FLUXOGRAMAS E PROGRAMAÇÃO COBOL

Gusman / Vasconcellos

2.ª edição

FLUXOGRAMAS E PROGRAMAÇÃO COBOL

GILZA GUSMAN

Graduada em Administração de Empresas pelas Faculdades Integradas Bennett.

- Gerente de Suporte Técnico da Companhia Internacional de Seguros.
- Instrutora de Cobol nos Cursos de Extensão na área de Informática promovidos pelo CPUERJ.

AUGUSTO DE VASCONCELLOS

Graduado em Engenharia Eletrônica pelo Instituto Tecnológico da Aeronáutica.
Consultor em Processamento de Dados.

LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

Rio de Janeiro-RJ • São Paulo-SP

Capa:
AG Comunicação Visual Assessoria e Projetos Ltda.

1ª edição: 1977
Reimpressões: 1977, 1978, 1979, 1980, 1981, 1982, 1983 e 1984
2ª edição: 1985

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

Gusman, Gilza.
G99f Fluxogramas e programação COBOL / Gilza
2. ed. Gusman, Augusto de Vasconcellos. — 2a. ed. — Rio de Janeiro: LTC — Livros Técnicos e Científicos Editora S.A., 1985.

Apêndices
Bibliografia

1. COBOL (Linguagem de programação para computadores) 2. Computadores eletrônicos I. Vasconcellos, Augusto de II. Título

CDD — 001.64
001.6424
CDU — 800.92COBOL
681.3

85-0304

ISBN: 85-216-0427-0

LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

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

Prefácio

Esta obra foi desenvolvida a partir da experiência adquirida pela autora como instrutora de disciplinas tais como "Introdução à Ciência da Computação" e "Programação de Computadores em Linguagem COBOL", integrantes de cursos de formação básica de programadores e de extensão na área de Informática, ministrados em entidades oficiais (UERJ) e particulares (LTD-DATAMEC) de ensino, como consultora autônoma, e como Analista de Suporte de Sistemas.

A obra, devido à estruturação adotada, é indicada tanto para iniciantes na área como também para programadores de micro-mini computadores e de computadores de grande porte, já envolvidos na área.

Ao longo da obra, é fornecida, de modo integrado, a tecnologia de elaboração de programas e a sua implementação utilizando a Linguagem COBOL.

Os capítulos que compõem a obra estão estruturados de acordo com uma seqüência lógica que permitirá aos leitores meios para o processamento dos seus programas desde o início.

Nos dois primeiros capítulos são abordados os conceitos fundamentais relativos a tecnologia de fluxogramação de programas, que familiarizam o leitor com as técnicas, e nos demais capítulos é abordada, detalhadamente, a sintaxe das instruções integrantes da Linguagem COBOL.

Para permitir uma melhor integração dos conceitos teóricos com a prática, ao longo de cada um dos capítulos, é apresentado um conjunto de exemplos, os quais permitirão ao leitor promover um melhor entendimento do assunto.

Gilza Gusman

Registro

A parte referente às instruções e comandos, que se encontra neste livro, utiliza material de domínio público, e o Comitê Executivo solicita que qualquer organização que pretenda escrever um manual de COBOL reproduza o seguinte, como parte da seção de apresentação do trabalho:

"Esta publicação baseia-se no Sistema COBOL desenvolvido em 1959 por um comitê composto de usuários do governo e de fabricantes de computadores.

As organizações participantes do desenvolvimento original foram:

Air Material Command, USAF;
Bureau of Standards, DC;
David Taylor Model Basin, USN;
EDP Division, Minneapolis Honeywell Reg. Co.;
Burroughs Corporation;
IBM;
RCA;
Sylvania Electric Products, Inc.;
UNIVAC.

Este manual é o resultado de contribuições de todas essas organizações. No entanto, nenhuma garantia é dada por qualquer contribuidor quanto à precisão ou ao funcionamento de linguagem.

Perguntas relativas a procedimentos e métodos para sugestões de alterações devem ser dirigidas ao Comitê Executivo da Conferência sobre Linguagens de Sistemas de Dados.

Os autores e os possuidores de direitos autorais do material aqui inserido: FLOWMATIC (Sperry Rand), DATA AUTOMATION SYSTEMS (Sperry Rand), COMMERCIAL TRANSLATOR (IBM), FACT (Honeywell) autorizam o seu uso nas especificações COBOL em manuais e publicações similares. Qualquer organização interessada em reproduzir o relatório COBOL e especificações iniciais, usando idéias tiradas deste relatório ou tomando-as como base para um manual, é livre para fazê-lo. Deverá, contudo, reproduzir esta seção como parte da introdução a seu documento."

Sumário

1

Diagramas de Fluxo

Para representar uma seqüência de etapas necessária à execução de um processo (chamada algoritmo), criou-se uma linguagem gráfica que recebeu o nome de "diagrama de fluxo" ou "fluxograma". Em nosso caso, os fluxogramas serão utilizados para representar as diversas ações que o computador deverá cumprir para executar a tarefa proposta pelo programador.

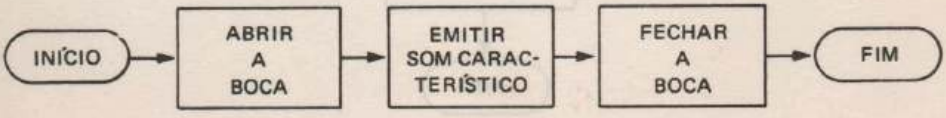
1.1. OS SÍMBOLOS

Com a finalidade de evitar que cada programador crie sua própria simbologia, desde cedo se fizeram várias tentativas no sentido de padronizar os símbolos utilizados e, recentemente, a Associação Brasileira de Normas Técnicas divulgou um projeto de norma, adotado aqui. (Ver Apêndice I)

É fácil ver que, não sendo os fluxogramas dedicados exclusivamente à programação, existem símbolos destinados a fluxogramas de análise de sistemas, de organização e métodos etc. No nosso caso, restringir-nos-emos aos símbolos utilizados nos fluxogramas de programas de computador.

Primeiramente, devemos conhecer o símbolo de INÍCIO, que também é utilizado para FIM ou INTERRUPÇÃO.

Vejamos como descrever, em fluxograma, a operação "bocejar":



Como vemos, essa operação trivial já nos exige algumas definições adicionais, como segue:

- o fluxograma deve ser elaborado a partir do canto superior esquerdo do papel desenvolvendo-se de cima para baixo ou da esquerda para a direita. Qualquer variação nessa regra deve vir caracterizada com setas indicativas da direção;

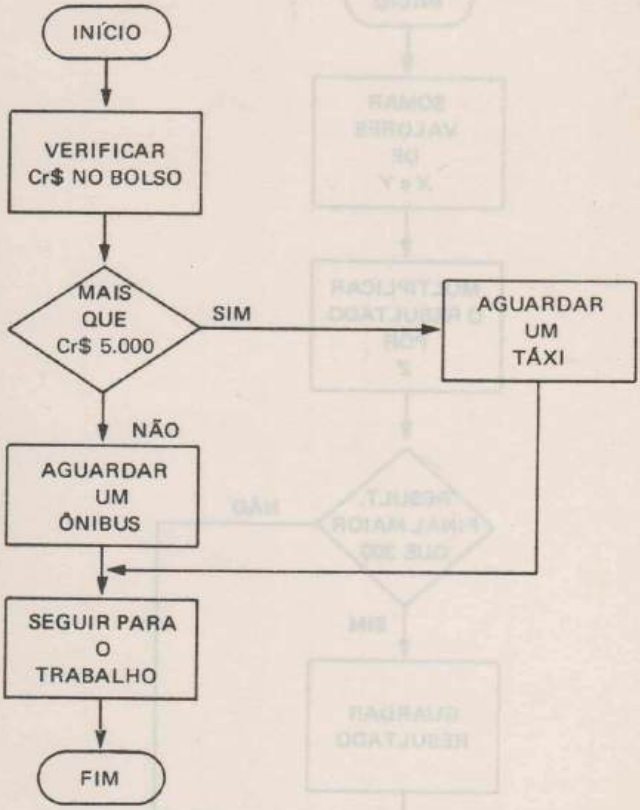
- além disso, fomos forçados a utilizar um outro símbolo, o retângulo, que representa todas as variedades de funções de processamento.

De posse dessas definições, podemos já efetuar uma incursão nos domínios da Matemática, elaborando o fluxograma correspondente à solução do problema:

$$\frac{X+Y}{Z} = A$$

Esse pequeno fluxograma poderia, perfeitamente, fazer parte de um extenso programa de computador, excetuando-se, é claro, os símbolos de INÍCIO e FIM. Não obstante, para que possamos elaborar fluxogramas representativos de situações mais complexas, é necessário um símbolo que represente uma operação de decisão, definindo o caminho a seguir sempre que houver mais de um.

Suponhamos uma situação em que o tipo de veículo de transporte a tomar para ir ao trabalho esteja sendo decidido.



Note-se a atuação do símbolo de decisão, criando os dois caminhos para o fluxo, bem como a conseqüente necessidade de mais uma definição: a de como fazer junção de linhas de ligação, meramente tocando uma linha secundária na principal. Aparece também aqui a utilização das setas, uma vez que contrariamos a direção

normal das linhas de fluxo e, além disso, a junção poderia dar margem a dúvidas quanto ao caminho a seguir.

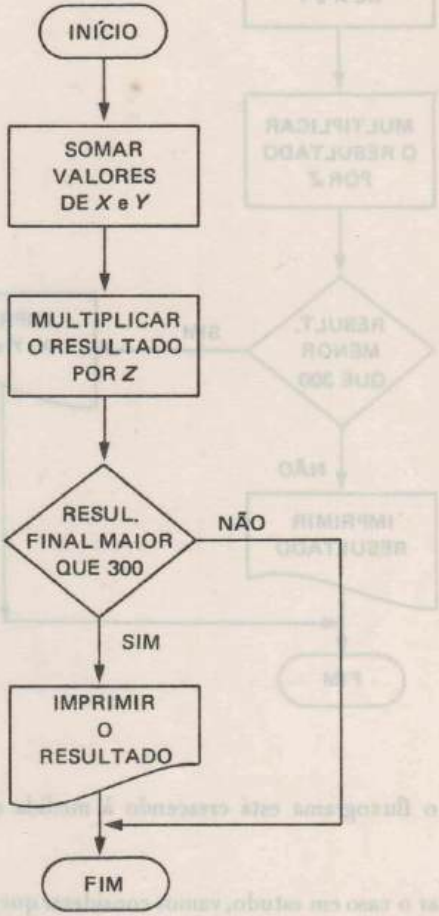
Vejamos um outro exemplo: estamos somando quantidades, porém somente nos interessa reter determinados resultados, definidos por

$$(X + Y)\, Z > 300.$$

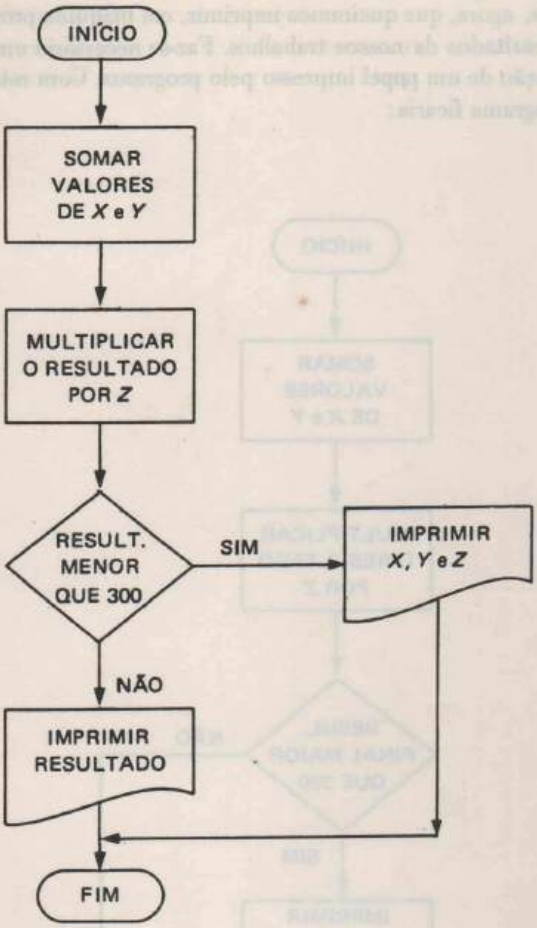
O fluxograma para um cálculo seria:

A decisão, nesse caso, define se devemos ou não guardar o resultado da operação, em função da condição analisada (testada).

Suponhamos, agora, que queiramos imprimir, em máquina própria, que faz parte do sistema, resultados de nossos trabalhos. Faz-se necessário um símbolo que represente a produção de um papel impresso pelo programa. Com esta opção incluída, nosso último programa ficaria:



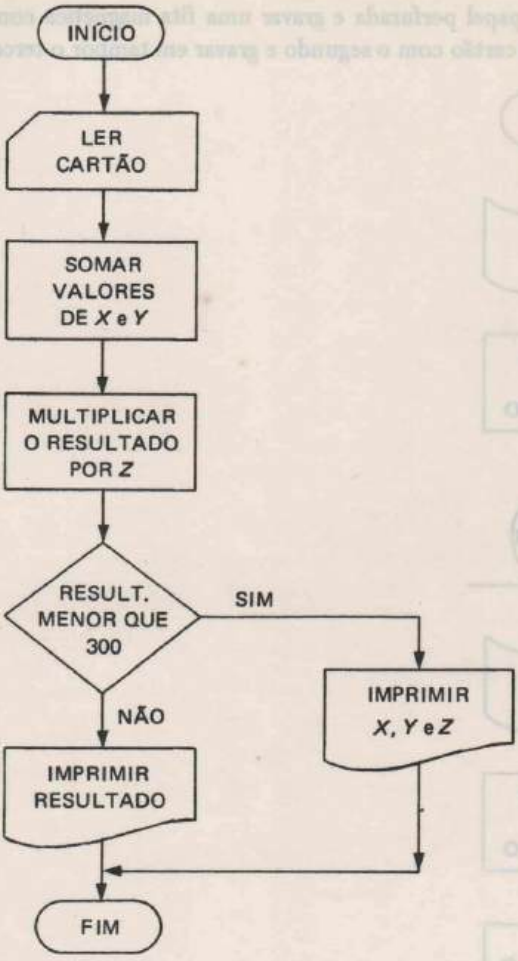
Para firmar os conceitos, suponhamos o mesmo problema, porém com uma nova condição: imprimir as parcelas, se o resultado for menor que 300.



Nota-se que o fluxograma está crescendo à medida que novas condições são consideradas.

Para completar o caso em estudo, vamos considerar que os valores de X e Y estão perfurados em um cartão. Isso faz com que tenhamos que definir o símbolo que doravante representará um cartão ou um conjunto de cartões perfurados.

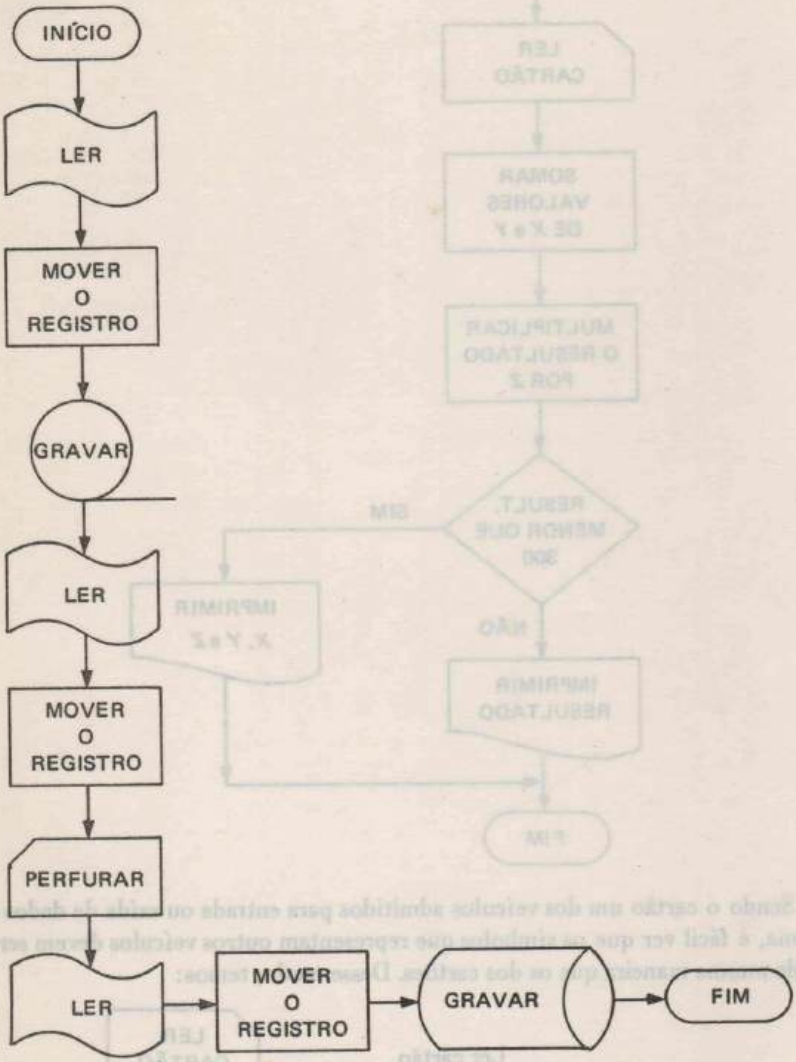
O símbolo utilizado para indicar a leitura de cartões é também usado para indicar a produção de cartões como resultado.



Sendo o cartão um dos veículos admitidos para entrada ou saída de dados num sistema, é fácil ver que os símbolos que representam outros veículos devem ser usados da mesma maneira que os dos cartões. Desse modo, temos:

Ler cartão e gravar fita magnética

Ler fita de papel perfurada e gravar uma fita magnética com o primeiro registro, perfurar um cartão com o segundo e gravar em tambor o terceiro:

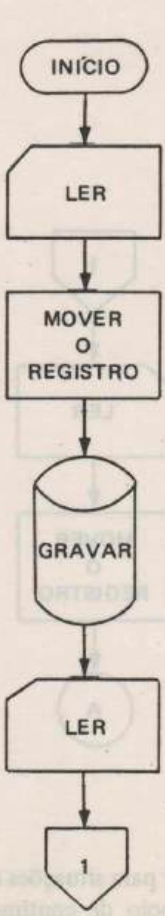


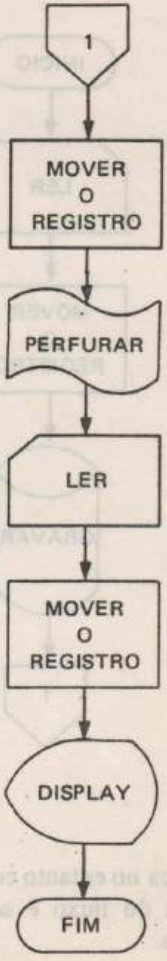
Esse fluxograma nos introduz aos símbolos representativos de fita de papel e de tambor magnético. Verificamos, além disso, que a utilização do símbolo próprio nos permite deixar de escrever o nome do veículo de dados dentro do símbolo, bastando indicar a operação que terá lugar.

Foi preciso, no caso, utilizar o símbolo de processamento para se poder retirar o conteúdo do registro lido da fita de papel, movendo-o, dentro da memória, de uma área de leitura para uma área utilizada para gravação.

Vamos agora ler três cartões, gravar o conteúdo do primeiro em disco, perfurar o do segundo em fita de papel e apresentar o conteúdo do terceiro em uma forma ilustrada de saída.

A forma ilustrada (*display*) pode ser uma tela de televisão, uma máquina de escrever, um traçador de curvas etc.



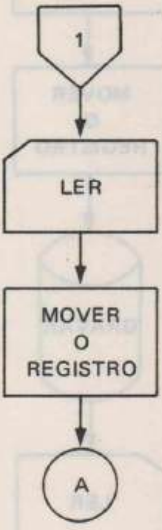


O símbolo de saída ilustrada definiu que os dados seriam apresentados na unidade chamada "console", e introduzimos também um símbolo especial, usado para continuação do fluxo.

Como proceder se o diagrama não couber numa só página?

