
```

16.8. FORMATO GERAL DO COMANDO PERFORM

Opção 1:

PERFORM nome-de-parágrafo-1 [*THRU* nome-de-parágrafo-2]

Opção 2:

PERFORM nome-de-parágrafo-1 [*THRU* nome-de-parágrafo-2] { nome-de-dado / inteiro } *TIMES*

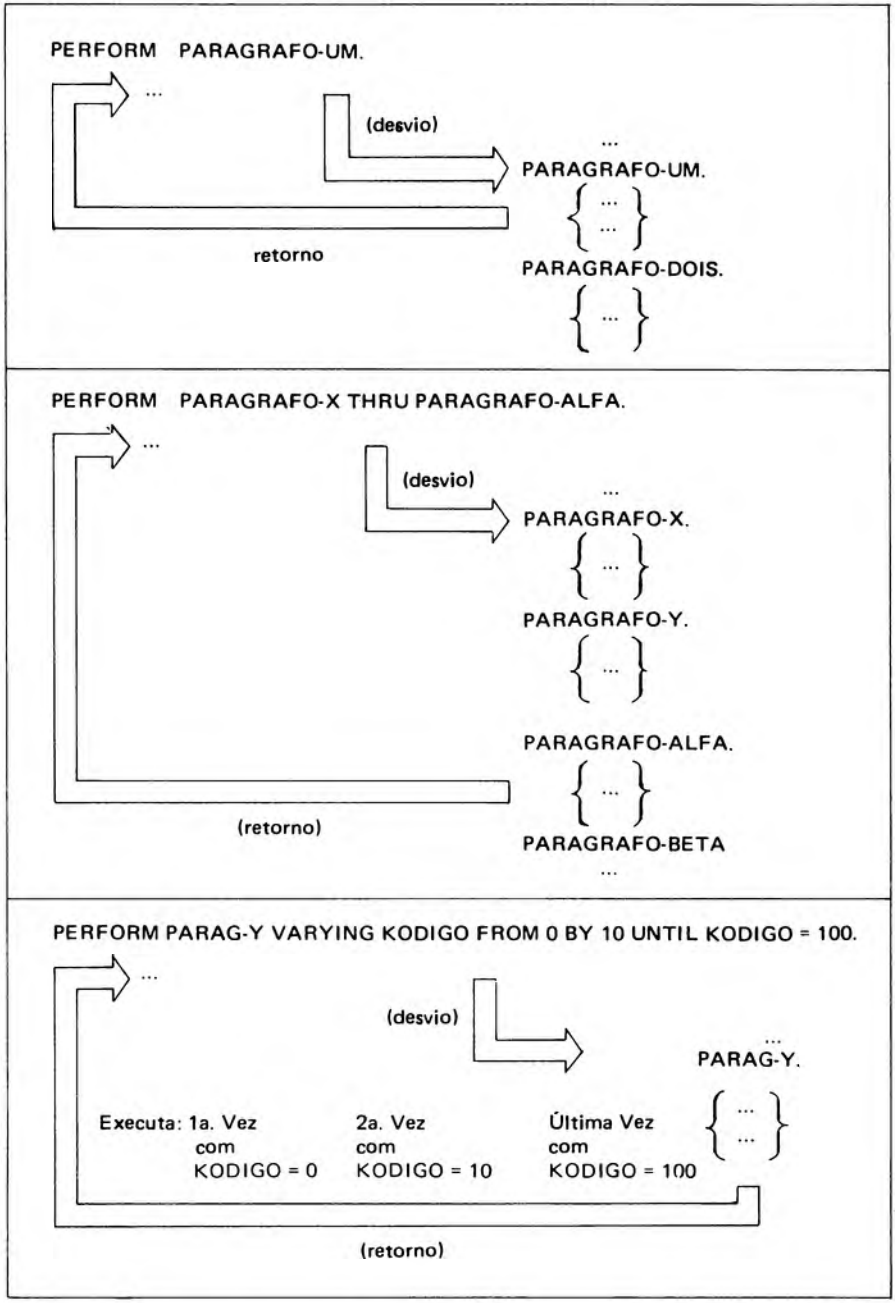
Opção 3:

PERFORM nome-de-parágrafo-1 [*THRU* nome-de-parágrafo-2] *UNTIL* condição

Opção 4:

PERFORM nome-de-parágrafo-1 [*THRU* nome-de-parágrafo-2]
VARYING nome-de-dado-1 *FROM* { *nome-de-dado-2* / literal-1 } *BY* { nome-de-dado-3 / literal-2 }
UNTIL condição (*)

(*) condição = expressão lógica.



150 Figura 16.1. *Ilustração do uso do comando PERFORM.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Para que servem e quais são os comandos de controle de seqüência?
2. Qual a diferença entre o comando GO TO e GO TO condicional?
3. O que faz o comando STOP? E o comando ALTER?
4. De quantas maneiras diferentes pode ser usado o comando PERFORM? Citar exemplos de cada caso.

EXERCÍCIOS DE APLICAÇÃO

1. Qual será a ação do programa para cada um dos comandos seguintes:
   a) PERFORM CALCULO-2.
   b) PERFORM CALCULO-2 1 TIMES.
   c) PERFORM CALCULO-2 THRU CALCULO-3 VARYING X FROM 1 BY 10 UNTIL X = 121.
2. Em um parágrafo chamado ROTINA-DE-CALCULO queremos calcular o juro de 1% ao mês sobre o VALOR dado. Em dois meses o juro será de 1% sobre o (VALOR + juro do primeiro mês) e assim por diante. Escrever o parágrafo completo e o comando PERFORM para calcular os juros referentes a doze meses de empréstimo.
3. O campo chamado CODIGO pode assumir valores 1, 2, 3 e 4. Qual seria o comando GO TO que testa esses valores e desvia o programa para os parágrafos P-1, P-2, P-3 e P-4 respectivamente?

17 PROCEDURE DIVISION (COMANDOS CONDICIONAIS IF)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

TIPOS DE CONDIÇÕES OU EXPRESSÕES CONDICIONAIS

FORMATO GERAL DO COMANDO IF

- IF (expressão lógica) [THEN] (comando).
- Teste de Relacionamento Simples.
  Exemplo: IF SIZEX IS GREATER THAN 150 GO TO PAR-UM.
- Teste de Sinal.
  Exemplo: IF X IS POSITIVE ADD X TO A.
- Teste de Classe.
  Exemplo: IF X IS NUMERIC MOVE X TO A.
- Teste com Nome de Condição para simplificar o comando IF.
  Exemplo: IF OLHOS-AZUIS GO TO ROTINA-OLHOS-AZUIS.
- Operadores implícitos

COMANDO IF-THEN-ELSE-NEXT SENTENCE

Exemplo: IF VALOR IS LESS THAN 15 NEXT SENTENCE;
ELSE MOVE VALOR TO X.

SEQÜÊNCIA DE COMANDOS IF

Exemplos

USO RECURSIVO DO COMANDO IF

SELEÇÃO DE BLOCO DE COMANDOS

Exercícios

17.1. TIPOS DE CONDIÇÕES OU EXPRESSÕES CONDICIONAIS

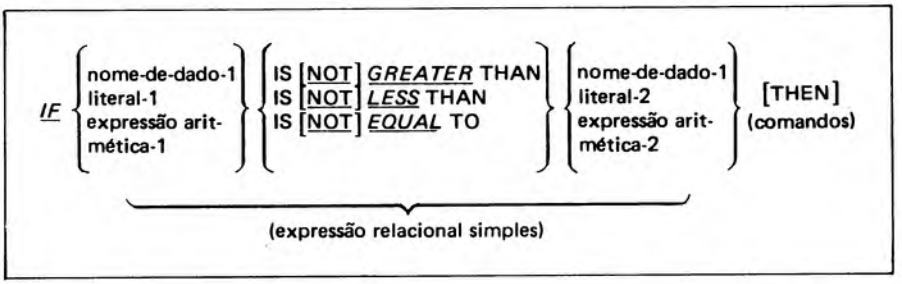
No Capítulo 7, estudamos as expressões lógicas ou condicionais que assumem o valor TRUE ou FALSE. Estas expressões são formadas por nome-de-dados, constantes ou fórmulas ligadas por operadores relacionais e operadores lógicos. (Por exemplo: GREATER THAN, EQUAL TO, AND, OR, NOT).

17.2. FORMATO GERAL DO COMANDO IF

IF (expressão condicional ou lógica) [THEN] (comandos).

O comando IF testa o valor da expressão condicional nela escrito, verificando se o resultado é TRUE ou FALSE (condição satisfeita ou não). Se for TRUE, ele executa o comando imediato seguinte ou os comandos seguintes contidos na mesma sentença até encontrar o ponto final. Se a condição for FALSE, todos esses comandos são ignorados e o programa prossegue executando o primeiro comando após essa sentença.

17.2.1. Teste de relacionamento simples



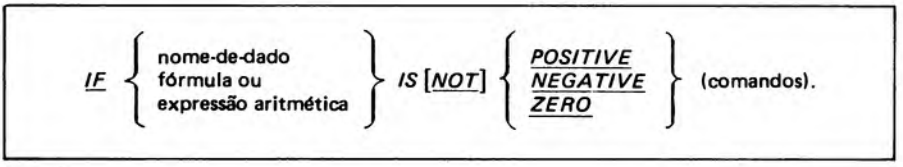
A comparação entre itens com valores numéricos obedece à relação algébrica dos valores. Se os itens forem alfanuméricos (letras, sinais e dígitos), a comparação é efetuada caráter por caráter obedecendo à seqüência alfanumérica adotada pelo computador.

Exemplos:

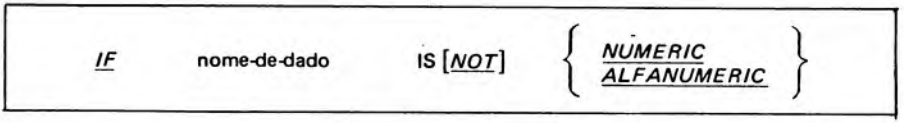
```
IF CATEGORIA IS EQUAL TO CODIGO-1 GO TO PARA-COD-1.
IF (X + A) * 3 IS NOT LESS THAN 100 ADD A TO X.
IF SIZO IS LESS THAN 150 AND CODIGO IS EQUAL TO NOME GO TO FIM.
IF SIZO IS GREATER THAN 150 OR EXTRA GO TO X.
```

Nos dois últimos exemplos, a expressão condicional foi ampliada usando operadores lógicos do tipo AND, OR.

17.2.2. Teste de sinal



17.2.3. Teste de classe



17.2.4. Teste com nome de condição para simplificar o comando IF

No DATA DIVISION usamos o nível 88 para expressar os valores possíveis que um dado pode assumir. Por exemplo, se a variável COR-DE-CABELO pode assumir valores distintos 1, 2 e 3 para cores CASTANHA, LOURA e RUIVA respectivamente, teríamos:

```
DATA DIVISION
. . . . . 03        COR-DE-CABELO
              88    CASTANHA VALUE IS 1.
              88    LOURA VALUE IS 2.
              88    RUIVA VALUE IS 3.
```

e podemos escrever

```
IF CASTANHA GO TO ROTINA-CASTANHA,
```

ou

```
IF LOURA GO TO ROTINA-LOURINHA.
```

Se não usássemos nomes de condições com nível 88, teríamos que usar os comandos.

```
IF COR-DE-CABELO IS EQUAL TO 1 GO TO ROTINA-CASTANHA.
```

ou

```
IF COR-DE-CABELO IS EQUAL TO 2 GO TO ROTINA-LOURINHA.
```

Outro exemplo seria:

```
IF CASTANHA OR RUIVA WRITE MENSAGEM.
```

17.2.5. Operadores implícitos

Alguns compiladores permitem o uso de forma implícita de comparação lógica sem a necessidade de usar a expressão completa.

Exemplos:

```
IF AGE IS GREATER THAN 5 AND LESS THAN 15 — no lugar de
IF AGE IS GREATER THAN 5 AND AGE IS LESS THAN 15.

IF A = 2 OR 4 OR 6                ou
IF A = 2, 4 OR 6                  no lugar de
IF A = 2 OR A = 4 OR A = 6.
```

17.3. COMANDO IF-THEN-ELSE-NEXT SENTENCE

Uma versão mais completa do comando IF permite dizer o que fazer também quando a condição não for satisfeita:

```
IF expressão condicional ou lógica { comando-1 | NEXT SENTENCE } { ;OTHERWISE | ;ELSE } { comandos-2. | NEXT SENTENCE }
```

Exemplos:

a) IF VALOR IS NEGATIVE ADD 2 TO CONTADOR ELSE ADD 5 TO CONTADOR.

b) IF VALOR IS LESS THAN 15 NEXT SENTENCE;
OTHERWISE MOVE A TO B; MOVE SPACES TO X; ADD 1 TO COUNT.
(Se VALOR for menos que 15, passe para a próxima sentença; se não o for, execute todos os comandos MOVE e ADD citados; NEXT SENTENCE será o comando após o comando IF e virá após o ponto decimal do comando IF, isto é, após o comando ADD.)

17.4. SEQÜÊNCIA DE COMANDOS IF

Na forma geral:

IF expressão condicional ou lógica { comando-1 / *NEXT SENTENCE* } { ; *OTHERWISE* / ; *ELSE* } { comando-2. / *NEXT SENTENCE* }

Tanto o comando-1 como o comando-2 podem ser um outro comando IF, que por sua vez poderá conter outros comandos IF, e assim por diante. Assim, temos uma seqüência de comandos IF encadeados.

Exemplo:

(1º Comando IF) { IF CODIGO IS EQUAL TO 1 NEXT SENTENCE; OTHERWISE (2º Comando IF) { IF ALTURA IS GREATER THAN 10 ADD 10 TO ALTURA.

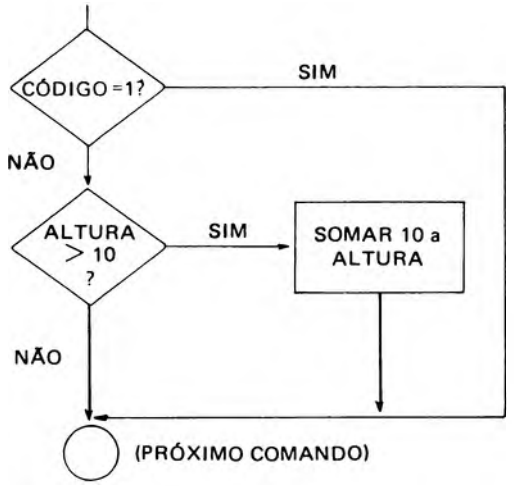


Figura 17.1. *Fluxograma.*

17.5. USO RECURSIVO DO COMANDO IF

No exemplo anterior houve o uso recursivo de um comando IF dentro de outro comando IF.

Esse fenômeno, muito comum em programação, é denominado de encadeamento ou "ninhos" (nested) de comandos e já ocorreu com os exemplos do comando PERFORM.

No formato geral do comando IF

```
IF condição [THEN] { comando-1      } { OTHERWISE } { comando-2      }
                   { NEXT SENTENCE  } { ELSE      } { NEXT SENTENCE  }
```

designemos por C1, C2, . . . as expressões condicionais ou condição e por S1 S2, . . . os comandos COBOL.

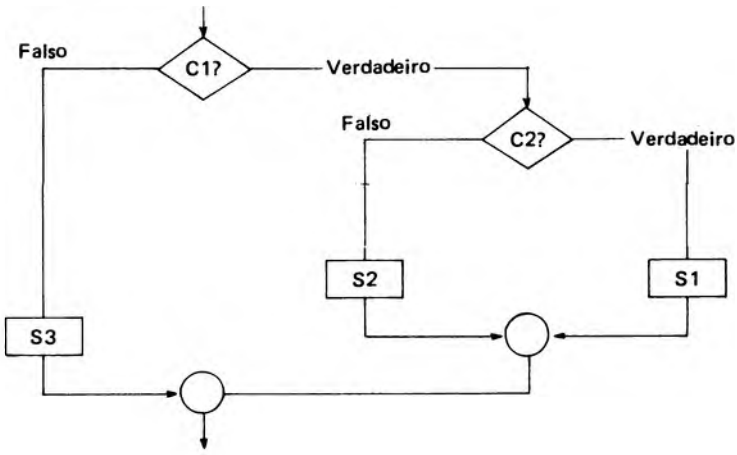
Podemos formar diversos tipos de blocos estruturados resultantes do uso recursivo de comandos IF.

17.5.1. Exemplo 1

```
IF                 IF  C1 THEN IF  C2   S1
externo                        ELSE  S2
           ELSE S3.
```

IF interno

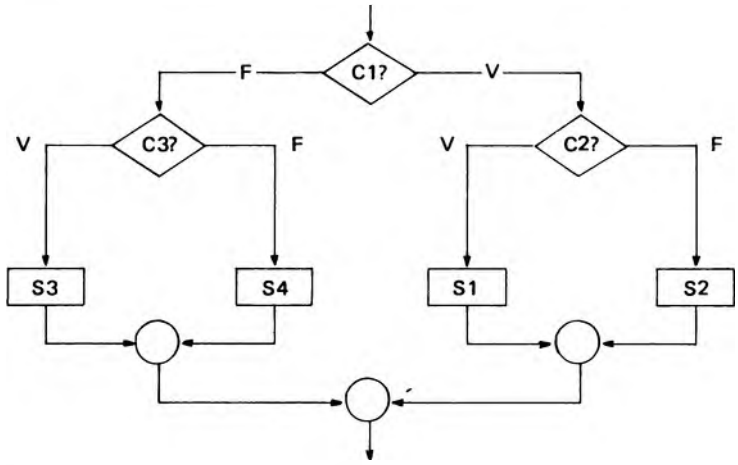
FLUXOGRAMA:



17.5.2. Exemplo 2

```
IF                IF C1 THEN   IF   C2     S1   } IF interno
externo {                           ELSE   S2   }
          {
            ELSE              IF   C3     S3   } IF interno
                                   ELSE   S4.  }
```

FLUXOGRAMA:



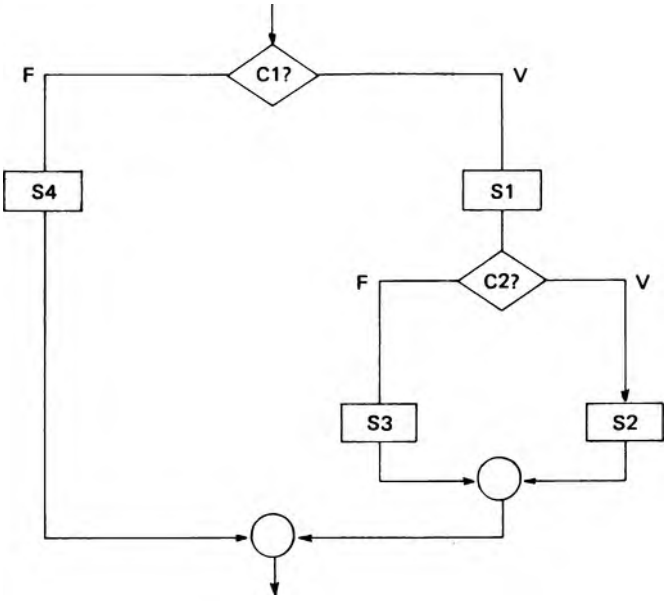
17.5.3. Seleção de bloco de comandos

No formato geral do comando IF, tanto o "comando-1" como o "comando-2" pode conter um bloco de mais de um comando COBOL.

Sendo assim, é possível formar um único comando IF do tipo:

```
IF C1 THEN    S1
              IF C2 S2   } Bloco correspondente a "comando-1".
              ELSE S3    }
  ELSE        S4
```

cujo fluxograma é:



O compilador COBOL sempre executa o programa associando cada cláusula ELSE ao IF mais próximo, não havendo ambigüidade na interpretação do comando. Devemos sempre lembrar que cada comando COBOL (IF, READ, WRITE) termina com um único ponto final, mesmo que contenha outros comandos (Ver exemplos do Capítulo 5).

EXERCÍCIOS DE RECAPITULAÇÃO

1. Dar exemplos de expressões condicionais.
2. Quais são os operadores Relacionais e operadores Lógicos usados em uma expressão condicional?
3. Dizer o que faz cada um dos seguintes comandos:

```
IF CASADO IS EQUAL TO 1 ADD 2 TO CODIGO.
IF CASADO IS EQUAL TO 'C' GO TO CASADO-ROTINA.
IF CAMPO-A IS GREATER THAN CAMPO-B NEXT SENTENCE;
   ELSE IF AREA-1 IS GREATER THAN AREA-A GO TO PARAG-1.
```

EXERCÍCIOS DE APLICAÇÃO

1. Queremos testar se um valor no CAMPO-X é positivo. Se for positivo, queremos que o programa prossiga para o parágrafo P-POS; caso contrário, para o parágrafo P-NEG. Escreva os comandos corretos.
2. Corrija os comandos errados:
   IF CONTA IS EQUAL TO VALOR MOVE X TO Y.
   IF CONTA DOES NOT EQUAL TO VALOR MOVE X TO Y.
   IF 30 IS NOT EQUAL TO 30.0 GO TO Y.
   IF X IS DIFFERENT OF VALOR GO TO A.
3. Os valores dos itens A, B, C, D são respectivamente 3, 4, 7 e -9, e temos a sentença:
   IF A IS EQUAL TO B AND C IS LESS THAN B OR D IS POSITIVE MOVE C TO D; OTHERWISE ADD A, B, C TO D. Quais são os valores desses itens após a execução dos comandos acima?
4. Converter os programas seguintes em Programa Estruturado, escrevendo os programas e os fluxogramas correspondentes:
   (C1 são expressões lógicas e S1 são comandos)

a)
```
IF C1 GO TO P-3.
S4
GO TO P-5.
P-3. S2
IF C2 GO TO P-4.
S3
GO TO P-5.
P-4. S1
P-5. . . .
```

b)
```
     IF C1 GO TO P-1
     S4
     GO TO P-6.
P-1. IF C2 GO TO P-4
     ELSE GO TO P-5.
P-4  S1
     GO TO P-6.
P-5. S2
     S3
P-6. . . .
```

18 PROCEDURE DIVISION (COMANDOS DE ENTRADA/SAÍDA: READ, WRITE, OPEN, CLOSE, DISPLAY, ACCEPT)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

COMANDOS DE ENTRADA OU DE SAÍDA
COMANDO OPEN
- Formato Geral
  Exemplo: OPEN INPUT ARQUIVO-MESTRE.

A opção I-O.
REVERSED.
NO REWIND.
COMANDO CLOSE
- Formato Geral
  Exemplo: CLOSE ARQUIVO-1, ARQUIVO-2.

COMANDO READ
- Formato geral
  Exemplo: READ ARQUIVO-MESTRE INTO AREA-UM AT END GO TO PARA-GRAFO-FIM.

COMANDO WRITE
- Formato Geral.
  Exemplo: WRITE REGISTRO-FUNCIONARIO FROM AREA-FUNCIONARIO.

A opção ADVANCING LINES e "mnemônico".
A opção INVALID KEY
COMANDO ACCEPT
- Formato Geral
  Exemplo: ACCEPT CORREÇÃO FROM CONSOLE.

COMANDO DISPLAY
- Formato Geral
  Exemplo: DISPLAY 'ERRO UM' UPON PRINTER.

EXEMPLO ILUSTRADO
Exercícios

18.1. COMANDOS DE ENTRADA OU DE SAÍDA

São comandos usados para efetuar a transferência dos dados que estão em um meio ou arquivo externo para locais reservados dentro da memória do computador e vice-versa. Recomenda-se, neste ponto, a leitura do Capítulo 4,

Introdução à Programação COBOL, onde se encontram ilustrados os processos de Entrada e Saída de dados.

A entrada de dados é executada pelo comando READ ou ACCEPT que indica o nome do arquivo externo a ser lido, e traz, cada vez em que é executado. o conteúdo de um registro completo (cartão, registro de fita ou disco) desse arquivo para o local da memória definido com nível 01 pelo DATA DIVISION.

A saída de dados é executada pelo comando WRITE ou DISPLAY que, cada vez em que é executado, transporta o conteúdo de um registro completo, que foi previamente montado na memória em local reservado pelo DATA DIVISION para um registro (uma linha de impressora, um cartão perfurado, um registro de fita ou disco, ou uma linha na máquina de escrever do console) do arquivo externo.

18.2. COMANDO OPEN

18.2.1. Formato geral

```
OPEN { [INPUT  {nome-do-arquivo-1 [REVERSED]}...]
       [OUTPUT {nome-do-arquivo-2 [WITH NO REWIND]}...]
       [I-O    {nome-do-arquivo-3 [WITH NO REWIND]}...] }
```

Este comando prepara ou abre um arquivo de Entrada ou Saída. Todo arquivo deve ser aberto antes de seu primeiro uso e fechado (comando CLOSE) após o seu último uso. O que é feito pelo OPEN (ou CLOSE) não diz respeito ao programador, pois a própria máquina se encarrega de efetuar o serviço apropriado para cada tipo de saída. Mesmo para Entrada/Saída, usando Leitora de Cartões ou Impressora, que aparentemente não exigem preparação antes da leitura ou após a escrita (ao contrário do disco ou fita magnética, que exigem leitura do rótulo ou etiqueta, teste de rotação, enrolamento da fita), o uso do OPEN e CLOSE é obrigatório. O OPEN não lê o primeiro registro do arquivo, mas somente o rótulo ou etiqueta.

Exemplos:

OPEN INPUT ARQUIVO-MESTRE. (abre o arquivo de entrada chamado ARQUIVO-MESTRE)

OPEN OUTPUT ARQUIVO-SAIDA. (idem arquivo de saída)

OPEN INPUT ARQUIVO-MESTRE OUTPUT ARQUIVO-SAIDA. (abre os 2 arquivos acima)

OPEN INPUT ARQUIVO-1, ARQUIVO-2, ARQUIVO-3. (abre 3 arquivos de entrada ao mesmo tempo)

Os nomes de arquivos usados (no OPEN e CLOSE) devem ser os mesmos usados no nível FD do DATA DIVISION.

8.2.2. A opção I-O

O arquivo aberto servirá tanto para saída como para entrada, o que pode :ontecer com arquivos de acesso direto como o disco magnético.

18.2.3. Reversed

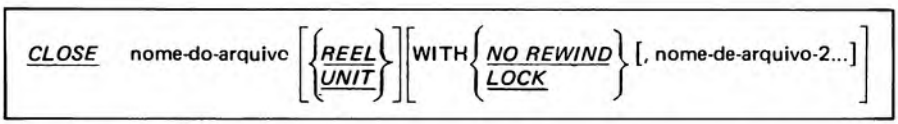
É usada quando queremos ler um arquivo de fita magnética do fim para o começo.

18.2.4. No Rewind

Usada quando queremos que a fita, ao chegar ao fim, não se reenrole automaticamente.

18.3. COMANDO CLOSE

18.3.1. Formato geral



É o oposto do OPEN, isto é, encerra e toma certas providências (como reenrolar a fita), após o último uso do arquivo mencionado. Este por sua vez deve ter sido aberto previamente por um comando OPEN.

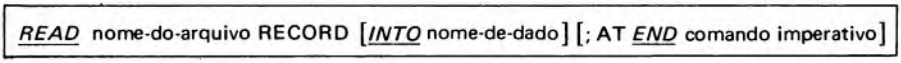
O comando CLOSE apresenta também outras variações com o uso das palavras WITH LOCK, WITH NO REWIND, REEL. Entretanto, estas opções devem ser efetuadas após consultar o operador do computador, ou o manual do Centro de Processamento de Dados. Se o comando CLOSE WITH LOCK for usado, o arquivo não poderá ser lido ou gravado sem a ação especial do operador.

Exemplo:

```
CLOSE ARQUIVO-MESTRE, ARQUIVO-1. (Fecha 2 arquivos)
```

18.4. COMANDO READ

18.4.1. Formato geral



O comando lê um registro completo de um arquivo externo de entrada e o traz para o local da memória reservado e definido pelo DATA DIVISION. O arquivo chamado deve ser previamente aberto pelo comando OPEN. O nome do arquivo usado é o mesmo definido no nível FD do DATA DIVISION.

O termo RECORD é opcional, mas serve para lembrar que está sendo lido um registro (RECORD) do arquivo.

Exemplos:

```
READ ARQUIVO-ENTRADA RECORD. (lê um registro do ARQUIVO-
-ENTRADA e coloca no local reservado ao registro definido através do nível 01
do DATA DIVISION).
```

READ ARQUIVO-ENTRADA RECORD INTO AREA-1 (idêntico ao caso anterior, mas o registro lido é imediatamente copiado em outro local AREA-1, que está definido no WORKING-STORAGE SECTION ou outro arquivo de saída).

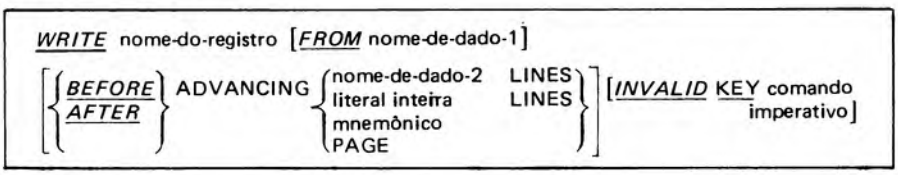
READ MESTRE RECORD INTO AREA-1 AT END GO TO PARAGRAFO-FIM. (idêntico ao caso anterior; o arquivo lido é MESTRE e após a leitura do *último registro* do arquivo, o programa desviará para o parágrafo PARAGRAFO-FIM).

A opção com AT END é sempre recomendada na prática.

O READ só manipula nome do arquivo formado sempre por um único tipo (tamanho e classe) de registro de nível 01, recebendo em um único local da memória, sucessivamente, os valores lidos. Se o arquivo lido tiver mais de um tipo de registro, o programador deverá fazer certos artifícios para poder distinguir um tipo de registro do outro, pois o conteúdo de todos os registros serão colocados sempre no mesmo local (nível 01) da memória.

18.5. COMANDO WRITE

18.5.1. Formato geral



Transfere o conteúdo da memória referente a um registro (nível 01 do DATA DIVISION) para o arquivo de saída. O arquivo usado pelo WRITE deve estar previamente aberto pelo comando OPEN. É importante notar que o WRITE usa o *nome do registro* (nível 01) e não o nome do arquivo como no READ, podendo, portanto, escrever mais de um tipo diferente de registro em um mesmo arquivo.

Exemplos:

WRITE REGISTRO-ITEM. (escreve o REGISTRO-ITEM de um arquivo de saída).

WRITE REGISTRO-ITEM FROM AREA-1. (o conteúdo de REGISTRO-ITEM é trazido de AREA-1, e então escrito no arquivo de saída).

Se o arquivo possui dois tipos diferentes de registros, podemos ter:

```
DATA DIVISION.
FILE SECTION.
FD ARQUIVO-SAIDA...
01 REGISTRO-DE-NOME-ITEM
   ...
01 REGISTRO-DE-QUANTIDADE
```

e podemos escrever no ARQUIVO-SAIDA o primeiro registro pelo comando WRITE REGISTRO-DE-NOME-ITEM.
e o segundo registro por WRITE REGISTRO-DE-QUANTIDADE.

18.5.2. A opção ADVANCING LINES e "mnemônico"

É a opção de avançar (no caso de Impressora) "n" linhas de impressão antes ou depois da operação de saída.

O termo "mnemônico" mais usado é PAGE (que permite mudança de página no formulário de impressão) e também o BOTTON (que posiciona a Impressora na última linha da página).

Exemplo:

```
WRITE REGISTRO-ITEM FROM AREA-1 BEFORE ADVANCING 10 LINES.
WRITE CABECALHO-TITULO AFTER ADVANCING PAGE.
WRITE REGISTRO-ITEM AFTER ADVANCING X LINES. (X é uma variável.)
```

18.5.3. A opção INVALID KEY

Serve para testar e se precaver contra condições inválidas que podem ocorrer com determinados tipos de arquivos, tais como registros cheios, registros que não estão de acordo com o método de acesso usado.

18.6. COMANDO ACCEPT

18.6.1. Formato geral

```
ACCEPT  nome-de-dado  [FROM nome-do-dispositivo]
```

A entrada normal de dados é efetuada pelo comando READ. O comando ACCEPT possibilita o recebimento através de certos dispositivos especiais como a máquina de escrever do console, terminal de teletype etc. É usado somente para pequeno volume de dados. Muitos centros que processam programas continuamente não permitem o uso do comando ACCEPT (ou DISPLAY) devido à interrupção de processamento e necessidade de interferência do operador.

Exemplos:

```
ACCEPT CORRECAO FROM CONSOLE. (CORRECAO é um nome definido no DATA DIVISION e CONSOLE deve estar especificado na seção SPECIAL-NOMES do ENVIRONMENT DIVISION).
```

18.7. COMANDO DISPLAY

18.7.1. Formato geral

```
DISPLAY { nome-de-dado-1 } [ { nome-de-dado-2 } ... ] [UPON nome-do-dispositivo]
        { literal-1      }   { literal-2      }
```

Escreve o conteúdo de uma ou mais variáveis ou códigos fixos (constantes) em determinado dispositivo de saída.

É usado para pequeno volume de dados de saída, como aviso ao operador, mensagem de erro etc.

Exemplo:

```
DISPLAY AVISO UPON CONSOLE. (o conteúdo da variável AVISO
será escrito na máquina do console).
DISPLAY 'PASSO 1' UPON PRINTER. (o código PASSO 1 é escrito na
máquina chamada PRINTER).
Não é recomendável usar os mesmos dispositivos de Entrada ou Saída
usados pelo READ ou WRITE, nos comandos ACCEPT e DISPLAY.
```

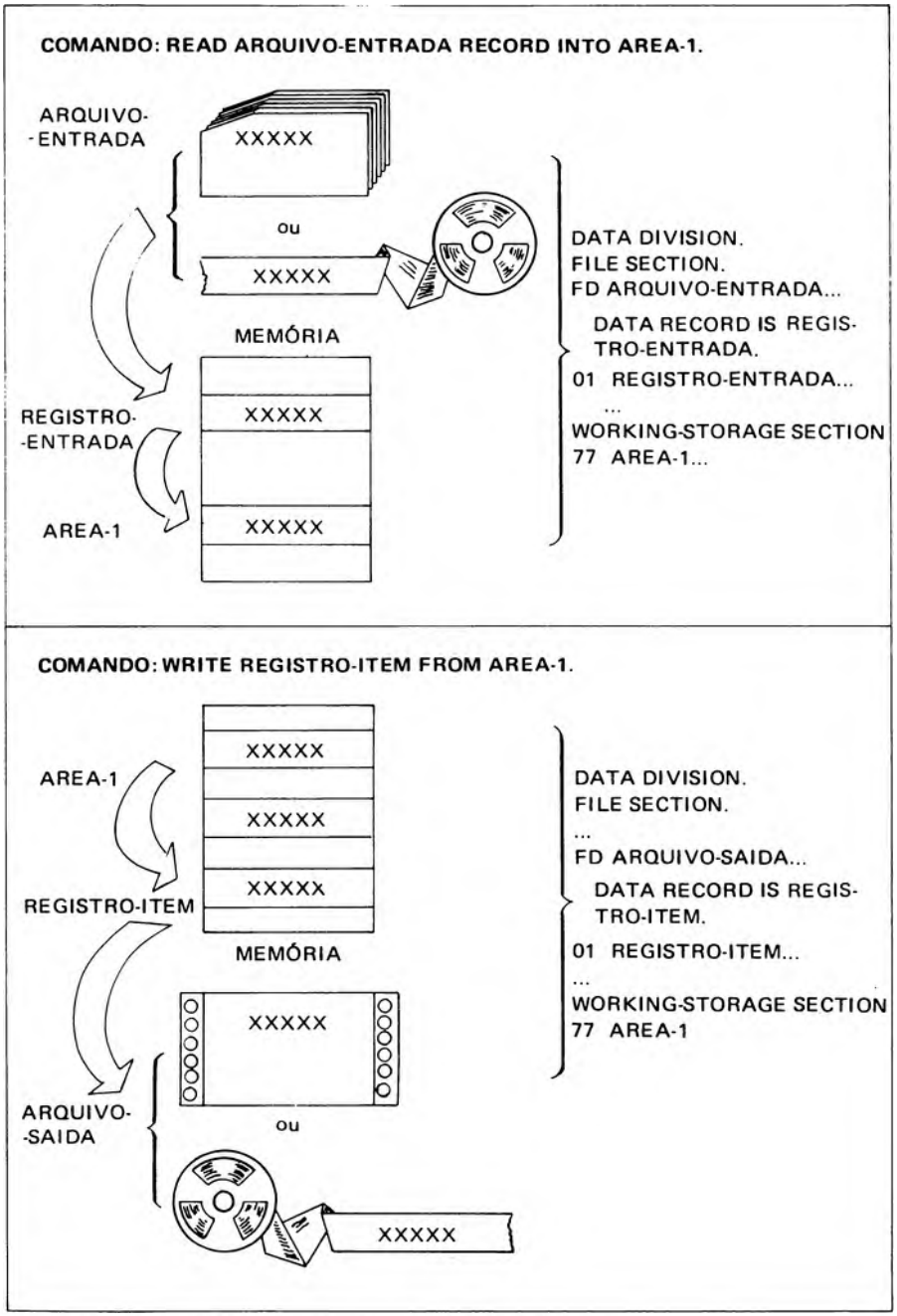


Figura 18.1. *Ilustrações dos comandos READ e WRITE.*

EXERCÍCIOS DE RECAPITULAÇÃO

1. Dizer o que fazem os comandos OPEN e CLOSE.
2. Dizer o que fazem os comandos READ e WRITE.
3. Dizer o que fazem os comandos ACCEPT e DISPLAY.
4. Qual a diferença entre os comandos READ e ACCEPT?
5. Qual a diferença entre os comandos WRITE e DISPLAY?

EXERCÍCIOS DE APLICAÇÃO

Um arquivo de Cartões perfurados contém em cada cartão o CÓDIGO (10 colunas) e NOME (70 colunas restantes) de um grupo de clientes:

1. Escrever os comandos de Entrada e Saída (OPEN, READ etc.) necessários para ler cada um dos cartões desse arquivo e imprimir em uma linha da impressora dando nomes aos registros e arquivos usados.
2. Escrever o DATA DIVISION correspondente ao exercício 1.
3. Queremos escrever, no topo da listagem, o cabeçalho com os dizeres "LISTAGEM DE CÓDIGO E NOME DOS CLIENTES." Escrever o DATA DIVISION e os comandos de Entrada e Saída necessários.

19 PROCEDURE DIVISION (COMANDO MOVE E MOVE CORRESPONDING)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

19.1. MOVIMENTO DE DADOS DE UM CAMPO PARA OUTRO

O comando usado é o MOVE.

Como tanto o *campo-origem* do dado como o *campo-destino* do dado devem estar definidos por um PICTURE conveniente, o movimento do dado acarreta edições de dados dependendo do PICTURE envolvido (vide Capítulo 22: EDIÇÃO DE DADOS POR PICTURE), a menos que ambos os PICTURE's sejam iguais.

19.2. COMANDO MOVE

Transfere o valor de um campo ou de uma constante para um ou mais campos. Os dados movidos não são destruídos.

Exemplos:

MOVE NUMERO TO AREA-1. (O valor de NUMERO é transferido para o local AREA-1).

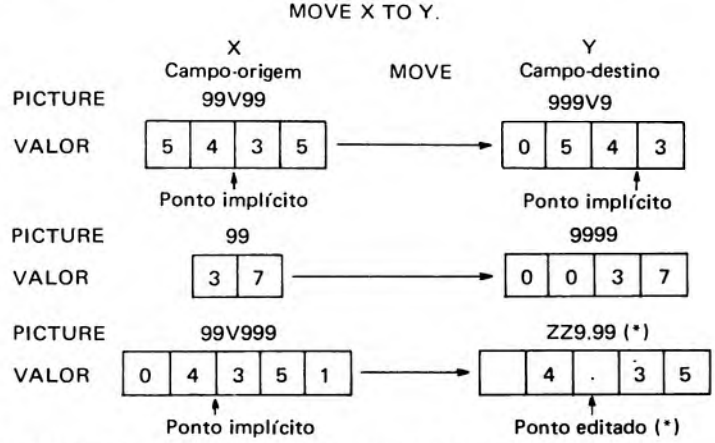
MOVE 200 TO CODIGO-1, CODIGO-2, CODIGO-3. (O valor 200 é colocado nos 3 locais diferentes).

19.3. MOVIMENTO DE DADOS NUMÉRICOS

São alinhados respeitando-se a posição do ponto decimal (implícito ou ponto "." editado) indicado pelo PICTURE do campo-destino. Os caracteres $, vírgula ponto, *, supressão de zeros etc. a serem editados devem ser colocados no PICTURE do campo-destino.

Se não houver indicação de ponto decimal, isto é, o valor numérico for inteiro sem ponto decimal, o movimento é efetuado dos dígitos da direita para os da esquerda, encostando-se o resultado no canto direito do campo-destino, a menos que a descrição JUSTIFIED LEFT do DATA DIVISION seja usada para esse dado.

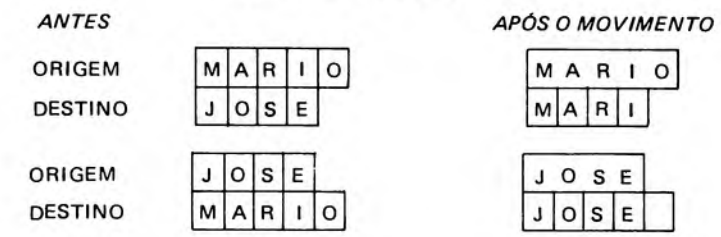
Exemplo:



19.4. MOVIMENTO DE DADOS ALFANUMÉRICOS

Os caracteres são movimentados da esquerda para a direita e encostados à esquerda do campo-destino (a menos que JUSTIFIED RIGHT seja usado). Dependendo do tamanho dos campos, há perda de caracteres ou preenchimento com espaços em branco.

Exemplo:



(*) (Descrição a serem vistas no Capítulo 22, edição de dados por PICTURE.)

19.5. COMANDO MOVE CORRESPONDING

Transfere os dados de um item de grupo (que é aquele que pode ser subdividido em outros subitens) para outro item de grupo.

Todo os subitens do campo-origem que tiverem o mesmo nome dos subitens do campo-destino (e somente estes) são movidos para o seu subitem homônimo.

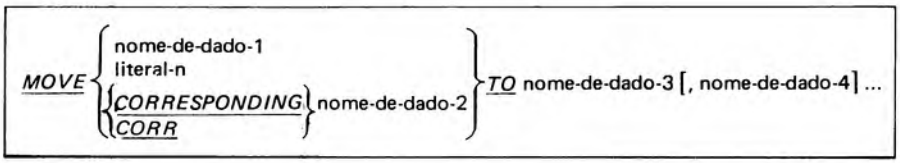
Exemplo:

```
MOVE CORRESPONDING NIVEL-DE-CONTROLE TO NIVEL-DE-ESTOQUE.
```

O movimento dos dados é o seguinte:

Os dados de nível 77 ou 88 não podem ser usados pelo MOVE CORRESPONDING, pois não se subdividem em outros subitens.

19.6. FORMATO GERAL DO COMANDO MOVE



EXERCÍCIOS

1. Escrever os comandos MOVE para transferir os conteúdos dos itens NÚMERO-LIDO, CÓDIGO-LIDO e VALOR-LIDO de um arquivo, respectivamente, para os itens NÚMSAÍDA, CÓDIGO-SAÍDA e VALOR-SAÍDA de um outro arquivo, sendo o DATA DIVISION:

```
01   LEITURA.
     02   NUMERO-LIDO...
     02   CODIGO-LIDO...
     02   PRECO-UNITARIO...
     02   VALOR-LIDO...
01   SAIDA.
     02   NUM-SAIDA
     02   VALOR-SAIDA
     02   CODIGO-OPERACAO
     02   CODIGO-SAIDA
```

2. Quais seriam as modificações necessárias no DATA DIVISION para executar as mesmas operações usando o comando MOVE CORRESPONDING?
3. Os dados seguintes estão descritos no DATA DIVISION como:

```
02   A    PICTURE IS 999V99      VALUE IS 100.
02   B    PICTURE IS 999         VALUE IS 35.
02   C    PICTURE IS 999V999     VALUE IS 35.345.
...
02
     03 X PICTURE IS 9(5) VALUE IS ZEROS.
     03 Y PICTURE IS 9(3)V9(2) VALUE IS 201.35.
```

Mostrar o conteúdo de cada dado após a execução de cada um dos comandos MOVE:

```
MOVE C TO A, B.
MOVE A TO X.
MOVE Y TO B.
MOVE X TO Y.
```

20 PROCEDURE DIVISION (COMANDOS EXIT E NOTE)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

COMANDOS QUE ESTABELECEM CONEXÃO COM O COMPILADOR

COMANDO EXIT

Exemplo:

COMANDO NOTE

Exemplos

20.1. COMANDOS QUE ESTABELECEM CONEXÃO COM O COMPILADOR

Existem comandos especiais que estabelecem conexão com o compilador COBOL, de modo que determinadas funções sejam executadas. Temos os comandos EXIT e NOTE.

20.2. COMANDO EXIT

É usado como ponto de retorno de um parágrafo usado como subprograma através do uso do comando PERFORM. Equivaleria ao comando RETURN usado no SUB-ROUTINE do FORTRAN, com a diferença de que em COBOL nem sempre é obrigatório o uso de EXIT para retorno ao ponto de chamada. Normalmente há o retorno ao ponto de chamada (do comando PERFORM) após a execução do último comando do parágrafo usado. Entretanto, quando o parágrafo possui mais de uma alternativa de ramificação proporcionada por comandos condicionais IF, passa a existir mais de um ponto de retorno. Tais pontos de retorno devem terminar, então, em outro parágrafo formado só pelo comando EXIT.

Exemplo:

```
        PERFORM PARTE-UM.

PARTE-UM. ...
        IF A IS EQUAL TO B GO TO RETORNO
        OTHERWISE MOVE B TO C.                    (retorno se A ≠ B)
RETORNO. EXIT.                                    (retorno se A = B)
```

Se não existisse o comando IF no parágrafo PARTE-UM, o retorno ocorreria sem o uso do EXIT.

```
        PERFORM PARTE-UM.

PARTE-UM. . . .
        MOVE B TO C. (retorno normal)
```

20.3. COMANDO NOTE

Serve para inserir comentários que sairão impressos na listagem do programa.

O formato deste comando é:

NOTE comentários

e o verbo NOTE impede que o comentário seja interpretado como um comando. Se a sentença com o comando NOTE e o comentário for a primeira do parágrafo, não deverá haver outros comandos no parágrafo. O comando NOTE pode ser usado no meio de outras sentenças.

Exemplos:

```
PARAGRAFO-A.   NOTE INICIO DE CALCULO DE JUROS.