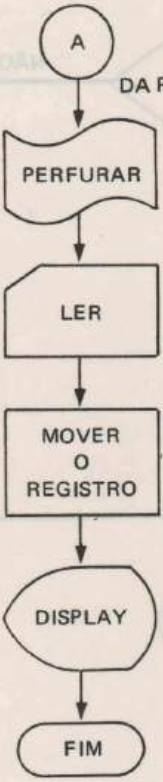
Para resolver esse problema, existe um símbolo próprio denominado "conector", com o qual se representa uma saída ou uma entrada em outra parte do fluxograma. Ilustraremos esse conceito com o exemplo anterior, supondo que fosse necessário duas páginas para conter todo o diagrama.

A primeira página conteria:



A prática no entanto consagrou o uso do conector para situações em que o sentido normal do fluxo é subvertido, ficando o símbolo de continuação para os demais casos.

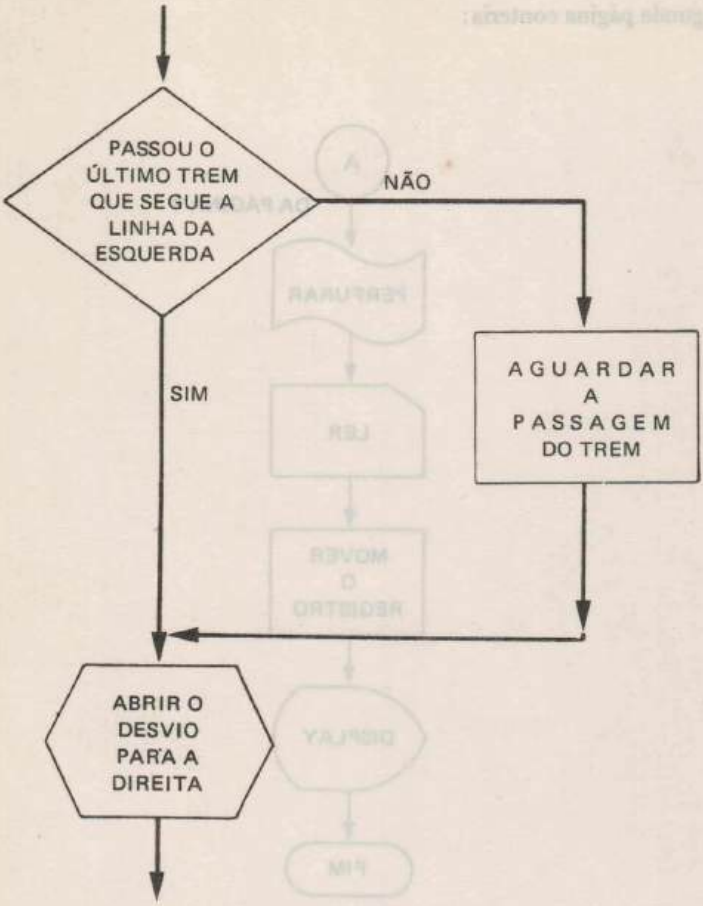
A segunda página conteria:



A letra "A", que aparece dentro do conector, deve-se ao fato de que na página 2 poderia haver mais que um conector, caso o fluxograma fosse mais complexo.

Estamos chegando ao fim dos símbolos que representam operações simples num fluxograma. Existem, no entanto, outros que representam situações um pouco mais complexas, e que devem ser abordados.

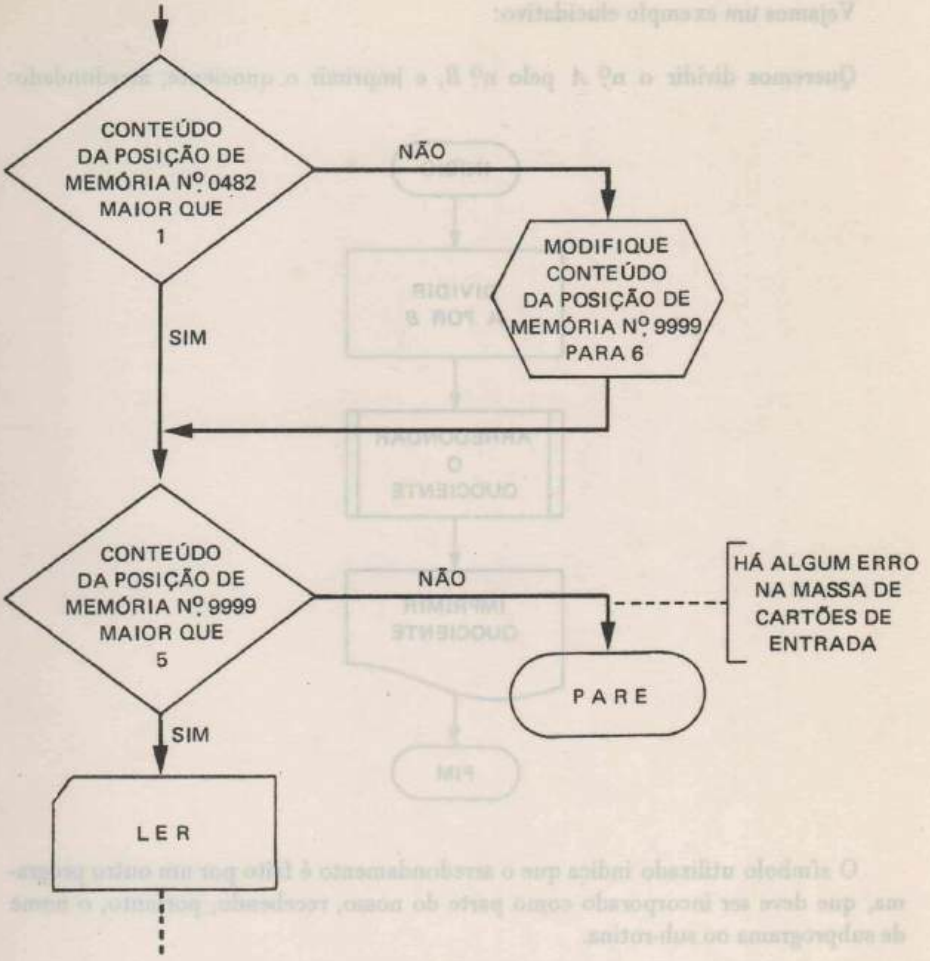
Suponhamos o trabalho de um funcionário de estrada de ferro, encarregado de um desvio, que estava dormindo em serviço.



O diagrama foi evidentemente simplificado para evitar o possível acidente ferroviário. Mesmo assim, nota-se a importantíssima função do novo símbolo apresentado, que representa uma modificação que altera a linha de funcionamento do programa.

Como o fluxograma está ficando excessivamente longo, abolimos os símbolos de início e fim por razões puramente didáticas, uma vez que num programa de computador eles terão que existir.

Vejamos outra situação:



Note-se a utilização do símbolo que estamos estudando, que criou condições para uma definição posterior quanto aos rumos que o programa deveria tomar.

Aproveitando a situação, introduzimos a maneira de apresentar um comentário, e exemplificamos o uso do símbolo de INÍCIO e FIM, para um caso de INTERRUPÇÃO.

Dentro dessa família de operações complexas, encontramos um símbolo que representa um processamento composto de uma ou mais operações ou de seqüências de programa definidas em outra parte (subprograma ou sub-rotina).

Vejamos um exemplo elucidativo:

Queremos dividir o nº *A* pelo nº *B*, e imprimir o quociente, arredondado:

O símbolo utilizado indica que o arredondamento é feito por um outro programa, que deve ser incorporado como parte do nosso, recebendo, portanto, o nome de subprograma ou sub-rotina.

1.2. A RESOLUÇÃO DOS PROBLEMAS

Uma consulta às normas da ABNT mostrará que existem diversos símbolos de fluxogramas que não foram abordados até aqui. Isso se deve ao fato de que esses símbolos são utilizados em diagramas feitos pelos analistas, e não se aplicam aos fluxogramas de programas de computador.

Uma vez expostos os elementos componentes da linguagem do fluxograma, cabe-nos tentar utilizá-los para representar as soluções dadas aos problemas que nos apresentem.

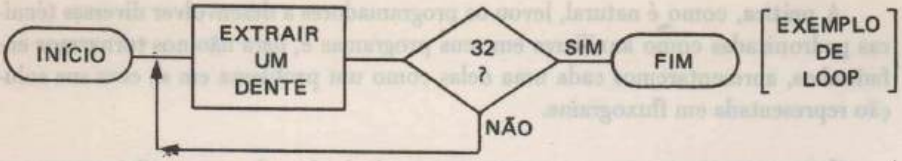
A prática, como é natural, levou os programadores a desenvolver diversas técnicas padronizadas como auxiliares em seus programas e, para não nos tornarmos enfadonhos, apresentaremos cada uma delas como um problema em si, com sua solução representada em fluxograma.

É importantíssimo que nos conscientizemos do fato de que um fluxograma *representa* a solução de um problema, mas *não é* a solução. Esta, uma vez encontrada, lançará mão de uma ou mais técnicas de programação, e o conjunto será *representado* por um fluxograma.

Suponhamos uma situação que exija ações iguais, repetidas. Por exemplo, um equipamento desenhado para extração de dentes. Informado da necessidade de extrair 32 dentes de um paciente, sua programação seria:

Teríamos 32 vezes repetido esse comando, terminando assim:

Esse diagrama, no entanto, pode ser simplificado por uma técnica chamada "iteração" (ou *loop*), que consiste em usar diversas vezes o mesmo trecho de um programa. O nosso diagrama ficaria:

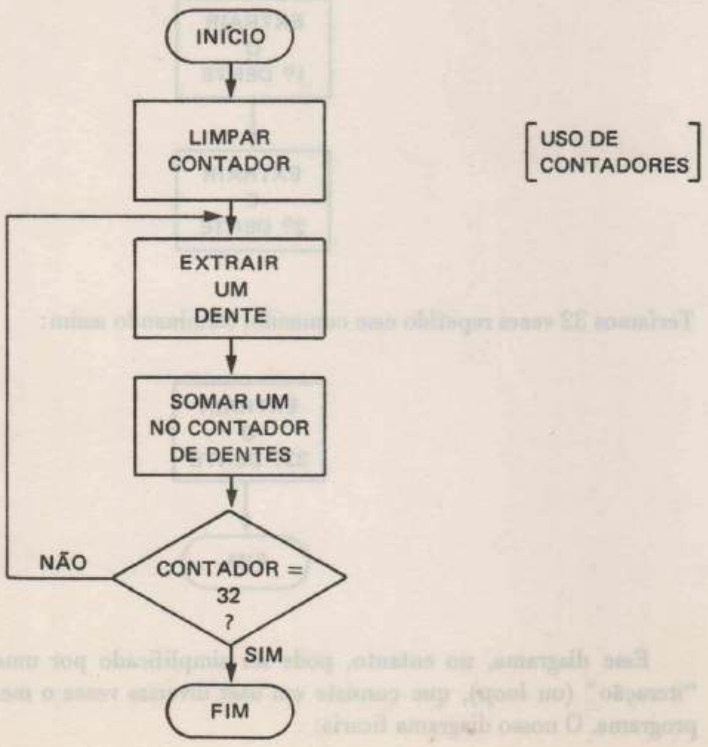


Nota-se de imediato a tremenda simplificação introduzida. Se, no entanto, pretendermos utilizar a nossa máquina em um segundo paciente, ela se recusará a trabalhar, pois seu programa informa que já extraiu 32 dentes, e, portanto, completou seu trabalho. Para evitar isso, utilizam-se duas outras técnicas:

a. "Contadores", que são locais onde se faz uma acumulação destinada a indicar quantas vezes a operação em pauta já foi usada.

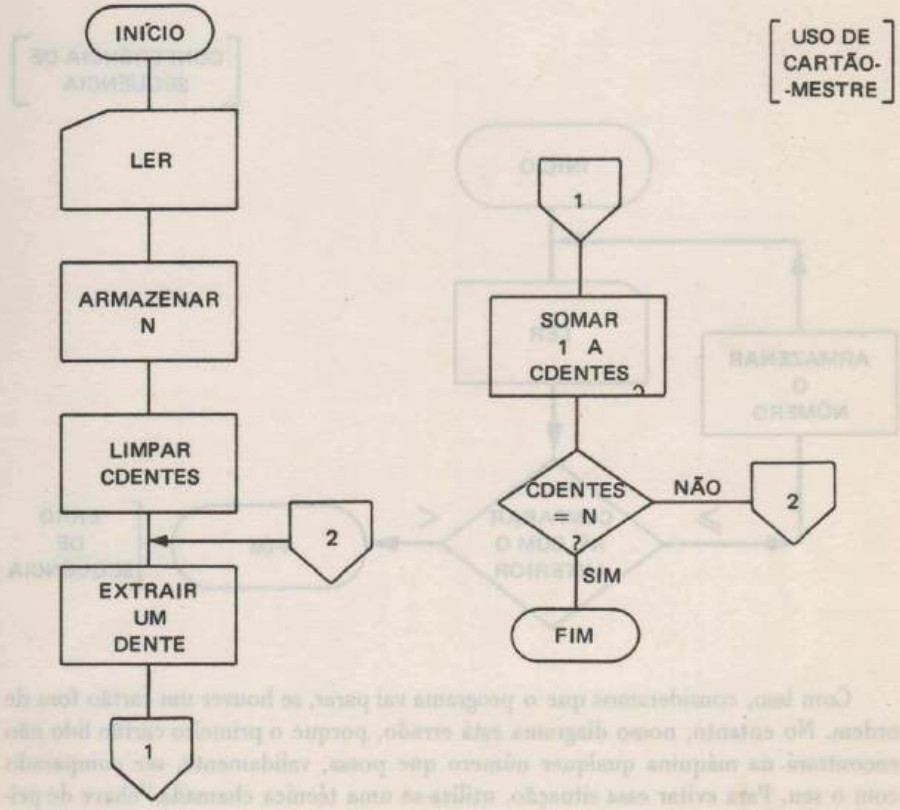
b. "Iniciação", que consiste em preparar os controles do programa (em nosso exemplo, os contadores), para a reutilização da solução, sem que os valores finais da utilização anterior interfiram.

A figura explicará melhor os conceitos:



Com essas adições ao programa, para se trabalhar em um novo paciente bastará apertar novamente o botão de INÍCIO, pois a máquina se reposiciona automaticamente.

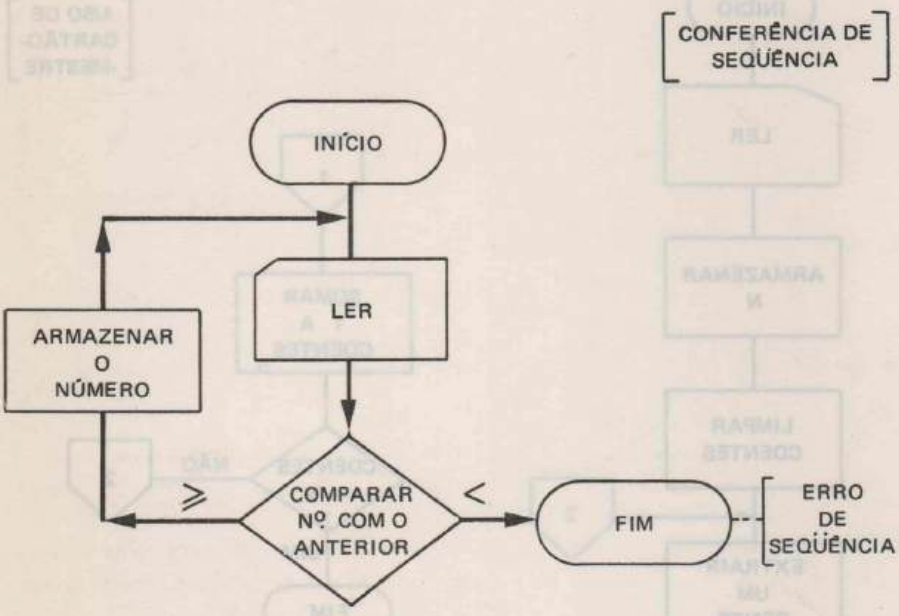
Todos estamos vendo que nossa máquina está muito limitada. Afinal, nem todos têm 32 dentes a arrancar. Essa limitação poderá ser removida, se fizermos com que o número de vezes que o programa vai ser percorrido seja definido pelo operador. Assim, podemos introduzir em nosso programa um cartão contendo as definições necessárias para a tomada de decisões. É o chamado "cartão-mestre". No caso, esse cartão conterá somente um número (N), que indicará o número de vezes que o *loop* será percorrido, e com isso a versatilidade do programa (e da máquina) ficará grandemente aumentada.



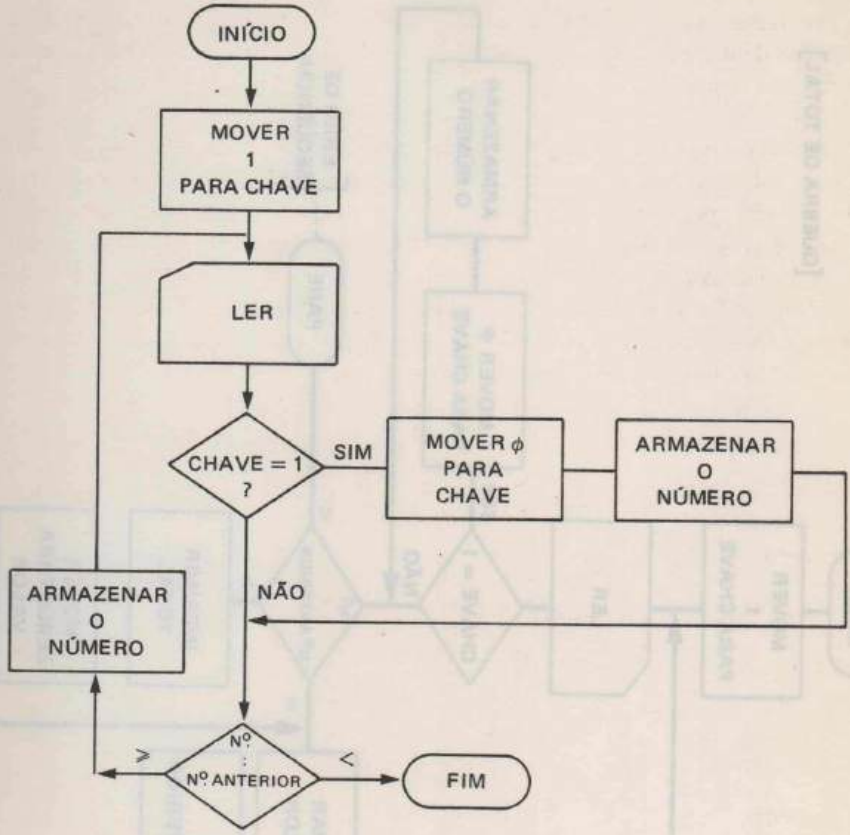
Com esse exemplo, introduzimos mais um conceito de extrema importância para o programador, que é o da escolha de nomes significativos para as áreas usadas

nos programas. Isso foi feito chamando de CDENTES a área em que fazíamos a conta dos dentes extraídos. Esse nome nos traz de imediato à lembrança a utilização dada ao campo.

À medida que avançamos em nossas técnicas, deparamos com situações cada vez mais interessantes, como é o caso da conferência de seqüência. Isso se faz necessário quando queremos verificar se, num conjunto, um determinado número está em ordem crescente (ou decrescente) em relação aos que o antecederam. Por exemplo, no conjunto 1, 4, 12, 8, 16, houve uma quebra de seqüência entre o número 12 e o número 8. Como poderia isso ser testado com um fluxograma? Suponhamos que temos cartões perfurados, cada um com um número diferente, e queiramos verificar se estão em seqüência crescente. O fluxograma que nos ocorre é:



Com isso, consideramos que o programa vai parar, se houver um cartão fora de ordem. No entanto, nosso diagrama está errado, porque o primeiro cartão lido não encontrará na máquina qualquer número que possa, validamente, ser comparado com o seu. Para evitar essa situação, utiliza-se uma técnica chamada "chave de primeira vez", que consiste em transferir para a área de armazenagem o primeiro número encontrado. Para isso, usa-se uma área auxiliar a que demos o nome de "chave".