PARAGRAFO-B    ADD TOTAL TO SOMA. WRITE SOMA.
               NOTE QUE SOMA NAO ULTRAPASSE LIMITE-A.
```

Nas versões mais modernas, este comanto NOTE foi substituído, bastando colocar simplesmente um asterisco (*) na coluna 7 de cartão, para que este seja tratado como comentário.

21 PROGRAMA COMPLETO EM COBOL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

O PROBLEMA
FLUXOGRAMA
PROGRAMA COBOL
Exercícios

21.1. O PROBLEMA

Duas fitas de entrada são lidas, sendo uma chamada FITA-DE-PEDIDO e a outra chamada FITA-DE-ESTOQUE que possuem registros da seguinte forma:

1 REGISTRO DE FITA-DE-PEDIDO

CÓDIGO DO ITEM (10 caracteres)	QUANTIDADE PEDIDA (5 dígitos)

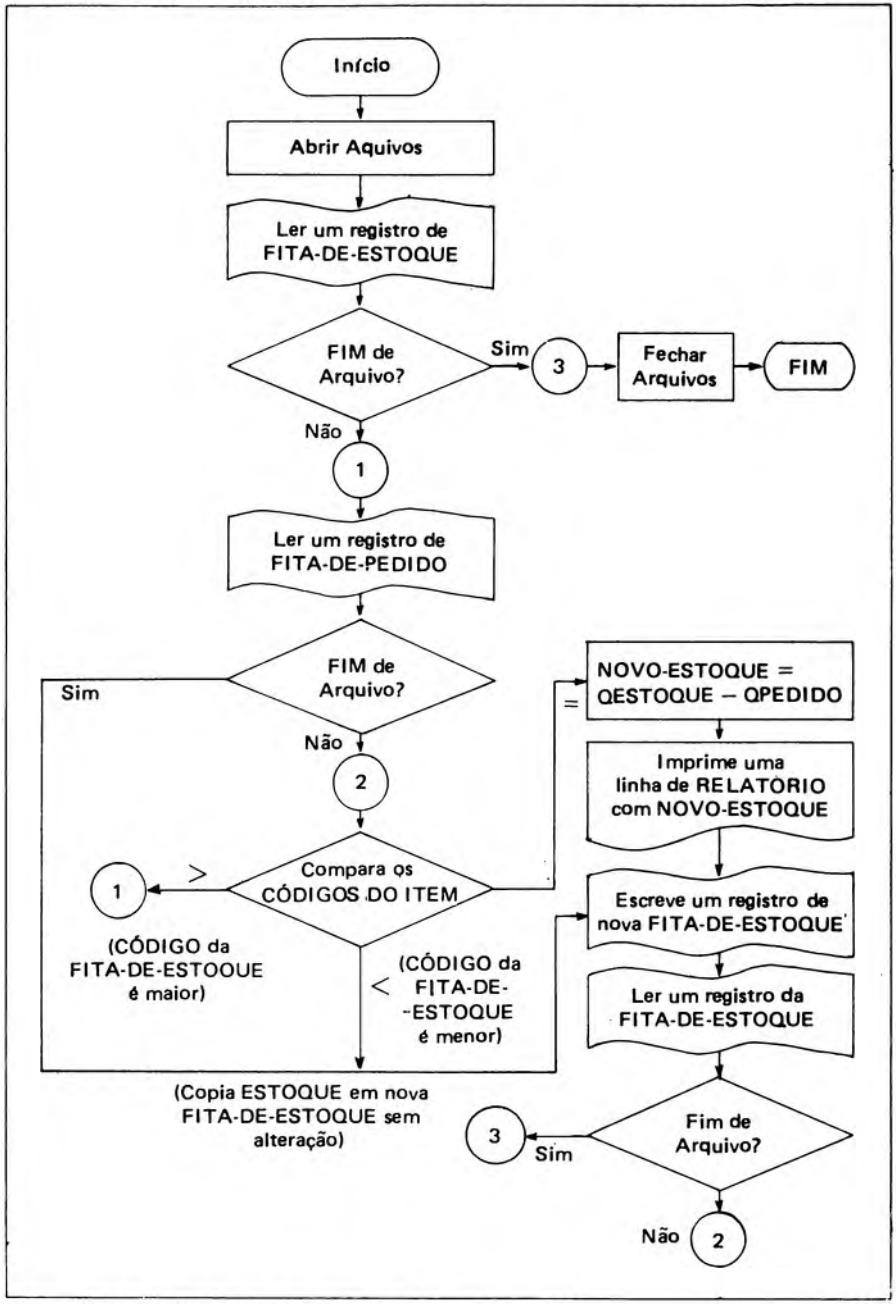
1 REGISTRO DE FITA-DE-ESTOQUE

CÓDIGO DO ITEM (10 caracteres)	QUANTIDADE EM ESTOQUE (5 dígitos)

Cada arquivo de fita está blocado com 10 registros cada bloco e está ordenado em seqüência crescente do CÓDIGO DO ITEM, mas o arquivo FITA-DE-PEDIDO não contém pedidos de todos os itens estocados de uma vez.

Cada pedido da FITA-DE-PEDIDO deve ser lido e comparado com a FITA-DE-ESTOQUE e, se os CÓDIGOS DO ITEM coincidirem, deve-se subtrair a quantidade pedida da quantidade em estoque e obter um novo valor de QUANTIDADE-EM-ESTOQUE.

21.2. FLUXOGRAMAS: ESTRUTURADO E NÃO-ESTRUTURADO



174 Figura 21.1. *Fluxograma da versão não-estruturada.*

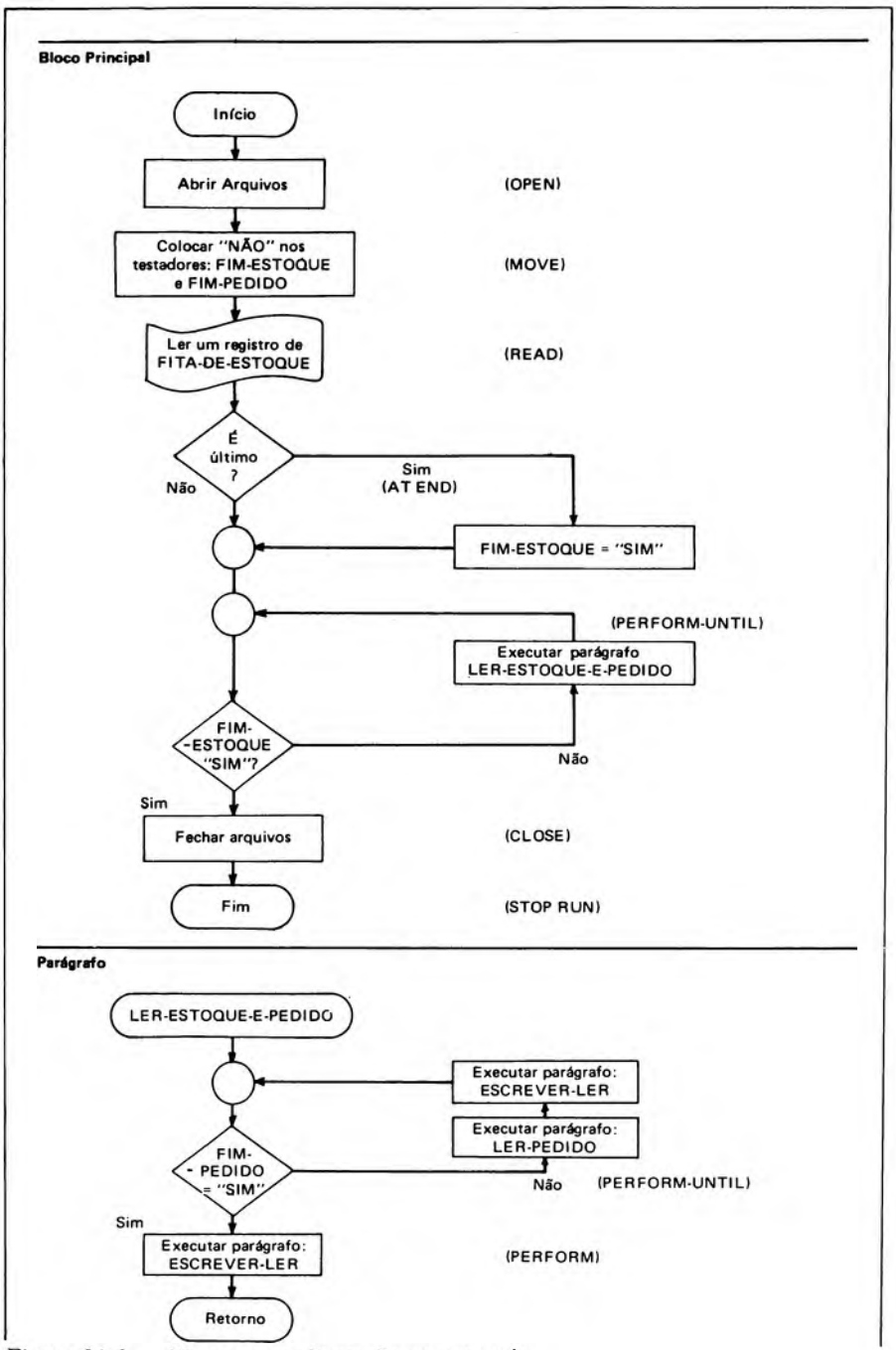
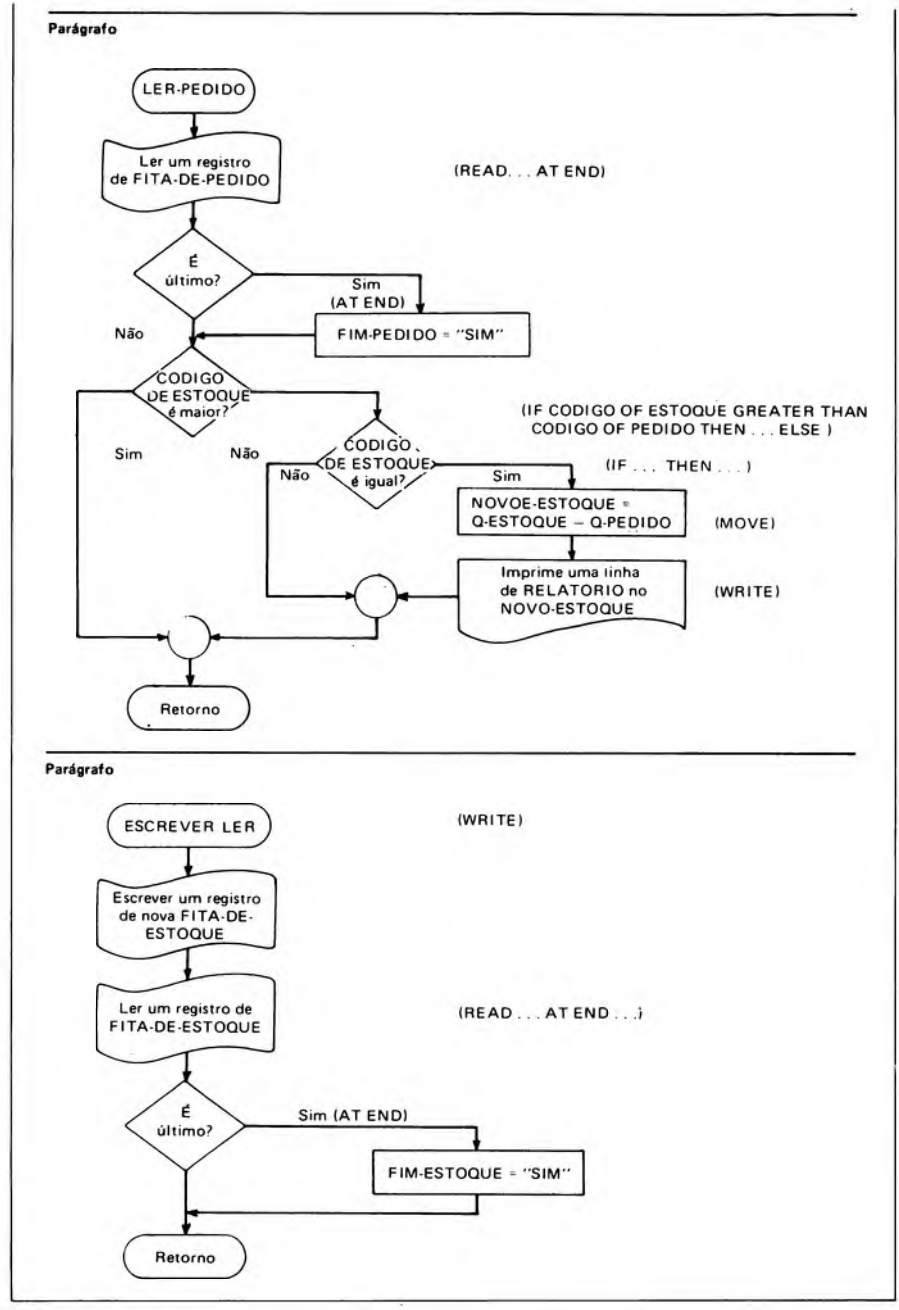


Figura 21.2. *Fluxograma da versão estruturada.*



176 Figura 21.2. *Continuação*

Pedem-se dois tipos de saída:

Primeira Saída: relatório pela Impressora contendo, em cada linha, os seguintes elementos:

10 espaços	CÓDIGO DO ITEM (10 caract.)	30 espaços	QUANTIDADE PEDIDA (5 dígitos)	30 espaços	QUANTIDADE EM ESTOQUE (5 dígitos)	30 espaços

Segunda Saída: gerar uma nova fita FITA-DE-ESTOQUE com a atualização dos registros afetados.

21.3. PROGRAMA COBOL (VERSÃO NÃO-ESTRUTURADA)

```
A    B
IDENTIFICATION DIVISION.
PROGRAM-ID. 'ATUALIZA'.
AUTHOR. XXXX.
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. SYSTEM-X.
OBJECT-COMPUTER. SYSTEM-X.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT FITA-DE-PEDIDO ASSIGN TAPE-1.
     SELECT FITA-DE-ESTOQUE ASSIGN TAPE-2.
     SELECT NOVA-FITA-ESTOQUE ASSIGN TAPE-3.
     SELECT RELATORIO ASSIGN PRINTER.
DATA DIVISION.
FILE SECTION.
FD   FITA-DE-PEDIDO BLOCK CONTAINS 10 RECORDS
     LABEL RECORDS ARE STANDARD DATA RECORD IS PEDIDO.
01   PEDIDO.
     02   CODIGO PICTURE IS X(10).
     02   QUANTIDADE PICTURE IS 9(5).
FD   FITA-DE-ESTOQUE BLOCK CONTAINS 10 RECORDS
     LABEL RECORDS ARE STANDARD DATA RECORD IS ESTOQUE.
01   ESTOQUE.
     02   CODIGO PICTURE IS X(10).
     02   QUANTIDADE PICTURE IS 9(5).
FD   RELATORIO BLOCK CONTAINS 120 CHARACTERS
     LABEL RECORDS ARE STANDARD DATA RECORD IS LINHA.

01   LINHA.
     02   FILLER PICTURE IS X(10).
     02   CODIGO PICTURE IS X(10).
     02   FILLER PICTURE X(30).
     02   Q-PEDIDO PICTURE IS 9(5).
     02   FILLER PICTURE IS X(30).
     02   Q-NOVO-ESTOQUE PICTURE IS 9(5).
     02   FILLER PICTURE IS X(30).
FD   NOVA-FITA-ESTOQUE BLOCK CONTAINS 10 RECORDS
     LABEL RECORDS ARE STANDARD DATA RECORD IS ESTOQUE-NOVO.
01   ESTOQUE-NOVO.
     02   CÓDIGO PICTURE IS X(10).
     02   QUANTIDADE PICTURE IS 9(5).
WORKING-STORAGE SECTION.
```

```
PROCEDURE DIVISION.
INICIO.  OPEN INPUT FITA-DE-PEDIDO FITA-DE-ESTOQUE
         OUTPUT NOVA-FITA-ESTOQUE RELATORIO.
     READ FITA-DE-ESTOQUE INTO ESTOQUE-NOVO AT END GO TO FINAE.
LEITURA-DE-PEDIDO.
     READ FITA-DE-PEDIDO RECORD AT END GO TO COPIA-ESTOQUE.
COMPARE.
     IF CODIGO OF ESTOQUE IS GREATER THAN CODIGO OF PEDIDO
     GO TO LEITURA-DE-PEDIDO.
     IF CODIGO OF ESTOQUE IS LESS THAN CODIGO OF PEDIDO
     GO TO COPIA-ESTOQUE.
     SUBTRACT QUANTIDADE OF PEDIDO FROM QUANTIDADE OF ESTOQUE
     GIVING Q-NOVO-ESTOQUE.
     MOVE CODIGO OF PEDIDO TO CODIGO OF LINHA.
     MOVE QUANTIDADE OF PEDIDO TO Q-PEDIDO.
     MOVE Q-NOVO-ESTOQUE TO QUANTIDADE OF ESTOQUE-NOVO.
     WRITE LINHA OF RELATORIO AFTER ADVANCING 2 LINES.
COPIA-ESTOQUE.
     WRITE ESTOQUE-NOVO.
     READ FITA-DE-ESTOQUE RECORD INTO ESTOQUE-NOVO AT END
     GO TO FINAE. GO TO COMPARE.
FINAE.
     CLOSE FITA-DE-PEDIDO FITA-DE-ESTOQUE NOVA-FITA-ESTOQUE
     RELATORIO. STOP RUN.
```

Observações:

- Todo código da FITA-DE-PEDIDO deve existir na FITA-DE-ESTOQUE. A FITA-DE--PEDIDO termina antes da FITA-DE-ESTOQUE após o que há somente cópia da FITA--DE-ESTOQUE na NOVA-FITA-DE-ESTOQUE.
- O nome do último parágrafo era FINAL que, entretanto, foi recusado pelo computador por ser PALAVRA-RESERVADA.
- O teste do programa também detectou outra falha de programação e que está citado no Exercício 4.

EXERCÍCIOS

Escrever o programa completo do:

1. Problema enunciado e resolvido como exemplo completo do DATA DIVISION.
2. Problema enunciado como EXERCÍCIO do DATA DIVISION.
3. No programa apresentado neste capítulo, explicar comando por comando, tanto no DATA DIVISION como no PROCEDURE DIVISION, o que aconteceria se os primeiros registros dos arquivos de entrada fossem:

FITA-DE-ESTOQUE

CODIGO	QUANTIDADE	CODIGO	QUANTIDADE	CÓDIGO	QUANTIDADE	
AX0101	00505	AX0104	01501	BX0200	02503	...

FITA-DE-PEDIDO

AX0104	00032	BX0200	00024	BX0202	00010	...

Nota — Colocar zeros à esquerda nos valores numéricos pois com espaços em branco, o campo é considerado alfanumérico.

4. O processamento real do programa completo deste capítulo mostrou que o mesmo não tem meio de processar casos em que, para o mesmo CÓDIGO DE ITEM, a QUANTIDADE PEDIDA é maior do que a QUANTIDADE EM ESTOQUE disponível. Acrescentar este teste tanto no FLUXOGRAMA como no PROGRAMA COBOL, fazendo-os imprimir a mensagem 'NÃO DISPONÍVEL EM ESTOQUE' pela Impressora.
5. Escrever o Procedure Division da versão estruturada do programa. Compare a versão estruturada com a não estruturada escrevendo as vantagens e as desvantagens de uma versão sobre a outra.
6. Repetir o processamento manual, solicitado no Exercício 3, com a versão estruturada.

Parte III

COBOL AVANÇADO

22 DATA DIVISION (EDIÇÃO DE DADOS POR PICTURE)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

A EDIÇÃO DE DADOS
CARACTERES E SINAIS USADOS NA EDIÇÃO
- Z: supressão de zeros à esquerda.
- $: insere $ à esquerda do valor.
- + ou -: edita o sinal + ou -.
- ,: insere vírgula na posição indicada.
- *: insere * à esquerda do valor.
- ϕ ou B: substitui caracteres por zero ou espaço.
- A edição do ponto decimal explícito.
- CR ou DB: coloca CR ou DB à esquerda ou à direita.

SINAL $ NA POSIÇÃO FLUTUANTE
SINAL POSITIVO OU NEGATIVO NA POSIÇÃO FLUTUANTE
COMO OCORRE A EDIÇÃO
EXEMPLOS
Exercícios

22.1. A EDIÇÃO DE DADOS

Os itens numéricos que serão impressos devem ser editados de modo conveniente para melhorar seu significado. Por exemplo, um valor monetário precisa estar precedido por Cr$ e conter a vírgula dos centavos. Certos grupos de caracteres podem ser inseridos em um item numérico. Devemos lembrar que o valor numérico não possui sinal + ou - ou ponto decimal (vírgula no Brasil) explícitos os quais devem ser colocados por edição. Valores numéricos negativos estão, em geral, representados na memória através de notação própria chamada "complemento de dois". A posição do ponto decimal foi implicitamente assumida pelo caráter V do PICTURE associado. Sendo assim, os valores numéricos NÃO POSSUEM, na memória, os sinais (+ ou -) ou o ponto decimal, os quais para serem escritos ou impressos devem ser acrescentados ao valor, através da edição por PICTURE.

A edição, via de regra, só é usada para itens numéricos, sendo efetuada por meio de caracteres apropriados usados no PICTURE.

22.2. CARACTERES E SINAIS USADOS NA EDIÇÃO

Os caracteres de edição que podem ser usados são apresentados a seguir.

22.2.1. Z

Suprime todos os zeros à esquerda de um número nas posições em que exista esse caráter. O "Z" não deve estar precedido pelo caráter 9 ou B ou ϕ (zero) no PICTURE.

Exemplos:

Valor do item	PICTURE	RESULTADO
2345	ZZ99	2345
0321	ZZ99	321
0032	ZZ99	32
0000	ZZ99	00

22.2.2. $

É usado na posição mais à esquerda do PICTURE e insere esse sinal à esquerda do valor numérico.

Exemplos:

Item	PICTURE	RESULTADOS
2345	$ZZ99	$ 2345
0032	$ZZ99	$ 32

22.2.3. A edição dos sinais + ou -

Podem ser colocados à direita ou à esquerda do valor absoluto do item.

O sinal editado do PICTURE acompanha o sinal interno do valor numérico, só aparecendo no resultado se o valor interno da memória concordar com o sinal indicado pela edição. Caso contrário aparecerá o sinal interno do valor do item ou espaço se o valor for positivo.

Exemplos:

Valor do item	PICTURE	RESULTADO
(+) 5435	+9(4)	+5435
(+) 5435	9(4)	5435
(-) 3212	9(4)-	3212-
(-) 3212	+9(4)	-3212 (sinal não concorda)
(+) 3212	-9(4)	3212 (idem)

Nota: (+) e (-) são sinais internos do valor na memória.

22.2.4. A edição da vírgula (,)

Insere uma vírgula na posição indicada. A vírgula que fica à esquerda do valor pode ser suprimida pelo Z. Devemos lembrar que na notação inglesa a vírgula é usada para separar os algarismos de cada milhar (mil, milhão etc.) e não a posição da parte decimal. Mas essa convenção pode ser alterada pela SPECIAL-NAMES. DECIMAL POINT IS COMMA.

Exemplos:

Item	PICTURE	RESULTADO
5499	$ 99,99	$ 54,99
0078	Z,ZZZ	78

22.2.5. Asterisco (*)

Nas posições dos zeros à esquerda do número é possível inserir asteriscos que servem para proteção de valores em cheques.

Exemplos:

Item	PICTURE	RESULTADO
543275	$**9 (4),99	$**5432,75
5432	**99	5432
0032	**99	**32
0000	****	espaço em branco
0000	**99	**00

22.2.6. A substituição por zero (0) e espaço (B)

Insere algarismo zero ou espaço em branco na posição indicada pelo algarismo zero ou letra B do PICTURE.

Exemplos:

Item	PICTURE	RESULTADO
5432	99BB99	54ØØ32
5432	990099	540032
12JUN83	XXBXXXBXX	12ØJUNØ83

21.2.7. A edição do ponto decimal implícito

Insere um ponto real ou explícito na posição correspondente ao ponto decimal implícito do valor numérico.

Exemplos:

5432	$9999.	$5432.
0009	$ZZ.ZZ	$ 9.00
87,65	ZZ9.99	87.65
(ponto implícito)		

22.2.8. CR ou DB

Representam Crédito ou Débito e devem ser usados na posição mais à direita do PICTURE. Se o valor do item for negativo, o sinal usado será impresso. Se for positivo, nada aparecerá.

Exemplos:

Item	PICTURE	RESULTADO
(-) 5432 (negativo)	$ 99,99CR	$ 54,32CR
(-) 5432 (negativo)	$ 99,99DB	$ 54,32 DB
(+) 5432 (positivo)	$ 99,99CR	$ 54,32

22.3. SINAL $ (CIFRÃO) EM POSIÇÃO FLUTUANTE

Serve para colocar um sinal $ logo após o dígito mais significativo depois de suprimir os zeros à esquerda. Entretanto, se o caráter Z for usado para suprimir zeros à esquerda, o sinal $ será colocado à esquerda dessas posições.

Exemplos:

Item	PICTURE	RESULTADO
5432	$$9(5)	$5432
0054	$$99	$54
0054	$$ZZ9	$ 54
0000	$$99	$00
0000	$$$$	espaços

22.4. SINAIS POSITIVOS OU NEGATIVOS FLUTUANTES

Suprimem os zeros à esquerda, colocando um sinal + ou -, de acordo com a descrição já feita.

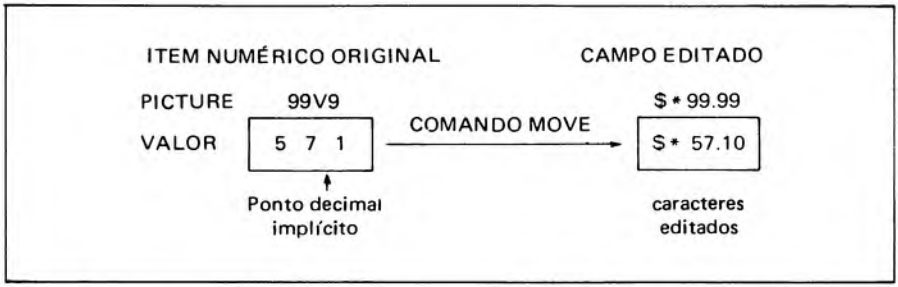
Exemplos:

Item	PICTURE	RESULTADO
(-) 5432 (negativo)	----99	-5432
(-) 0054 (negativo)	----99	-54
(+) 0054 (positivo)	----99	54
(-) 0023 (negativo)	+++99	-23
0000	++++	espaços

22.5. COMO OCORRE A EDIÇÃO

A edição só acontece por ocasião do movimento de um dado numérico de seu campo original para outro campo de memória executado por um comando MOVE ou equivalente (alguns comandos como o ADD-GIVING, READ-INTO e WRITE-FROM também podem movimentar dados, entre 2 campos da memória). Os caracteres a serem editados devem figurar no PICTURE do campo receptor do dado.

Exemplo:



A seguir temos na Figura 22.1 exemplo de itens de dado que foram lidos e colocados na memória por um PICTURE de entrada e depois editados por outro PICTURE de saída.

LEITURA		NA MEMÓRIA (Pontos são implícitos)	SAÍDA	
ITEM	PICTURE		PICTURE	DADO EDITADO
571	9(3)V	571 ↑	$Z, ZZ99.99	$␣␣␣571.00
(–) 571 (negativo)	S99V9	57 ↑ 1 (negativo)	$ * 99.99	$ * 57.10
(–) 571 (negativo)	S99V9	57 ↑ 1 (negativo)	$ * 99.99DB	$ * 57.10DB
(+) 571 (positivo)	99V9	57 ↑ 1 (positivo)	$ * 99.99DB	$ * 57.10
(–) 571 (negativo)	S99V9	57 ↑ 1 (negativo)	+99.99	–57.10
00650	9(2) V9(3)	0 ↑ 650	$*** 9.99	$*** 0.65

(␣ indica espaço)

Figura 22.1. *Itens de dados: leitura e inserção na memória por um PICTURE de entrada e edição por um PICTURE de saída.*

EXERCÍCIOS DE APLICAÇÃO

1. Preencher o valor e o posicionamento correto dos dados finais que devem aparecer na área de destino.

AREA ORIGEM		AREA DESTINO	
PICTURE	VALOR	PICTURE	VALOR
S999	578	-ZZ,ZZ9.99	578.00
S9(6)	000345	$ZZZ,ZZ9.99	
9(6)	000345	$ ***, ** 9.99	
9(4)V9(2)	578945	$ ***, ** 9.99	
S9(4)V(2)	123456	-ZZZZ9.99	
S9(4)V(2)	-123456	-ZZZZ9.99	
S9(6)	123456	ZZZZZZ.00	

2. Deseja-se imprimir pela impressora linha com os seguintes dizeres fixos e valores variáveis indicados por (Valor) (suprimir os zeros à esquerda; os dizeres fixos estão sublinhados).
   NUMERO (Valor) (10 espaços) QUANTIDADE (Valor) DZS. (10 espaços) PREÇO UNIT. CR$ (Valor), (10 espaços) PREÇO TOTAL CR$ (Valor), (10 espaços) OBSERVAÇÃO (20 espaços com conteúdo variável).
   Dar a descrição completa do registro.
   Nota – (Valor) são os algarismos a serem preenchidos pelo programa do computador, e cuja descrição deve ser fixada pelo programador.
3. Explicar a descrição do seguinte registro de dado:

```
01  PO-REC.
    02  PO-CARR-CONTR      PICTURE ++999.
    02  PO-MESS            PICTURE X(80) JUSTIFIED RIGHT.
    02  FILLER             PICTURE X(2).
    02  PO-PART.
        03  PART-1         PICTURE XX VALUE IS 'CD'.
        03  PART-2         PICTURE XX.
        03  PART-3         PICTURE XX VALUE IS 'MM'
    02  PO-COUNT           PICTURE $ZZ,ZZ9.
    02  FILLER             PICTURE X(38).
```

23 PROCEDURE DIVISION (COMANDO EXAMINE... TALLYING E EXAMINE... REPLACING) E (INSPECT... TALLYING...)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

COMANDO EXAMINE

- Examina o dado e conta ou substitui caracteres.
- Formato Geral.
- Exemplos.

A forma EXAMINE...TALLYING.

A forma EXAMINE...REPLACING.

A forma EXAMINE...TALLYING...REPLACING.

O COMANDO INSPECT

Exercícios.

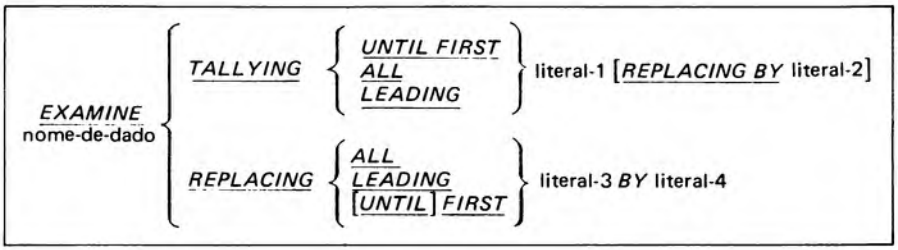
23.1. COMANDO EXAMINE

Este comando serve para *examinar* os caracteres que formam um campo ou item de dado e contar (TALLY) o número de ocorrência de um dado caráter no campo ou substituir (REPLACE) um caráter por outro. Um fato importante que deve ser notado é que o EXAMINE deve ser usado somente com itens de dados classificados como USAGE IS DISPLAY sendo errado usar em itens do tipo USAGE IS COMPUTATIONAL. O exame dos caracteres é sempre efetuado da esquerda para a direita no campo de dado.

A contagem do caráter escolhido é efetuada em um registro especial de contagem do COBOL chamado TALLY que fica disponível ao programador para uso no programa. O contador TALLY pode contar valores com até 5 dígitos e seu conteúdo é renovado automaticamente toda vez que o comando EXAMINE... TALLYING é usado.

23.1.1. Formato geral

Existem várias opções de uso do comando EXAMINE:



Exemplos:

Consideremos o item de dado chamado EXEMPLO-1 formado pelos seguintes caracteres:

3	3	–	C	O	D	3	5

23.1.2. A forma EXAMINE ... TALLYING

EXAMINE EXEMPLO-1 TALLYING ALL '3'. (conta todos os caracteres '3'. Conteúdo do TALLY = 00003).

EXAMINE EXEMPLO-1 TALLYING UNTIL FIRST 'D'. (conta número de caracteres até encontrar 'D'. Conteúdo do TALLY = 00005).

EXAMINE EXEMPLO-1 TALLYING LEADING '3'. (conta o número de vezes em que ocorre '3' antes de encontrar outro caráter. Conteúdo do TALLY = 00002).

23.1.3. A forma EXAMINE ... REPLACING (esta forma não altera o conteúdo do TALLY)

EXAMINE EXEMPLO-1 REPLACING ALL 'D' BY 'W'. (novo conteúdo de EXEMPLO-1 será 33-COW35).

EXAMINE EXEMPLO-1 REPLACING LEADING '3' BY 'X'. (novo conteúdo de EXEMPLO-1 será XX-COD35).

EXAMINE EXEMPLO-1 REPLACING UNTIL FIRST 'D' BY 'X'. (novo conteúdo de EXEMPLO-1 será XXXXXD35).

EXAMINE EXEMPLO-1 REPLACING FIRST '3' BY 'X'. (substitui apenas o primeiro '3' por 'X' e EXEMPLO-1 conterá X3-COD35).

23.1.4. A forma EXAMINE ... TALLYING ... REPLACING

EXAMINE EXEMPLO-1 TALLYING ALL '3' REPLACING BY 'X'. (conteúdo de TALLY = 00003. Conteúdo de EXEMPLO-1: XX-CODX5).

Outras opções têm significado análogo.

23.2. O COMANDO INSPECT

Foi introduzido no COBOL-74 e possui basicamente as mesmas funções do EXAMINE com a diferença que, em vez de usar o registro especial TALLY como contador, o programador pode utilizar seu próprio contador.
Exemplos:

```
EXAMINE   EXEMPLO-1    TALLYING ALL '3'.
...
MOVE ZERO TO CONTADOR.
INSPECT   EXEMPLO-1    TALLYING FOR ALL '3'
```

Os comandos acima são equivalentes.
Como se vê, no INSPECT, o programador é o responsável pela definição (no DATA DIVISION) do seu contador. Todos os exemplos do comando EXAMINE possuem comandos equivalentes com o INSPECT.

EXERCÍCIOS

1. O item CALENDARIO contém os seguintes caracteres: 3Ø/NOVEMBRO/198Ø. Diga o que acontece após execução de cada um dos seguintes comandos:

   ```
   EXAMINE CALENDARIO TALLYING ALL 'Ø' REPLACING BY SPACES.
   EXAMINE CALENDARIO REPLACING ALL '/' BY '*'.
   EXAMINE CALENDARIO TALLYING UNTIL FIRST 'B'.
   ```

2. Dado o item VALOR contendo os seguintes caracteres 000360570 escreva o comando EXAMINE para:
   - contar a ocorrência de caracteres 0 (zero) e substituí-los por caracteres '*'.
   - substituir os zeros à esquerda do número por '*' e colocar '$' na primeira posição (mais de um comando EXAMINE).

24 VARIÁVEIS SUBSCRITAS (CLÁUSULAS OCCURS, REDEFINES E O COMANDO PERFORM) E ÍNDICES (CLÁUSULA INDEXED E COMANDOS SET E SEARCH)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

24.1. OS SUBSCRITOS OU ÍNDICES

Servem para simplificar a manipulação de valores de mesmo formato agrupados em forma de tabelas ou matrizes. Sem o subscrito ou índice, teríamos que definir um a um esses valores no DATA DIVISION cada qual com nome diferente, o que acarretaria trabalho imenso para tabelas de 20, 50, 100 ou mais itens.

Os valores de uma tabela ou matriz podem ser subscritos ou indexados através de um único nome da tabela seguido por um ou mais índices ou subscritos que são literais inteiros ou nomes de dados com valores inteiros.

24.2. REGRA DE SEPARAÇÃO DOS ÍNDICES COM VÍRGULAS E ESPAÇOS

As variáveis subscritas devem ser escritas na seguinte forma:

nome-da-tabela-ou-matriz ␣ (índice)	(1 dimensão)
nome-da-tabela-ou-matriz ␣ (índice, ␣ índice)	(2 dimensões)
nome-da-tabela-ou-matriz ␣ (índice, ␣ índice, ␣ índice)	(3 dimensões)

Observação: "␣" representa a ocorrência de pelo menos um espaço e que é de uso obrigatório como separador entre dois nomes ou valores.

Exemplos:

```
TABELA (1)
TABELA (1, INDICE-2, INDICE-3)
MATRIZ (5,    10,    3)
```

Entretanto, esta rigidez de separação de índices e nomes de tabela não ocorre em todos os sistemas usando COBOL.

24.3. A CLÁUSULA OCCURS

Todas as variáveis subscritas usam a cláusula OCCURS na DATA DIVISION como veremos nos exemplos.

24.4. TABELAS OU VARIÁVEIS SUBSCRITAS DE UM NÍVEL OU UMA DIMENSÃO

É o caso mais simples de grupo de valores de mesmo tipo formando uma tabela simples ou um vetor.

Por exemplo, sejam 10 valores numéricos de 5 dígitos cada perfurados em um cartão:

1.º valor	2.º valor		10.º valor
99999	99999		99999

A descrição desses valores no DATA DIVISION através de variáveis subscritas é:

```
DATA    DIVISION.
FD      ......
01      TABELA-CARTÃO.
        02 TABELA PICTURE 9(5) OCCURS 10 TIMES.
```

No PROCEDURE-DIVISION, cada elemento lido pode ser usado na forma subscrita:

```
ADD TABELA (1)    TO SOMA. (soma o 1.º valor com SOMA)
ADD TABELA (2)    TO TABELA (10). (soma o 2.º valor de TABELA
no 10.º valor)
```

O subscrito pode ser um nome de dado, como:

```
ADD TABELA   (POSICAO)  TO SOMA.
```

e nesse caso o nome POSICAO deve ser definido no DATA DIVISION e assumir valor de 1 a 10.

24.5. TABELAS OU VARIÁVEIS SUBSCRITAS DE DOIS NÍVEIS OU DIMENSÕES

A seguinte tabela possui duas dimensões ou níveis:

NÚMERO DE ALUNOS FORMADOS

ANO	1970		1971		1972		1973		1974	
SEXO	MASC	FEM	MASC	FEM	MASC	FEM	MASC	FEM	MASC	FEM
	...	...	...	...	...	...	...	...	...	...

Em notação matricial a tabela teria a forma bi-dimensional com 5X2= 10 elementos:

ANO \ SEXO	MASC	FEM
1970		.....
1971		
1972		
1973		
1974	.....	

A definição dessa tabela é:

```
01  ALUNOS-FORMADOS-TOTAL.
    02  ALUNOS-FORMADOS-POR-ANOS OCCURS 5 TIMES.
        03  ALUNOS-FORMADOS PICTURE 9999
            OCCURS 2 TIMES.
```

Exemplos

Um *elemento genérico* da tabela seria identificado por:

- ALUNOS-FORMADOS (ANO, SEXO) onde ANO e SEXO são índices inteiros que variam de 1 a 5 e 1 a 2 respectivamente; e
- ALUNOS-FORMADOS (5, 1) conteria o número de alunos do sexo masculino formados em 1974.

Podemos referir a itens de nível superior usando menos índices, não especificando portanto o nível eliminado, e teríamos:

- ALUNOS-FORMADOS-POR-ANO (3) refere-se a *2 valores* referentes ao ano 1972, uma para MASC e outra para FEM.
- ALUNOS-FORMADOS-TOTAL refere-se à tabela toda (10 elementos) pois define o nível mais alto da tabela.

A operação com esses níveis seria, por exemplo:

- MOVE ZEROS TO ALUNOS-FORMADOS (1, 1). (coloca zeros no 1º elemento da tabela: nº de MASC no ano 1970).
- MOVE ZEROS TO ALUNOS-FORMADOS-POR-ANO (3). (coloca zeros nos 2 elementos MASC e FEM do ano 1972).
- MOVE ZEROS TO ALUNOS-FORMADOS-TOTAL (preenche toda a tabela com zeros).

24.6. TABELA DE TRÊS NÍVEIS

O COBOL permite até 3 níveis de índices ou variáveis subscritas. Como exemplo, seja a tabela seguinte:

TABELA DE TEMPERATURA

REGIÃO	A						B					
MES	ABRIL			MAIO			ABRIL			MAIO		
TEMPERATURA	BAIXA	MEDIA	ALTA	B	M	A	B	M	A	B	M	A

A tabela tem 2 X 2 X 3 = 12 elementos e é definida por:

```
01 TABELA-TEMPERATURA.
   02 REGIAO OCCURS  2 TIMES
      03 MES OCCURS  2 TIMES
         04 TEMPERATURA PICTURE 9(4) OCCURS 3 TIMES.
```

Os elementos são usados de modo análogo a duas dimensões.

24.7. USO DA CLÁUSULA "REDEFINES" PARA VALORES INICIAIS DA TABELA

O COBOL *não permite* o uso da cláusula VALUE IS na mesma sentença com o OCCURS, razão por que não foi possível dar valores iniciais aos elementos de uma tabela subscrita.

Exemplo:

Queremos usar os valores da seguinte tabela:

GRUPO	TAXA
1	1,25
2	2,00
3	2,50
4	3,00
5	3,45

Como não é permitido usar VALUE IS na definição, teríamos apenas:

```
01 TABELA-TAXAS.
   02 TAXA PIC 99V99 OCCURS 5 TIMES.
```

O que deve ser feito é definir primeiro uma área no WORKING-STORAGE SECTION, com 5 itens recebendo os respectivos valores iniciais:

```
WORKING-STORAGE SECTION.

01 TABELA-TAXA-INICIAL.
   02 TAXA-1 PIC   99V99   VALUE IS 1.25.
   02 TAXA-2 PIC   99V99   VALUE IS 2.00.
   02 TAXA-3 PIC   99V99   VALUE IS 2.50.
   02 TAXA-4 PIC   99V99   VALUE IS 3.00.
   02 TAXA-5 PIC   99V99   VALUE IS 3.45.
```

Depois, então, definir:

```
01 TABELA-TAXAS.  REDEFINES TABELA-TAXA-INICIAL.
   02 TAXA PIC 99V99 OCCURS 5 TIMES.
```

Agora as áreas TABELA-TAXAS e TABELA-TAXA-INICIAL ocupam a mesma posição da memória assumindo os mesmos valores (semelhante ao comando EQUIVALENCE do FORTRAN).

24.7.1. Observações sobre o uso de OCCURS e REDEFINES

- Um campo definido no DATA DIVISION com OCCURS deve sempre aparecer na forma subscrita na PROCEDURE DIVISION e inversamente não se usam variáveis subscritas sem o uso do OCCURS.
- Não se usa OCCURS no nível 01.
- O valor do índice ou subscrito deve ser um número inteiro maior que zero.
- A cláusula REDEFINES deve sempre ser a primeira descrição usada.
- Não se usa o REDEFINES no nível 01 na FILE SECTION.
- Todos os valores de uma mesma tabela de valores com subscritos devem possuir o mesmo formato (PICTURE), o mesmo acontecendo com as tabelas definidas pelo REDEFINES.

24.8. USO DO COMANDO "PERFORM" COM VARIÁVEIS SUBSCRITAS

A variável ou contador que controla a repetição (looping) da execução do comando PERFORM pode ser usada como índice da variável subscrita.

Exemplos:

a) Sejam os parágrafos:

```
PARAGRAFO-1
   ADD TABELA (SUBSCRITO-1) TO VALOR-FINAL.
   . . .
PARAGRAFO-3.
```

onde TABELA é uma variável subscrita definida no DATA DIVISION.

O comando PERFORM PARAGRAFO-1 THRU PARAGRAFO-3 VARYING SUBSCRITO-1 FROM 1 BY 1 UNTIL SUBSCRITO-1 = 10.

executa os parágrafos citados por 10 vezes, variando o índice SUBSCRITO-1.

b) O PERFORM pode usar mais de um índice, como no caso:

```
PERFORM PARAGRAFO-1 THRU PARAGRAFO-X VARYING INDICE-1
FROM 1 BY 1 UNTIL INDICE-1 = 10
AFTER INDICE-2 FROM 2 BY 2 UNTIL  INDICE-2 = 10.
```

onde o INDICE-1 assume 10 valores de 1 a 10 e
INDICE-2 assume valores 2, 4, 6, 8 e 10.

Para 3 índices, acrescenta-se a frase AFTER INDICE-3 . . .

Este procedimento é idêntico ao uso do comando "DO" do FORTRAN com variáveis subscritas.

24.9. OBSERVAÇÃO SOBRE O USO DE QUALIFICADOR "OF" OU "IN"

Não se pode usar índices em um qualificador (elemento após o OF ou IN) ou em um elemento qualificado (elemento antes do OF ou IN), embora seja permitido usar índices no nome inteiro qualificado, colocando os índices após o nome qualificado.

Exemplo:

Seja a tabela definida por

```
01 TABELA.
   02 NIVEL-A OCCURS 2 TIMES.
      03 NIVEL-B OCCURS 4 TIMES.
         04 NIVEL-ITEM PIC 9(5) OCCURS 10 TIMES.
```

As seguintes formas são corretas e equivalentes:

```
NIVEL-ITEM IN NIVEL-B OF NIVEL-A (1, 2, 2)
NIVEL-ITEM OF NIVEL-B (1, 2, 2)
NIVEL-ITEM OF NIVEL-A (1, 2, 2)
NIVEL-ITEM (1, 2, 2)
```

As seguintes formas são incorretas:

```
NIVEL-ITEM (1, 2, 2) OF NIVEL-B IN NIVEL-A
NIVEL-ITEM (2) OF NIVEL-B (2) IN NIVEL-A (1)
NIVEL-ITEM (2) OF NIVEL-B (1, 2)
```

24.10. USO DE ÍNDICES (INDEX) COM OS COMANDOS SET E SEARCH

Alguns compiladores podem trabalhar tanto com índices como subscritos. Um índice é semelhante ao subscrito e operado do mesmo modo. Entretanto, do ponto de vista da eficiência do programa objeto, os índices são preferíveis aos subscritos, e, portanto, devem ser usados quando disponíveis.

Os valores dos índices podem ser manipulados pelos comandos SET e SEARCH.

Temos também de usar uma nova cláusula – INDEXED BY – em conjunção com o OCCURS, para indicar que os itens da tabela serão usados através desses índices. Cada índice exige uma definição na WORKING-STORAGE SEC-

TION com o nome do índice seguido pela cláusula USAGE IS INDEX, sem o uso de PICTURE.

Exemplo:

```
WORKING-STORAGE SECTION.
77 INDEX-A USAGE IS INDEX.                    (definição de um índice)
   ...
01 ACCNT-TABELA.
   02 ACCNT-DADOS OCCURS 100 TIMES INDEXED BY INDEX-A.
      03 ACC-NUMERO PICTURE 9(5).              (dados da tabela usa índices)
   ...
PROCEDURE DIVISION.
   ...
   SET INDEX-A TO 1.                          (define um valor do índice)
   ...
   SEARCH ACCNT-DADOS, WHEN ACC-NUMERO (INDEX-A)
      EQUAL TO ACC-X GO TO PROCESSAR.         (usa um valor indexado)
   ...
```

24.10.1. O comando SET

Este comando fornece um valor qualquer, incrementa ou decrementa os valores de um índice definido por USAGE IS INDEX.

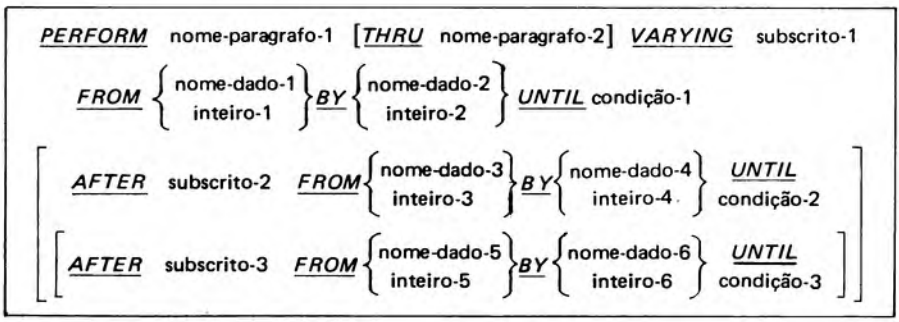
Exemplos:

```
SET INDEX-X TO 2.
SET INDEX-Q UP BY 1.                          (incrementa)
SET K-INDICE DOWN BY 2.                       (decrementa)
```

24.10.2. O comando SEARCH

Este comando pesquisa automaticamente uma tabela de valores indexados pela cláusula INDEXED BY. O índice é aumentado automaticamente até que um valor desejado (de ACC-NUMERO) seja encontrado. A opção SEARCH ALL melhora a eficiência da busca pois usa um algorítmo especial denominado BUSCA BINÁRIA.

24.11. FORMATO GERAL (COMANDO PERFORM COM SUBSCRITOS)



EXERCÍCIOS

1. Definir uma tabela de 15 elementos numéricos e escrever o comando que calcule a soma desses elementos em um item chamado SOMA.
2. Explicar o que faz o seguinte trecho de programa:

```
DATA DIVISION.
01 TABELA.
   02 LINHAS OCCURS 10 TIMES.
      03 COLUNAS PIC 9(5) OCCURS 10 TIMES.
...
PROCEDURE DIVISION.
...PERFORM PARAGRAFO-X VARYING L FROM 1 BY 1
UNTIL L = 10 AFTER C FROM 1 BY 1 UNTIL C = 10.
...
PARAGRAFO-X.
   ADD COLUNAS (L, C) TO COLUNAS (1, 1).
   ADD COLUNAS (L, C) TO COLUNAS (L, 1).
```

25 USO DE SUB-ROTINAS EXTERNAS EM COBOL (COMANDOS ENTER, CALL E A LINKAGE SECTION)

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

SUB-ROTINAS E SUBPROGRAMAS

COMANDOS ENTER E CALL PARA CHAMAR SUB-ROTINAS

LINKAGE SECTION
- Para definir os parâmetros ou argumentos da sub-rotina.

RETORNO DA SUB-ROTINA

EXEMPLO DE SUB-ROTINA EM COBOL
Sub-rotina.
Programa Principal.

SUB-ROTINA CHAMANDO SUB-ROTINA

DIFERENTES FORMAS DE CHAMAR SUB-ROTINAS
Sistema ICL-1900 (inglesa).
Sistema IBM (americana).

Exercícios

25.1. SUB-ROTINAS E SUBPROGRAMAS

Já vimos que o comando PERFORM permite chamar e executar determinado número de parágrafos do mesmo programa como se fossem subprogramas ou *sub-rotinas internas,* sendo que o retorno ao ponto de chamada pode ser executado pelo comando EXIT ou pelo final do último parágrafo executado.

O COBOL permite, além disso, o uso de *sub-rotinas externas* que são outros programas independentes escritos em qualquer outra linguagem disponível no sistema, inclusive em COBOL.

A chamada da sub-rotina é executada pelos comandos ENTER e CALL e os argumentos ou parâmetros passados, do programa principal para a sub-rotina, devem estar descritos na sub-rotina na seção chamada LINKAGE SECTION do DATA DIVISION.

25.2. COMANDOS ENTER E CALL

A chamada de uma sub-rotina externa por um programa COBOL é em geral efetuada pelo seguinte grupo de comandos:

```
          ...
nome-do-parágrafo. ENTER  nome-da-linguagem.
                   CALL   nome-da-sub-rotina [USING lista de parâmetros]
nome-do-parágrafo. ENTER  COBOL.
```

Os dois comandos ENTER, um para entrada na outra linguagem da sub-rotina e outro para volta ao COBOL, devem estar escritos cada um em linhas separadas formando parágrafos isolados. Além disso:

- (nome-da-linguagem): pode ser FORTRAN, ASSEMBLER, PL/I, COBOL, e é a linguagem usada pela sub-rotina;
- (nome-da-sub-rotina): é um nome de identificador externo que *não deve* estar definido no DATA DIVISION do programa principal;
- (lista de parâmetros): é uma seqüência de nome de dados usados como parâmetros e que são passados à sub-rotina chamada; esta lista de nomes deve estar definida no DATA DIVISION (FILE SECTION ou WORKING-STORAGE SECTION) do programa principal e também no LINKAGE SECTION da sub-rotina chamada se esta for um programa em COBOL. Em geral *não se permite* usar variáveis subscritas como parâmetros de sub-rotinas.

25.3. LINKAGE SECTION

É a seção do DATA DIVISION onde são descritos os dados recebidos como parâmetros ou argumentos de uma sub-rotina. O LINKAGE SECTION só aparece em programas em COBOL usados como sub-rotinas. Os dados recebidos como parâmetros podem ser de nível 77 ou de nível 01, se contiverem subitens, e devem ter cada um a mesma descrição PICTURE do parâmetro usado no programa principal.

Os nomes dos parâmetros podem ser diferentes dos nomes usados no programa principal, mas a ordem de definição deve ser a mesma.

25.4. RETORNO DA SUB-ROTINA

O comando final de uma sub-rotina que indica o retorno ao programa principal pode ser indicado pelo comando RETURN ou EXIT PROGRAM dependendo do computador usado.

25.5. EXEMPLO DE SUB-ROTINA EM COBOL

Escrever uma sub-rotina em COBOL que calcule o valor da fórmula:

```
VOLUME = (LADO ** 2) * ALTURA
```

e usar esta rotina para calcular o volume para:

```
LADO = 20 , ALTURA = 10 e
LADO = 30 , ALTURA = 5.
```

25.5.1. Sub-rotina

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
      SUB-VOLUME. (nome-da-sub-rotina)
   . . .
ENVIRONMENT DIVISION.
   . . .
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
   77 VOLUME  PIC  999V99.   (descrição dos parâmetros recebidos)
   77 LADO    PIC   99V99.
   77 ALTURA  PIC   99V99.
PROCEDURE DIVISION.
PARTE-CALCULO.
   COMPUTE VOLUME ROUNDED = (LADO * 2) * ALTURA.
RETORNO.
   RETURN.                          (ou EXIT PROGRAM)
```

25.5.2. Programa principal

```
   . . .
DATA DIVISION.
   . . .
WORKING-STORAGE SECTION.
   77 VOLUME-UM   PIC  999V99.  (1º Grupo de parâmetros)
   77 LADO-UM     PIC   99V99   VALUE IS 20.00.
   77 ALTURA-UM   PIC   99V99   VALUE IS 10.00.
   77 VOLUME-DOIS PIC  999V99.  (2º Grupo de parâmetros)
   77 LADO-DOIS   PIC   99V99   VALUE IS 30.00.
   77 ALTURA-DOIS PIC   99V99   VALUE IS  5.00.
PROCEDURE DIVISION.
   . . .
PAR-1.  ENTER COBOL. (Primeira chamada da sub-rotina)
        CALL SUB-VOLUME USING VOLUME-UM LADO-UM
        ALTURA-UM.
PAR-VOLTA. ENTER COBOL.

   . . .
PAR-2.  ENTER COBOL. (Segunda chamada da sub-rotina)
        CALL SUB-VOLUME USING VOLUME-DOIS
        LADO-DOIS ALTURA-DOIS.
PAR-VOLTA2. ENTER COBOL.

   . . .
```

25.6. SUB-ROTINA CHAMANDO SUB-ROTINA

A chamada de outra sub-rotina externa por uma sub-rotina é em geral permitida até determinado nível. Entretanto o manual específico do computador em uso deve ser consultado para usar a forma correta.

25.7. DIFERENTES FORMAS DE CHAMAR SUB-ROTINAS

O que foi explicado é sempre válido de modo geral, mas o modo de usar o ENTER e o CALL pode variar de computador para computador.

25.7.1. Sistema ICL-1900 (inglesa)

Usa as formas:

```
CALL nome-da-rotina [USING lista-de-parâmetros]
```

para sub-rotinas escritas em COBOL e

```
ENTER nome-da-linguagem nome-da-sub-rotina [USING lista-de-parâmetros]
```

para sub-rotinas escritas em outras linguagens.

Exemplo:

```
ENTER FORTRAN ROTINA-RAIZ-QUADRADA USING ARGUMENTO.
```

25.7.2. Sistema IBM

Usa as formas (as sub-rotinas são escritas em qualquer linguagem):

```
ENTER LINKAGE.
CALL nome-da-rotina [USING lista-de-parâmetros]
ENTER COBOL
```

para chamar a sub-rotina do programa principal e

```
ENTER LINKAGE.
ENTRY nome-da-rotina-1 [USING lista-de-parâmetros]
ENTER COBOL.
```

para chamar uma sub-rotina a partir de uma outra sub-rotina.

EXERCÍCIOS

1. Escrever a sub-rotina externa que calcula a soma de 20% e 30% de um valor dado e chamar pelo programa principal. Explicar o mesmo cálculo usando o PERFORM.
2. Escrever uma sub-rotina externa que calcule o valor de $x^2 + x^3 + x^4 + x^5 + x^6$ sendo x dado pelo programa principal. A sub-rotina deve usar uma segunda sub-rotina para calcular a operação $(x)^k$ com x e k dados pela primeira sub-rotina.

26 MÉTODOS DE ACESSO AOS DADOS DE UM ARQUIVO: AS CLÁUSULAS ACCESS E ORGANIZATION

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

TIPOS DE ORGANIZAÇÃO DE DADOS EM UM ARQUIVO
- Modo Seqüencial Padrão.
- Modo Indexado.
- Modo Direto.
- Modo Relativo.

MÉTODOS DE ACESSO AOS ARQUIVOS.

MÉTODO DE ACESSO SEQÜENCIAL.

MÉTODO DE ACESSO ALEATÓRIO.

A CLÁUSULA ACCESS.

A CLÁUSULA ORGANIZATION.

OBSERVAÇÃO IMPORTANTE.

26.1. TIPOS DE ORGANIZAÇÃO DE DADOS EM UM ARQUIVO

Em um arquivo, os dados podem ser colocados ou organizados de diversas maneiras, a fim de poder aproveitar ao máximo as vantagens oferecidas pelo tipo físico (fita magnética, disco magnético, tambor magnético etc.) do dispositivo que forma o arquivo.

Em COBOL podem ser adotados os seguintes tipos de organização de dados:

26.1.1. Modo seqüencial padrão

Neste caso, os dados são colocados no arquivo em posição seqüencial de acordo com a sua ordem de criação. O modo seqüencial padrão pode ser adotado em qualquer tipo de arquivo e é adotado automaticamente quando a cláusula ORGANIZATION não for usada no ENVIRONMENT DIVISION.

26.1.2. Modo indexado

Neste caso, a posição de cada dado no arquivo é determinada por índices ou keys mantidos pelo sistema operacional do computador que, usando uma técnica apropriada de pesquisa e busca de dados, coloca ou busca os dados no arquivo. Os arquivos de dados com organização indexada devem ser criados em dispositivos de acesso direto como o disco magnético e são especificados pela cláusula ORGANIZATION IS INDEXED no ENVIRONMENT DIVISION.

26.1.3. Modo direto

Neste caso, a posição dos dados é determinada através de um esquema de endereçamento relativo de pistas (*tracks*) do arquivo formado. O modo direto deve ser designado para dispositivos de acesso direto e especificado pela cláusula ORGANIZATION IS DIRECT.

26.1.4. Modo relativo

Neste caso é usado um esquema de endereçamento relativo de registros do arquivo, onde a posição dos dados é determinada com relação ao primeiro registro do arquivo. Os dispositivos usados devem ser de acesso direto e a cláusula de especificação é ORGANIZATION IS RELATIVE.