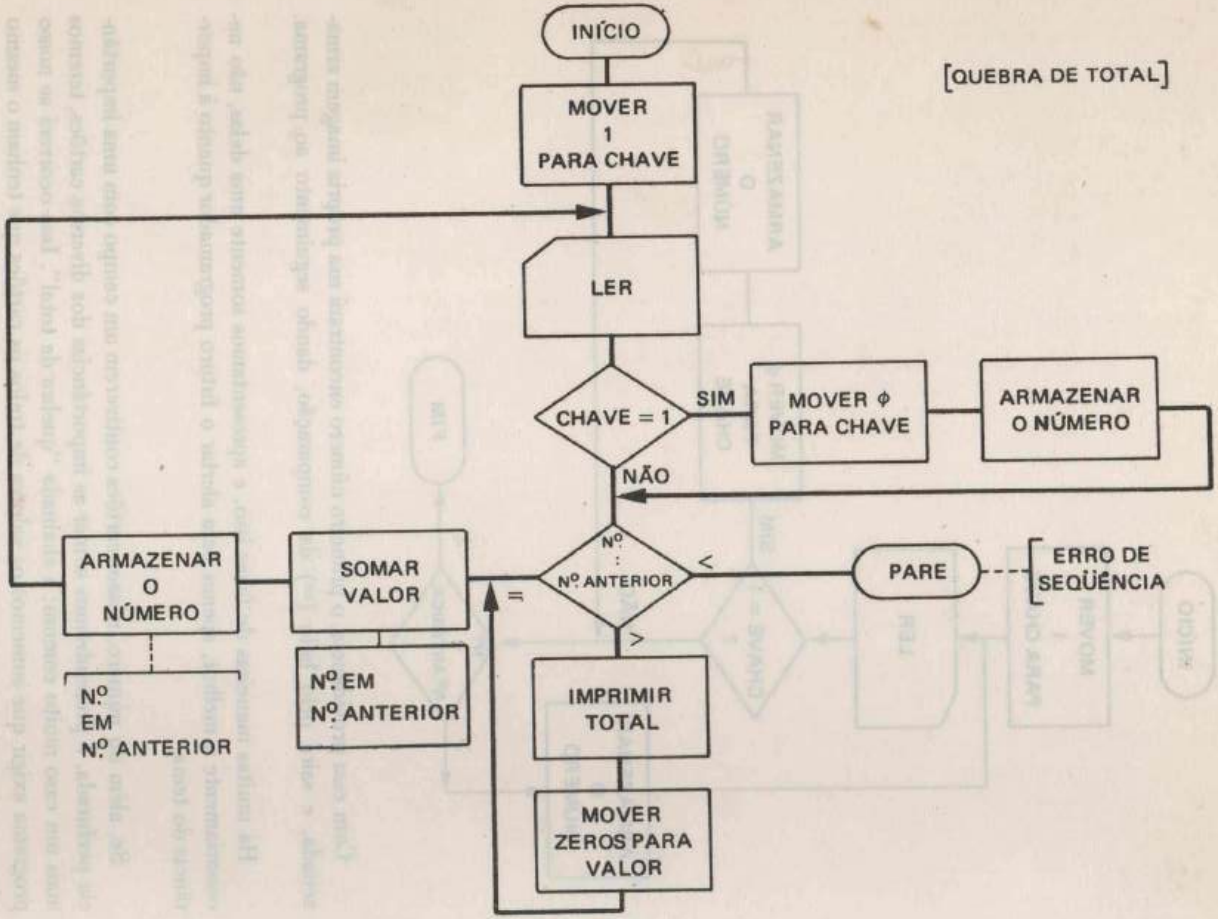


Com essa providência, o primeiro número encontrará sua própria imagem armazenada, e sairá pelo lado (=) da comparação, dando seguimento ao programa.

Há muitas maneiras de fazer isso, e apresentamos somente uma delas, não necessariamente a melhor, apenas para alertar o futuro programador quanto à importância do tema.

Se, além do número, nossos cartões contiverem um campo com uma importância perfurada, e pretendermos somar as importâncias dos diversos cartões, teremos mais um caso muito comum: a chamada "quebra de total". Isso ocorrerá se nosso programa exigir que somemos os valores de todos os cartões que tenham o mesmo número e imprimamos o total acumulado sempre que o cartão lido apresentar um número maior que o anterior. Evidentemente, um número menor significará que houve falha na seqüência.

[QUEBRA DE TOTAL]



Num caso real, o programador, via de regra, não dará ordem de parada sempre que encontrar um erro. Em vez disso, tomará alguma providência que assinale o ocorrido, e tentará prosseguir a execução do programa.

Em nosso exemplo, se, além de conferir a seqüência, tivéssemos que verificar a validade do número perfurado no cartão, uma das coisas que se poderiam fazer seria construir uma tabela com todos os números aceitáveis e, a cada cartão lido, verificar se o número do cartão existia na tabela.

Vejamos a mesma coisa em um exemplo menos complexo. Suponhamos que um gênio da eletrônica tivesse construído uma máquina que, ao lado de uma passarela, inspecionaria todas as candidatas que passassem e daria um aviso sempre que a passante estivesse enquadrada nas preferências de seu construtor. Para isso, parte de sua programação constaria de uma pesquisa de tabela.

Vejamos como construir a tabela, codificando cada tipo que as garotas poderiam apresentar:

a. Morenas	–	1	
Louras	–	2	
Ruivas	–	3	Não aceito
Amarelas	–	4	Não aceito
Negras	–	5	
b. Altas	–	1	
Medianas	–	2	Não aceito
Baixas	–	3	Não aceito
c. Atraentes	–	1	
Simpáticas	–	2	
Feias	–	3	Não aceito
Repulsivas	–	4	Não aceito

Com isso, os tipos aceitáveis teriam os códigos:

111, 112, 211, 212, 511, 512.

Qualquer outra classificação seria rejeitada pelo construtor, e, portanto, a máquina não deveria chamá-lo.

Dentro dessa definição, cada vez que uma garota passasse, a máquina elaboraria para ela uma classificação numérica, com três dígitos, e percorreria seu programa in-

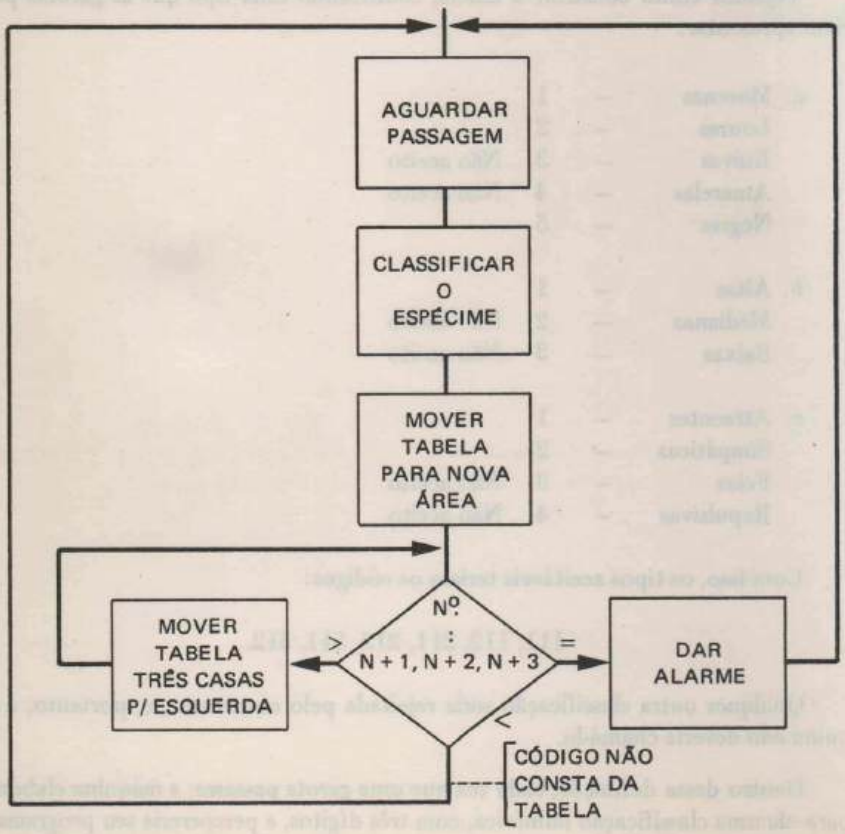
terno, consultando a tabela e dando aviso se o código atribuído fosse encontrado como constante da tabela de aceitáveis.

Como se processaria essa pesquisa? Vamos armazenar nossa tabela a partir da posição de memória de número M.

O primeiro código ocuparia as posições M, M + 1 e M + 2, o segundo, as posições M + 3, M + 4 e M + 5 etc.

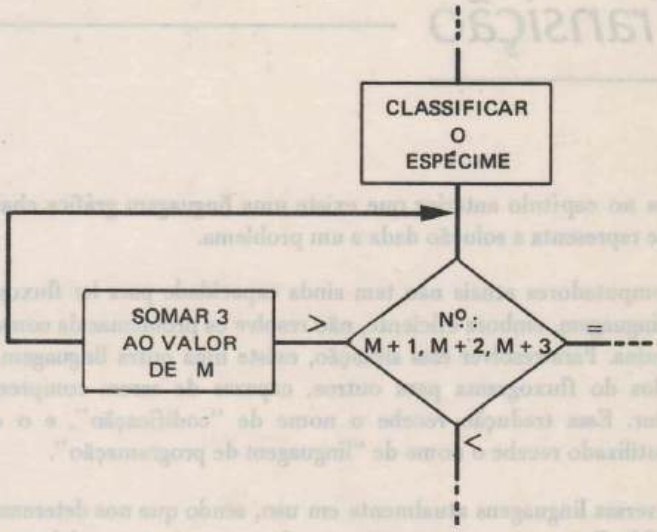
A maneira mais elementar de se pesquisar a tabela seria mandá-la para outra área de memória (começando em N, por exemplo), comparar o número do cartão com as três primeiras posições da tabela e deslocá-la de três em três posições, para futuras comparações.

O trecho de fluxograma seria:



Caso existisse a possibilidade de a máquina criar um código maior que o mais alto código aceito, bastaria colocar no fim da tabela 999, e fazer o programa testar, na saída "menor", se já tínhamos chegado ao fim.

Outro processo, até mais usado, seria avançar o endereço da parte da tabela que estivesse sendo usada na comparação, em vez de mover a própria, deixando-a na área M. O trecho de programa ficaria



Os processos que estamos abordando recebem o nome genérico de "pesquisa de tabela", e existem muitas outras maneiras de efetuar essa pesquisa.

2

A Transição

Vimos no capítulo anterior que existe uma linguagem gráfica chamada fluxograma, que representa a solução dada a um problema.

Os computadores atuais não tem ainda capacidade para ler fluxogramas, pelo que essa linguagem, embora eficiente, não resolve os problemas da comunicação homem-máquina. Para resolver essa situação, existe uma outra linguagem, que traduz os símbolos do fluxograma para outros, capazes de serem compreendidos pelo computador. Essa tradução recebe o nome de "codificação", e o conjunto de símbolos utilizado recebe o nome de "linguagem de programação".

Há diversas linguagens atualmente em uso, sendo que nos deteremos na linguagem COBOL. Para que se possa compreender sua estrutura, elaboraremos alguns exemplos de fluxogramas, de modo a ilustrar a transição fluxograma/codificação.

2.1. TÉCNICAS BÁSICAS

Seja um conjunto de cartões perfurados, referente ao serviço de faturamento de uma empresa, e ordenado segundo uma numeração atribuída a cada cliente. Ao conjunto de cartões dá-se o nome de "arquivo", e dele diz-se estar classificado por "número do cliente".

A representação da organização dada aos diversos campos que constituem cada modelo recebe o nome de *layout*.

Queremos ler o arquivo, conferir a seqüência, avisando ao operador caso apareça algum erro, sem parar o processamento, e imprimir o conteúdo de cada cartão

conforme o modelo abaixo. Ao final do serviço, deve-se imprimir uma linha que indique o valor total das faturas representadas pelos cartões. Ao impresso produzido pelo computador dá-se o nome de "relatório".

"LAYOUT" DO CARTÃO

"LAYOUT" DO RELATÓRIO

				001
NCLIENTE	NFATURA	DATA FATURA	NOME	VALOR
00001	10101	10/09/74	SHEILA	25.050,00
00300	11781	10/09/74	TBNAO	51.700,20
		TOTAL		76.750,20

A primeira linha do relatório recebe o nome de "cabeçalho", cada uma das que compõem o corpo do relatório recebe o nome de "detalhe" e a última linha recebe o nome de "total".

Vejamos então como ficaria o fluxo representativo da solução encontrada, definindo antes os nomes dados a vários campos que vão aparecer.

a. Áreas contendo dados lidos do cartão que está sendo processado:

NCLI : Número do cliente
NFAT : Número da fatura
DIA : O próprio
MÊS : O próprio

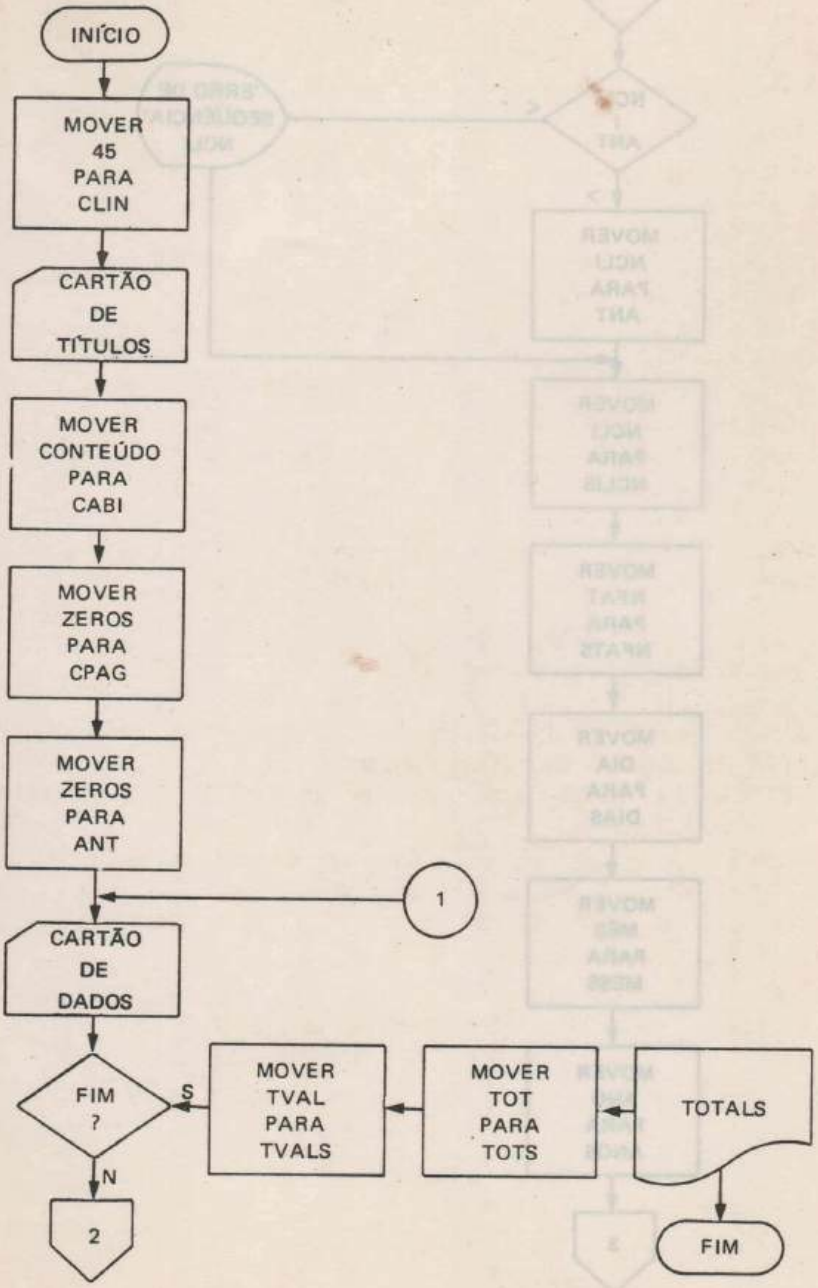
ANO : O próprio
NOME : Nome do cliente
VAL : Valor da fatura

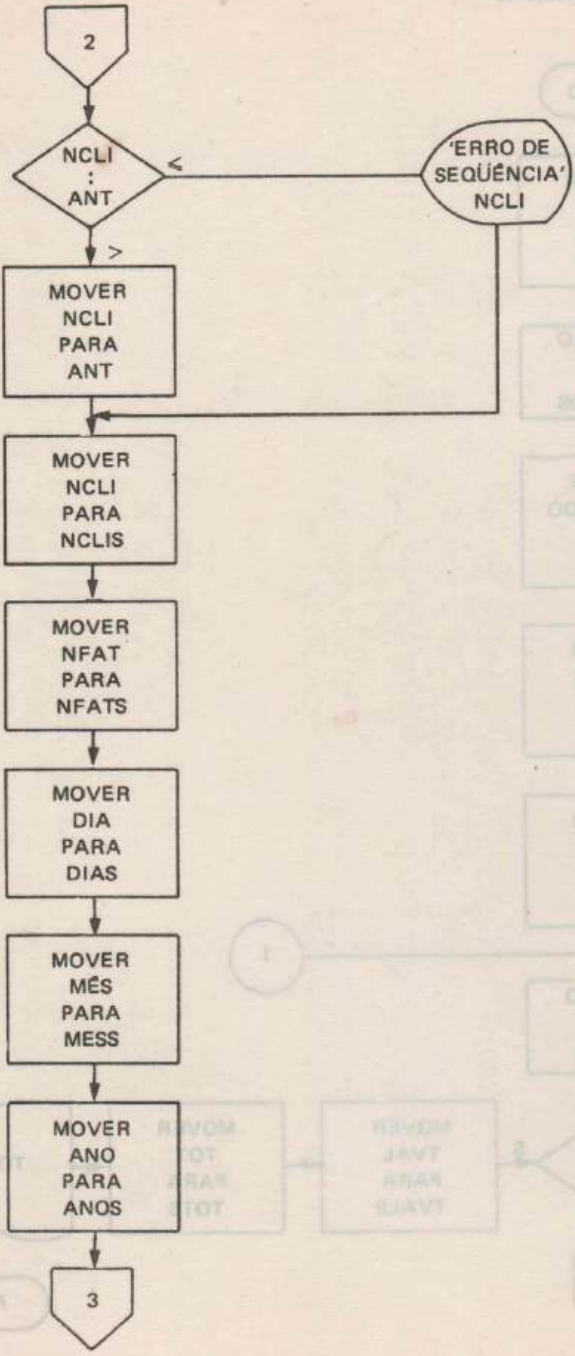
b. Áreas contendo dados que vão aparecer no relatório de saída:

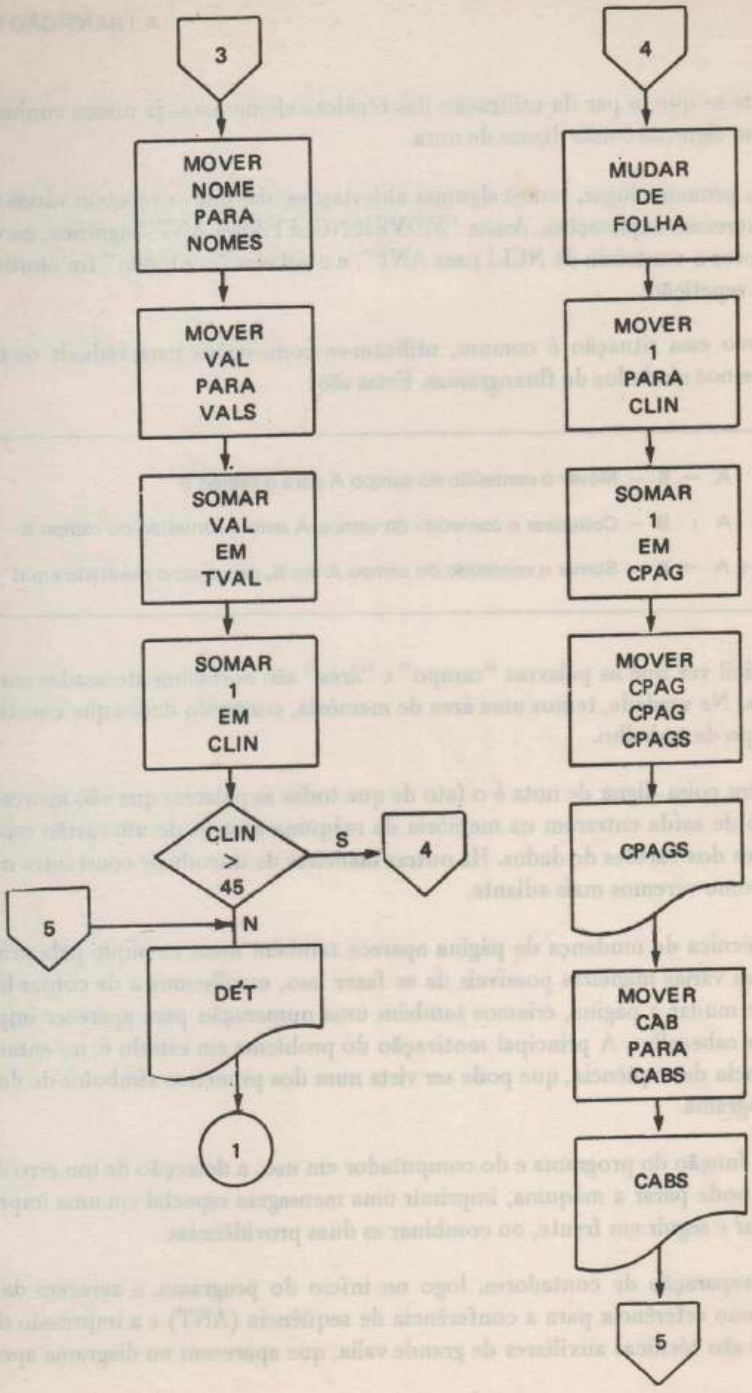
CPAGS : Contador de páginas
NCLIS : Número do cliente
NFATS : Número da fatura
DIAS : Dia
MESS : Mês
ANOS : Ano
NOMES : Nome do cliente
CABS : Conjunto de palavras que constituem a linha de cabeçalho
VALS : Valor da fatura
TVALS : Valor da soma das faturas
TOTALS: Linha completa de total, com a palavra TOTAL e o valor
TOTS : Palavra "TOTAL", dentro da linha TOTALS

c. Áreas auxiliares definidas por necessidade do programa:

ANT : Número do cliente, lido do cartão processado no ciclo anterior
TVAL : Somador para os valores das faturas
DET : Conjunto de todas as áreas que constituem uma linha-detalhe no relatório
CABI : Áreas contendo as palavras que vão aparecer nas linhas de cabeçalho e de total
TOT : Palavra TOTAL
CLIN : Contador de linhas
CPAG : Contador de páginas
CAB : Conjunto de palavras que compõem o cabeçalho


INÍCIO
MOVER 45 PARA CLIN
CARTÃO DE TÍTULOS
MOVER CONTEÚDO PARA CABI
MOVER ZEROS PARA CPAG
MOVER ZEROS PARA ANT
1
CARTÃO DE DADOS
FIM ?
S
N
MOVER TVAL PARA TVALS
MOVER TOT PARA TOTS
TOTALS
2
FIM


2
NCLI : ANT
<
'ERRO DE SEQÜÊNCIA' NCLI
>
MOVER NCLI PARA ANT
MOVER NCLI PARA NCLIS
MOVER NFAT PARA NFATS
MOVER DIA PARA DIAS
MOVER MÊS PARA MESS
MOVER ANO PARA ANOS
3


3
MOVER NOME PARA NOMES
MOVER VAL PARA VALS
SOMAR VAL EM TVAL
SOMAR 1 EM CLIN
CLIN > 45
S
4
5
N
DET
1
4
MUDAR DE FOLHA
MOVER 1 PARA CLIN
SOMAR 1 EM CPAG
MOVER CPAG CPAG CPAGS
CPAGS
MOVER CAB PARA CABS
CABS
5

Note-se que, a par da utilização das técnicas elementares já nossas conhecidas, aparecem algumas coisas dignas de nota.

Em primeiro lugar, temos algumas abreviações, das que se repetem várias vezes, e que merecem explicações. Assim "MOVER NCLI PARA ANT" significa, na verdade, "mover o conteúdo de NCLI para ANT", e a palavra "conteúdo" foi omitida em vista da repetição.

Como essa situação é comum, utilizam-se convenções para reduzir os textos inseridos nos símbolos de fluxogramas. Estas são:

A → B	Mover o conteúdo do campo A para o campo B
A : B	Comparar o conteúdo do campo A com o conteúdo do campo B
+ A → B	Somar o conteúdo do campo A em B, deixando o resultado em B

É fácil ver que as palavras "campo" e "área" são normalmente usadas como sinônimos. Na verdade, temos uma área de memória, contendo dados que constituem um campo de trabalho.

Outra coisa digna de nota é o fato de que todas as palavras que vão aparecer no relatório de saída entraram na memória da máquina através de um cartão especial, lido antes dos cartões de dados. Há outras maneiras de introduzir constantes na memória, como veremos mais adiante.

A técnica de mudança de página aparece também nesse exemplo pela primeira vez e, das várias maneiras possíveis de se fazer isso, escolhemos a de contar linhas. Além de mudar a página, criamos também uma numeração para aparecer impressa antes do cabeçalho. A principal motivação do problema em estudo é, no entanto, a conferência de seqüência, que pode ser vista num dos primeiros símbolos de decisão do fluxograma.

Em função do programa e do computador em uso, a detecção de um erro de seqüência pode parar a máquina, imprimir uma mensagem especial em uma impressora auxiliar e seguir em frente, ou combinar as duas providências.

A preparação de contadores, logo no início do programa, a zeragem da área usada como referência para a conferência de seqüência (ANT) e a impressão de cabeçalhos são técnicas auxiliares de grande valia, que aparecem no diagrama apresentado.

Recapitulando, diremos que este programa exemplificou o seguinte conceito:

- uso de contadores;
- mudança de página;
- alimentação de constantes;
- impressão de cabeçalho;
- definição de áreas;
- simbologia auxiliar;
- conferência de seqüência;
- controle de fim de arquivo.

Para tanta coisa, até que o fluxo foi bem simples! A verdade é que poderia ser ainda mais simples, se levássemos em conta que, ao se codificar um programa, se podem introduzir na memória do computador diversas mensagens que vão aparecer depois no impresso, como o cabeçalho, a palavra TOTAL etc.

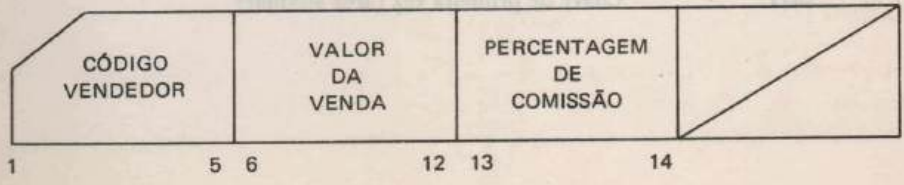
2.2. TOTAL INTERMEDIÁRIO

Vejamos, com a apresentação de mais um exemplo, como todos esses conceitos se consolidam, notando que nosso trabalho de programação está agora dividido em duas grandes tarefas:

- definição de dados;
- definição de procedimentos (fluxograma).

Seja uma massa de cartões representando as vendas feitas por diversas pessoas, classificada por código do vendedor. Queremos calcular o total dos valores das comissões e das vendas *por vendedor*; quando houver mudança ("quebra") de CÓDIGO DO VENDEDOR, imprimir os totais e prosseguir. Serão controladas as linhas de detalhe, pois só podem aparecer 40 por folha.

LAYOUT DO CARTÃO

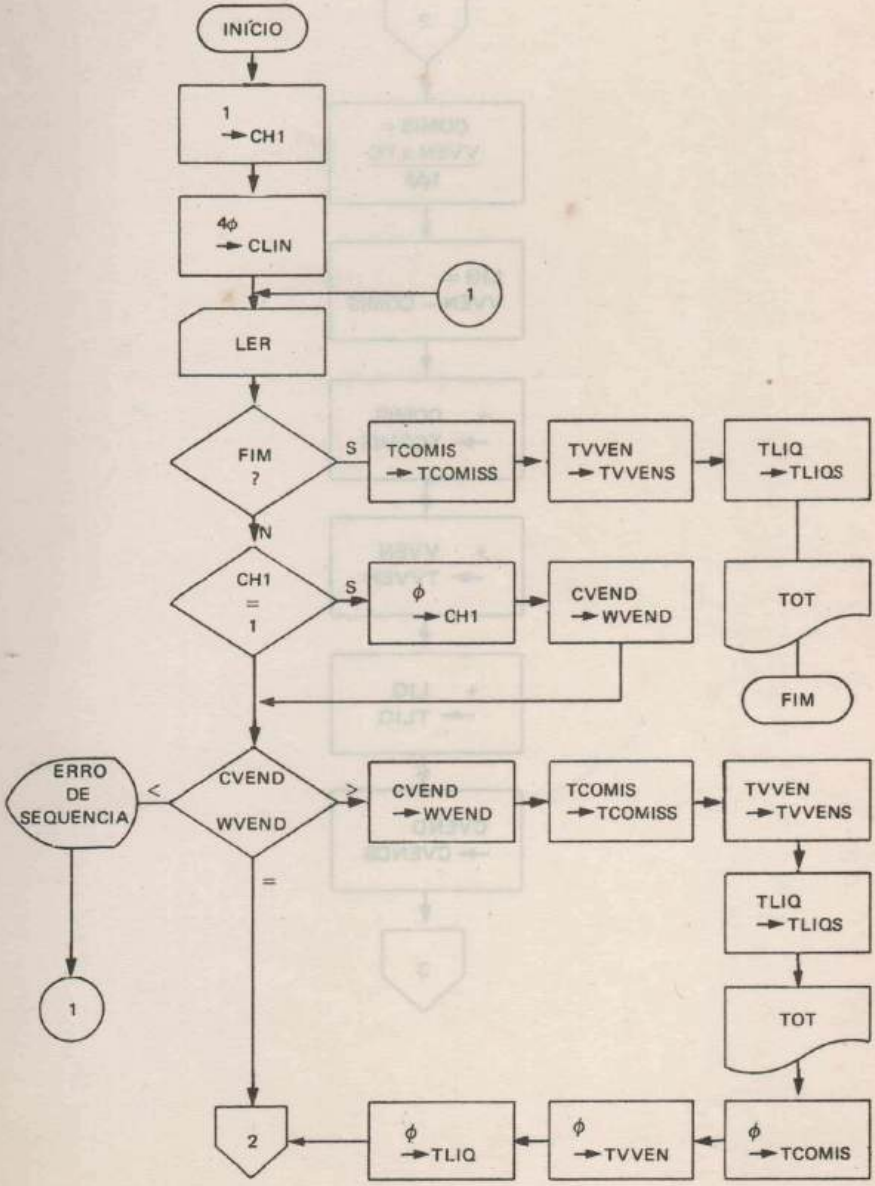


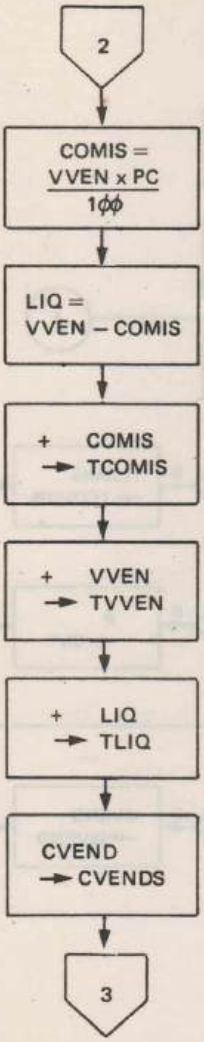
LAYOUT DO RELATÓRIO

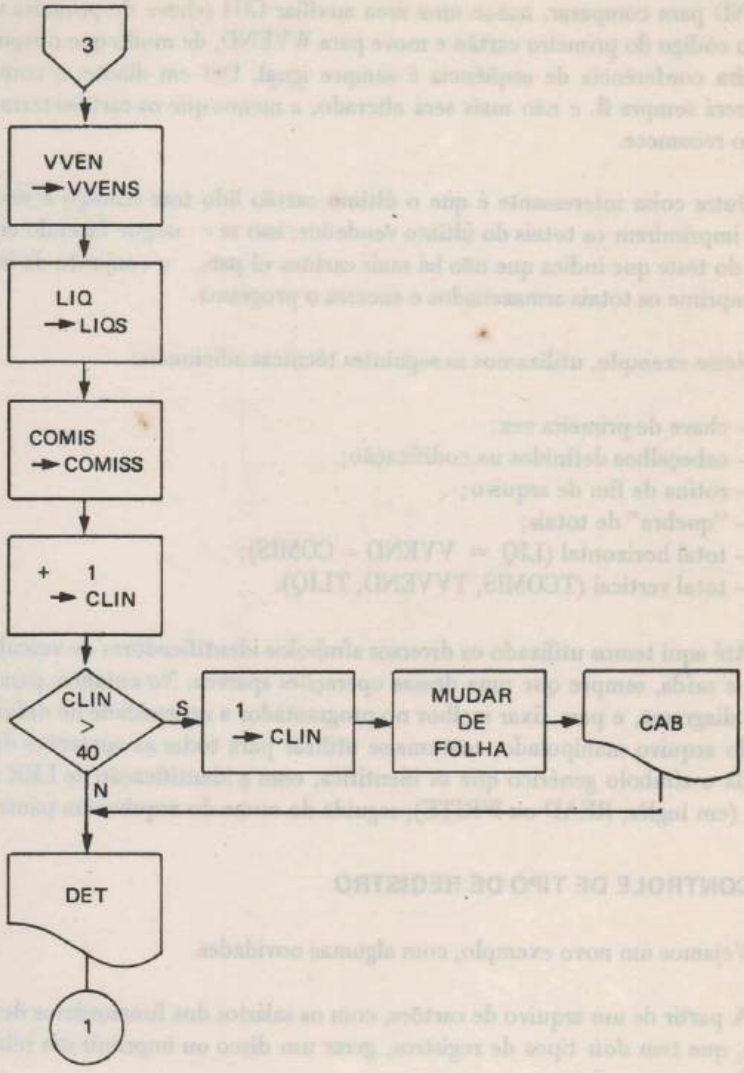
CVENDEDOR	VVENDA	VCOMISSÃO	LÍQUIDO
00001	100.000,00	20.000,00	80.000,00
00001	20.000,00	4.000,00	16.000,00
TOTAL	120.000,00	24.000,00	96.000,00
00017	10.000,00	1.000,00	9.000,00
00017	300.000,00	9.000,00	291.000,00
TOTAL	310.000,00	10.000,00	300.000,00

Definição dos dados

COMIS	:	Comissão do vendedor (calculado)
COMISS	:	Comissão do vendedor (para impressão)
CLIN	:	Contador de linhas
DET	:	Linha de detalhe
CAB	:	Linha de cabeçalho
TOT	:	Linha do total
PC	:	Percentagem (lido do cartão)
VVEN	:	Valor da venda (lido do cartão)
VVENS	:	Valor da venda (para impressão)
TCOMIS	:	Total da comissão (calculado)
TCOMISS	:	Total da comissão (para impressão)
CVEND	:	Código do vendedor (lido do cartão)
CVENDS	:	Código do vendedor (para impressão)
WVEND	:	Código do vendedor (para trabalho de comparação)
LIQ	:	Valor líquido da venda (calculado)
LIQS	:	Valor líquido da venda (para impressão)
TVVEN	:	Valor total da venda (calculado)
TVVENS	:	Valor total da venda (para impressão)
TLIQ	:	Valor total das vendas líquidas (calculado)
TLIQS	:	Valor total das vendas líquidas (para impressão)
CH1	:	Chave de primeira vez (área auxiliar)


INÍCIO
1 → CH1
4ϕ → CLIN
1
LER
FIM ?
S
TCOMIS → TCOMISS
TVVEN → TVVENS
TLIQ → TLIQS
N
CH1 = 1
S
ϕ → CH1
CVEND → WVEND
TOT
FIM
ERRO DE SEQUENCIA
<
CVEND WVEND
>
CVEND → WVEND
TCOMIS → TCOMISS
TVVEN → TVVENS
=
TLIQ → TLIQS
TOT
1
2
ϕ → TLIQ
ϕ → TVVEN
ϕ → TCOMIS


2
COMIS = VVEN x PC / 1φφ
LIQ = VVEN – COMIS
+ COMIS → TCOMIS
+ VVEN → TVVEN
+ LIQ → TLIQ
CVEND → CVENDS
3



A figura mostra o fluxograma correspondente ao procedimento que devemos seguir para resolver o problema proposto. Chamamos a atenção para o fato de que novas técnicas apareceram.

Para evitar que o código de vendedor lido do primeiro cartão precipite a impressão de totais ainda não existentes, pela ausência de um código anterior em

WVEND para comparar, usa-se uma área auxiliar CH1 (chave de primeira vez), que pega o código do primeiro cartão e move para WVEND, de modo que o resultado da primeira conferência de seqüência é sempre igual. Daí em diante, o conteúdo de CH1 será sempre Ø, e não mais será alterado, a menos que os cartões terminem e o serviço recomece.

Outra coisa interessante é que o último cartão lido traz consigo a necessidade de se imprimirem os totais do último vendedor; isso se consegue fazendo com que a saída do teste que indica que não há mais cartões vá para um conjunto de instruções que imprime os totais armazenados e encerra o programa.

Nesse exemplo, utilizamos as seguintes técnicas adicionais:

- chave de primeira vez;
- cabeçalhos definidos na codificação;
- rotina de fim de arquivo;
- "quebra" de totais;
- total horizontal (LIQ = VVEND – COMIS);
- total vertical (TCOMIS, TVVEND, TLIQ).

Até aqui temos utilizado os diversos símbolos identificadores de veículos de entrada e saída, sempre que uma dessas operações aparece. No entanto, para simplificar o diagrama, e para fixar melhor no programador a necessidade de definição precisa do arquivo manipulado, costuma-se utilizar para todas as operações de entrada e saída o símbolo genérico que as identifica, com a identificação de LER ou GRAVAR (em inglês, READ ou WRITE), seguida do nome do arquivo em pauta.

2.3. CONTROLE DE TIPO DE REGISTRO

Vejamos um novo exemplo, com algumas novidades.

A partir de um arquivo de cartões, com os salários dos funcionários de uma empresa, que tem *dois* tipos de registros, gerar um disco ou imprimir um relatório, de acordo com o seguinte:

1. Se a data do processamento for igual à data no registro (cartão tipo 1) – dar saída em disco.

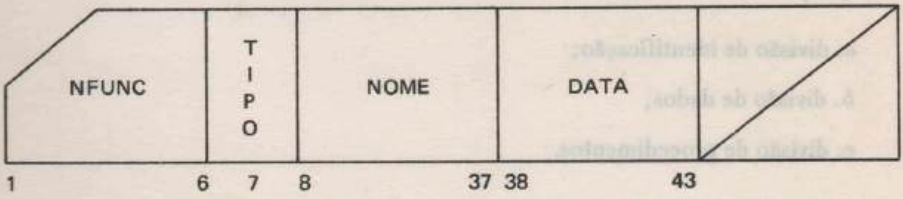
2. Se a data do processamento for diferente da data no registro (cartão tipo 1) – imprimir no relatório com contador de páginas, e 40 linhas-detalhe por página.

A data do processamento (DATAP) será introduzida pelo operador através do console. Deverão ser controladas a seqüência e a existência dos dois tipos de registros.

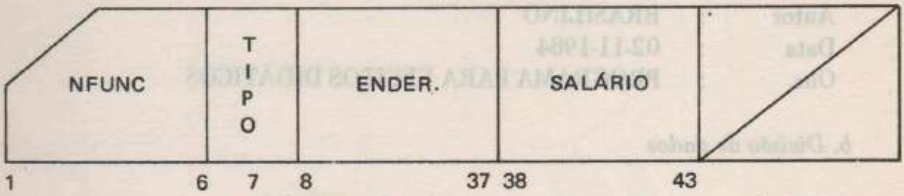
LAYOUT

INPUT (entrada)

a. CARTÃO
TIPO 1

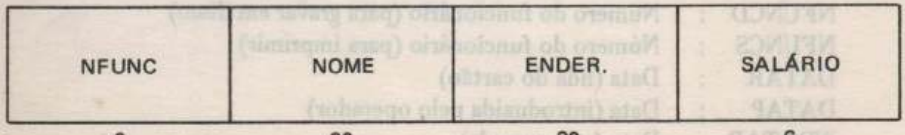


b. CARTÃO
TIPO 2



OUTPUT (saída)

a. DISCO



b. RELATÓRIO

RELATÓRIO		PÁG. 001
NFUN	NOME	SALÁRIO
001013	OSWALDO NEY	2.850,00
034710	LUIZ ALVES	8.500,00

Para completar a documentação e, conseqüentemente, para facilitar o entendimento, por terceiros, do que foi feito, vamos criar mais uma divisão em nosso trabalho, que passa a se constituir de:

a. divisão de identificação;

b. divisão de dados;

c. divisão de procedimentos.