26.2. MÉTODOS DE ACESSO AOS ARQUIVOS

Para cada tipo de organização de dados adotado em um arquivo é necessário usar um programa contendo o método ou rotina de acesso apropriada de leitura ou de escrita dos dados desse arquivo.

Os métodos de acesso podem ser de dois tipos básicos:

- de *acesso seqüencial,* quando os dados são lidos ou escritos de modo seqüencial e,
- de *acesso aleatório,* quando os dados são lidos ou escritos no arquivo, de modo pré-especificado pelo programador.

Para cada tipo de organização de dado do arquivo, o sistema que fornece a linguagem COBOL possui várias técnicas ou métodos de acesso de dados no arquivo. Por exemplo, no sistema IBM os métodos possíveis de acesso são:

Para Organização	Método de Acesso	
	Seqüencial	Aleatório
Seqüencial		
Padrão	QSAM	–
Indexada	QISAM	BISAM
Direta	BSAM1 (Direta)	BDAM1 (Direta)
Relativa	BSAM2 (Relativa)	Bsam2 (Relativa)

Os significados das iniciais são:

QSAM: Queued Sequential Access Method.
QISAM: Queued Indexed Sequential Access Method.
BISAM: Basic Indexed Sequential Access Method.
BSAM: Basic Sequential Access Method.
BDAM: Basic Direct Access Method.

A escolha da técnica de formação ou organização de arquivos e, conseqüentemente, do método de acesso depende do tipo de programa, sua finalidade e complexidade, sendo tarefa reservada a analistas e programadores com bastante conhecimento do computador que está em uso.

26.3. A CLÁUSULA ACCESS

Esta cláusula do ENVIRONMENT DIVISION indica a maneira pela qual os registros e dados de um arquivo são lidos ou escritos.

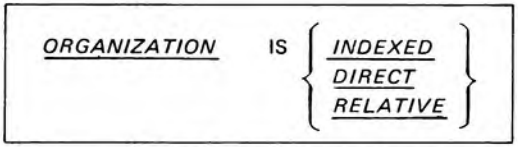
O formato geral é

Se esta cláusula não for usada, assume-se que ACESS IS SEQUENTIAL. Se ACESS IS RANDOM for usada, o arquivo usado deve estar em um dispositivo de acesso direto como o disco magnético e a cláusula ORGANIZATION deve ser usada.

26.4. A CLÁUSULA ORGANIZATION

Esta cláusula do ENVIRONMENT DIVISION indica o tipo de organização de dado adotado por um arquivo particular.

Sua forma geral é



Se a cláusula ORGANIZATION for omitida, assume-se que o arquivo possui organização seqüencial padrão.

26.5. OBSERVAÇÃO IMPORTANTE

Se o programador desejar usar um tipo de organização de dados diferente da SEQÜENCIAL PADRÃO, deve sempre consultar o MANUAL APROPRIADO DA LINGUAGEM COBOL que está sendo usada, a fim de se inteirar das especificações e propriedades do método de acesso usado, e da maneira com que cada registro de arquivo é criado. No capítulo seguinte apresentamos estudo ilustrado de uso de arquivo em disco magnético.

27 ARQUIVOS EM DISCOS MAGNÉTICOS: UM PROGRAMA PARA CRIAR E ATUALIZAR ARQUIVO DE ACESSO ALEATÓRIO OU DIRETO

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

OS ARQUIVOS DE ACESSO SEQÜENCIAL: CARTÕES, FITAS MAGNÉTICAS E IMPRESSORAS
O DISCO MAGNÉTICO
EXEMPLO 1: PROGRAMA COBOL PARA CRIAR UM ARQUIVO-MESTRE EM DISCO MAGNÉTICO
Descrição do Problema
Comentários sobre os comandos e cláusulas para o uso do arquivo em disco.

EXEMPLO 2: PROGRAMA QUE ATUALIZA UM ARQUIVO-MESTRE DE CONTROLE DE ESTOQUE EM DISCO MAGNÉTICO
Descrição do problema
Arquivo mestre: DISK-MAST
O Arquivo de Transações TR-FILE
Fluxograma
Programa COBOL do Exemplo de Atualização
Comentários sobre as cláusulas e comandos do exemplo-2
COMENTÁRIOS GERAIS SOBRE OS EXEMPLOS: VERSÃO ESTRUTURADA
Exercícios

27.1. OS ARQUIVOS DE ACESSO SEQÜENCIAL: CARTÕES, FITAS MAGNÉTICAS E IMPRESSORAS

Os arquivos de dados definidos em forma de cartões, formulários impressos e fitas magnéticas constituem os arquivos com característica essencialmente SEQÜENCIAL, isto é:

- os registros de dados (cartões, linhas de impressão ou registro de fita) são lidos ou escritos um a um, em seqüência, do começo ao fim;
- para procurar um registro no meio do arquivo ou no fim, é necessário ler todos os registros que o antecedem;

- o retorno a um registro lido ou escrito é impossível (cartões ou linhas de impressão) ou inconveniente (fita magnética);
- os registros devem estar previamente classificados antes do seu processamento através de um programa normal.

A maior vantagem dos arquivos de acesso seqüencial é o seu BAIXO CUSTO, porém a velocidade de suas unidades de processamento é relativamente baixa em comparação aos arquivos de acesso chamado diretos ou aleatórios, como o disco magnético. Seu uso é principalmente recomendado em firmas ou serviços de porte médio.

As velocidades típicas de operação em arquivos de acesso seqüencial são:

Leitura de cartões perfurados:	1000 a 2000 cartões/minuto ou cerca de 1500 caracteres/segundo.
Impressão de formulários:	1000 a 2000 linhas/minuto ou cerca de 2000 caracteres/segundo.
Leitura ou gravação em fita magnética	50.000 a 180.000 caracteres/segundo.

Até o momento, apresentamos sempre programas COBOL utilizando arquivos de acesso exclusivamente SEQÜENCIAL.

27.2. O DISCO MAGNÉTICO

É o arquivo de acesso direto ou aleatório mais utilizado tanto de maneira fixa como removível do sistema. Outros tipos de arquivos de acesso direto como o tambor magnético e o *data-cell* são arquivos fixos e de menor uso, servindo como memória de trabalho adicional de certos computadores.

As características principais do disco magnético são:

ASPECTO FÍSICO: Disco de metal recoberto com material magnetizável em ambas as faces. Possui dimensão variável de 10 a 20 polegadas. Em cada face possui pistas (tracks) circulares e concêntricas onde os dados são gravados. Existem de 100 a 1000 pistas em cada face dependendo do tamanho e tipo do disco.

TIPO: Existem discos fixos e discos removíveis. Cada unidade (Driver) de leitura e gravação de disco pode manipular de 1 até 10 discos removíveis (Figura 27.1.). Os disquetes (DISKETTES) são discos de porte menor e mais baratos utilizados em microcomputadores.

VANTAGENS: Alta velocidade de transferência de dados: 100.000 a 400.000 caracteres/segundo.
Alta capacidade de armazenar dados: de 10×10^6 a 400×10^6 caracteres por disco.
Acesso (uso) de seus dados pode ser feito tanto de modo direto (aleatório) ou seqüencial.

DESVANTAGEM: Custo das unidades é mais alto que outros tipos de arquivos.

USO: Recomendado para processamentos que:
- requerem operação rápida e freqüente. Exemplo: freqüência diária.
- índice de uso dos registros é alto a ponto de compensar o uso do arquivo SEM CLASSIFICAÇÃO PRÉVIA.

27.3. EXEMPLO 1

PROGRAMA COBOL PARA CRIAR UM ARQUIVO-MESTRE EM DISCO MAGNÉTICO (Adaptado de: McCracken e Garbassi. *Programação COBOL*. Atlas, S. Paulo, com permissão da editora.)

27.3.1. Descrição do problema

Ler os cartões de um arquivo de cartões e formar um arquivo de acesso direto em disco magnético, isto é, copiar um arquivo de cartões em disco magnético.

FLUXOGRAMA: ver Figuras 27.1 e 27.2.

PROGRAMA COBOL: ver Figuras 27.1 e 27.2.

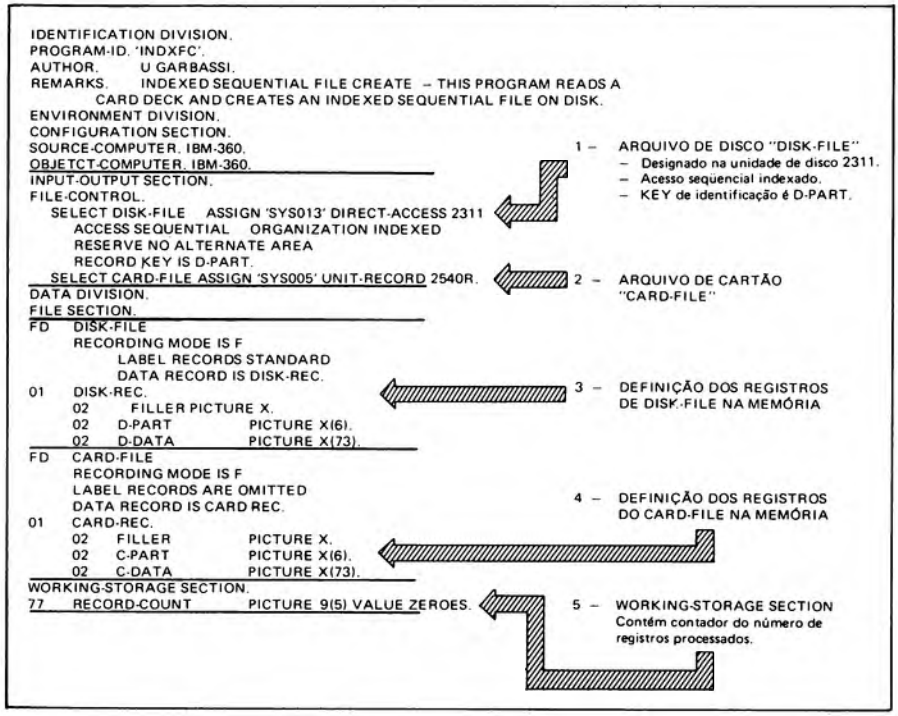
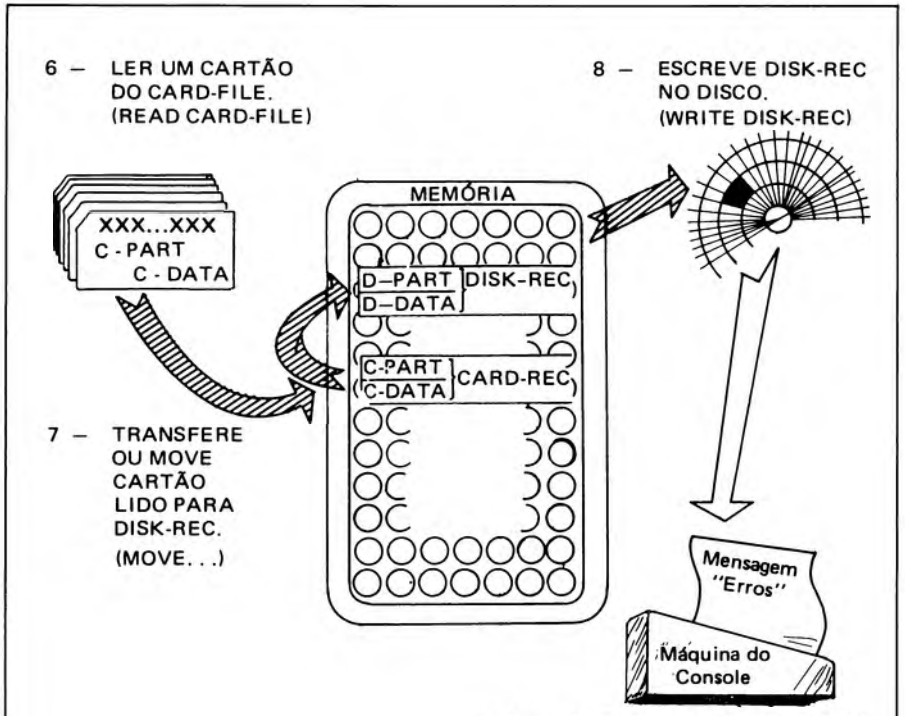


Figura 27.1. *Um programa para criação de um arquivo-mestre de acesso aleatório (disco magnético): as três primeiras divisões.*




```
      PROCEDURE DIVISION.
           OPEN INPUT CARD-FILE OUTPUT DISK-FILE.
      READ-IN.
(6)        READ CARD-FILE AT END GO TO EOJ-RTN.
(7)        MOVE CARD-REC TO DISK-REC.
(8)        WRITE DISK-REC  INVALID KEY
(9)             DISPLAY 'INVALID KEY' C-PART· GO TO READ-IN.
           DISPLAY DISK-REC.
           ADD 1 TO RECORD-COUNT.
           GO TO READ-IN.
      EOJ-RTN.
           DISPLAY 'NUMBER OF RECORDS ON DISK' RECORD-COUNT.
           CLOSE CARD-FILE DISK-FILE.
           STOP RUN.
```

Figura 27.2. *Um programa para criação de arquivo mestre de acesso aleatório: PROCEDURE DIVISION. (Adaptada da ob. cit.)*

27.3.2. Comentários sobre os comandos e cláusulas para o uso do arquivo em disco

ORGANIZATION IS INDEXED – esta cláusula cria um arquivo de disco com organização indexada, isto é, a procura de seus registros é efetuada através de uso de índices ou KEYS.

ACCESS IS SEQUENTIAL – cláusula opcional que serve somente para indicar que estamos lendo dados ordenados seqüencialmente. Pode ser omitida.

RECORD KEY D-PART – Indica o nome D-PART como índice ou KEY de identificação dos registros do disco.

WRITE DISK-REC INVALID KEY DISPLAY. . .
O KEY ou o índice de identificação do arquivo deve estar em seqüência ordenada, e não deve existir índices em duplicata. Se ocorrer alguma dessas irregularidades durante a gravação do arquivo será necessário estar prevenido, por exemplo, escrevendo a mensagem 'INVALID KEY' É. . ."

DISPLAY DISK-REC – escreve no terminal do console cada registro criado no disco, e é útil durante o teste do programa, mas deve ser eliminado para execução normal do programa em problema real, pois consome muito tempo e papel do terminal.

27.4. EXEMPLO 2:

PROGRAMA QUE ATUALIZA UM ARQUIVO-MESTRE DE CONTROLE DE ESTOQUE EM DISCO MAGNÉTICO (Adatada da ref. já citada).

27.4.1. Descrição do problema

Atualizar os itens (registros) de mercadorias estocadas que estão em um ARQUIVO-MESTRE chamado DISK-MAST de acordo com as modificações fornecidas pelos registros de um ARQUIVO DE TRANSAÇÕES ou MODIFICAÇÕES chamado TR-FILE. Este programa é semelhante ao programa de atualização de itens estocados em arquivo seqüência, estudado no capítulo sobre UM PROGRAMA COMPLETO EM COBOL.

27.4.2. Arquivo-mestre: DISK-MAST

Cada registro do DISK-MAST (ver Figura 27.3.) é formado pelos campos de:

D-PART:	número da peça que identifica o item e que é usado como índice ou KEY.
D-DESCR:	descrição do item.
D-QOH:	quantidade disponível.
D-QOO:	quantidade pedida mas ainda não recebida.
D-RP:	ponto ou valor da quantidade em estoque abaixo do qual será feito um pedido de compra ou fabricação; inclui quantidade em estoque e quantidade pedida anteriormente.
D-RQ:	quantidade ótima ou econômica que deve ser comprada ou fabricada em cada pedido efetuado.
D-UNITS:	unidade usada.
D-UNIT-PRICE.	preço unitário.
D-YTD-SALES:	vendas acumuladas do item.
D-CODX:	código que indica se é matéria-prima (código 1) ou produto acabado (código 2).
D-STATUS:	campo onde será colocado um valor muito alto (chamado HIGH-VALUE), se quisermos eliminar ou ignorar todo o registro do disco.

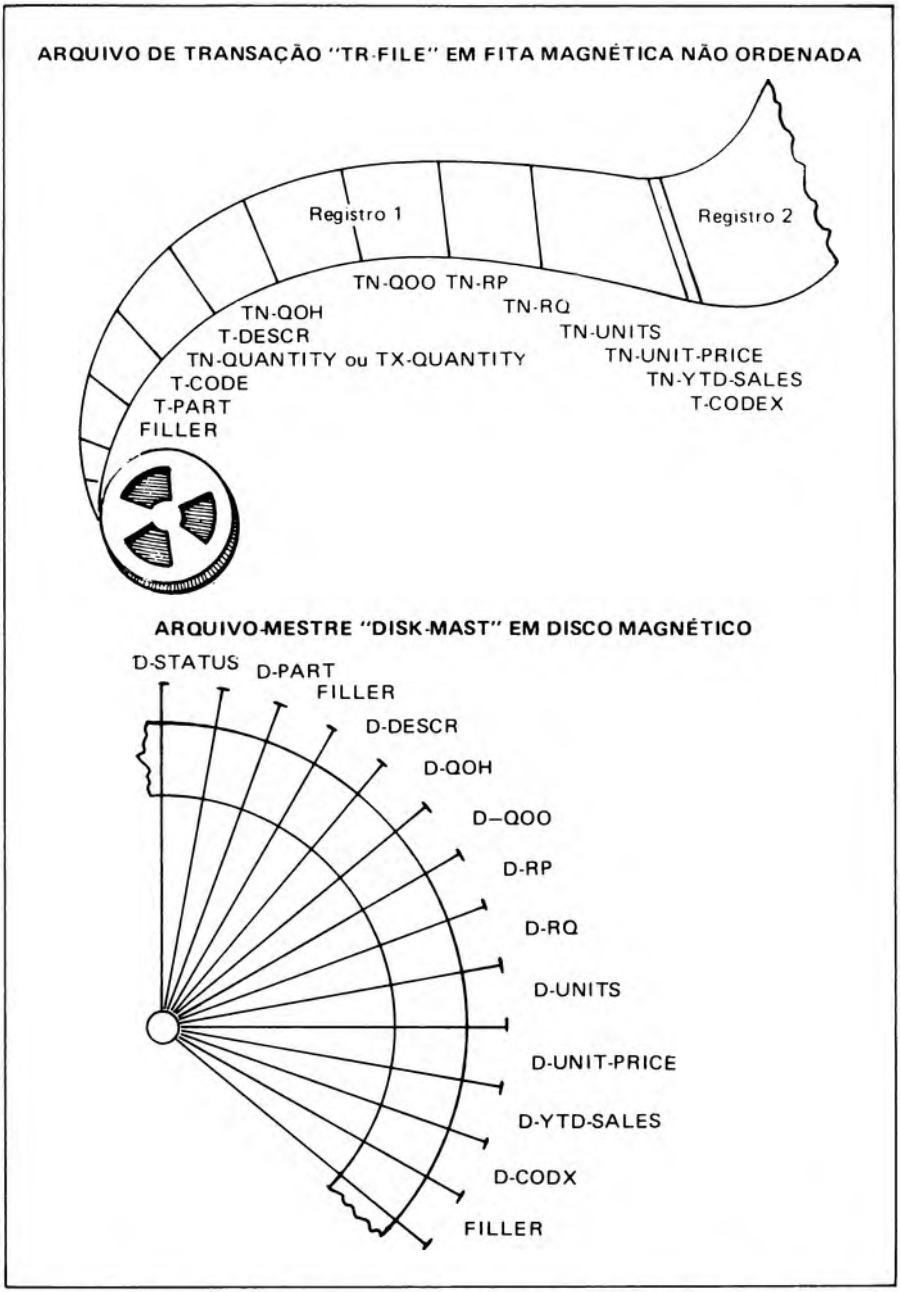


Figura 27.3. *Registros dos arquivos: MESTRE (em disco) e de TRANSAÇÃO (em fita magnética) do exemplo 2.*

27.4.3. Arquivo de Transações TR-FILE

Contém as operações diárias de atualização, que podem ser do seguinte tipo:

CÓDIGO 1 – INCLUSÃO de um novo item no arquivo mestre.
CÓDIGO 4 – PEDIDO de compra para um item.
CÓDIGO 6 – CANCELAMENTO ou ELIMINAÇÃO de um item do arquivo mestre.

Os demais códigos 2, 3 e 5 que formam respectivamente as operações de AJUSTE DA QUANTIDADE, RECEBIMENTO DE UM PEDIDO FEITO, e SAÍDA OU GASTO NORMAL DO ITEM não foram incluídos neste exemplo. Exemplos dessas operações podem ser estudados na referência utilizada, bem como no capítulo: PROGRAMA COMPLETO EM COBOL já citado neste livro.

Os códigos de operação estão colocados no campo T-CODE de cada registro. Os demais campos de cada registro estão ilustrados na Figura 27.3., e definidos no DATA SECTION do programa e são análogos aos campos dos registros do arquivo mestre (DISK-MAST).

Os registros destes arquivos NÃO ESTÃO ORDENADOS pelo número da peça.

27.4.4. Fluxograma

Ver Figura 27.4.

27.4.5. Programa COBOL do exemplo de atualização

Ver Figuras 27.4 e 27.5.

27.4.6. Comentários sobre as cláusulas e comandos do exemplo-2

ACCESS IS RANDOM – permite processar as transações diretamente e sem classificação prévia.

SYMBOLIC KEY IS W-PART – O índice ou KEY (número da peça) do registro de transação lido é T-PART, que é transferido para um local chamado W-PART do WORKING-STORAGE SECTION. Então, o comando READ DISK-MAST usa esse key W-PART para procurar e ler, do disco, o registro que contém esse valor.

RECORD-KEY IS D-PART – é o índice ou key do registro que está no disco.

INVALID KEY – esta parte do comando READ ou WRITE do disco é ativada em condição não prevista. Por exemplo, não existe no arquivo-mestre um key D-PART com o mesmo valor do W-PART, e nesse caso uma mensagem ou alguma operação adicional é feita.

REWRITE – comando usado para REESCREVER no disco o registro que foi atualizado por operação normal ou que deve ser cancelado através da colocação de valor especial chamado HIGH-VALUE no campo especial D-STATUS. Ao ser lido, esses registros cancelados pelo HIGH-VALUE serão ignorados pelo programa.

WRITE – comando que escreve um registro novo (não se trata daquele registro que estava no disco e vai ser reescrito) no disco.

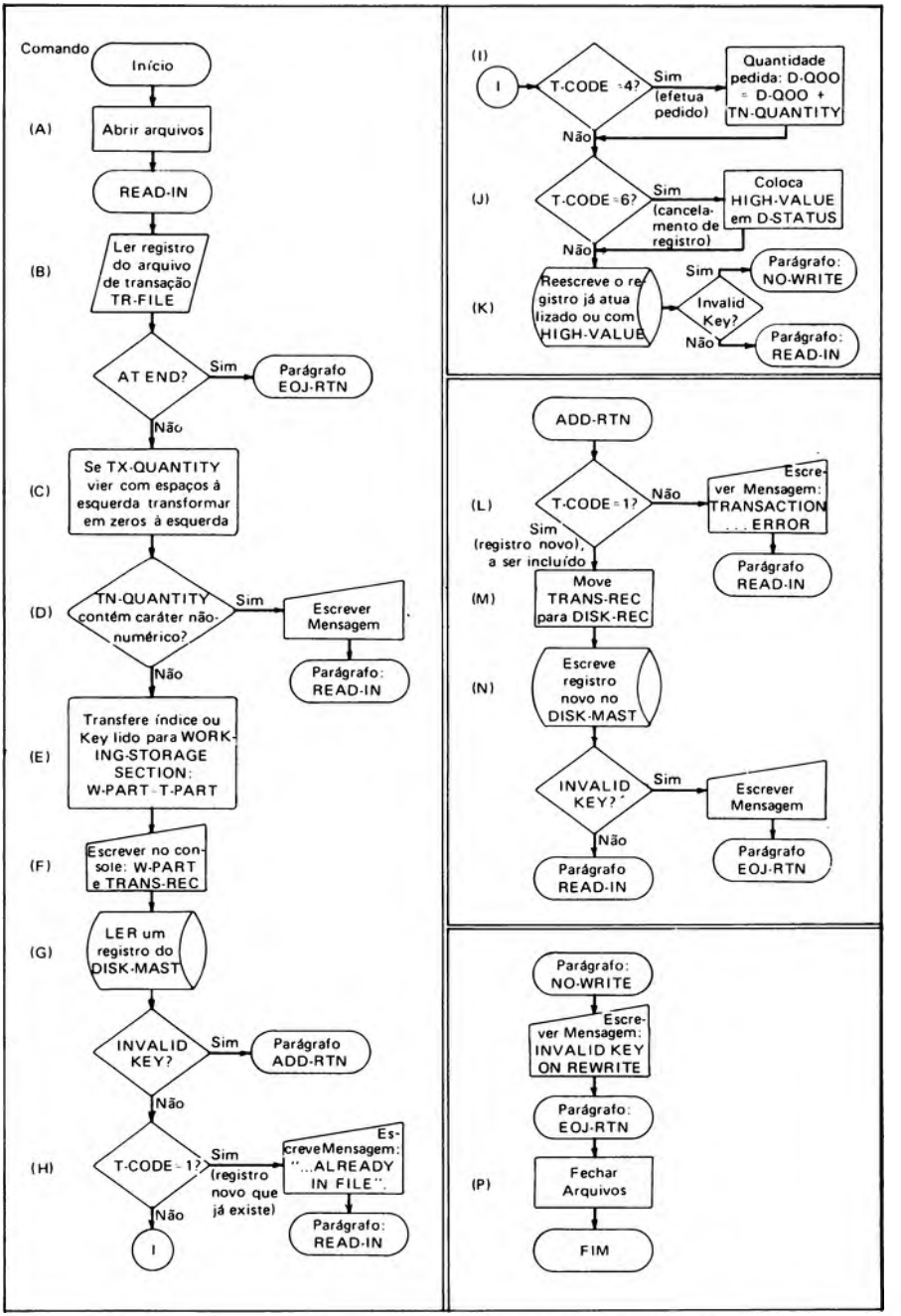


Figura 27.4. *Fluxograma do Exemplo-2: Atualização de Arquivo de Disco.*

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'INDXFU'.
AUTHOR. U GARBASSI.
REMARKS. THIS PROGRAM READS A TRANSACTION FILE IN RANDOM SEQUENCE
      AND UPDATES AN INDEXED SEQUENTIAL FILE ON DISK.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360.
OBJET-COMPUTER.  IBM-360.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT DISK-MAST      ASSIGN 'SYS013' DIRECT-ACCESS 2311
         ACCESS IS RANDOM
         ORGANIZATION INDEXED
         RESERVE NO ALTERNATE AREA
         SYMBOLIC KEY W-PART
         RECORD KEY IS D-PART.
      SELECT TR-FILE       ASSIGN 'SYS005' UNIT-RECORD 2540R.
DATA DIVISION.
FILE SECTION.
FD  DISK-MAST
         RECORDING MODE IS F
         LABEL RECORDS ARE STANDARD
         DATA RECORD IS DISK-REC.
01 DISK-REC.
   02 D-STATUS                          PICTURE X.
   02 D-PART                        PICTURE X(6).
   02 FILLER                        PICTURE X(5).
   02 D-DESCR                       PICTURE X(15).
   02 D-QOH                                 PICTURE 9(4).
   02 D-QOO                                 PICTURE 9(4).
   02 D-RP                                  PICTURE 9(4).
   02 D-RQ                                  PICTURE 9(4).
   02 D-UNITS                               PICTURE X(4).
   02 D-UNIT-PRICE                  PICTURE 9999V9999.
   02 D-YTD-SALES                   PICTURE 9(8).
   02 D-CODX                        PICTURE 9.
   02 FILLER                        PICTURE X(16).
FD TR-FILE
   RECORDING MODE IS F
   LABEL RECORDS ARE OMITTED
   DATA RECORDS IS TRANS-REC.
01 TRANS-REC.
   02 FILLER                        PICTURE X.
   02 T-PART                        PICTURE X(6).
   02 T-CODE                        PICTURE 9.
   02 TN-QUANTITY                           PICTURE S9999.
   02 TX-QUANTITY REDEFINES TN-QUANTITY    PICTURE X(4).
   02 T-DESCR                       PICTURE X(15).
   02 TN-QOH                                PICTURE 9(4).
   02 TN-QOO                                PICTURE 9(4).
   02 TN-RP                                 PICTURE 9(4).
   02 TN-RQ                                 PICTURE 9(4).
   02 T-UNITS                               PICTURE X(4).
   02 TN-UNIT-PRICE                 PICTURE 9999V9999.
   02 TN-YTD-SALES                  PICTURE 9(8).
   02 T-CODEX                       PICTURE 9.
   02 FILLER                        PICTURE X(16).
WORKING-STORAGE SECTION.
77 W-PART                           PICTURE X(6).
```