Dentro dessa filosofia, eis como se desenvolveria nosso programa:

a. Divisão de identificação

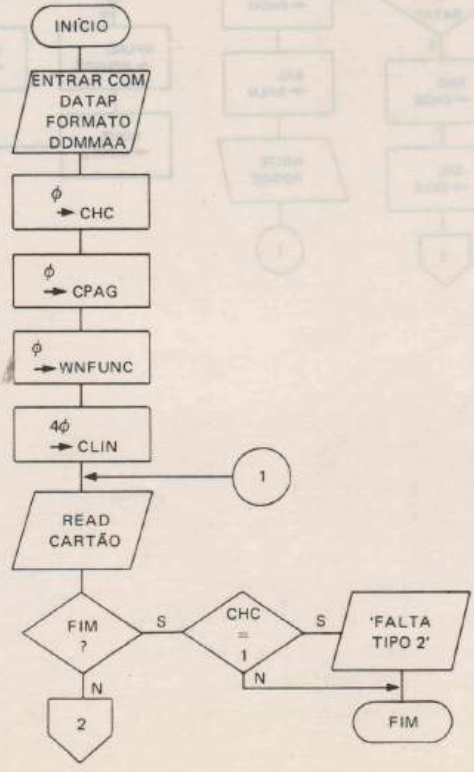
Programa	:	DOIS REGISTROS
Autor	:	BRASILINO
Data	:	02-11-1984
Obs.	:	PROGRAMA PARA EFEITOS DIDÁTICOS

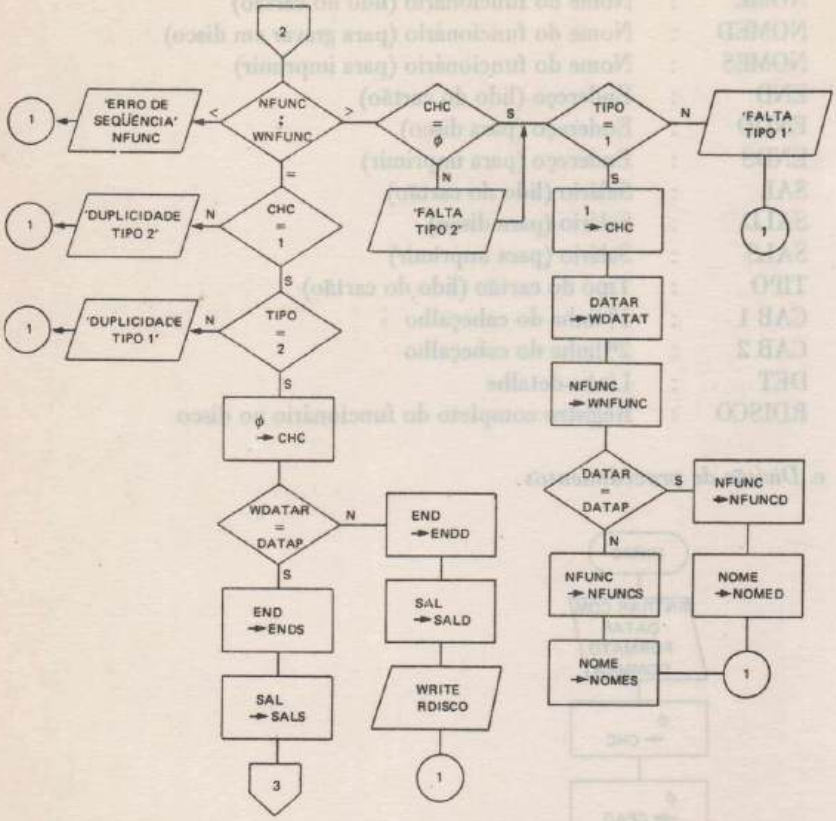
b. Divisão de dados

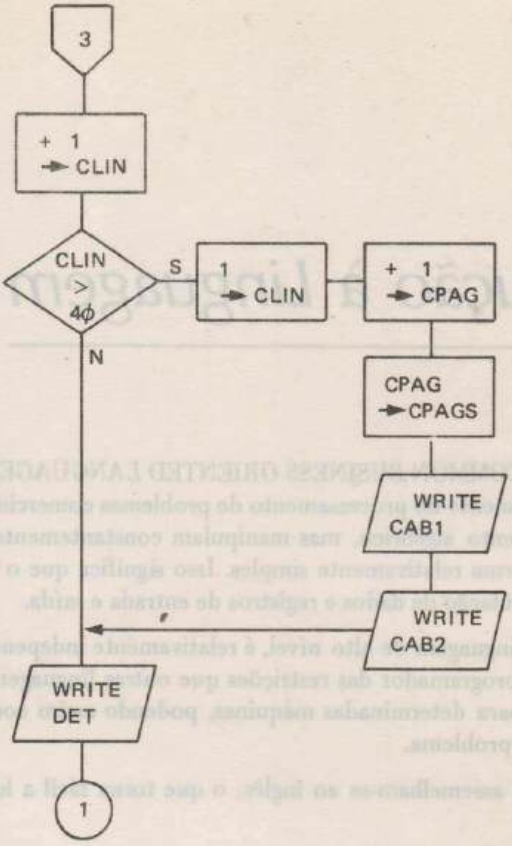
CHC	:	Chave indicativa do tipo de cartão lido
CLIN	:	Contador de linhas
CPAG	:	Contador de páginas
CPAGS	:	Contador de páginas (para impressão)
NFUNC	:	Número do funcionário (lido do cartão)
WNFUNC	:	Número do funcionário (armazenado)
NFUNCD	:	Número do funcionário (para gravar em disco)
NFUNCS	:	Número do funcionário (para imprimir)
DATAR	:	Data (lida do cartão)
DATAP	:	Data (introduzida pelo operador)
WDATAR	:	Data (armazenada)

NOME	:	Nome do funcionário (lido do cartão)
NOMED	:	Nome do funcionário (para gravar em disco)
NOMES	:	Nome do funcionário (para imprimir)
END	:	Endereço (lido do cartão)
ENDD	:	Endereço (para disco)
ENDS	:	Endereço (para imprimir)
SAL	:	Salário (lido do cartão)
SALD	:	Salário (para disco)
SALS	:	Salário (para imprimir)
TIPO	:	Tipo do cartão (lido do cartão)
CAB 1	:	1ª linha do cabeçalho
CAB 2	:	2ª linha do cabeçalho
DET	:	Linha-detalhe
RDISCO	:	Registro completo do funcionário no disco

c. Divisão de procedimentos




2
1
'ERRO DE SEQÜÊNCIA' NFUNC
NFUNC : WNFUNC
<
>
=
CHC = ϕ
S
N
TIPO = 1
N
'FALTA TIPO 1'
1
'FALTA TIPO 2'
1 → CHC
1
'DUPLICIDADE TIPO 2'
N
CHC = 1
S
DATAR → WDATAT
1
'DUPLICIDADE TIPO 1'
N
TIPO = 2
S
NFUNC → WNFUNC
ϕ → CHC
DATAR = DATAP
S
NFUNC → NFUNCD
N
WDATAR = DATAP
N
END → ENDD
NFUNC → NFUNCS
NOME → NOMED
S
END → ENDS
SAL → SALD
NOME → NOMES
1
SAL → SALS
WRITE RDISCO
3
1



No fluxograma apresentado, nota-se o aparecimento da intervenção do operador para introduzir a data atual, além da multiplicidade de mensagens que poderão aparecer no console. Cada uma dessas mensagens será definida quando da codificação do programa, de modo que isso não nos trará qualquer problema.

3

Introdução à Linguagem

O COBOL (*COMMON BUSINESS ORIENTED LANGUAGE*) é uma linguagem eficiente, especialmente no processamento de problemas comerciais. Estes envolvem pouco processamento algébrico, mas manipulam constantemente grandes arquivos de registros de forma relativamente simples. Isso significa que o COBOL enfatiza a descrição e manipulação de dados e registros de entrada e saída.

Sendo uma linguagem de alto nível, é relativamente independente de máquina, ou seja, libera o programador das restrições que outras linguagens impõem, por serem específicas para determinadas máquinas, podendo assim concentrar-se nos aspectos lógicos do problema.

As sentenças assemelham-se ao inglês, o que torna fácil a leitura e o entendimento.

Quando um programa é lido pelo computador, precisa ser traduzido em linguagem de máquina e exige, portanto, um programa tradutor (compilador).

A linguagem contém um conjunto básico de palavras reservadas e símbolos. As combinações entre eles são transformadas pelo compilador num grupo de instruções em linguagem de máquina; é o programa-objeto. Evidentemente, a elaboração de um programa terá que seguir algumas normas e limitações impostas pela própria linguagem; a apresentação do que é e do que não é aceito por um compilador constitui a própria essência de qualquer manual.

3.1. ESTRUTURA

A linguagem é estruturada na forma de um texto que tem um significado particular para o compilador, podendo o programador definir palavras que tenham signi-

ficado para si. Assim, por exemplo, se ele quer o total de salário em um determinado programa, pode escrever:

ADD SALARIO TO TOTAL-SALARIO,

onde *ADD* e *TO* são instruções padronizadas do COBOL, e *SALARIO* e *TOTAL-SALARIO* são nomes para campos definidos pelo programador.

3.1.1. Conjunto de Caracteres

A linguagem aceita uma combinação de determinadas letras e números para formar um comando. O conjunto completo consiste em 51 caracteres aceitos.

Caracteres	*Significado*
0, 1, ..., 9	dígitos
A, B, ..., Z	letras (alfabeto inglês)
	espaço
+	símbolo de adição
–	símbolo de subtração (hífen)
*	asterisco
/	barra
=	símbolo de igualdade
$	símbolo monetário
,	vírgula
;	ponto e vírgula
.	ponto decimal
(	parêntesis esquerdo
)	parêntesis direito
>	símbolo de "maior que"
<	símbolo de "menor que"
" ou '	aspas ou apóstrofo

Para formar palavras, só podem ser usados:

0, 1, ..., 9;
A, B, ..., Z;
– (hífen);

Os demais caracteres são usados para pontuação e outras tarefas especiais que veremos mais adiante.

A *palavra* é composta pela combinação desses caracteres, até o máximo de 30, e não pode começar ou terminar com hífen.

O espaço não pode aparecer entre os caracteres de uma palavra, somente como *separação* de palavras.

3.1.2. Tipos de palavras

1. Palavras reservadas (RESERVED WORDS)

São palavras da sintaxe da linguagem, não podendo ser usadas em nomes para campos definidos pelo programador. São de três tipos:

a. Palavras-chave (KEY WORDS): sua presença é obrigatória para o início ou trecho de um programa em COBOL. Neste livro, elas aparecem sublinhadas, e são o que realmente podemos chamar de "comandos", pois designam as "atitudes" que o computador deverá tomar. Em nosso exemplo:

ADD SALARIO *TO* TOTAL-SALARIO,

o verbo ADD (somar) representa o comando aritmético que executa as operações de adição, não podendo ser usado no corpo do programa para qualquer outro fim.

Quando damos esse comando *ADD*, o compilador interpreta que deve gerar instruções de máquina que executarão operações de soma (na realidade, podem ser várias instruções), além de várias atitudes que o computador deve executar para completar uma operação aritmética. Daí a vantagem das linguagens de alto nível, nas quais o programador não precisa preocupar-se com tais "atitudes", porque estas são geradas automaticamente; para isso, porém, é preciso que existam formatos específicos para cada instrução, e KEY WORDS a serem empregadas somente nas condições designadas, para que não venham a ocorrer erros na preparação do programa-objeto.

Naturalmente, tais erros são indicados pelo compilador, mas, para formarmos programas-fonte corretos, necessitamos saber todas as regras que regem a linguagem.

b. Palavras opcionais (OPTIONAL WORDS): são palavras que só aumentam o sentido da sentença criada pelo programador e que podem ser dispensadas, se necessário, pois não alteram a tradução feita pelo compilador. Neste livro elas não aparecem sublinhadas.

c. *Conectivas* (CONNECTIVES). das quais há dois tipos:

– *qualificadoras* – usadas para associar um nome com seu qualificador. As conectivas qualificadoras são *OF* e *IN*.

Exemplo:

```
MOVE DIA IN DATA-NAS TO DIANAS
MOVE DIA IN DATA-ADM TO DIAADM
```

indica que no mesmo programa existem dois campos definidos com o mesmo nome (DIA), mas que pertencem a áreas diferentes (DATA-NAS e DATA-ADM).

– *lógicas* – são usadas para compor condições. São:

AND, OR, AND NOT e OR NOT

Exemplo:

```
IF A EQUAL TO B (Se A = B)
AND C EQUAL TO D (e C = D)
    GO TO CERTO. (Vá para CERTO)
MOVE A TO B (Mova A para B)
```

significa que o programa só se desviará para a rotina CERTO se A for igual a B e C for igual a D. É necessário que as duas condições sejam satisfeitas.

2. Nomes (NAMES)

São palavras compostas pelo programador para serem usadas dentro de um programa. Estão classificadas em três tipos básicos:

a. *nome-de-dado* (data-name): é uma palavra escolhida pelo programador para identificar um item de dado contendo pelo menos um caracter alfabético. Os seguintes nomes obedecem à mesma regra de formação:

– nome-de-arquivo (FILE-NAME)

Exemplo:

ARQFUN (nome dado para o arquivo de funcionário)

– nome-de-registro (record-name)

Exemplo:

REGFUN (nome dado a um registro do arquivo de funcionário)

– nome-de-relatório (report-name)

Exemplo:

RELAT (nome dado para um relatório)

– nome-de-mnemônico (mnemonic-name)

Exemplo:

IMPRESSORA (nome que significa "unidade de impressão").

Observar que os nomes devem ser escolhidos de forma a serem entendidos tanto pelo programador que o fez como por qualquer outro. Como, na realidade, não faz diferença que se use um nome ou outro, poderíamos atribuir XKFY – 01 para o indexador ou PHXZ – 45 para o arquivo de funcionário, só que seria difícil saber o significado destes.

b. nome-de-condição (condition-name): é o nome dado para um valor específico, conjunto ou intervalo de valores, dentro daqueles que um determinado item de dado pode assumir. O item de dado é chamado "variável condicional".

Exemplo:

```
CODIGO
PROGRAMADOR          VALUE 1. (PROGRAMADOR, VALOR 1)
ANALISTA             VALUE 2. (ANALISTA, VALOR 2)
```

PROGRAMADOR E ANALISTA são nomes dados às condições que CODIGO pode assumir (1 e 2).
CODIGO é a variável condicional (item de dado).

c. nome-de-procedimento (procedure-name), também chamado nome-de-parágrafo (paragraph-name) ou nome-de-seção (section-name). Pode ser composto somente por caracteres numéricos. Dois nomes de procedimentos numéricos serão

equivalentes se, e somente se, forem compostos pelo mesmo número de dígitos e tiverem o mesmo valor. Por exemplo:

para desviar um programa para um parágrafo de nome 1, pode-se escrever: GO TO 1, onde *1* é um nome-de-parágrafo diferente de *001*, em GO TO 001.

Os nomes de bibliotecas (library-names) e os nomes de programa (program-name) seguem as mesmas regras dos nomes de procedimentos.

EXEMPLOS E REGRAS DE FORMAÇÃO DE NOMES-DE-DADOS

Nome-de-dado	PERMITIDO	MOTIVO
ALFA	SIM	1. Tem menos de 30 caracteres. Não contém caracteres especiais entre os caracteres do nome-de-dado como também no início e/ou fim. Não possui espaços entre os caracteres do nome-de-dado. O único caracter especial válido é o hífen (-), mas não pode aparecer no início e/ou fim do nome-de-dado. Este caso também não contraria tal regra. Também atende a especificação de possuir pelo menos 1 caracter alfabético; no caso, temos quatro caracteres. Não é formado somente por números.
SOMA TORIO	NÃO	2. Não podem existir brancos nos nomes-de-dados.

		Brancos servem como separadores de palavras; no caso, denotaria dois nomes-de-dados.
– INDIC	NÃO	3. Não podem existir hífens no fim e/ou início de nomes-de-dados.
A	SIM	4. Atende a regra de possuir pelo menos um caracter alfabético, além de não contrariar nenhuma regra do item 1.
A21	SIM	5. Vide item 1.
357	NÃO	6. Não podem existir nomes-de-dados formados somente por números.
SOMA-TOTAL	SIM	7. Vide item 1.
BE/TA	NÃO	8. Não pode haver caracteres especiais entre os caracteres que formam um nome-de-dado.

EXEMPLOS E REGRAS DE FORMAÇÃO DE NOMES-DE-PROCEDIMENTOS

Nome-de-procedimento	PERMITIDO	MOTIVO
PARAGRAFO-1Ø	SIM	1. Tem menos de 30 caracteres. Não contém caracteres especiais entre os caracteres do nome-de-procedimento, como também no início e/ou fim. Não possui espaços entre os caracteres do nome-de-procedimento.

O único carácter especial válido é o hífen (-), mas não pode aparecer no início e/ou fim do nome-de-procedimento. Este caso também não contraria tal regra.

DESVIO 3 NÃO

2. Não podem existir brancos entre os caracteres que formam um nome-de-procedimento.

0001 SIM

3. Ao contrário dos nomes-de-dados, os nomes-de-procedimentos podem ser formados somente por números.

Todas as regras que se aplicam a nomes-de-dados se referem ao nome-de-procedimento, excetuando esta.

Observação: Tanto para nomes-de-dados como para nomes-de-procedimentos, a regra destes possuírem no máximo 30 caracteres pode ser contrariada da seguinte forma:

Caso o programador construa uma palavra com mais de 30 caracteres, somente os 30 primeiros serão considerados pelo compilador para identificação do campo.

3.2. ESPECIFICAÇÕES PARA INTERPRETAÇÃO

No ensino da linguagem COBOL, fornecemos as regras de utilização dos comandos padronizando os formatos de cada instrução e explicando as normas que os regem.

Primeiramente, estabeleceremos tais conhecimentos e consolidaremos idéias para entrarmos em maiores detalhes, com a finalidade de reduzir a um mínimo a margem de dúvidas.

Para o entendimento dos formatos, há símbolos que devemos saber como interpretar:

— *Palavras sublinhadas*

As palavras reservadas são escritas em maiúsculas e são sublinhadas quando obrigatórias na formação dos comandos.

– *Chaves* { }

Indicam que deverá ser escolhido obrigatoriamente um dos operandos entre as chaves.

– *Colchetes* [ ]

Estará entre colchetes tudo que for opcional para o COBOL.

– *Reticências* ...

Indicam que se pode repetir qualquer número de vezes que estiver anterior às reticências.

– *Palavras não sublinhadas*

Representam as palavras opcionais, que podem ou não fazer parte da formação do comando, sem nenhum problema para seu funcionamento. Servem apenas para dar maior sentido ao comando.

3.3. UMA FAMÍLIA DE PROGRAMAS

Existe um conjunto de programas para dar resposta aos problemas propostos. Conhecê-lo-emos à medida que formos sabendo o que acontece com um determinado programa até atingirmos o ponto de colocá-lo em condição de resolver algum problema.

3.3.1. Programa-fonte

Sabemos que há várias etapas para o planejamento e a construção de um determinado programa. Durante a criação ou modificação de um sistema, chega-se ao ponto em que um ou mais programas precisam ser definidos. Essa definição consiste na proposição do problema a ser resolvido pelo computador, e aí serão informados os cálculos, as decisões e os movimentos que o programador deve comunicar à máquina.

Nesta etapa, o programador estudará o problema e construirá um fluxograma, partindo daí para a transcrição do programa em uma determinada linguagem, que

no nosso caso é COBOL (a codificação é feita em folhas desenhadas especialmente para a codificação dos comandos); esta será transportada para um meio de entrada de dados que um determinado computador é capaz de entender. Geralmente usamos os "cartões perfurados", ou os programas são digitados diretamente em terminais de vídeo conectados a um computador.

Esse programa codificado e transformado para um determinado meio de entrada de dados é o que chamamos PROGRAMA-FONTE.

3.3.2. Compilador e programa-objeto

O computador não entende palavras escritas em inglês, português ou qualquer outra língua. Não obstante, é capaz de interpretar tais palavras como comandos, desde que elas sejam traduzidas para uma linguagem de máquina. Esta é a única que ele pode entender ("por enquanto" é a linguagem binária[1]), e é por meio dela que ele é capaz de interpretar um certo caracter e/ou comando formado por uma quantidade de caracteres. Esse conjunto é projetado de tal forma que cada caracter tenha sua configuração específica e única.

Para que um determinado PROGRAMA-FONTE seja interpretado pelo computador e colocado em sua memória para execução das ordens aí contidas, ele deve ser traduzido por um outro programa chamado COMPILADOR. Este, que é um programa que também já passou pelo processo de tradução, é capaz de interpretar os comandos da linguagem na qual o programa está codificado e transformá-los em linguagem de máquina. Ao programa em linguagem de máquina traduzido pelo compilador chamamos PROGRAMA-OBJETO.

---

[1] Vasconcellos, Augusto de – *Computadores Eletrônicos Digitais*. Livros Técnicos e Científicos Editora S.A. e LTD/DATAMEC. Rio de Janeiro, 1974.

3.3.3. Supervisor

O programa-objeto não é suficiente para resolver nosso problema. No dia-a-dia de um computador há vários programas-objeto que entram e saem da memória, e cada um deve executar as mais diversas ações. Além disso, existe a possibilidade de termos mais de um programa em funcionamento na memória, dependendo da capacidade e das características do equipamento. Ora, quem deve controlar esse tráfego de informações? A resposta é o programa "supervisor". Sua principal função é fornecer rotinas comuns a todos os programas-objeto; estas são rotinas de entrada e saída de dados, coordenadas pelo programa supervisor.

Se nosso programa emite uma ordem de leitura/gravação de dados, as instruções geradas pelo compilador estabelecem a ligação entre ele e o programa de controle, proporcionando ao supervisor, assim, a condição de executar as "atitudes" de leitura/gravação dos dados.

As interrupções de programas, quando ocorre qualquer erro no programa ou na própria máquina, são controladas pelo supervisor, que se encarrega de emitir mensagens de erro e de identificação do problema para serem analisados pelo programador.

Como vemos, nosso programa não funciona "sozinho"; ele fica todo o tempo sob o controle do supervisor, que identifica as instruções do programa-objeto e proporciona condições para que elas sejam executadas.

3.4. FOLHA DE CODIFICAÇÃO

O uso da folha de codificação é uma forma padronizada de codificar programas em qualquer linguagem.

A folha de codificação COBOL é desenhada especialmente para atender às necessidades de padronização, como também para que os problemas sejam codificados de forma a serem interpretados pelo compilador. Este "espera" que determinados detalhes da linguagem venham colocados em lugares específicos; só dessa forma será capaz de interpretar o programa-fonte.

Conforme a gravidade do desrespeito às regras da codificação COBOL, o compilador acusará erros no PROGRAMA-FONTE, que poderão impedir ou não a formação de um programa-objeto apropriado.

O compilador permite que se emita uma listagem do programa-fonte, que será a imagem de como ele foi codificado.

FOLHA DE CODIFICAÇÃO COBOL

Sistema		Instruções de Perfuração	Folha 3 de
Programa 1		2	4 Identificação 73 80
Programador	Data		

5 8

SEQUÊNCIA (PAG.) 1 3 SERIAL 4 6 CONT 7 8 A 6 B 7

12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

1. Identificação do programa: nome do sistema, do programa, do programador e a data em que teve início a codificação.

Um programa terá várias folhas, naturalmente, mas basta que as informações sejam colocadas apenas na primeira.

2. As instruções de perfuração visam a indicar alguma atitude a ser tomada pela perfuradora para que nenhuma confusão seja causada.

Exemplo:

Em geral, a instrução mais usada é a do programador estabelecer a diferença entre o 0 (zero) e a letra O. Como a codificação de programas é um conjunto de palavras e números, pode haver muita confusão. Assim, na Instrução de Perfuração deve ser colocado o seguinte:

Ø = zero

O = letra.

3. Neste ponto é colocada a numeração das folhas.

Exemplo:

Se o programa tem 50 folhas de codificação e estamos codificando a folha 8, teremos "Folha 08 de 50".

NOTA: Os itens de 1 a 3 são de caracter ilustrativo, pois não fazem parte do cartão a ser perfurado.

4. Neste quadro deve-se colocar o código do programa que geralmente vem descrito na definição. Esses códigos atingem no máximo 8 caracteres, e devem ser codificados nas colunas 73 a 80 da folha de codificação COBOL.

5. Representa a seqüência dos cartões em ordem ascendente. Nas colunas de 1 a 3 temos o número da folha de codificação, e nas colunas de 4 a 6 temos o número de cada cartão.

Exemplo:

PAG	SER	
001	010	
001	020	
.	.	1ª folha
.	.	
.	.	
001	250	

002	010	2ª folha
002	020	
	.	
	.	
002	250	

A numeração dos cartões pode ser de 1 em 1, de 2 em 2 etc., mas geralmente se usa a seqüência de 10 em 10, para deixar margem para introdução futura de novos comandos.

O número de seqüência deve ser usado, apesar de opcional, porque facilita bastante a identificação de cada cartão, para o caso de ser necessário fazer alguma mudança e principalmente para evitar misturas dos comandos.

6. Margem *A*: esta área é das colunas 8 a 11; serve para início de Nomes de Divisões, Seções, Parágrafos e Procedimentos e Indicadores de Nível e Números de Níveis tais como Ø1, 77.

7. Margem *B*: esta área é das colunas 12 a 72 e serve para início ou continuação de comandos em COBOL.

8. A coluna 7 é usada para identificar cartões de comentários através de um asterisco (*); a partir daí pode ser escrito qualquer comentário a respeito dos procedimentos do programa, que serão listados para efeito de documentação, mas não interpretados pelo compilador.

Essa coluna serve também para indicar continuação; para esse fim, coloca-se um hífen (-) para demonstrar ao compilador que uma determinada linha é continuação da linha anterior, ou seja, o cartão com hífen.

3.5. ORGANIZAÇÃO

Um programa completo em COBOL é organizado em quatro divisões.

1. *De identificação* (IDENTIFICATION DIVISION). Serve para registrar datas de elaboração e compilação, nome do autor etc., ou seja, dados para documentação.

2. *De equipamentos* (ENVIRONMENT DIVISION). Define o computador usado para compilar e executar o programa, além de nomes de unidades e técnicas para entrada e saída de dados.

3. *De dados* (DATA DIVISION). Especifica nomes, características e agrupamentos dos campos de dados que o programa vai utilizar.

4. *De procedimentos* (PROCEDURE DIVISION). Define a seqüência de passos que devem ser seguidos pelo computador definido na *environment division*, para resolver o problema proposto usando os dados especificados na *data division*.

4

Divisões de Especificações

São duas divisões que deverão ser colocadas no início do programa, numa ordem definida. A primeira, a Divisão de Identificação, fornece uma série de comentários a respeito do programa, mas sua função principal é identificar o programa-fonte (programa-problema) para o programa de controle.

Todos os aspectos do processamento que dependem das características de um computador específico são definidos na segunda divisão, a de Equipamento, do que se conclui que esta será a divisão onde haverá proporcionalmente maiores alterações, caso se queira executar um mesmo programa em outro computador que não o originalmente previsto.

É importante notar que essas divisões não fazem parte da lógica do programa, servindo apenas para *especificar* certas atitudes ao computador.

4.1. IDENTIFICATION DIVISION

A Divisão de Identificação é a primeira de um programa COBOL, e dá nomes aos programas-fonte e aos programas-objeto, aceitando, também, *como comentário* o nome do programador, a data em que o programa foi escrito e/ou compilado, o que o programa faz etc. Apresenta a seguinte estrutura:

```
{IDENTIFICATION } DIVISION.
{ID DIVISION.   }

{PROGRAM-ID.      nome-do-programa}

[ AUTHOR.         [comentário] .... ]
```

[*INSTALLATION*.	[comentário] ....]
[*DATE-WRITTEN*.	[comentário] ....]
[*DATE-COMPILED*.	[comentário] ....]
[*SECURITY*.	[comentário] ....]
[*REMARKS*.	[comentário] ....]

Exemplo (referente às margens *A* e *B* da folha de codificação):

```
A  B
IDENTIFICATION DIVISION.

PROGRAM-ID.      EXEMPLO.

AUTHOR.          LAURA.

INSTALLATION.    CIS.

DATE-WRITTEN.    28/02/85.

DATE-COMPILED.   03/03/85.

SECURITY.        SEM DISPOSITIVOS DE SEGURANCA.

REMARKS.         ISTO E SO UM EXEMPLO COMPLETO
                 DA DIVISAO DE IDENTIFICACAO.
```

A Divisão de Identificação precisa começar com as palavras reservadas IDENTIFICATION (ou ID) DIVISION seguidas de um ponto e espaço, codificadas a partir da margem *A*. Depois, para identificar o programa, seu nome precisa ser sempre especificado no primeiro parágrafo, que é PROGRAM-ID. Todos os outros parágrafos são opcionais.

O *nome-do-programa* (program-name) identifica o programa-objeto para o programa de controle. É formado segundo as regras dos nomes-de-procedimento, onde os oito primeiros caracteres identificam o nome-do-programa que deve ser único dentro do programa.

Se o primeiro caracter do nome-do-programa for numérico (o sistema espera que seja alfabético), este será convertido assim:

0	(zero)	para *J*
1	a 9	para *A* a *I*, respectivamente,

e, como o sistema não admite o hífen como caracter válido para o program-name, este será convertido para o caracter 0 (zero) se estiver a partir da segunda posição até a oitava.

Exemplo:

PROG-01 será PROG001

O parágrafo *DATE-COMPILED*, se usado, implica a inserção, pelo sistema, da data considerada durante a compilação do programa, que sairá na listagem do programa-fonte. Além disso, se usado, temos:

INSTALLATION – pode ser dado o nome da empresa à qual se destina o programa;

DATE-WRITTEN – pode ser colocada a data em que o programador iniciou a codificação;

REMARKS – especifica os objetivos do programa.

4.2. ENVIRONMENT DIVISION

As funções da Divisão de Equipamentos são descrever as características do computador a ser utilizado e definir os arquivos usados no programa, podendo também descrever técnicas e controles especiais para entrada ou saída de dados.

Essa divisão é composta de duas seções: Configuração (CONFIGURATION SECTION) e de Entrada e Saída (INPUT-OUTPUT SECTION).

4.2.1. Configuration Section

A Seção de Configuração fornece informações sobre o computador e é dividida em três parágrafos:

SOURCE-COMPUTER,
OBJECT-COMPUTER,
SPECIAL-NAMES.

Tanto a Seção de Configuração como os parágrafos a ela associados são opcionais; os parágrafos SOURCE-COMPUTER e OBJECT-COMPUTER são tratados como comentários.

4.2.1.1. Source-Computer

O parágrafo SOURCE-COMPUTER descreve em qual computador o programa-fonte será compilado, e é apenas documentacional, sendo tratado como comentário pelo compilador COBOL. Apresenta o seguinte formato:

SOURCE-COMPUTER. nome-do-computador.

4.2.1.2. Object-Computer

O parágrafo OBJECT-COMPUTER descreve o computador no qual o programa-objeto será executado. Apresenta o seguinte formato:

OBJECT-COMPUTER. nome-do-computador.

4.2.1.3. Special-Names

Neste parágrafo podem-se definir técnicas especiais para a programação, como, por exemplo, a alteração da simbologia padronizada pela linguagem. Apresenta o formato:

SPECIAL-NAMES.

[nome-de-função IS nome-simbólico]

[*DECIMAL-POINT IS COMMA*].

Os nomes de função válidos são:

SYSIPT
SYSPUNCH ou SYSPCH
SYSLST
CONSOLE
Cϕ1 a C12 (Controla o canal)
CSP (Suprime espaço)
Sϕ1 a Sϕ5 (Seleciona escaninho)

Os nomes SYSIPT (leitora de cartões), SYSPUNCH (perfuradora de cartões), SYSLST (impressora) e CONSOLE podem ser associados a nomes-simbólicos (definidos pelo programador) para serem utilizados nos comandos ACCEPT e DISPLAY.

Os nomes C01 a C12 se referem às perfurações na "fita de carro", conhecidas como canais. E, por fim, os nomes CSP, S01 a S05 da leitora/perfuradora de cartões.

Decimal-Point is Comma. Esta cláusula inverte a função do ponto decimal (usado nos países anglo-saxônicos) para vírgula, nos literais numéricos e campos numéricos editados.

Exemplo:

Inglês	Português
1,000.00	1.000,00
$20,000,000.00	$20.000.000,00

O uso dessa cláusula permite ao programador definir sua edição com o formato em Português.

Exemplo:

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.  IBM-4341.

OBJECT-COMPUTER.  IBM-4341.

SPECIAL-NAMES.  C01  IS  CANAL-1
                DECIMAL-POINT IS COMMA.
```

4.2.2. Input-Output Section

Na seção de Entrada e Saída é que se define cada arquivo, dizendo-se em que unidade de entrada ou saída vão funcionar, além de técnicas especiais de utilização Ela está dividida em dois parágrafos: FILE-CONTROL; I-0-CONTROL.

4.2.2.1. File-Control

É neste parágrafo que cada arquivo é selecionado e designado a uma unidade de entrada ou saída. Aqui são usadas as seguintes cláusulas, que devem ser codificadas a partir da Margem *B*:

SELECT nome-do-arquivo *ASSIGN* TO identificação
[cláusula RESERVE]
[cláusula ACCESS]

4.2.2.2. Cláusula SELECT e ASSIGN

Usa-se a cláusula SELECT para nomear cada arquivo do programa, recebendo cada um somente um SELECT. Se, por exemplo, temos no nosso programa um arquivo em cartões e outro em relatório (impresso), cujos nomes são, respectivamente ARQUIVO-CART e ARQUIVO-REL, faremos:

```
SELECT    ARQUIVO-CART
SELECT    ARQUIVO-REL
```

Mas isso não é suficiente, pois é necessário ainda dizer a que unidade externa cada arquivo está designado, usando para isso a cláusula ASSIGN:

```
ASSIGN  TO  identificação
```

A identificação especifica a unidade em particular, a classe da unidade, a organização dos dados nessa unidade e o nome externo do arquivo.

A razão específica do comando SELECT/ASSIGN é *ligar* nossos arquivos lógicos com as unidades periféricas do computador. Quando selecionamos um determinado arquivo para uma determinada unidade, esta deve possuir características de funcionamento para nosso arquivo. Se, por exemplo, processarmos um arquivo seqüencialmente em disco, deveremos especificar na identificação que nosso arquivo estará colocado numa unidade de disco e que deverá ser processado de forma seqüencial, mas não diremos em que unidade ele estará realmente. Essa designação é feita em tempo de execução do programa, por meio de técnicas muito simples, mas a vantagem disso está no seguinte:

Se formos obrigados a estabelecer uma determinada unidade de disco *X* ou determinada unidade de fita *Y*, o que é que nos garante que no momento de execução

do programa estas estarão em condições de processamento? Poderão estar em manutenção, ou ocupadas por arquivos de outro programa, ou poderão estar defeituosas; enfim, poderão sofrer uma quantidade de problemas que as tornarão não operativas. Então o que fazemos é designar o programa para uma unidade lógica com o nome SYS, cujo nome (número) será complementado por três dígitos representando a unidade simbólica.

Esses números estão "guardados" dentro de uma tabela do programa de controle, que será capaz de interligá-los por meio de um comando para uma determinada unidade que esteja em condições operativas. A identificação é que será capaz de designar a unidade lógica, o tipo da unidade e de que forma o arquivo será processado pelo programa. Apresenta o seguinte formato:

SYSNNN-classe-unidade-organização,

onde:

NNN é um número composto por três dígitos, variando entre 000 e 240 (inclusive), representando a unidade simbólica para a qual cada arquivo é designado;

classe é um campo de duas posições que representa o tipo da unidade, sendo:

DA acesso-direto (disco magnético – acesso direto);
UT *utility* (disco ou fita magnética – acesso seqüencial);
UR *unit-record* (cartão ou relatórios impressos).

Os arquivos designados para unidades *UT* e *UR* são de organização seqüencial, e para *DA* podem ser de organização seqüencial ou direta;

unidade é um campo de quatro ou cinco dígitos que representam o número atribuído pelo fabricante a uma unidade em particular.

Para *DA* temos: 2311, 2314, 2321, 3330;
Para *UT* temos: 2400, 2311, 2314, 2321, 3420;
Para *UR* temos: 1224R, 1442P, 1404, 1403, 2501, 2502R, 2502P, 2540R, 2540P.

NOTA: *R* (READER) indica leitora, e *P* (PUNCH), perfuradora;

organização é um campo de uma posição que especifica a organização do arquivo, que é definido por:

S – para arquivos seqüenciais

A – para arquivos diretos com endereçamento real etc.

No nosso exemplo, teríamos:

```
SELECT ARQUIVO-CART    ASSIGN SYS001-UR-2540R-S.
SELECT ARQUIVO-REL     ASSIGN SYS002-UR-1403-S.
```

Isso significa que: temos dois arquivos, cujos nomes são ARQUIVO-CART e ARQUIVO-REL, que estão designados para as unidades simbólicas SYS001 e SYS002 e pertencem à classe *UR*, ou seja, unidades de cartão ou relatório; que essas unidades têm os números 2540R (que é uma leitora de cartões) e 1403 (que é uma impressora de relatórios) e que ambos os arquivos são de organização seqüencial.

4.2.2.3. Cláusula RESERVE

Com a cláusula RESERVE, o programador pode modificar o número de áreas de entrada e saída (*buffers*) alocadas pelo compilador para arquivos seqüenciais.

$$\underline{RESERVE} \left\{ \begin{matrix} 1 \\ \underline{NO} \end{matrix} \right\} \qquad \text{ALTERNATE} \left[\begin{matrix} \text{AREA} \\ \\ \text{AREAS} \end{matrix} \right]$$

Para um arquivo é necessário que haja pelo menos um *buffer*; se *omitimos* a cláusula ou usamos a opção do inteiro *1*, significa que um *buffer* adicional é assumido. Se a opção de *NO* for usada, significa que nenhum *buffer* adicional será assumido.

4.2.2.4. O que são "buffers"?

Quando definimos os registros nos programas, o compilador prepara áreas que serão usadas pelo programa de controle para guardar o conteúdo de diversos registros lidos, de onde os retira, um a um, para o processamento. Não são as instruções de leitura/gravação dos programas COBOL que executam tais procedimentos; elas indicam que o compilador deve gerar instruções de máquina que façam a leitura/gravação dos nossos arquivos. Tais atitudes são tomadas pelo programa de controle. Assim, se possuímos duas áreas alternadas para processamento, enquanto os dados de uma estão sendo computados, a outra está sendo carregada com os dados do próximo registro, e, ao término do processamento de uma área, automaticamente o

controle passa a processar os dados da área alternada. Por sua vez, a área que acabou de ser processada será carregada com dados de mais um registro, e assim por diante.

Na gravação, o funcionamento é idêntico: enquanto uma área de saída está sendo preenchida, a outra está sendo gravada. O objetivo disso é minimizar o tempo de processamento, evitando um "tempo" de espera dos dados a serem processados, para os próximos serem carregados dentro do mesmo *buffer*, no caso de nenhuma área alternada ter sido reservada. Em compensação, dependendo da máquina com que estamos lidando, do nível da linguagem COBOL que estamos utilizando e do programa que estamos projetando, podemos ter problemas de pouca memória para armazenagem do nosso programa; então sacrificamos uma parte do tempo de processamento para usar a opção *NO*, com a qual se reduzirá a área de armazenagem para o arquivo.

Em resumo, o uso dessa opção dependerá de uma necessidade específica, mas, pela experiência, geralmente ela poderá ser omitida nos arquivos seqüenciais (reservando duas áreas), sendo usada somente no caso de se necessitar de *redução de memória*.

4.2.2.5. Cláusula ACCESS

Com esta cláusula definimos o modo de acesso aos registros de um arquivo.

$$\underline{ACCESS} \quad \text{MODE} \quad \underline{IS} \quad \left\{ \begin{matrix} \underline{SEQUENTIAL} \\ \underline{RANDOM} \end{matrix} \right\}$$

Quando é especificado ACCESS IS SEQUENTIAL, significa que os registros são colocados ou obtidos seqüencialmente, e isso serve para arquivos em fita, disco, cartão ou relatório. Caso esta cláusula seja omitida, o compilador assumirá que o acesso será seqüencial.

Exemplos:

1.
```
SELECT ARQ-FITA ASSIGN TO SYS007-UT-2400-S
ACCESS MODE IS SEQUENTIAL.
```

2.
```
SELECT ARQ-FITA ASSIGN TO SYS007-UT-2400-S.
```

Nesses exemplos, ambas as descrições designam um arquivo chamado ARQ-FITA ligado à unidade lógica SYS007, cujo tipo de arquivo é UTILITY, sendo a unidade de fita magnética do tipo 2400, com o arquivo em organização seqüencial. Reservaram-se duas áreas alternadas e foi especificado que os registros serão lidos ou gravados de maneira seqüencial. Como sabemos, para arquivos seqüenciais as cláusulas ACCESS e RESERVE são opcionais.

5

Estrutura de Arquivos

Sabemos que um arquivo de dados é um conjunto de registros, e estes um conjunto de campos, ordenados segundo um processo preestabelecido.

Como existem diversos tipos de organização, haveria uma tendência a usar a de especificações mais simplificadas e, com o tempo, todas as outras seriam abandonadas. Qual delas utilizar? Isso deve ser objeto de estudo, para que a maneira particular de processamento de cada uma se encaixe e preencha as necessidades de nosso serviço.

A organização seqüencial deve ser utilizada em arquivos que tenham como meio de entrada/saída cartões, formulários, fita de papel, fita magnética e discos magnéticos (estes são os meios de entrada/saída mais comuns no país), enquanto que a organização Indexada e a Direta só podem ser utilizadas para arquivos de entrada/saída em discos magnéticos.

Por quê?

Sendo as unidades de discos magnéticos de acesso direto, isto é, o acesso a uma informação independente de sua posição, podemos "ler" diretamente um registro de nº 5.000 sem precisarmos "ler" os 4.999 registros anteriores, ao passo que a característica da organização seqüencial seria ler o 1º, 2º, 3º etc., até encontrarmos o registro de nº 5.000.

Podemos comparar o acesso direto com um disco de música, onde, se queremos ouvir a terceira faixa, pegamos o braço do toca-discos e colocamos na faixa desejada, e o acesso seqüencial com uma fita cassete, onde tal opção não existe.

Nas organizações Indexada e Direta podemos fornecer o código, e por um processamento "qualquer" temos acesso direto ao registro 5.000. A esse tipo de acesso damos o nome de acesso direto, aleatório ou RANDÔMICO.

ORGANIZAÇÃO	*ACESSO*	*CARACTERÍSTICAS*
SEQÜENCIAL	SEQÜENCIAL	1. Os registros só podem ser pesquisados um a um. Para pesquisa de um registro, é necessário que se passe pelos anteriores.
INDEXADA	SEQÜENCIAL RANDÔMICO	Idem 1 2. Os registros não precisam ser pesquisados um a um, não é necessário que se passe pelos anteriores.
DIRETA	SEQÜENCIAL RANDÔMICO	Idem 1 Idem 2

Para relembrar o significado dos termos utilizados, temos:

organização – é o método pelo qual os registros são "arrumados" dentro do arquivo para que seja possível uma busca a seus conteúdos;

acesso – é o modo como eles são "apanhados" (lidos, processados, gravados) no arquivo.

Na organização *seqüencial* os registros são arrumados de forma que só podem ser "apanhados" por um acesso seqüencial, e nas organizações Indexada e Direta os registros são arrumados de forma a permitirem que um acesso seqüencial seja aleatório.

É necessário que o arquiteto do projeto verifique se o arquivo será pesquisado sempre com muitos ou com poucos itens, como também se a freqüência de necessidade dessas informações será muito grande. Por exemplo:

a. O arquivo é freqüentemente pesquisado com poucos itens e essas informações devem receber respostas imediatas ou quase imediata. O acesso randômico a registros poderia ser utilizado com grande sucesso.

b. O arquivo não é pesquisado com freqüência, e, quando isso acontece, uma grande quantidade de registros é processada e as respostas não precisam ser enviadas imediatamente ao setor interessado.

O acesso seqüencial deveria ser utilizado poupando tempo e uma programação mais elaborada.

5.1. DATA DIVISION

É na Divisão de Dados que se descrevem e nomeiam todas as informações que serão processadas pelo programa-objeto, ou seja, aí estão a descrição de arquivos e registros, a definição de áreas de trabalho (contadores, acumuladores, linha de cabeçalho, detalhe e total etc.), os nomes dos campos e também a definição do tipo de dado (alfabético, numérico, alfanumérico etc.); enfim, preparam-se as informações que serão processadas pelo programa.

Essa divisão deve ser escrita a partir da Margem *A* com palavras reservadas DATA DIVISION seguidas de um ponto e um espaço, e se encontra dividida em seções, das quais veremos as seguintes:

FILE SECTION (Seção de Arquivo)
WORKING-STORAGE SECTION (Seção de Trabalho).

Essas seções começam a partir da Margem *A*, com o nome da seção (FILE ou WORKING-STORAGE) seguido da palavra reservada SECTION e de um ponto e um espaço. Observe-se que os nomes das seções são também palavras reservadas, mas as seções são opcionais, podendo ser omitidas quando não forem necessárias.

5.1.1. File Section

Nesta seção se faz a descrição de cada arquivo utilizado no programa, bem como de todos os seus registros. A seção apresenta o seguinte aspecto:

DATA DIVISION
FILE SECTION
FD nome-do-arquivo .... .
1 nome-do-registro ... ...

Para a descrição de *cada arquivo* é necessário usar um, e somente um, Indicador de Nível (FD), o que equivale a dizer que, para cada SELECT usado na INPUT-

OUTPUT SECTION da ENVIRONMENT DIVISION, haverá *um* Indicador de Nível correspondente.

Esses indicadores devem ser escritos na Margem *A*, seguidos de um nome de arquivo (file-name) escrito a partir da Margem *B*.

Exemplo:

```
FILE SECTION.
FD   ARQUIVO-FITA
```

Isso significa que foi dado um SELECT para um arquivo chamado ARQUIVO-FITA, precisando-se agora descrever as suas características; para tanto usamos cláusulas que se apresentam como:

```
FD  nome-do-arquivo
    BLOCK CONTAINS
    RECORD CONTAINS
    RECORDING MODE
    LABEL RECORD
    DATA RECORD
```

Observe-se que o nome-do-arquivo é o mesmo usado na cláusula SELECT.

5.1.1.1. Cláusula BLOCK

Esta cláusula especifica o tamanho de um registro físico contido no arquivo. Apresenta o seguinte formato:

BLOCK CONTAINS [inteiro-1 *TO*] inteiro-2 {CHARACTERS / *RECORDS*}

É codificada a partir da Margem *B*.

Quando a opção *RECORDS* é usada, significa que o *inteiro-2* é o fator de bloco do arquivo.

Exemplo:

Suponhamos que tenhamos um arquivo com registros de 80 posições e queiramos formar em fator de bloco 10; formaremos um registro físico de 800 caracteres; tendo com esta cláusula duas maneiras de defini-lo:

a. BLOCK CONTAINS 10 RECORDS – o que significa que o nosso arquivo está grupado de 10 em 10 registros, ou seja, num bloco temos 10 registros.

b. BLOCK CONTAINS 800 CHARACTERS – sendo utilizada a opção CHARACTERS, expressamos o tamanho do registro físico com o seu valor em caracteres.

A cláusula BLOCK CONTAINS deve ser omitida nos arquivos de cartões ou relatórios e pode ser omitida quando queremos que nosso arquivo tenha um fator de bloco 1, o que significa que o tamanho de nosso registro lógico tem o mesmo tamanho do registro físico.

Exemplo:

Suponhamos, como no caso anterior, que nosso arquivo tenha registros de 80 posições. Se verificarmos a cláusula BLOCK, o compilador assumirá que os registros são deblocados ou têm "fator de bloco 1". Caso nenhuma das opções (CHARACTERS ou RECORDS) seja especificada, o compilador assumirá como sendo CHARACTERS.

Exemplo:

```
BLOCK CONTAINS 10
```

Este exemplo descreve um registro físico com 10 caracteres.

Um detalhe importante é que, na prática, geralmente usamos a opção RECORDS, isto é, usamos o fator de bloco e não o número de caracteres de bloco.

5.1.1.2. Cláusula RECORD

A cláusula RECORD CONTAINS especifica o tamanho do registro de dados.

RECORD CONTAINS [inteiro-1 *TO*] inteiro-2 CHARACTERS

Sabemos que o tamanho de cada registro de dados é completamente definido na descrição de registros, portanto esta cláusula não é obrigatória; achamos, porém, que deve ser sempre usada para facilitar a manutenção do programa, porque o tamanho do registro de dados pode ser facilmente identificado por ela, ao invés de ter que ser calculado pelo número de posições na descrição de registros, principalmente quando trabalhamos com registros fixos.

Dessa cláusula podemos explicar o seguinte:

1. Usando-se inteiro-1 e inteiro-2, o primeiro refere-se ao registro de menor tamanho e o segundo ao registro de maior tamanho.

Exemplo:

Suponhamos que tenhamos um arquivo que contenha registro de 100 até 181 posições; é claro que podemos ter registros de 101 até 150 etc., porém definimos o tamanho do menor e do maior registros.

RECORD CONTAINS 100 TO 181 CHARACTERS

2. Usando somente inteiro-2, estaremos indicando que todos os registros de dados no arquivo têm tamanho fixo.

Exemplo:

Se tivermos um arquivo de dados com registros de tamanho de 300 posições, definiremos:

RECORD CONTAINS 300 CHARACTERS

5.1.1.3. Cláusula RECORDING

Esta cláusula determina o formato dos registros, e apresenta-se desta forma:

RECORDING MODE IS modo,

onde a letra caída pode ser *F*, *V* ou *U*, sendo o *F* usado para registros de tamanho fixo, *V* para registros de tamanho variável e *U* para registros de tamanho indefinido.

Exemplo:

RECORDING MODE IS *F*

significa que vamos "tratar" com um arquivo de tamanho fixo.

5.1.1.4. Cláusula LABEL

Esta cláusula especifica se existem "rótulos" que identifiquem o arquivo ou não.

$$\underline{LABEL}\ \begin{Bmatrix}\underline{RECORD}\\ \underline{RECORDS}\end{Bmatrix}\ \begin{Bmatrix}\text{IS}\\ \text{ARE}\end{Bmatrix}\ \begin{Bmatrix}\underline{OMITTED}\\ \underline{STANDARD}\end{Bmatrix}$$

A opção *OMITTED* significa que o arquivo não tem *label* e deve ser especificada para arquivos que pertencem às unidades tipo UR (cartões, relatórios).

A opção *STANDARD*, quando usada, especifica que existe um *label* para o arquivo, e que este se apresenta de forma padronizada, que será testada pelo sistema.

5.1.1.5. Cláusula DATA

A cláusula *DATA RECORD* atribui um nome ao registro de um arquivo.

$$\underline{DATA}\ \begin{Bmatrix}\underline{RECORD}\\ \underline{RECORDS}\end{Bmatrix}\ \begin{Bmatrix}\text{IS}\\ \text{ARE}\end{Bmatrix}\ \text{nome-registro-1}\ \left[\text{nome-registro-2}\right]\ \ldots$$

Se usarmos mais de um nome de registro, significa que teremos mais de um tipo de registro para um mesmo arquivo, o que não implica uma área de memória para cada um, mas uma mesma área com várias descrições (vários tipos de registros). Esta cláusula é tida como comentário.

Exemplos:

```
FD    ARQUIVO-CARTAO
      RECORD CONTAINS 80 CHARACTERS
      RECORDING MODE IS F
      LABEL RECORD IS OMITTED
      DATA RECORD IS REG-ARQUIVO-CARTAO.

 1    REG-ARQUIVO-CARTAO ... .

FD    ARQUIVO-RELATO
      RECORD CONTAINS 90 CHARACTERS
      RECORDING MODE IS F
      LABEL RECORD IS OMITTED
      DATA RECORDS ARE REG-REL1 REG-REL2.

 1    REG-REL1.
```

```
1       REG-REL2.
        .
        .
        .
FD      ARQUIVO-FITA
        BLOCK CONTAINS 20 RECORDS
        RECORD CONTAINS 200 CHARACTERS
        RECORDING MODE IS F
        LABEL RECORD IS STANDARD
        DATA RECORD IS REG-ARQUIVO-FITA.
        .
1       REG-ARQUIVO-FITA.
```

5.2. WORKING-STORAGE SECTION

Esta seção pode conter descrições de registros e de dados que não fazem parte dos arquivos externos, mas que servem como áreas auxiliares para o processamento interno.

Para descrever as entradas, usamos os números de nível de 01 a 49 e o número de nível especial 77. Este, quando utilizado, deve ser a primeira entrada dentro da *Working-Storage*. Em outras palavras, todos os níveis 77 vêm juntos, logo após o nome da seção.

Exemplos:

```
WORKING-STORAGE SECTION.
77    CONTADOR-PAGINA      PIC 99          VALUE Ø.
77    TOTAL-SALARIO        PIC 9(7)V99     VALUE Ø.
1     DETALHE.
      .
      .
      .
```

6

Estrutura de Dados

É fácil observar a formação seqüencial de arquivos constituídos por cartão ou relatório, pois só podemos ler cartão por cartão ou imprimir linha por linha, ou seja, não podemos ler quaisquer cartões sem que os anteriores tenham sido processados.

Registro Lógico. É o registro que contém um conjunto de campos com informações correlatas.

Registro Físico ou Bloco. É um conjunto de registros lógicos gravados de uma só vez. Suponha que tenhamos 10 registros lógicos formando um registro físico; teremos, então, um registro físico de 1.000 posições, e a este número de registros lógicos que formam um registro físico damos o nome de FATOR DE BLOCO.

Exemplo:

Tam. Reg. Lógico	Tam. Reg. Físico	FATOR DE BLOCO
100	1.000	1.000 ÷ 100 = 10
800	2.400	2.400 ÷ 800 = 3
80	1.600	1.600 ÷ 80 = 20

6.1. NÚMERO DE NÍVEL

Os números de nível são usados para estruturar um registro lógico ou áreas de trabalho, podendo ser subdivididos de 01 a 49, formando, assim, os itens elementares e/ou itens de grupo.

6.1.1. Item elementar

É o item que *não* é subdividido em outros itens e pode ser parte de um registro ou o próprio registro inteiro.

Exemplo:

Se temos um campo chamado NOMEF, fazemos:

03 NOMEF ... (estrutura do campo),

onde *03* é o número de nível, *NOMEF* é o nome do campo e ... é onde definiremos a estrutura.

6.1.2. Item de grupo

É o item que é subdividido em outros itens (elementares e/ou de grupo).

Exemplo:

```
03   DATA-ATUAL.
     05 DIA  ... (estrutura do campo).
     05 MÊS  ... (estrutura do campo).
     05 ANO  ... (estrutura do campo).
```

onde *DATA-ATUAL* é o nosso item de grupo e os campos *DIA*, *MÊS* e *ANO* são itens elementares subordinados a ele (observem que a estrutura de cada campo está definida juntamente com o próprio item elementar).

O tamanho de um item de grupo é sempre definido pelos itens elementares a ele subordinados. Considerando a subdivisão dos itens de grupo, podemos exemplificar: suponhamos um registro chamado REG-CART, onde existem os campos de NFUNC (número do funcionário), NOMEF (nome do funcionário) e DATAAD (data de admissão), visualmente assim:

REG-CART				
NFUNC	NOMEF	DATAAD		
		DIAAD	MESAD	ANOAD

Codificaremos assim:

```
01    REG-CART.
      03  NFUNC  ... (estrutura do campo).
      03  NOMEF  ... (estrutura do campo).
      03  DATAAD.

          05 DIAAD  ... (estrutura do campo).
          05 MESAD  ... (estrutura do campo).
          05 ANOAD  ... (estrutura do campo).
```

onde REG-CART e DATAAD são itens de grupo e NFUNC, NOMEF, DIAAD, MESAD e ANOAD são itens elementares. O tamanho do item REG-CART está definido pela soma dos tamanhos dos itens elementares NFUNC, NOMEF, DIAAD, MESAD, ANOAD e, por sua vez, o tamanho do item de grupo *DATAAD* está definido pela soma de seus itens elementares (DIAAD, MESAD, ANOAD).

Sempre teremos que começar uma descrição de registro ou área com um nível 1 ou 01 (para o compilador, 1 e 01 têm o mesmo valor quando se trata de número de nível); se este precisar ser subdividido, o programador deverá ter cuidado ao "quebrar" os níveis, pois, quando isso acontecer, os níveis "menores" serão subordinados ao de "maior" nível ("menor" significa um número numericamente maior que o anterior, porém menor em nível).

Exemplo:

<table>
<tr><td colspan="4">GRUPO-A</td></tr>
<tr><td rowspan="2">ITEM-B</td><td rowspan="2">ITEM-C</td><td colspan="2">GRUPO-D</td><td rowspan="2">ITEM-G</td></tr>
<tr><td>ITEM-E</td><td>ITEM-F</td></tr>
</table>

```
01  GRUPO-A.          (item de grupo)
    03 ITEM-B . . . .  (item elementar subordinado ao item GRUPO-A)
    03 ITEM-C . . . .  (item elementar subordinado ao item GRUPO-A)
    03 GRUPO-D.        (item de grupo subordinado ao item GRUPO-A)
       05 ITEM-E . . . .  (item elementar subordinado ao item GRUPO-D)
       05 ITEM-F . . . .  (item elementar subordinado ao item GRUPO-D)
    03 ITEM-G . . . .  (item elementar subordinado ao item GRUPO-A)
```

Notar que os números de nível não precisam ser, necessariamente, em seqüência unitária.

Além desses números de nível (01 a 49) existem os Números Especiais de Nível, que são: *66*, *77* e *88*.

66 – Reagrupa os itens elementares ou de grupo, dando-lhes um novo nome e uma nova estrutura.

77 – Define os itens elementares independentes, que não são subdivisão de nenhum item e não podem ser subdivididos.

88 – Usado para associar nomes de condições a valores predeterminados de um certo campo.

Os números de nível 01 e 77 devem começar na Margem *A* seguidos do nome de dados a eles associados, sendo os demais escritos a partir da Margem *B*.

6.2. IDENTIFICAÇÃO E ESTRUTURA DE CAMPOS

Para se descrever um campo, é necessário usar um número de nível, seguido de um nome de dado e de uma série de cláusulas independentes. Estas são usadas de acordo com as necessidades do programador, e podem ser escritas em qualquer ordem, com uma exceção: a cláusula REDEFINES.

O formato geral para a descrição de um registro (ou de uma área ou de um campo) é composto de identificação e estrutura do campo:

Identificação

01-49	{ nome-de-dado FILLER }

Estrutura

cláusula	PICTURE
cláusula	VALUE
cláusula	BLANK WHEN ZERO
cláusula	JUSTIFIED
cláusula	REDEFINES
cláusula	USAGE

6.2.1. Identificação do campo

01-49 é o número escolhido neste intervalo (inclusive).

A opção de um nome-de-dado significa que deve ser atribuído um nome ao campo para sua referência.

Exemplo:

Um campo de salário líquido, a que daremos o nome de SALLQ, será codificado assim:

03 SALLQ ... (estrutura do campo).

onde *03* é o número de nível e SALLQ é o nome-de-dado.

Quando, dentro de um registro, temos uma área que não será referenciada por nenhum comando da divisão de procedimentos, podemos defini-la com a palavra FILLER.

Exemplo:

03 FILLER ... (estrutura do campo).

6.2.2. Estrutura do campo

É nesta parte que definiremos o tamanho do campo (número de posições de memória que irá ocupar), o tipo, o formato interno (binário, compactado) e valores iniciais que o campo pode assumir.

6.3. CLÁUSULA PICTURE

Para caracterizar e dar o tamanho de um campo usamos esta cláusula, que apresenta o seguinte formato:

$$\left\{\begin{array}{l}\underline{PICTURE} \\ \\ \underline{PIC}\end{array}\right\} \text{ IS formato}$$

O *formato* de uma PICTURE especifica o tamanho e o tipo de dado, que podem ser: alfabético, alfanumérico, numérico e editado.

Não se deixam espaços no conjunto de caracteres que definem o "formato" de uma cláusula PICTURE, e este conjunto não pode exceder de 30 caracteres.

6.3.1. Formato alfabético

É usado para especificar campos que contenham somente caracteres de *A* até *Z* (alfabeto inglês) e espaços. É representado pelo conjunto de caracteres:

A ( ) número-inteiro

Suponhamos um campo de sigla de Estado, ao qual daremos o nome *SIGLA*, cujo tamanho seja de duas posições e pertença à categoria alfabética; teremos:

```
03 SIGLA PICTURE AA.
```

onde o caracter *A* define a categoria do campo e o número de caracteres *A* define o tamanho do campo. Também poderia ser definido como:

```
03 SIGLA PICTURE A (2).
```

onde o tamanho do campo é definido pelo *número inteiro* entre parênteses.

Na prática, e por comodidade, quando o tamanho do campo ultrapassa três posições, usamos a segunda descrição.

6.3.2. Formato alfanumérico

Este é utilizado para especificar campos que contenham qualquer caracter do conjunto EBCDIC. É representado pelo conjunto de caracteres:

X ( ) número-inteiro

Suponhamos um campo de nome de funcionário, ao qual chamaremos NOMEF, que tenha 18 posições e cuja categoria seja alfanumérica.