```
        PROCEDURE DIVISION.
        START-RTN.
(A)         OPEN INPUT TR-FILE
                I-O DISK-MAST.
        READ-IN.
(B)         READ TR-FILE AT END GO TO EOJ-RTN.
(C)         EXAMINE TX-QUANTITY      REPLACING LEADING SPACES BY ZEROES.
(D)         IF TN-QUANTITY NOT NUMERIC DISPLAY
                'TRANSACTION QUANTITY INVALID'
                GO TO READ-IN.
(E)         MOVE T-PART TO W-PART.
(F)         DISPLAY W-PART TRANS-REC.
            READ DISK-MAST    INVALID KEY GO TO ADD-RTN.
(G)         IF T-CODE EQUAL TO 1 DISPLAY
                'ADDITION – RECORD ALREADY IN FILE'
(H)             GO TO READ-IN.
(I)         IF T-CODE EQUAL TO 4 ADD TN-QUANTITY TO D-QOO.
(J)         IF T-CODE EQUAL TO 6 MOVE HIGH-VALUE TO D-STATUS.
            REWRITE DISK-REC INVALID KEY GO TO NO-WRITE.
(K)         GO TO READ-IN.
        ADD-RTN.
            IF T-CODE NOT EQUAL TO 1 DISPLAY
            'TRANSACTION ERROR – MAT-CHING RECORD NOT FOUND'
(L)             GO TO READ-IN.
(M)         MOVE TRANS-REC TO DISK-REC.
            WRITE DISK-REC INVALID KEY DISPLAY 'ADDITION NOT ACCEPTED'
(N)             GO TO EOJ-RTN.
            GO TO READ-IN.
        NO-WRITE.
(O)         DISPLAY 'INVALID KEY ON REWRITE' GO TO EOJ-RTN.
        EOJ-RTN.
(P)         CLOSE TR-FILE DISK-MAST.
            STOP RUN.
```

Nota: Comparar letras (A), (B), (C), (D) etc. dos comandos com os blocos do fluxograma.

Figura 27.5. *Versão de acesso direto do programa para atualização do arquivo-mestre de controle de estoque. (McCracken e Garbassi – ob. citada)*

EXAMINE TX-QUANTITY REPLACING LEADING SPACES BY ZEROES.

Se os dados lidos do arquivo de transação contiverem espeços (o que é muito comum e mais fácil para a digitação) à esquerda, o dado é considerado como do tipo DISPLAY e não serve para operação aritmética. É necessário transformar o dado em tipo COMPUTATIONAL, substituindo os espaços por zeros.

IF TN-QUANTITY NOT NUMERIC:

Pela mesma razão, se algum caráter não numérico existir no campo lido, é necessário haver indicação de erro.

Tanto o TX-QUANTITY como o TN-QUANTITY foram definidos no DATA DIVISION, ocupando a mesma área da memória, através da cláusula REDEFINES, a fim de aceitar tanto valores puramente numéricos como não numéricos.

27.5. COMENTÁRIOS GERAIS SOBRE OS EXEMPLOS: VERSÃO ESTRUTURADA

Os programas-exemplo estudados são modelos simples e práticos para a utilização de arquivo de acesso direto em disco magnético.

Os programas NÃO ESTÃO ESTRUTURADOS, pois foram adaptados da referência original de D. MacCracken e U. Garbassi, baseados nas edições anteriores à divulgação das técnicas de estruturação.

Os leitores podem transformar os mesmos em versão estruturada facilmente seguindo a transformação executada no Capítulo 4, com o exemplo de PROGRAMA COBOL-1.

EXERCÍCIOS DE RECAPITULAÇÃO

1. O que são arquivos de acesso seqüencial? Dar suas características e exemplos.
2. O que são arquivos de acesso direto em DISCOS MAGNÉTICOS? Dar suas características, vantagens e desvantagens.
3. Descrever e comentar os seguintes comandos e cláusulas do Exemplo-1:

```
SELECT DISK-FILE ASSIGN 'SYS013' DIRECT ACCESS 2311
ACCESS SEQUENTIAL ORGANIZATION INDEXED
RECORD KEY IS D-PART.
RECORDING MODE IS F
77 RECORD-COUNT PICTURE 9(5) VALUE ZEROES.
WRITE DISK-REC INVALID KEY DISPLAY 'INVALID KEY'
        C-PART GO TO READ-IN.
DISPLAY DISK-REC.
```

4. Descrever e explicar os arquivos de transação TR-FILE e o mestre DISK-FILE do Exemplo-2; comentando como são definidos tanto fisicamente (ver Figura 27.3.) como no programa COBOL (ver Figura 27.5.).
5. Descrever e comentar, comando a comando, o PROCEDURE DIVISION do Exemplo-2 comparando as Figuras 27.4 e 27.5.

EXERCÍCIOS DE APLICAÇÃO

1. Efetuar as alterações necessárias no fluxograma e programa COBOL do Exemplo-2 – Atualização do arquivo-mestre de controle de estoques a fim de incluir as operações seguintes:
   CODIGO 2 – AJUSTE DA QUANTIDADE EM ESTOQUE A UM CERTO NÍVEL DADO.
   CODIGO 3 – RECEBIMENTO DE UM PEDIDO DE COMPRA FEITO, isto é, aumento na quantidade em estoque e
   CODIGO 5 – SAIDA OU GASTO NORMAL DO ITEM EM ESTOQUE, isto é, atendimento de um Pedido do usuário desse estoque.

Sugestão: Para a operação do CODIGO 5, ver exemplo do programa COBOL no Capítulo 21 – PROGRAMA COMPLETO EM COBOL.

2. Escrever o fluxograma e o programa COBOL dos exemplos 1 e 2 deste capítulo em sua VERSÃO ESTRUTURADA.
   Sugestão: Seguir exemplos do Capítulo 5 e 21.

28 CONCEITOS BÁSICOS SOBRE BANCO DE DADOS (DATA-BASE) E SEU USO EM COBOL

SÍNTESE DA MATÉRIA CONTIDA NESTE CAPÍTULO

A NECESSIDADE DE UM SISTEMA DE GERENCIAMENTO DE DADOS: TRÊS TIPOS DE OPERAÇÃO

Processamento de Dados consiste em: criar e atualizar arquivos, coletar e classificar dados e gerar relatórios.

SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS (SGBD)

Finalidades:

- Formação segura e eficiente de arquivos de dados que evitam duplicação e interferência de dados.
- Relacionamento natural de registros de arquivos diferentes, de acordo com a aplicação.
- Uso de linguagem simples para definir e manipular arquivos e gerar relatórios.

POR QUE USAR O SGBD?

Causas

- Alto custo de manutenção e atualização de programas.
- Uso de programas inadequados ou incompletos.
- Acúmulo de novos programas a serem implantados.
- Alto custo de desenvolvimento de novos programas.
- Ocorrência de programas semelhantes, mas não exatamente iguais.

O BANCO DE DADOS (BD) E SUA DEFINIÇÃO

- O uso de entidades ou conjuntos de dados agrupados de acordo com a necessidade da aplicação.

APLICAÇÃO USANDO BANCO DE DADOS

- Exemplo ilustrado.

A LINGUAGEM DE DEFINIÇÃO DE DADOS (LDD)

- Finalidades: Descrever as entidades, o tipo e valor de dados e manusear através de comandos os itens de dados.

DESCRIÇÃO DE UMA ENTIDADE DO BANCO DE DADOS

- Exemplo

EXEMPLO DE DEFINIÇÃO E MANUSEIO DE BANCO DE DADOS COM O COBOL: O DATA-BASE SECTION

EXEMPLO DE APLICAÇÃO

OS SGBD EXISTENTES NO MERCADO

- TOTAL, SYSTEM 2000, MARK IV, IDMS, DMS II, IMS etc.

Exercícios

28.1. A NECESSIDADE DE UM SISTEMA DE GERENCIAMENTO DE DADOS: TRÊS TIPOS DE OPERAÇÃO

Em Processamento de Dados temos certas operações que ocorrem com grande freqüência e em nível cada vez mais complexo. Essas operações são as seguintes:

- criar e atualizar arquivos;
- coletar, classificar, selecionar dados e
- gerar relatórios.

O desenvolvimento do *Software* para processamento de dados comerciais sempre teve em mente executar com maior eficiência e facilidade esses três tipos de operação. Para tal, procurou-se criar linguagens de programação cada vez mais poderosas como o Macro Assembler, o RPG (Report Program Generator) e o COBOL ou usar sub-rotinas e programas de utilidade incorporados ao Sistema Operacional do computador e que servem para a criação e administração de arquivos.

Como tais operações se tornaram cada vez mais complexas e repetitivas, houve a necessidade de criar um sistema mais adequado e eficiente para a administração e gerenciamento de dados comerciais.

Apareceram, então, os Sistemas de Gerenciamento de Dados ou, melhor dizendo, os Sistemas de Gerenciamento de Banco de Dados (Data-Base ou Base de Dados) que é um conjunto de programas que oferecem recursos para a execução das operações acima citadas, de modo mais automatizado e eficiente do que era possível com uma linguagem COBOL ou RPG.

28.2. SISTEMA DE GERENCIAMENTO OU ADMINISTRAÇÃO DE BANCO DE DADOS (SGBD)

Um SGBD oferece basicamente facilidades para:

- formação segura e eficiente dos diversos tipos de arquivos, evitando duplicação e interferência dos dados;
- relacionamento natural, de acordo com a aplicação, de registros de dados existentes em arquivos separados; e
- utilização de linguagem simples para a definição e manipulação de arquivos e dos dados.

28.3. POR QUE USAR O SGBD?

A partir de um certo volume de dados ou de programas de aplicação, é inevitável que, em um serviço tradicional de processamento de dados, ocorram os seguintes problemas:

- alto custo de manutenção e atualização dos programas;
- uso de programas ou geração de relatórios com dados inadequados ou incompletos;
- acúmulo de novos programas e serviços a serem implantados;
- alto custo de desenvolvimento de novos programas;
- ocorrência de diversos programas de manipulação de dados que executam tarefas semelhantes, mas não exatamente iguais.

Os problemas acima enfatizaram o uso de um SGBD que oferece facilidades para formulação de programas novos e reformulação de programas velhos, pois permite:

- estabelecer relação entre os itens de dados em diversos níveis;
- obter informação dos dados em diversos tipos ou lógicas de seqüência e não apenas na seqüência natural de armazenamento;
- obter informação formada por qualquer agrupamento desejado de dados.

28.4. O BANCO DE DADOS (BD) E SUA DEFINIÇÃO

Para usar um SGBD que é um conjunto de programas para manipulação de dados é necessário formar um BANCO DE DADOS (BD) que é um conjunto de arquivos e registros de dados relacionados através de uma certa estrutura lógica que pode ser usada por qualquer programa de usuário, com a necessária clareza e segurança.

Por clareza, entende-se a precisão e a facilidade de definir ou solicitar os dados desejados e por segurança entende-se a proteção contra eliminação acidental, invasão de áreas alheias ou duplicidade de tais dados.

Um Banco de Dados é formado por Entidades ou Conjuntos de Dados agrupados com certo objetivo e finalidade.

Lembramos que os dados de uma entidade não provêm, necessariamente, de um mesmo arquivo físico ou de um mesmo tipo de arquivo, podendo ser formados por dados pertencentes a arquivos fisicamente diferentes ou arquivos de características diferentes. O Banco de Dados permite essa flexibilidade de relacionamento entre os dados ou itens de dados.

28.5. APLICAÇÃO USANDO BANCO DE DADOS

A Figura 28.1 mostra um exemplo ilustrativo de confecção de um relatório usando dados de arquivos diferentes.

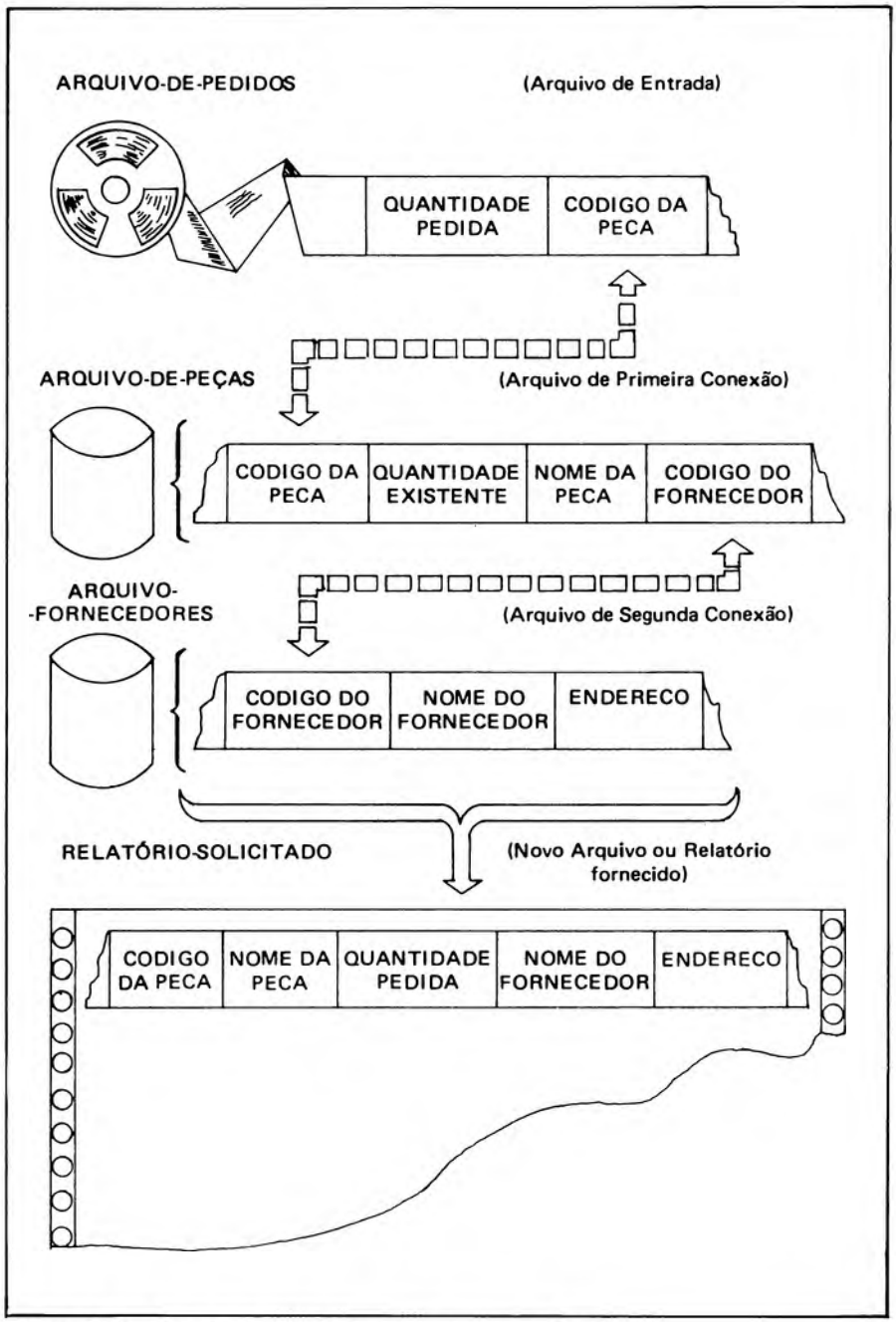


 Figura 28.1. *Arquivos e relatório de um banco de dados.*

Um programador usando um Sistema de Gerenciamento de Banco de Dados (SGBD) definiria como seu Banco de Dados, os nomes dos arquivos envolvidos e os respectivos campos ou registros que entram na formação do novo registro do relatório solicitado. O programador, em seu programa, teria somente o trabalho de solicitar a impressão do novo registro definido para formar o RELATÓRIO.

O SGBD possui recursos (programas) que se encarregam de efetuar a conexão entre os arquivos, ter acesso aos dados, cuidar da lógica do programa e definir e gerar o relatório solicitado.

28.6. A LINGUAGEM DE DEFINIÇÃO DE DADOS (LDD)

Todo o Sistema de Gerenciamento de Dados (SGBD) e seu respectivo Banco de Dados (BD) deve possuir capacidade para:

- descrever as entidades ou conjunto de dados;
- definir o tipo e o valor dos dados;
- manusear (através de comandos) as entidades e os itens de dados.

Para a descrição das entidades ou conjunto de itens de dados, o SGBD possui uma LINGUAGEM DE DEFINIÇÃO DE DADOS (LDD) e para definição do tipo e valor dos dados e solicitação de relatórios e outras aplicações (processamento propriamente dito dos dados) essa linguagem deve ser usada em conjunto com uma linguagem de programação conhecida, tal como o COBOL, FORTRAN ou PL/1.

28.7. DESCRIÇÃO DE UMA ENTIDADE DO BANCO DE DADOS

Uma entidade chamada PRESIDENTE pode ser descrita ou definida pelos itens:

- PRES-NOME
- NOME-ESPOSA
- DATA-NASCIMENTO que possui os subitens
  - DIA
  - MES
  - ANO
- FILHOS que possui subitens
  - NOME-DO-FILHO e DATA-NASCIMENTO.

O item FILHOS é um item repetitivo pois o PRESIDENTE pode ter 0, 1, 2 ou mais filhos.

A descrição ou definição formal dessa entidade seria efetuada usando os comandos de definição da LDD:

```
PRESIDENTE = < PRES-NOME, NOME-ESPOSA, DATA-NASCIMENTO, {FI-
LHOS } >onde DATA-NASCIMENTO =< DIA, MES, ANO >
e FILHOS =< NOME-DO-FILHO, DATA-NASCIMENTO>
```

Os sinais { } indicam a ocorrência repetitiva e opcional do item.

Não seria necessário frisar que qualquer item que compõe a entidade PRESIDENTE pode ter sido extraído de um arquivo distinto. Por exemplo, os itens FILHOS podem estar fisicamente formando um arquivo próprio de FILHOS DOS PRESIDENTES.

28.8. EXEMPLO DE DEFINIÇÃO E MANUSEIO DE BANCO DE DADOS COM O COBOL: O DATA BASE SECTION

Quando usamos um SGBD juntamente com a linguagem COBOL teremos, no DATA DIVISION do programa em COBOL, uma seção especial chamada DATA BASE SECTION onde os dados da entidade previamente definidos pela Linguagem de Definição de Dados (LDD) recebem a sua descrição completa (do tipo de dado, comprimento, valor etc.) a fim de poderem ser usados como arquivos e registros de dados desse programa em COBOL.

Sendo assim, se um programador desejar usar em seu programa COBOL a entidade PRESIDENTE anteriormente definida pela LDD, deverá preparar um programa COBOL do tipo apresentado na Figura 27.2.

```
IDENTIFICATION DIVISION.
      (igual ao programa COBOL comum)
ENVIRONMENT DIVISION.
-------------------------------------------------------------
      (declarar os arquivos de ENTRADA/SAIDA do COBOL e que não são do
      BANCO DE DADOS; o manual do sistema deve ser consultado)
-------------------------------------------------------------
DATA DIVISION.
FILE SECTION.
FD ... (definição dos arquivos e registros normais do programa COBOL)
DATA BASE SECTION.
DB AREA-PRESIDENTE                    (Nome do Banco de Dados que usa a
01    PRESIDENTE.                      entidade chamada PRESIDENTE)
      02    PRES-NOME PICTURE X(20).
      02    NOME-ESPOSA PICTURE X(10).          (Descrição
      02    DATA-NASCIMENTO.                    dos Dados das
            03    DIA PIC 99.                   entidades do
            03    MES PIC 99.                   Banco de Dados)
            03    ANO PIC 99.
      02    FILHOS OCCURS O TO N TIMES.
            03    NOME-DO-FILHO PIC X(10).
            03    DATA-NASCIMENTO 9(6).
01    . . . . . (próxima entidade)
-------------------------------------------------------------
PROCEDURE DIVISION.
      . . . . . (comandos COBOL e comandos de manipulação do
                 Banco de Dados).
```

Figura 28.2. *Programa COBOL em que se utiliza a entidade PRESIDENTE.*

28.9. EXEMPLO DE APLICAÇÃO

A aplicação usando Banco de Dados apresentada na seção 28.5 e pela Figura 28.1 poderia então ser definida e descrita.

A aplicação pressupõe que existe no Banco de Dados 3 arquivos (ARQUIVO-DE-PEDIDOS, ARQUIVO-DE-PEÇAS, E ARQUIVO-FORNECEDORES) e formação de uma nova entidade na forma do RELATÓRIO-SOLICITADO e que contém os itens:

- CODIGO-DA-PEÇA
- NOME-DA-PEÇA
- QUANTIDADE-PEDIDA
- NOME-DO-FORNECEDOR
- ENDEREÇO

sendo cada item extraído de um dos três arquivos citados.

Para a definição do Banco de Dados dessa aplicação teríamos os seguintes comandos de descrição (comparar com a Figura 28.1):

```
AREA-NAME IS RELATORIO-DE-PECAS-EM-ESTOQUE        (nome genérico dado à
                                                   aplicação em questão)
ARQUIVO-DE-PEDIDOS =
   < QUANTIDADE-PEDIDA, CODIGO-DA-PECA >;         (Entidade ou Arquivo 1)
ARQUIVO-DE-PECAS =
   < CODIGO-DA-PECA, QUANTIDADE-EXISTENTE, NOME-DA-PECA
     CODIGO-DO-FORNECEDOR >;                      (Entidade ou Arquivo 2)
ARQUIVO-FORNECEDORES =
   < CODIGO-DO-FORNECEDOR, NOME-DO-FORNECEDOR,
     ENDERECO >;                                  (Entidade ou Arquivo 3)

RELATORIO-SOLICITADO =
   < CODIGO-DA-PECA, NOME-DA-PECA, QUANTIDADE-PEDIDA,
     NOME-DO-FORNECEDOR, ENDERECO >;              (Entidade ou Arquivo 4)
```

O programa COBOL que efetuará os processamentos necessários para gerar o RELATÓRIO-SOLICITADO terá a seguinte forma:

```
IDENTIFICATION DIVISION.
      ....
ENVIRONMENT DIVISION.
      ....
DATA DIVISION.
FILE SECTION.
FD    ...                          (Definição de arquivos normais do programa COBOL)
FD    ...
DATA-BASE SECTION.
DB    RELATÓRIO-DE-PECAS-EM-ESTOQUE.        (nome do Banco de Dados da
                                             aplicação)
01    ARQUIVO-DE-PEDIDOS.
      02    QUANTIDADE-PEDIDA   PIC 9(5).
      02    CODIGO-DA-PECA      PIC 9(5).
01    ARQUIVO-DE-PECAS.
      02    CODIGO-DA-PECA      PIC 9(5)
      02    QUANTIDADE-EXISTENTE PIC 9(10).      (Entidades)
      02    NOME-DA-PECA PIC X(15).
      02    CODIGO-DO-FORNECEDOR PIC X(10).
01    ARQUIVO-FORNECEDORES.
      02    CODIGO-DO-FORNECEDOR PIC X(10).
      02    NOME-DO-FORNECEDOR PIC X(20).
      02    ENDERECO PIC X(20).
01    RELATORIO-SOLICITADO.
      02    CÓDIGO-DA-PECA PIC 9(5).
      02    NOME-DA-PECA PIC X(15).
      02    QUANTIDADE-PEDIDA PIC 9(5).
      02    NOME-DO-FORNECEDOR PIC X(20).
      02    ENDERECO PIC X(20).
WORKING-STORAGE SECTION.
      ....
PROCEDURE DIVISION.
      ..... (comandos de processamento)
```

28.10. OS SGBD EXISTENTES NO MERCADO

Nos exemplos de descrição e aplicação do Banco de Dados não foi possível usar a descrição ou comandos mais precisos ou completos, pois são exemplos introdutórios e genéricos e que não estão ligados a qualquer SGBD específico.

Não foi possível, portanto, fornecer precisamente todo o tipo de descrição e relacionamento envolvidos entre os dados, o que está além das pretensões deste livro. Pretende-se fornecer apenas uma visão geral e esquemática do relacionamento entre um SGBD e um programa COBOL, através da Linguagem de Definição de Dados (LDD).

A tecnologia, tanto do *Hardware* como do *Software*, empregada em um SGBD, é tão complexa e variada que seria impossível dar uma idéia precisa do uso de um Banco de Dados, sem fornecer uma descrição completa e extensa

de um SGBD específico. Entretanto, cada SGBD procura oferecer ao usuário regras simples e fáceis para definição e utilização dos Bancos de Dados.

A utilização do SGBD torna-se cada vez mais intensa e a implicação de seu uso juntamente com programas em linguagem COBOL é bastante grande.

Para se ter idéias mais precisas e completas sobre o uso de um SGBD, é necessário dedicar-se à utilização de um dos diversos SGBD existentes no mercado tais como:

GIS, DL/I e IMS (da IBM); DMS II (da Burroughs); TOTAL, ADABAS, IDMS, MARK IV e SYSTEM 2000 (existente em diversos computadores).

EXERCÍCIOS DE RECAPITULAÇÃO

1. Citar as operações mais freqüentes em Processamento de Dados.
2. Quais são as facilidades oferecidas por um SGBD?
3. Enumere os fatores que levam ao uso de um sistema sofisticado como o SGBD em um Centro de Processamento de Dados.
4. Explicar o que é SGBD e seu Banco de Dados.
5. O que é Linguagem de Definição de Dados e para que serve?
6. Explicar como um SGBD é usado em conjunto com um programa em COBOL.

EXERCÍCIOS DE APLICAÇÃO

1. Usando o exemplo de Aplicação da seção 28.5, definir a geração, através de um SGBD, de outros tipos possíveis de RELATÓRIO, fornecendo sua definição pela Linguagem de Descrição de Dados e pelo DATA BASE SECTION do COBOL.

Apêndice A

RESUMO DAS DESCRIÇÕES E FORMAS GERAIS

I – RESUMO DO DATA DIVISION

Data Division Outline

```
DATA DIVISION.
FILE SECTION.
   File Description entries
      Record Description entries
   Sort Description entries
      Record Description entries
   Saved Area entries
      Record Description entries
WORKING-STORAGE SECTION.
   Record Description entries
CONSTANT SECTION.
   Record Description entries
REPORT SECTION.
   Report Description entries
      Report Group Description entries
      Report Element Description entries
```

II – DESCRIÇÃO DE ARQUIVOS E ÁREAS

File Description Entry

```
FD  file-name
    LABEL { RECORD  IS  } { STANDARD }
          { RECORDS ARE } { OMITTED  }
                          { data-name }.

    [RECORDING MODE IS mode]
    [BLOCK CONTAINS integer { CHARACTERS }]
                            { RECORDS    }
       [RECORD CONTAINS [integer-1 TO]
            integer-2 CHARACTERS]
    DATA { RECORD  IS  } record-name ...
         { RECORDS ARE }
```

```
[ {REPORT   IS  } report-name... ]
[ {REPORTS  ARE } ]
```

Sort Description Entry

```
SD sort-file description-name
   DATA {RECORD  IS  } record-name ...
        {RECORDS ARE }
   [RECORDING MODE IS mode]
   [RECORD CONTAINS [integer-1 TO]
            integer-2 CHARACTERS].
```

Saved Area Description Entry

```
SA saved-area-name
   [RECORD CONTAINS [integer-1 TO]
        integer-2 CHARACTERS].
```

Working-Storage Section Outline

```
WORKING-STORAGE SECTION.
   77-Level entries
   Record Description entries
```

Constant Section Outline

```
CONSTANT SECTION.
   77-Level entries
   Record Description entries
```

Report Description Entry

```
RD report-name
   [CODE-clause] [CONTROL-clause]
   [PAGE-clause].
```

III – DESCRIÇÃO DE REGISTROS E ITENS DE DADO (NÍVEIS 01, 02, ...)

Group Item

```
level-number {data-name} [OCCURS...] [USAGE...] [CLASS...].
             {FILLER   }
```

Alphanumeric Item

```
level-number {data-name} [OCCURS...] [SIZE...] ] [SYNCHRONIZED...]
             {FILLER   }
   [PICTURE...] [USAGE IS DISPLAY,]
                [VALUE IS non-numeric-literal]
   [ CLASS IS {ALPHANUMERIC} ]
              {ALPHABETIC  }
```

Report Item

```
level-number {data-name} [OCCURS...]
             {FILLER   }
[SYNCHRONIZED...] ] [editing clauses...]
[SIZE...] [CLASS IS ALPHANUMERIC]
[USAGE IS DISPLAY] [PICTURE IS...]
[VALUE IS numeric-literal].
```

External Decimal Item

level-number { data-name / FILLER } [OCCURS...] [SYNCHRONIZED...]
[SIZE...] [USAGE IS DISPLAY] [PICTURE...]
[VALUE IS numeric-literal]
[CLASS IS NUMERIC].

Internal Decimal Item

level-number { data-name / FILLER } [OCCURS...]
[SIZE...] [USAGE IS COMPUTATIONAL]
[PICTURE...] [VALUE IS numeric-literal]
[CLASS IS NUMERIC] [SYNCHRONIZED RIGHT].

IV – RESUMO DO PROCEDURE DIVISION

PROCEDURE DIVISION.
DECLARATIVES.
[{section-name SECTION. declarative-sentence.
{paragraph-name. {sentence} ...}... }...]
END DECLARATIVES.
{[section-name SECTION. [priority].]
{paragraph-name. {sentence}... }... }...

V – COMANDOS DE MANIPULAÇÃO DE DADOS

Data Manipulation Statements

+ADD { data-name-1 / numeric-literal-1 } [, data-name-2 / , numeric-literal-2] ... { TO / GIVING }
data-name-3 [ROUNDED] [, data-name-4 [ROUNDED]]...
[ON SIZE ERROR imperative statement]
COMPUTE data-name-1 [ROUNDED] [, data-name-2 [ROUNDED]] ...

{ FROM / = / EQUALS } arithmetic-expression [ON SIZE ERROR imperative-statement]

+ DIVIDE { data-name-1 / numeric-literal-1 } INTO { data-name-2 GIVING data-name-3 / numeric-literal-2 GIVING data-name-3 }
[ROUNDED] [ON SIZE ERROR imperative-statement]

EXAMINE data-name { TALLYING { UNTIL FIRST / ALL / LEADING } literal-2 [REPLACING BY literal-2]
REPLACING { ALL / LEADING / UNTIL FIRST } literal-1 BY literal-2 }

MOVE { data-name-1 / literal / CORRESPONDING data-name-2 } TO data-name-3 ...

```
MULTIPLY { data-name-1       } BY { data-name-2 [GIVING data-name-3]       }
         { numeric-literal-1 }    { numeric-literal-2 GIVING data-name-3 }
       [ROUNDED] [ON SIZE ERROR imperative-statement]

†SUBTRACT { data-name-1       } [ , data-name-2       ] ... FROM
          { numeric-literal-1 }   [ , numeric-literal-2 ]
    { data-name-3       [GIVING data-name-4] }
    { numeric-literal-n  GIVING data-name-n  }
      [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Extended Data Manipulation Verb: TRANSFORM.

VI – COMANDOS DE CONTROLE DE PROGRAMA

Control Statements

```
ALTER procedure-name-1 TO PROCEED TO procedure-name-2
      [, procedure-name-3 TO PROCEED TO procedure-name-4]....
GO TO [procedure-name]
GO TO procedure-name-1 [, procedure-name-2]...
      DEPENDING ON data-name
IF condition [THEN] { statement-1 }
                    { NEXT SENTENCE }
   { OTHERWISE } { statement-2 }
   { ELSE      } { NEXT SENTENCE }
PERFORM procedure-name-1 [THRU procedure-name-2]
PERFORM procedure-name-1 [THRU procedure-name-2] UNTIL condition-1
PERFORM procedure-name-1 [THRU procedure-name-2] { data-name-1 } TIMES
                                                 { integer-1   }
†PERFORM procedure-name-1 [THRU procedure-name-2] VARYING data-name-2
      FROM { data-name-3 } BY { data-name-4 } UNTIL condition-1
           { literal-1   }    { literal-2   }
      STOP { literal }
           { RUN     }
```

Extended Control Verbs: HOLD, PROCESS, SEARCH, SET.

VII – COMANDOS DE ENTRADA E SAÍDA

Input/Output Statements

```
ACCEPT data-name [FROM mnemonic-name]
CLOSE { file-name-1 [ REEL ] [ WITH { NO REWIND } ] } ...
                    [ UNIT ]       { LOCK      }
DISPLAY { literal-1   } [ , literal-2   ] ... [UPON mnemonic-name]
        { data-name-1 }   [ , data-name-2 ]
OPEN [INPUT { file-name REVERSED
            { WITH NO REWIND }...]
[OUTPUT {file-name [WITH NO REWIND]}...]
[ I-O          {file-name}... ]
[ INPUT-OUTPUT
```

†READ file-name RECORD [INTO data-name]
AT END imperative-statement
WRITE record-name [FROM data-name]
[INVALID KEY imperative-statement]
Extended Input/Output Verbs: GENERATE, INITIATE, RELEASE, RETURN, SEEK, SORT, TERMINATE.

VIII – COMANDOS DE ESTRUTURAÇÃO

Program Structure Statements

ENTER Language-name [routine-name].
paragraph-name. EXIT

{ paragraph-name, / section-name SECTION } INCLUDE library-name.

NOTE character-string.

IX – COMANDOS DE ESPECIFICAÇÃO

Specification Statements

Specification statements in the Procedure Division are termed declaratives and must be included in the declaratives section.

†USE AFTER STANDARD ERROR PROCEDURE ON

{ file-name-1 [, file-name-2] ... / INPUT / OUTPUT / INPUT-OUTPUT / I-O }

Extended Specification Verb: DEFINE.

(†) – Comandos precedidos por este sinal admitem formas mais gerais.

Apêndice B

AS PALAVRAS RESERVADAS EM COBOL

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BLANK
BLOCK
BY

CALL
CF
CH
CHANGED
CHARACTERS
CHECKING
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORRESPONDING
CREATING
CYCLES

DATA
DATE-COMPILED
DATA-WRITTEN
DEFINE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DIRECT
DIRECT-ACCESS
DISPLAY
DISPLAY-ST
DIVIDE
DIVISION

ELSE
END
ENDING
ENTER
ENTRY
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXHIBIT
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILLER
FINAL
FIRST
FOOTING
FOR
FORM-OVERFLOW
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

ID
IDENTIFICATION
IF
IN

INCLUDE
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS

JUSTIFIED

KEY

LABEL
LABELS
LAST
LEADING
LESS
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MODE
MORE-LABELS
MOVE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC

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

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PICTURE
PLUS
POSITIVE
PRINT-SWITCH
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM-ID
PROTECTION

QUOTE
QUOTES

RANDOM
RD
READ
READY
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMARKS
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTRICTED
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SD
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEQUENTIAL
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STOP
SUBTRACT
SUM
SYMBOLIC
SYSIN
SYSOUT
SYSPUNCH

TALLY
TALLYING
TERMINATE
THAN
THEN
THRU
TIMES
TO
TRACE
TRACK-AREA
TRACKS
TRANSFORM
TRY
TYPE

UNIT
UNIT-RECORD
UNITS
UNTIL
UPON
USAGE
USE
USING
UTILITY

VALUE
VARYING

WHEN
WITH
WORKING-STORAGE
WRITE
WRITE-STORAGE
WRITE-ONLY

ZERO
ZEROES
ZEROS

Apêndice C

TESTES DE MÚLTIPLA ESCOLHA

1. Um campo ou item de dado é caracterizado pelos seguintes elementos:
   a) Nome, ponto, valor e sinal.
   b) Nome, tipo, comprimento.
   c) Valor numérico, valor alfabético e valor alfanumérico.
   d) Nome, tipo, comprimento e endereço.

2. Qual das afirmações seguintes está correta?
   a) Um arquivo de dados é formado por registros de dados que, por sua vez, são formados por campos de dados.
   b) Um arquivo de dados é formado pelos campos de dados e etiquetas de início e de fim de arquivos.
   c) Um arquivo de dados é formado por registros de mesmo tamanho.
   d) Um arquivo de dados é formado por um conjunto de fichas e cartões perfurados.

3. Dizer qual das afirmações está correta:
   a) O Procedure Division é a divisão que define os dados e efetua os cálculos em Programação COBOL.
   b) O Environment Division e o Identification Division são de uso opcional.
   c) O Data Division é a divisão que vem após o Procedure Division.
   d) O Procedure Division contém os parágrafos, sentenças e comandos de execução.

4. Os comandos para executar operações aritméticas em COBOL são:
   a) ADD, Subtract, Multiply e Divide.
   b) ADD, Subtract, Multiply, Divide, Move, ADD Corresponding e Divide Corresponding.
   c) ADD, Subtract, Multiply, Divide, Compute, ADD Corresponding e Substract Corresponding.

5. Um Programa COBOL pode conter as seguintes divisões (indicar a alternativa correta):
   a) Identification Division, Environment Division, Data Division e Working-Storage Section.
   b) Identification Division, Procedure Section, Data Division, Environment Division (que é opcional).
   c) Identification Division (identifica programa), Data Division (relaciona arquivos com memória e define áreas), Environment Division (relaciona arquivos com unidades periféricas), Procedure Division (executa operações).
   d) Identification Division (identificação é opcional), Environment Division (descreve o hardware), Data Division (relaciona arquivo com periféricos), Procedure Division (executa programa).

6. A Programação Estruturada é uma técnica de programação que deve ser adotada para:
   a) Eliminar o uso de comando Go To.
   b) Substituir o comando Go To pelo comando IF.
   c) Estruturar o programa em seqüências de estruturas básicas:
      - seqüência simples;
      - seleção de alternativas;
      - controle de repetições.
   d) Tornar o programa mais claro e legível.

7. As estruturas básicas para programação estruturada em COBOL são:
   a) ● Seqüência simples.
      - Estrutura de seleção: IF B DO S1.
      - Estrutura de repetição: PERFORM...UNTIL.
   b) ● Seqüência simples:
      - Estrutura de seleção: IF B DO S1
        IF B THEN S1 ELSE S2.
      - Estrutura de seleção: PERFORM ... WHILE.
   c) ● Seqüência simples:
      - Estrutura de seleção: IF B THEN S1.
        IF B THEN S1 ELSE S2.
      - Estrutura de repetição: PERFORM parágrafo ... UNTIL condição.
   d) ● Caso c sem a opção IF B THEN S1.

8. Exemplos de palavras opcionais são:
   a) VALUE, DIVIDE, MOVE, ON.
   b) OF, IN, ON, THEN,
   c) IS, ON, THEN.
   d) IS, ON, BY.

9. Exemplos de constantes ou literais numéricas são:
   a) 26,3 '010' 3547.7
   b) 55.4- 1000 35.45
   c) 1862.99 +35 - 37.8

10. Exemplo de expressão aritmética:
    a) TAXA IS GREATER THAN ZERO.
    b) (A + B) * PRODUTO.
    c) ADD X, Y, Z.
    d) (C + Y) OR (Y + B).

11. Exemplo de expressão lógica correta:
    a) (X IS EQUAL TO UM) AND (Z < 3).
    b) X DOES NOT EQUAL TO GRAU.
    c) COMPUTE X = A + B + C.
    d) ADD X, Y, Z.

12. A forma geral do comando IF:

    IF condição THEN { comando 1 / NEXT SENTENCE } { OTHERWISE / ELSE } { comando 2 / NEXT SENTENCE }

    permite usar a seguinte sentença sintaticamente correta:
    a) IF X THEN GO TO UM.
    b) IF X = Y THEN Y = 1 OTHERWISE NEXT SENTENCE ELSE Y = 30.
    c) IF X LESS THEN Y NEXT SENTENCE ELSE Y = X
    d) IF X LESS THAN Y NEXT SENTENCE ELSE COMPUTE Y = X.

13. No Data Division a descrição de dados é feita através dos seguintes níveis de dados:
    a) FD — para descrição de um arquivo.
       01 a 49 — para descrição de subcampos de dados.
       77 e 88 — para descrição de áreas do WORKING-STORAGE SECTION.

b) FD – para descrição de um arquivo.
   01 – para descrição do registro desse arquivo.
   02 a 88 – para descrição dos subcampos do registro.
c) FD – para descrição de um arquivo.
   01 – para descrição de um arquivo desse registro.
   02 a 49 – para descrição dos subcampos desse registro.
   77– para área de trabalho do WORKING-STORAGE SECTION.
   88 – para descrição de nome de condição.
d) FD – para descrição de um arquivo.
   RD – para descrição de um registro.
   01 a 049 – para descrição de subcampos.
   77 – para descrição de área de trabalho.
   88 – para descrição de nome de condição.

14. A cláusula PICTURE usa, para descrever os valores 25035 (ponto decimal assumido entre 0 e 3), JUNHO e XPT–/3 as seguintes descrições:
    a) 999.99, A (10), X(6).
    b) 999V99, AAAAA, X(6).
    c) XXXVXX, A(5), X(6).
    d) 9(3)V9(2), X(5), A(6).

15. O comando que executa o parágrafo PAR-1 cinco vezes repetidamente é:
    a) PERFORM PAR-1 THRU PAR-5.
    b) PERFORM PAR-1 VARYING X FROM 1 BY UNTIL X > 5.
    c) PERFORM PAR-1 THRU PAR-5 TIMES.
    d) PERFORM PAR-1 5 TIMES.

16. O comando
    IF AGE IS LESS THAN 21 AND GREATER THAN 65
       THEN COMPUTE X = 30
       ELSE COMPUTE X = 50
    resulta em:
    a) X = 50 ou X = 30 dependendo do valor AGE.
    b) X = 30.
    c) X = 50.
    d) Nem X = 30 nem X = 50, pois a condição nunca é satisfeita.

17. Qual da seqüência abaixo de comandos é aceitável se cada seqüência se aplica à operação de um arquivo?
    a) READ, READ.
    b) OPEN ACCEPT CLOSE.
    c) OPEN READ READ CLOSE.
    d) OPEN DISPLAY CLOSE.

18. Qual a alternativa que contém comandos equivalentes e corretos:
    a) MOVE ZERO TO AREA.
       COMPUTE AREA EQUAL ZERO.
    b) MOVE ZERO TO AREA.
       MOVE 0 TO AREA.
    c) MOVE ZERO TO AREA.
       MOVE 'ZERO' TO AREA.
    d) MOVE ZERO TO AREA.
       MOVE '0' TO AREA.

19. O comando.
    EXAMINE VALOR TALLYING UNTIL FIRST 'Q'
    aplicado a VALOR com PT89QNQ9Q resulta em:
    a) TALLY = 5.
    b) TALLY = 2.

c) TALLY = 4.
d) TALLY = 9.

20. A descrição de PICTURE para editar o valor 123456, transformando-o em 123,456.00, é:
a) ZZZ,999.
b) 999,999.
c) ZZZ,ZZZ,ZZ.
d) ZZZ,ZZZ.00

RESPOSTAS E SUGESTÕES

1. **b**; ver item 3.2.
2. Ver itens 3.5 e 3.3.
3. **d**; ver item 4.1.
4. Ver Capítulo 15.
5. **c**; ver item 4.7.
6. Ver item 5.1.
7. **c**; itens 5.5 e 5.6.
8. Ver item 6.1 e Apêndice A.
9. **c**; ver item 6.3.
10. Ver item 7.2.
11. **a**; ver item 7.3.
12. Ver item 7.7.
13. **c**; ver item 9.3.
14. Ver item 11.2.
15. **d**; ver item 16.6.
16. Ver item 17.2.5.
17. **c**; ver Capítulo 18
18. Ver item 19.6.5.3
19. **c**; ver item 23.1.
20. Ver item 22.5.

Apêndice D

GLOSSÁRIO DE TERMOS TÉCNICOS, CLÁUSULAS E COMANDOS

Abreviações: (IDEN) – IDENTIFICATION DIVISION
(ENV) – ENVIRONMENT DIVISION
(DATA) – DATA DIVISION
(PROC) – PROCEDURE DIVISION
(WORK) – WORKING-STORAGE SECTION
(DIV) – DIVISION

A	Capítulo
A – símbolo para definição de um caráter alfabético no PICTURE. Também serve para designar a MARGEM A da folha de codificação COBOL (DATA). Exemplo: 02 TAXA PICTURE IS AAAA.	4,11 e 22
ACTUAL KEY – (ver KEY.)	
ACCEPT – comando para ler (aceitar) dados através da máquina de escrever do console (PROC). Exemplo: ACCEPT AVISO FROM CONSOLE.	18
ACCESS (MODE) – cláusula que indica o modo SEQUENTIAL ou RANDOM da leitura de um arquivo (ENV). Exemplo: ACCESS MODE IS SEQUENTIAL.	26 e 27
ADD e ADD CORRESPONDING – comando aritmético de adição de dados ou campos de dados (PROC). Exemplo: ADD A TO B GIVING C. ADD CORRESPONDING M TO X.	4 e 15
ADVANCING LINES – verbo opcional utilizado com o comando WRITE, nas versões AFTER ou BEFORE ADVANCING LINES (PROC). Exemplo: WRITE LINHA-COMENTARIO AFTER ADVANCING LINES.	18

Capítulo

AFTER – (ver ADVANCING LINES). –

ALL – constante figurativa indicando TUDO que vier a seguir. 16
Exemplo: ALL SPACES, ALL 'Z' etc. (DATA E PROC).

ALPHABETIC – designação de tipo de dado ou caráter alfabético formado por letra A a Z (DATA). –
Exemplo: CLASS IS ALPHABETIC.

ALTER – comando que modifica o desvio definido por um comando GO TO. Não recomendado em programação estruturada (PROC). –

ALTERNATE AREA – definição opcional de área alternativa de memória usada no FILE-CONTROL da ENVIRONMENT DIVISION. 27

AND – operador lógico usado para ligar duas ou mais expressões lógicas, indicando a operação "E" ou ocorrência simultânea de eventos (PROC). 7
Exemplo: IF X = ZERO AND Z LESS THAN ZERO ADD A TO Z.

ANSI – abreviatura de AMERICAN NATIONAL STANDARD INSTITUTE. –

APPLY – define técnica ou uso especial de I-O-CONTROL a ser aplicada por uma Biblioteca prefixada no ENVIRONMENT DIVISION. 8
Exemplo: APPLY (nome da técnica) ON (nome da biblioteca).

ASSIGN – verbo utilizado na cláusula SELECT e serve para designar uma unidade de entrada ou saída a um arquivo de dados (ENV). 8
Exemplo: SELECT ARQUIVO-UM ASSIGN TO PRINTER.

AT END – verbo opcional utilizado no comando READ (PROC). 18
Exemplo: READ ARQUIVO AT END MOVE X TO Y.

AUTHOR – verbo opcional para colocar nome do autor no programa. (IDEN). 8
Exemplo: AUTHOR. FULANO DE TAL.

B

B – símbolo usado para editar um espaço em branco através do PICTURE; indica também MARGEM B do formulário COBOL (DATA). 22
Exemplo: PICTURE IS 99BB99.

BEFORE – opção usada no comando WRITE X BEFORE ADVANCING LINES (PROC).

BIT – valor binário zero ou um.

Capítulo

BLOCK CONTAINS – cláusula que define o número de registro ou caracteres contido em um bloco de dados. Bloco é a quantidade de dados ou registro de dados que uma unidade periférica pode ler ou escrever em uma operação READ ou WRITE (DATA). 10
Exemplo: BLOCK CONTAINS 120 CHARACTERES.

BY – verbo ou operador utilizado nos comandos MULTIPLY, SET e PERFORM (PROC). –
Exemplos: MULTIPLY X BY Y.
PERFORM PARAG-L VARYING X FROM 1 BY 1 UNTIL 100.
SET INDICE DOWN BY 5.

C

CALL – comando para chamar uma sub-rotina externa em conjunto com o comando ENTER e a LINKAGE SECTION (PROC). 25
Exemplo: CALL SUBSOMA.

CHARACTERS – opção usada para definir dados por caracteres em vez de registro (DATA). 10
Exemplo: BLOCK CONTAINS 100 CHARACTERS.

CLOSE – comando usado para fechar um arquivo de dados (PROC). 4 e 18
Exemplo: CLOSE ARQUIVO-SAIDA.

COLUMN – cláusula utilizada para identificar posição específica no REPORT SECTION (DATA). –

COMPUTATIONAL – definição da forma de representação através da cláusula USAGE, outra opção é DISPLAY (DATA). 12
Exemplo: USAGE IS COMPUTATIONAL.

COMPUTE – comando usado para calcular expressão aritmética (PROC). 4 e 15
Exemplo: COMPUTE X = Y - (C * 3)/D.

CONFIGURATION SECTION – seção que define a configuração ou componentes (hardware) do sistema de computação usado (ENV). 8

CONSOLE – sigla utilizada para designar a máquina de escrever do console (PROC). 18

CONSTANTES OU LITERAIS – são valores constantes assumidos literalmente pelo programa e que não se alteram. Podem ser:
Constantes ou literais numéricas: 10, 0.0055
Constantes ou literais não numéricas: 'ERRO UM', '0,26%'
Constantes ou literais figurativas: ZERO, SPACES, QUOTE, ALL, HIGH-VALUE, LOW-VALUE. 6

Capítulo

CONTROLS (ARE) – cláusula usada no REPORT SECTION (DATA). –

COPY – cláusula usada para copiar uma sentença predefinida na Biblioteca do sistema COBOL (ENV. e DATA). 8
Exemplo: COPY LIBRARY-X.

CORRESPONDING ou CORR – ver ADD CORRESPONDING e MOVE CORRESPONDING. 15

CR – símbolos usados para editar a sigla CR (crédito) através do PICTURE (DATA). 22
Exemplo: PICTURE IS 9999V99CR.

D

DATA DIVISION – divisão onde são definidas as variáveis e as áreas de dados, estabelecendo ligação entre a memória e os arquivos de dados. 4 e 9

DATA RECORD – cláusula opcional que define o nome dos registros contidos em um arquivo (DATA). 10
Exemplo: FD ARQUIVO-LER .... DATA RECORD IS NOME, NUMERO.

DATE-COMPILED ou WRITTEN – cláusulas que definem as datas de compilação ou preparação do programa (IDEN). 8
Exemplo: DATE-WRITTEN. 12/OUT/88.

DB – símbolo usado para editar a sigla DB (débito) através do PICTURE (DATA). 22
Exemplo: PICTURE IS 9999DB.

DECLARATIVES – seção do PROCEDURE DIVISION onde são declaradas as condições especiais de operação do programa. Estas condições especiais utilizam os comandos USE, DEFINE e INCLUDE. Apêndice A
Exemplo:
```
PROCEDURE DIVISION.
    DECLARATIVES.
    TAPE-ERROR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE
      ON INPUT.
    END DECLARATIVES.
```

DEFINE – comando especial para redefinir um verbo ou comando novo no programa. Usado na seção DECLARATIVES. –

DEPENDING ON – ver GO TO DEPENDING ON. 16

DISPLAY – comando usado para escrever comentários na máquina de escrever do console (PROC) ou definir tipo de dado usado que não pode ser usado em cálculo (DATA). 12 e 18
Exemplos:
```
DISPLAY 'ERRO-TIPO UM' UPON CONSOLE.
USAGE IS DISPLAY.
```

Capítulo

DIVIDE – comando aritmético de divisão (PROC). 4 e 15
Exemplo: DIVIDE X INTO Y.

DIVISION – indica uma das quatro divisões do programa COBOL. –

DOWN – palavra que decrementa o valor de um índice de variável indexada através do comando SET (PROC). 24
Exemplo: SET INDICE DOWN BY 5.

E

ELSE – palavra opcional que indica a alternativa FALSA de um comando IF (PROC). 17
Exemplo: IF X EQUAL TO ZERO MOVE Y TO Z ELSE COMPUTE X = 25.

END – ver AT END. –

ENTER – comando usado para definir o começo (entrada) de um programa escrito em outra linguagem (PROC). 25
Exemplo: (Programa COBOL)
ENTER ASSEMBLER.
(Programa em Assembler)
EXIT.
ENTER COBOL.
(Programa em COBOL).

ENVIRONMENT DIVISION – divisão onde são definidos os equipamentos de hardware (através da CONFIGURATION SECTION) e estabelecidas as relações entre os arquivos de dados e as unidades periféricas (através do INPUT-OUTPUT SECTION). 8

EQUAL (TO) – operador relacional usado para formar uma expressão lógica (PROC). 7 e 17
Exemplos: IF X IS EQUAL TO ZERO MOVE X TO Y.

EXAMINE – comando usado para contar (TALLYING) ou substituir (REPLACING) caracteres de um campo de dados (PROC). 23
Exemplos: EXAMINE NOME TALLYING ALL SPACES.
Análogo ao comando INSPECT.

EXIT – comando usado para o retorno automático de parágrafos executados pelo comando PERFORM como se fossem sub-rotinas. Usado também com o comando ENTER. 20 e 25

F

FD – sigla colocada na margem A para indicar a definição das características de um arquivo de dados no FILE SECTION (DATA). 10
Exemplo: FD ARQUIVO-ENTRADA, LABEL RECORD IS STANDARD.

Capítulo

FILE-CONTROL – estabelece ligação entre os arquivos e as unidades de entrada/saída no INPUT-OUTPUT SECTION do ENVIRONMENT DIVISION. 8
Exemplo: FILE-CONTROL.
SELECT ARQUIVO ASSIGN TO CARD-READER.

FILE SECTION – seção do DATA DIVISION que define as características de um Arquivo de Dados e estabelece ligação de seus registros com os locais de memória. (DATA). 10

FILLER – cláusula utilizada no DATA DIVISION para definir a ocorrência de espaços em branco. 4 e 12
Exemplo: 03 FILLER PICTURE X(10).

FINAL – palavra-chave usada no REPORT SECTION. (DATA). –

FOR MULTIPLE REEL – cláusula opcional usada na definição de fita magnética no FILE-CONTROL (ENV). 8

FROM – palavra usada nos comandos SUBTRACT, ACCEPT, WRITE etc. (PROC). –
Exemplo: SUBTRACT X FROM Y.
ACCEPT NOME FROM CONSOLE.

G

GENERATE – comando usado para imprimir relatórios definidos pelo REPORT SECTION (PROC). –

GIVING – opção utilizada nos comandos aritméticos para definir uma variável onde se coloca o resultado da operação. 15
Exemplos: ADD X, Y GIVING Z.

GO TO – comando de desvio condicional para o início de um parágrafo (PROC). 16
Exemplo: GO TO PARAGRAFO-UM.

GO TO DEPENDING ON – comando de desvio para várias alternativas possíveis (PROC). 16
Exemplo: GO TO PARAG-1, PARAG-2, PARAG-3 DEPENDING ON VALOR.

GREATER THAN – operador relacional usado na expressão lógica (PROC). 7 e 17
Exemplo: IF X GREATER THAN Y MOVE X TO Y.

H

HIGH-VALUE – nome da constante figurativa indicando o maior valor possível em COBOL (PROC). 6
Exemplo: MOVE HIGH-VALUE TO ITEM.

IS – palavra opcional usada para melhorar a legibilidade das expressões lógicas ou cláusulas.
Exemplos: IF X IS EQUAL TO Y MOVE X TO Z (PROC).
USAGE IS COMPUTATIONAL (DATA). –

J

JUSTIFIED (LEFT ou RIGHT) – cláusula para definir o posicionamento de dados, contrário ao padronizado, na movimentação de um campo para outro (DATA). 12

K

KEY – cláusula do FILE-CONTROL que define o campo-chave ou índice a ser utilizado no processo de leitura ou escrita de registros em um arquivo de dado de acesso direto ou aleatório (ENV). Pode ser: ACTUAL KEY, SYMBOLIC KEY, NOMINAL KEY etc. 27
Exemplo: SYMBOLIC KEY IS CODIGO-DA-PEÇA.

KEY-WORDS – palavras-chave que são de uso obrigatório em um comando ou cláusula. –
Exemplos: ADD, MOVE, PICTURE etc.

L

LABEL RECORDS – cláusula que define o tipo de rótulo ou etiqueta de identificação de um arquivo de dados (DATA). 10
Exemplo: FD ARQUIVO-UM LABEL RECORDS ARE STANDARD.

LEADING – palavra-chave usada nos comandos EXAMINE ou INSPECT indicando as posições mais à esquerda dentro de um campo de dado (PROC). 23
Exemplo: EXAMINE X REPLACING LEADING ZEROS BY SPACES.

LESS THAN – operador relacional usado na expressão lógica (PROC). 7
Exemplo: IF XL LESS THAN Y THEN MOVE XL TO Z.

LINES – ver ADVANCING LINES. 18

LINKAGE SECTION – seção especial do DATA DIVISION onde são definidos os argumentos ou parâmetros de sub-rotinas externas, e que são usados nos comandos CALL e ENTER. 25

LITERAIS – ver CONSTANTES.

LOCK – palavra-chave usada no comando CLOSE (PROC). 18
Exemplo: CLOSE FITA-UM WITH LOCK.

Capítulo

LOW-VALUE – constante figurativa que representa o menor valor possível assumido pelo COBOL (PROC). 6
Exemplo: MOVE LOW-VALUE TO MEDIA.

M

MERGE – comando especial, disponível em versões especiais do COBOL, e que intercala ou mistura automaticamente os registros de dois arquivos, formando um terceiro arquivo classificado (PROC). –
Usado em arquivos definidos pela sigla SD do FILE SECTION; ver SORT.

MOVE e MOVE CORRESPONDING – transfere valores de uma variável ou campo de dados para outro (PROC). 19
Exemplos: MOVE DADO-L TO DADO-X.
MOVE CORRESPONDING AREA-X TO AREA-2.

MULTIPLY – comando de multiplicação aritmética (PROC). 4 e 15
Exemplo: MULTIPLY X BY Y GIVING Z.

N

NEGATIVE – nome especial, indicando condição negativa (PROC). –
Exemplo: IF VALOR IS NEGATIVE WRITE MENSAGEM.

NEXT SENTENCE – palavra-chave usada para preencher uma das alternativas da condição VERDADEIRA ou FALSA no comando IF (PROC). 17
Exemplo: IF X = O THEN NEXT SENTENCE
ELSE WRITE X.

NOMES ou IDENTIFICADORES – palavras formadas pelo programador usando até 30 caracteres e servem para representar:
nome de dados ou variáveis;
nome de parágrafo ou seção;
nome de condição;
nomes especiais, como PRINTER, CARD. 6

NOMINAL KEY – ver KEY.

NOT – operador lógico que indica a negação de uma condição ou expressão lógica (PROC). 7
Exemplos: NOT EQUAL TO, NOT POSITIVE.

NOTE – comando utilizado para colocar comentários no meio do programa. Pode ser substituído pela colocação de um asterisco (*) na coluna 7 da folha de codificação COBOL. 20
Exemplo: NOTE INICIO DO TESTE.

NUMERIC – palavra-chave usada para testar ou indicar a condição de valor numérico (PROC). 7 e 17
Exemplo: IF DADO IS NUMERIC GO TO Y.

OBJECT-COMPUTER – cláusula que define o nome do computador que processa o programa-objeto (IDENTIFICATION DIVISION). 8

OCCURS – cláusula que define a repetição ou número de variáveis dentro de uma tabela ou variável subscrita (DATA). 24
Exemplo: 01 TABELA.
02 HORARIO OCCURS 5 TIMES.

OF – palavra conectiva para destacar um nome dentro de um campo hierarquizado de dados (PROC). 6
Exemplo: VALOR OF ITEM IS 100.
ADD TAXA OF JUROS TO SOMA.

OMITTED – palavra-chave usada na opção para definição de rótulo de um arquivo de dados (DATA). 10
Exemplo: LABEL RECORD IS OMITTED.

ON – palavra opcional usada para melhorar a legibilidade de sentenças como ON SIZE ERROR, DEPENDING ON etc. (PROC).
Exemplo: ADD X TO Y, ON SIZE ERROR DISPLAY 'ERRO'.

OPEN – comando usado para abrir um arquivo de dados (PROC). 18
Exemplo: OPEN INPUT ARQUIVO–X.

OR – operador lógico que liga duas expressões lógicas, indicando a ocorrência de um ou de outro ou de ambas as condições indicadas (PROC). 7 e 17
Exemplo: IF X = Y OR Z LESS THAN 100 MOVE X TO Z.

ORGANIZATION – cláusula do FILE-CONTROL do ENVIRONMENT DIVISION que indica o tipo de organização de dados adotado pelo arquivo. 26 e 27
Exemplo: ORGANIZATION IS INDEXED (ou DIRECT ou RELATIVE).

OTHERWISE – palavra-chave usada como alternativa da opção FALSA do comando IF; equivalente a ELSE (PROC). 17
Exemplo: IF X = O MOVE X TO Y OTHERWISE WRITE Y.

OUTPUT – palavra-chave usada para indicar operação ou arquivo de saída (PROC). 18
Exemplo: OPEN OUTPUT ARQUIVO-X.

P

P – símbolo usado pelo PICTURE para alterar a escala decimal de um valor, multiplicando ou dividindo o valor por dez (DATA). 22
Exemplo: PICTURE IS 99PV.

PAGE – opção do comando WRITE para avançar uma página do formulário (PROC). 18
Exemplo: WRITE X AFTER ADVANCING PAGE.

PALAVRA-CHAVE o mesmo que KEY-WORD e de uso obrigatório em uma sentença em COBOL, pois indica a ação ou operação do comando ou cláusula. 6
Exemplos: ADD, PICTURE, VALUE etc.

PALAVRA CONECTIVA – serve para ligar dois nomes, através das conectivas IN ou OF. 6
Exemplo: VALOR OF TOTAL

PALAVRA OPCIONAL – palavra utilizada para melhorar a legibilidade de uma sentença ou comando COBOL. 6
Exemplos: IS, ON etc.

PALAVRA RESERVADA – palavra cujo uso como nome de variável, condição ou de parágrafo é proibido. É formada pelas palavras-chaves, palavras conectivas e palavra opcional. Ver LISTA DE PALAVRAS-CHAVES no APÊNDICE B. 6

PARAGRAPH – designação de conjunto de comandos COBOL do PROCEDURE DIVISION e que recebe um nome. 4

PERFORM – comando que executa um ou mais parágrafos como se fossem sub-rotinas (PROC). 4, 6 e 16
Exemplos: PERFORM PARAGRAFO-SOMA.
PERFORM PARAGRAFO-DEZ THROUGH PARAGRAFO-VINTE.

PICTURE OU PIC – cláusula de designação do tipo, formato e tamanho de um campo de dado, através de um conjunto de símbolos especiais. Usado no DATA DIVISION. 4, 11 e 22
SÍMBOLOS UTILIZADOS:

A: define caráter alfabético, de A a Z.
X: define caráter alfanumérico: letras, números e sinais.
9: define caráter numérico de 0 a 9.
V: define posição de um ponto (ou vírgula) decimal implícito ou assumido.
P: posiciona valor por múltiplo ou submúltiplo de dez.
S: define a ocorrência de um sinal algébrico.
0 (zero): coloca um valor zero no campo.
B: coloca um espaço em branco no campo.
Z: suprime todos os zeros à esquerda do valor.
$**: suprime todos os zeros à esquerda e coloca um sinal **$.
* : suprime todos os zeros à esquerda e coloca um sinal (*); podem ser colocados vários sinais **$** ou (*).
. (ponto): coloca um ponto decimal no campo.
, (vírgula) coloca uma vírgula no campo.
CR ou DB: coloca sigla CR (crédito) ou DB (débito) no campo.

+ (mais) ou - (menos): coloca sinal mais ou menos no campo.

POSITIVE – palavra-chave indicando condição positiva (PROC). 17
Exemplo: IF CODIGO IS POSITIVE MOVE X TO Y.

PROCEDURE DIVISION – divisão do programa COBOL onde são colocados os comandos de execução. 4

PROGRAM-ID – cláusula obrigatória do IDENTIFICATION DIVISION e que define o nome do programa. 8

Q

QUOTE – constante figurativa que indica a ocorrência do sinal (') apóstrofe (PROC). 6
Exemplo: DISPLAY QUOTE XXX QUOTE.

R

RANDOM – opção RANDOM (aleatória) de um arquivo definido pela cláusula ACCESS (ENV). 26 e 27
Exemplo: ACCES IS RANDOM.

RD – sigla colocada na margem A da folha de codificação e serve para definir um relatório no REPORT SECTION (DATA). –
Exemplo: RD RELATORIO-VENDAS CONTROLS ARE FINAL.

READ – comando que lê um registro de arquivo de dados e coloca na memória (PROC). 18
Exemplo: READ ARQUIVO-UM; AT END COMPUTE X =9.

RECORD – palavra opcional usada em alguns comandos ou cláusulas. –
Exemplo: READ RECORD ARQUIVO

RECORD CONTAINS – cláusula usada na linha FD do FILE SECTION para definir o número de caracteres de um registro (DATA). 10
Exemplo: RECORD CONTAINS 100 CHARACTERS.

RECORD KEY – ver KEY. –

RECORDING MODE – cláusula usada na linha FD do FILE SECTION, e serve para definir o modo F (registros de tamanho fixo) ou V (registros de tamanho variável) de um arquivo (DATA). 10
Exemplo: RECORDING MODE IS F.

REDEFINES – cláusula que designa a mesma área de memória para mais de uma variável ou nome e atribui características diferentes (DATA). 24
Exemplo: 02 AREA-UM REDEFINES AREA-ANTERIOR.

Capítulo

REEL – palavra que designa certas características do carretel de fita magnética. 18
Exemplos: FOR MULTIPLE REEL RESERVE 2 ALTERNATE AREAS (ENV).
CLOSE ARQUIVO REEL WITH REWIND (PROC).

RENAMING – cláusula usada no FILE-CONTROL para redefinir um arquivo com mais de um nome (ENV). 8
Exemplo: SELECT ARQUIVO-UM RENAMING ARQUIVO--DOIS ASSIGN TO PRINTER.

REPLACING – palavra utilizada nos comandos EXAMINE e INSPECT (PROC). 23
Exemplo: EXAMINE X REPLACING ALL ZEROES BY SPACES.

REPORT SECTION – seção do DATA DIVISION usada para definir a forma e tipo de um relatório padronizado. Apêndice A

RERUN – cláusula do I-O-CONTROL do ENVIRONMENT DIVISION. 8

RESERVE – palavra usada no FILE-CONTROL DO ENV DIVISION. 8
Exemplo: RESERVE 2 ALTERNATE AREAS.

REWIND – palavra usada no comando que manipula arquivo de fita magnética (PROC). 18
Exemplo: CLOSE ARQUIVO-FITA WITH NO REWIND.

REWRITE – comando utilizado para gravar, no mesmo local onde foi lido, um registro de arquivo em disco magnético. O comando WRITE em disco seria para gravar um novo registro (PROC). 27
Exemplo: REWRITE REG-DISCO INVALID KEY DISPLAY 'ERRO'.

REVERSED – palavra utilizada no comando OPEN e serve para operar fita magnética do fim para o começo (PROC). 18
Exemplo: OPEN FITA-UM REVERSED.

ROUNDED – palavra utilizada para arredondar os valores resultantes de operação aritmética (PROC). 15
Exemplo: ADD X TO Y ROUNDED.

RUN – palavra utilizada no comando STOP RUN (PROC). 16

S

S – símbolo usado pelo PICTURE para indicar ocorrência de um sinal (DATA). 11 e 22
Exemplo: PICTURE IS S999V99.

Capítulo

SAME (RECORD) AREA - operação definida do I-O-CONTROL do ENVIRONMENT DIVISION para atribuir mesma área da memória a vários arquivos. 8
Exemplo: SAME RECORD AREA FOR ARQUIVO-L, ARQUIVO-M, ARQUIVO-N.

SD – linha de definição de um arquivo a ser classificada pelo comando SORT (DATA). –
Exemplo: SD ARQUIVO-CLASSIFICAR.

SEARCH – comando que efetua a busca de um dado na tabela indexada, definida pelos comandos e cláusulas SET e INDEX (PROC). 24
Exemplo: SEARCH CONTA WHEN NUMERO (INDICE) EQUAL TO VALOR.

SEARCH ALL – opção do comando SEARCH efetuando a pesquisa da tabela através de método de busca binária que acelera o processo (PROC). 24

SECTION – seções ou partes de diversas divisões. –

SELECT – cláusula do FILE-CONTROL que designa arquivo de dados a unidades periféricas de entrada/saída (ENV). 8
Exemplo: SELECT ARQUIVO ASSIGN TO PRINTER.

SEQUENTIAL – ver ACCESS e RANDOM. 26

SET – comando usado para fixar, incrementar ou decrementar o valor dos índices das variáveis indexadas definido pela cláusula INDEX (PROC). 24
Exemplo: SET INDICE TO 1.

SIZE ERROR – opção colocada nos comandos aritméticos para prevenir a ocorrência de transbordo (OVERFLOW) ou valores muito baixos (UNDERFLOW) nos resultados (PROC). 15
Exemplo: ADD X, Y TO C; ON SIZE ERROR COMPUTE X = 999.

SORT – comando especial usado para classificar um arquivo de dados definido pela linha SD do FILE SECTION (PROC). –
Exemplo: SORT ARQUIVO-X ON ASCENDING KEY.

SOURCE – cláusula usada no REPORT SECTION (DATA). –

SOURCE COMPUTER – cláusula que define o nome do computador onde o programa COBOL foi compilado (ENV). 8

SPACE ou SPACES – constante figurativa que indica a ocorrência de espaços em branco (PROC). 6
Exemplo: EXAMINE NOME TALLYNG LEADING SPACES.

SPECIAL-NAMES – seção do ENVIRONMENT DIVISION que define as condições especiais ou nomes especiais, diferentes dos padrões oferecidos pelo COBOL (ENV). 8

Exemplos: SPECIAL-NAMES.
COPY DESCRITOR-X.
DECIMAL POINT IS COMMA.

STANDARD – opção padronizada adotada pelo COBOL. –
Exemplo: LABEL IS STANDARD.

STOP RUN – comando que indica o fim de um programa COBOL (PROC). 16

SUBTRACT ou SUBTRACT CORRESPONDING – subtrai valor de uma área ou conjunto de áreas (PROC). 15
Exemplo: SUBTRACT A FROM X, GIVING Y.
SUBTRACT CORRESPONDING AREA-TOTAL FROM AREA.

SYMBOLIC KEY – ver KEY. 27

SYNCHRONIZED (LEFT ou RIGHT) – cláusula para posicionar os dados à esquerda ou à direita do campo (DATA). 12

T

TALLY – contador especial usado pelo comando EXAMINE (PROC). 23

TALLYING – opção do comando EXAMINE para contar, através do contador TALLY, os caracteres desejados (PROC).
Exemplo: EXAMINE CAMPO TALLYING ALL SPACES.

THEN – palavra opcional utilizada no comando IF (PROC). 17
Exemplo: IF X = O THEN MOVE X TO Y.

THRU – palavra usada no comando PERFORM ou cláusula VALUE. 16
Exemplos: PERFORM PARAGRAFO-1 THRU PARAGRAFO-ONZE.
VALUE IS 1 THRU 5.

TERMINATE – comando utilizado para terminar a geração de um relatório definido pelo REPORT SECTION (DATA). –

TIMES – palavra que indica a ocorrência de repetição da definição. –
Exemplos: PERFORM PARAG-5 10 TIMES (PROC).
02 X OCCURS 7 TIMES (DATA).

TO – palavra-chave usada nos comandos aritméticos e cláusulas. –
Exemplos: BLOCK CONTAINS 3 TO 5 RECORDS (DATA).
ADD X TO Y (PROC).

TYPE – cláusula usada no REPORT SECTION (DATA). –

U

Capítulo

UNTIL – palavra-chave usada nos comandos PERFORM, EXAMINE e INSPECT. (PROC). 16
Exemplos: PERFORM PARAG-UM UNTIL X = Y.
EXAMINE X TALLYING UNTIL FIRST ZERO.

USAGE – cláusula que define se o dado é do tipo COMPUTATIONAL (só usado para cálculos) ou DISPLAY (só usado para movimentação e escrita) (DATA). 12
Exemplo: USAGE IS COMPUTATIONAL-1.

USE – comando especial da seção DECLARATIVES do PROCEDURE DIVISION para definir trecho de uso especial do programa. Apêndice A

UP – indica ato de incrementar um índice (PROC). 24
Exemplo: SET INDICE-L UP BY 1.

UPON – ver DISPLAY. 18

V

V – símbolo que indica a posição implícita ou assumida de um ponto decimal definida pelo PICTURE (DATA). 4 e 11
Exemplo: PICTURE IS 999V99.

VALUE – cláusula que fornece o valor inicial de um dado ou do registro de identificação de um arquivo (DATA). 10 e 12
Exemplos: FD ARQUIVO-DOIS LABEL RECORD IS STANDARD VALUE OF ID IS 'MESTRE'.
03 DADO-X PICTURE 999 VALUE IS 5.

VARYING – palavra utilizada no comando PERFORM (PROC). 16
Exemplo: PERFORM PARAGRAFO-DOIS VARYING
INDICE FROM 1 BY 1 UNTIL INDICE = 50.

X

X – símbolo usado no PICTURE para representar um caráter alfanumérico (DATA). 4 e 11
Exemplo: PICTURE IS XXX.

W

WHEN – palavra usada no comando SEARCH. (PROC). 24

WORKING-STORAGE SECTION – seção do DATA DIVISION onde são definidos as variáveis e as áreas de trabalhos, bem como os nomes de condições (DATA). 13

Capítulo

WRITE – comando para escrever valores da memória em um registro de dado de um arquivo de saída (PROC). 18
Exemplo: WRITE LINHA-UM FROM DADO-OBTIDO.

Z

Z – símbolo usado pelo PICTURE para indicar a supressão de zeros à esquerda de um valor (DATA). 22
Exemplo: 02 VALORES PICTURE IS ZZ999.99.

ZERO ou ZEROS ou ZEROES – constante figurativa que indica o valor zero ou zeros. 6
Exemplos: EXAMINE CODIGO TALLYING ALL ZEROS (PROC).
02 ITEM-UM PICTURE 999 VALUE IS ZERO (DATA).

REFERÊNCIAS BIBLIOGRÁFICAS

Alves, A. P. *Programação*. São Paulo, Atlas, 1978.

Diversos autores. "Data Base Management Systems". *ACM — Computing Surveys,* número especial, 8 (1), março, 1976.

Gusman e Vasconcellos. *Fluxograma e Programação COBOL*. Rio de Janeiro, Ao Livro Técnico Ltda./DATAMEC.

McCracken, D. *A Simplified Guide to Structured COBOL Programming*. New York, Wiley & Sons.

McCracken, D. e U. Garbassi. *Programação COBOL*. São Paulo, Atlas, 1980.

Saxon, J. A. *COBOL: A Self-Instructional Manual*. New Jersey, Prentice-Hall.

Shimizu, T. *Processamento de Dados: Conceitos Básicos*. São Paulo, Atlas, 1980.

Shimizu, T. *Processamento de Dados nas Empresas*. São Paulo, Atlas, 1983.

Treanor, R. G. "Data Management — Fact and Fiction". *Data Base — Newsletter,* da ACM — SIGBDP, 3 (1), 1971.

Watters, J. *COBOL Programming*. Londres, Heinemann Books.

Impresso em offset

GRAFICA EDITORA HAMBURG

Avenida Bogaert, 64
Vila das Mercês São Paulo
Fone: 914-0233
CEP 04298

com filmes fornecidos pelo editor

PROCESSAMENTO DE DADOS

PROCESSAMENTO DE DADOS — Conceitos Básicos
Tamio Shimizu

Atende à necessidade de levar ao estudante, de forma extremamente simples, o que é o computador e quais os principais conceitos relacionados ao uso dessa máquina. Exemplos rotineiros, de rápida assimilação, quase sempre acompanhados de ilustrações elementares, aparecem ao longo do texto, familiarizando os estudantes com as aplicações dos computadores e removendo a idéia generalizada de que tais equipamentos são por demais sofisticados para serem entendidos e usados pela maior parte das pessoas. Traz exercícios para fixação dos conceitos e um minidicionário de termos técnicos.

PROGRAMAÇÃO COBOL
Daniel D. McCraken e Umberto Garbassi

Como a linguagem COBOL é amplamente utilizada para o processamento de dados empresariais e comerciais, este texto se destina a todos que queiram adquirir conhecimentos essenciais sobre a aplicação de computadores nas empresas. Ricamente ilustrado, contém grande variedade de exercícios de recapitulação, cujas questões se encontram resolvidas. Apresenta três casos para estudo: **Estatística de Vendas, Controle de Estoques** e **Folha de Pagamento.**

PROGRAMAÇÃO PL/1 — Manual de Instrução Programada
Edna H. Barbour

Apesar de seu caráter didático, este texto pode ser utilizado por todos os interessados em aprender a programar, sem o auxílio de um professor ou de qualquer instrução formal. Dada a ambivalência da linguagem PL/1, pode adaptar-se tanto a programas científicos e matemáticos quanto a programas empresariais. Sendo uma linguagem de alto nível, que se assemelha bastante ao inglês, requer menos tempo de memorização e de aprendizagem. Contém uma série de exercícios de programação bem como as respectivas soluções. Traz ainda dois apêndices, que tratam dos seguintes assuntos: exemplo de cartões de linguagem **Job Control** e palavras-chave utilizadas no texto.

PROGRAMAÇÃO
Aloisio Pinto Alves

Resultante da experiência profissional do autor na área de processamento de dados, este texto pode ser apontado como pioneiro na bibliografia nacional disponível, justamente por cuidar da **técnica de programação**, particularmente dos problemas operacionais vinculados à codificação, aos testes, à depuração de erros e à documentação. Envolve cinco capítulos: **Projetos**, que aborda conceitos básicos sobre programação estruturada e fluxogramas; **Codificação**, que contém tópicos relacionados à clareza de um programa; **Testes**, que apresenta sugestões quanto aos tipos de dados e à metodologia empregada nos testes; **Depuração de erros** — atividade distinta dos testes; **Documentação**, com suas qualidades fundamentais e suas finalidades.

publicações atlas