```
03 NOMEF PICTURE XXXXXXXXXXXXXXXXXX.
```

onde o caracter *X* define a categoria do campo e o número de caracteres *X* define o tamanho deste, podendo também ser definido como:

```
03 NOMEF PICTURE X(18).
```

onde o tamanho do campo é definido pelo *número inteiro* entre parênteses.

6.3.3. Formato numérico

Este formato é usado para especificar campos que contenham caracteres de 0 (zero) até 9, podendo conter um sinal operacional que denota se ele é positivo ou negativo. É apresentado pelo conjunto de caracteres:

```
9  V  P  S  (  )  número-inteiro
```

O conteúdo de um item numérico não pode exceder 18 dígitos.

Suponhamos um campo chamado NUMERO, cujo tamanho seja de 4 posições e pertença à categoria numérica; faremos:

```
03 NUMERO PICTURE 9999.
```

onde o caracter 9 define a categoria do campo e o número de caracteres 9 define o tamanho destes que também poderia ser definido como:

```
03 NUMERO PICTURE 9 (4).
```

Esse campo NUMERO poderá assumir os valores de 0000 (zero) até 9999 (nove mil, novecentos e noventa e nove).

O caracter *V* indica a localização do ponto (vírgula) decimal, podendo aparecer somente uma vez. *Não* representa uma posição de memória, sendo só uma indicação.

Exemplos:

S99	pode assumir valores de – 99 até + 99
99	pode assumir valores de 00 até 99

S99V99	pode assumir valores de – 99,99 até + 99,99
99V99	pode assumir valores de 00,00 até 99,99
S9 (3) V9 (5)	pode assumir valores de – 999,99999 até + 999,99999
9 (3) V9 (5)	pode assumir valores de 000,00000 até 999,99999
SV999	pode assumir valores de – ,999 até + ,999
V99	pode assumir valores de ,00 até ,99
99V	pode assumir valores de 00 até 99

(neste caso, o uso da caracter *V* é redundante).

6.3.4. Formato editado

Os itens de edição são grupos de caracteres que "criam" formatos para relatórios (impressos) de saída, podendo ser alfanuméricos-editados ou numéricos-editados.

6.3.4.1. Alfanumérico-editado

Este formato é usado para causar a inserção de *zeros* ou *brancos* entre os caracteres que compõem um campo alfanumérico ou alfabético (alfabético-editado pode perfeitamente ser incluído nesta classe). É representado pelo conjunto de caracteres:

A X 9 *B* *0* ( ) número-inteiro

Aqui só será definida a função dos símbolos *B* e *0*, pois os outros já foram definidos anteriormente: cada caracter *B* representa um espaço em branco que será inserido; cada caracter *0* (ZERO) representa um caracter zero que será inserido.

Exemplos:

	CONTEÚDO DO CAMPO	RESULTADO EDITADO
04 EXEMPLO1 PIC 000A (10).	TANAJURA	000TANAJURA
05 EXEMPLO2 PIC XXBXXBXX.	120374	12 03 74
04 EXEMPLO3 PIC X (3) BXBX (4).	BOIEVACA	BOI E VACA
03 EXEMPLO4 PIC A (5) BBB.	LEITO	LEITO
04 EXEMPLO5 PIC AABAABAA.	BARATA	BA RA TA

6.3.4.2. Numérico-Editado

Este formato é usado para causar a inserção ou supressão de caracteres e é representado pelo seguinte conjunto:

Z * +CR DB $, . B P V 0 9 ( ) número-inteiro

O caracter *Z* significa a supressão de zeros não significativos, devendo aparecer à esquerda do caracter 9, e a supressão dos zeros cessará logo que seja encontrado o primeiro caracter significativo, ou quando atingir o dígito correspondente ao *9* da PICTURE.

Exemplo:

	CONTEUDO	RESULTADO
03 NUMERO1 PIC ZZ9.	017	17
03 NUMERO2 PIC ZZZ.	001	1
03 NUMERO3 PIC Z (5).	00000	
03 NUMERO4 PIC ZZZ9.	0000	0

A utilização do caracter , ou . resulta na inserção destes entre os caracteres da PICTURE.

O caracter * é usado, normalmente, para proteção ou segurança do campo de saída, ou seja, para que não haja alterações ilícitas num determinado valor. Substitui cada zero não significativo à esquerda do número por asteriscos.

Exemplo:

Se num campo de conta a receber, que chamaremos de CREB, quiséssemos ter certeza de que nenhum número seria incluído à esquerda (o que aumentaria o valor de quem recebesse), faríamos:

Ø3 CREB PICTURE *****, **.

Considerando que o valor do campo fosse 500,00, teríamos como saída:

**500,00

O caracter $ é usado para a presença do sinal de cruzeiros; vejamos:

```
Ø3 QUANTIA PICTURE $ZZZ.999,99.
Conteúdo = 259906
Edição = $ 2.599,06
Ø3 QUANTIA PICTURE $$$.$$$,$$.
Conteúdo = 2305
Edição = $23,05
```

Uma posição de $ deve ser deixada para que, mesmo que o campo atinja seu maior valor, seja editado um caracter $.

Os caracteres + e − são usados para identificar um campo positivo ou negativo, sendo impresso respectivamente com o sinal interno, o mesmo acontecendo com *CR* (crédito) e *DB* (débito), que aparecem à extrema direita do número.

Se usarmos CR e o número for negativo, sairá impresso CR; caso o número seja positivo, nada será impresso.

Exemplos:

VALOR DO CAMPO	PICTURE	VALOR EDITADO
12345	99.999	12.345
00010	ZZ.ZZZ	10
00010	ZZZ,ZZ	,10
+ 37000	999,99DB	370,00
− 37000	999,99CR	370,00CR
00500	***,99	**5,00
000	***	***
000	ZZZ	
1313	ZZB99	13 13
1700	$99,99	$17,00
+ 35	99 +	35 +
− 35	99 +	35 −
+ 35	99 −	35

6.4. CLÁUSULA USAGE

Vimos que a cláusula PICTURE especifica o número de caracteres do dado; fica a cargo da cláusula USAGE especificar o formato com que o dado deverá ser armazenado na memória.

A razão da existência de várias formas de armazenagem de dados é que existem funções específicas para cada uma delas.

O formato da cláusula é

$$\left[\underline{USAGE}\ \text{IS}\right] \quad \left(\begin{array}{l} \underline{DISPLAY} \\ \left\{ \begin{array}{l} \underline{COMPUTATIONAL\text{-}3} \\ \underline{COMP\text{-}3} \end{array} \right\} \\ \left\{ \begin{array}{l} \underline{COMPUTATIONAL} \\ \underline{COMP} \end{array} \right\} \end{array} \right)$$

6.4.1. Opção display

A opção DISPLAY especifica o formato de dados onde os caracteres são armazenados um em cada *byte*, e pode ser aplicada em itens alfabéticos, alfanuméricos, numéricos e editados (alfanuméricos ou numéricos).

No caso desta cláusula ser omitida, será sempre assumida a opção DISPLAY (na prática, sempre a omitimos, utilizando esta opção).

Exemplo:

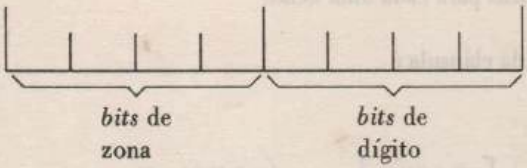
Suponha que tenhamos um campo com a seguinte definição:

```
Ø3 CAMPOA  PICTURE  XXXX  DISPLAY.
ou
Ø3 CAMPOA  PICTURE  XXXX.
```

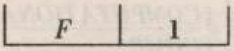
Neste caso, especificaremos 4 caracteres e precisaremos de 4 *bytes* para sua armazenagem. Na opção DISPLAY, cada *byte* é dividido em 2 partes.

Os 4 *bytes* da esquerda são considerados os *bits* de zona e os 4 *bits* da direita são considerados os *bits* que especificam a parte de dígito. Todos os caracteres no formato DISPLAY (também chamados zonado ou decimal-externo) são representados por uma parte de zona e outra de dígito, e é a parte de zona que os diferencia entre si, em combinação com a parte de dígito, que varia de Ø a 9.

Representamos um *byte* da seguinte forma:

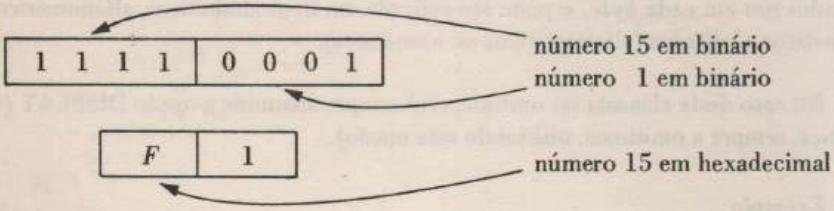


e colocamos como seu conteúdo o valor 1, obtendo o seguinte:



zona representativa dos caracteres numéricos; todos os números têm zona *F*.

Na verdade, sabemos que essa representação está em notação hexadecimal; como todos os caracteres são combinações binárias, a sua configuração verdadeira seria:

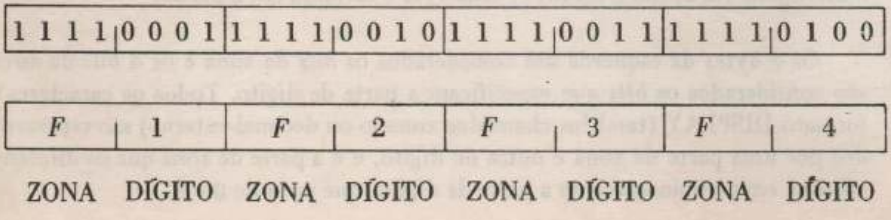


porém é mais cômodo fazer a notação hexadecimal, razão pela qual a usaremos.

Voltando ao exemplo do CAMPOA, suponha que tivéssemos como conteúdo os seguintes valores:

a) 1234

Sua representação interna seria:



b) ACKL

1 1 0 0	0 0 0 1	1 1 0 0	0 0 1 1	1 1 0 1	0 0 1 0	1 1 0 1	0 0 1 1
C	1	*C*	3	*D*	2	*D*	3
ZONA	DÍGITO	ZONA	DÍGITO	ZONA	DÍGITO	ZONA	DÍGITO

Por conveniência, explicamos a configuração interna dos itens alfanuméricos (o mesmo seria aplicado aos itens alfabéticos), porém, para os itens numéricos, a mesma teoria se aplica, mas precisamos de uma particularidade, que é a posição de sinal dos itens numéricos.

Suponha que precisemos de um campo com a seguinte definição:

```
    Ø2 CAMPO B  PICTURE  9 (3)  DISPLAY.
ou  Ø2 CAMPO B  PICTURE  999.
```

Nos itens numéricos, a posição de sinal fica identificada no *byte* mais à direita do número.

Exemplo:

a) Temos como conteúdo o valor 793

F	7	F	9	F	3

este é o *byte* onde sempre se identifica o sinal operacional do número. A zona desse *byte* especifica se o número é positivo, negativo ou está com valor absoluto. As zonas respectivas são as seguintes:

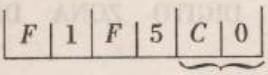
caso a zona seja *C*, o número será positivo;
caso a zona seja *D*, o número será negativo;
caso a zona seja *F*, o número estará com valor absoluto (módulo) e será considerado positivo.

No caso, o numero 793 está em valor absoluto.

b) No caso de termos a seguinte definição:

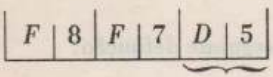
```
05 CAMPO C   PICTURE   S999   DISPLAY.
05 CAMPO C   PICTURE   S999.
```

Se tivermos o valor + 150 ou 150,



zona de sinal especificando que o número é positivo.

Se tivermos o valor – 875,



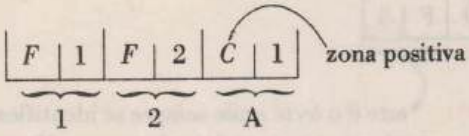
zona de sinal especificando que o número é negativo.

Sabemos que, quando especificamos a PICTURE sem *S*, é providenciado o valor absoluto, e talvez o leitor já tenha observado que, quando especificamos o *S* no último *byte*, as configurações representam uma letra nas seguintes condições:

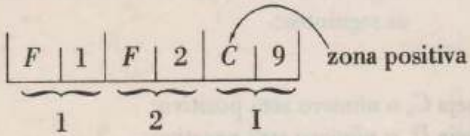
letras de *A* até *I* – valores positivos;
letras de *J* até *R* – valores negativos.

Exemplos:

a) + 121



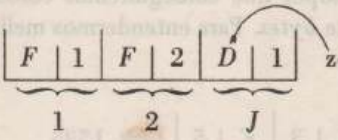
+ 129



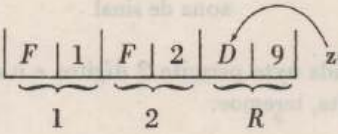
A razão disso está no fato de que as letras de *A* até *I* possuem zona *C*, e a maneira de diferenciar os caracteres é a combinação zona/dígito; os "arquivos" do computador estabeleceram, naturalmente, que esta seria a posição de identificação

de sinal, fazendo com que, no final de cada número sinalizado, tivéssemos uma letra de *A* até *R*.

b) – 121



– 129



6.4.2. Opção computational-3

A opção COMPUTATIONAL-3 especifica o formato de dados onde, em cada *byte*, podem ser armazenados até 2 dígitos. Só pode ser aplicada a itens numéricos. É de grande importância porque, com ela, podemos lançar mão de técnicas de economia de memória e tempo de execução.

Caso seja utilizada, esta opção deve ser especificada na definição do campo.

Na prática, quando utilizamos a opção COMPUTATIONAL-3 (também chamada decimal-interno), dizemos que o dado está COMPACTADO. Sua configuração interna é diferente da opção DISPLAY no sentido de que nela podemos definir dados com menos memória; além disso, é nesta configuração que se farão todas as operações aritméticas internas e os testes de comparação de todos os dados numéricos.

Aqui não existem *bits* de zona; cada *byte* é capaz de armazenar 2 dígitos e a posição de sinal fica identificada também no *byte* à direita, porém o sinal fica nos 4 *bits* mais à direita do *byte* da extrema direita.

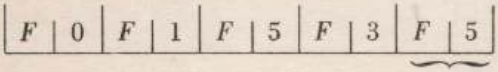
As configurações de *bits* de sinal são as mesmas que as da opção DISPLAY, ou seja, *C* = positivo, *D* = negativo, *F* = absoluto.

Exemplos:

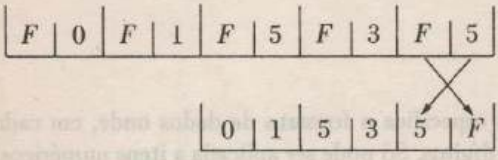
a) Temos o campo:

ϕ5 CAMPO-*D* PICTURE 9 (5) COMP-3.

Na opção DISPLAY, cada caracter ocupa 1 *byte*; nesta opção, cada 2 dígitos ocupam um *byte*; logo é de se supor que conseguiremos colocar os 5 dígitos do CAMPO-*D* num menor número de *bytes*. Para entendermos melhor, a configuração DISPLAY do CAMPO-*D* seria:



Como, na opção COMP-3, cada *byte* permite 2 dígitos e o sinal fica nos 4 *bits* mais à direita do *byte* mais à direita, teremos:



Observe bem as diferenças:

a) as zonas foram retiradas;

b) armazenamos 2 dígitos em cada *byte*;

c) usamos 3 *bytes* para armazenagem de campo de 5 dígitos;

d) houve uma troca na posição de sinal, pois na opção DISPLAY a posição de sinal fica nos 4 *bits* mais à *esquerda* do *byte* mais à *direita*.

6.4.3. Opção computational

A opção COMPUTATIONAL é usada para armazenar dados em valor binário. Causa, a princípio, alguma confusão. É claro que todas as opções são construídas com configurações binárias, mas nesta o dado é armazenado com seu valor em *BASE BINÁRIA* propriamente dita, e o número de *bytes* gastos pelo campo depende exclusivamente da PICTURE. Vejamos:

se tivermos de 1 a 4 dígitos, gastaremos sempre 2 *bytes* (meia-palavra);

se tivermos de 5 a 9 dígitos, gastaremos sempre 4 *bytes* (palavra);

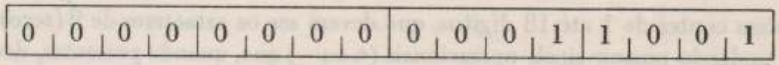
se tivermos de 10 a 18 dígitos, gastaremos sempre 8 *bytes* (palavra-dupla).

A opção COMPUTATIONAL pode ser aplicada apenas a itens numéricos, e a particularidade do sinal operacional está no *bit* mais à esquerda da meia-palavra, palavra ou palavra-dupla, conforme seja o caso. Sempre que se usar esta opção, deve-se definir a PICTURE com sinal (*S*).

Exemplo:

```
Ø5 CAMPOEX1 PICTURE S999 COMPUTATIONAL.
```

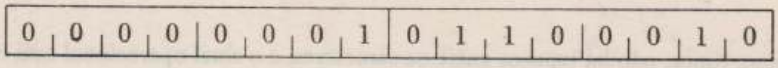
Suponha que tenhamos como conteúdo o valor 25; como especificamos 3 dígitos, gastaremos, neste caso, 2 *bytes* (meia-palavra). A configuração interna será:



2 *bytes* = meia-palavra

o *bit* mais à esquerda que representa a posição de sinal, e pode estar preenchido somente de duas formas: se com o valor Ø (zero), representa valores positivos; se com 1 (um), representa valores negativos.

Observe que o número 25 está armazenado em base binária. Do mesmo modo, o valor 356 seria armazenado assim:



Vejamos agora uma tabela exemplificando os três tipos de armazenagem de dados:

OPÇÃO				
VALOR	PICTURE	DISPLAY (zonado, decimal-externo)	COMPUTATIONAL-3 (compactado, decimal-interno)	COMPUTATIONAL
8	S9(5)	F0F0F0F1C8	00008C	0000 1000
−1	S9(4)	F0F0F0D1	00001D	1111 1111
117	S9(4)	F0F1F1C7	00117C	0111 0101
−117	S9(4)	F0F1F1D7	00117D	1000 1011

6.5. CONSTANTES

Uma constante é um dado cujo valor não está sujeito a mudanças; pode ser movimentado para um campo ou ser usado para estabelecermos valores iniciais desse campo, para fórmulas etc. Existem dois tipos: literais e figurativas.

6.5.1. Constantes literais

Um literal é composto por uma série de caracteres que determina seu valor, os quais podem ser numéricos ou não numéricos.

6.5.1.1. Literais numéricos

Podem conter de 1 até 18 dígitos, que devem ser os caracteres de Ø (zero) a 9 (nove), podendo possuir sinais operacionais (+ ou –) que, quando presentes, devem ser colocados como o caracter mais à esquerda; não pode haver espaços entre os caracteres do literal.

Os literais numéricos podem possuir um ponto/vírgula decimal.

Exemplos:

```
+ 13517
– 17737
  15000
```

Como vimos, não há espaços entre os caracteres, e o sinal operacional está colocado como o caracter mais à esquerda do literal. No caso de não existir sinal operacional, o literal será considerado positivo.

O uso da vírgula decimal ou do ponto decimal depende do emprego da cláusula DECIMAL-POINT IS COMMA.

Exemplos:

```
+ 13,56   (com DECIMAL-POINT IS COMMA)
– 13.57   (sem DECIMAL-POINT IS COMMA)
```

Pode existir somente um ponto/vírgula decimal, isto é, vírgula/ponto não podem ser usados como separadores de classes.

Exemplo:

135.136,37 (inválido)
135 136,37 (válido)

Uma outra regra é que o ponto/vírgula decimal não pode ser colocado como o caracter mais à direita ou à esquerda do literal.

Exemplo:

,05 (inválido)
1536. (inválido)
0,05 (válido)
1536 (válido)

6.5.1.2. Literais não numéricos

A característica dos literais não numéricos é que são sempre colocados entre apóstrofos (') ou aspas ("), o que vai depender do equipamento utilizado.

Um literal não numérico pode possuir qualquer caracter do conjunto EBCDIC dentro dos apóstrofos, exceto, naturalmente, o próprio apóstrofo, sendo que seu tamanho máximo deve ser de 120 caracteres entre os apóstrofos.

Os literais não numéricos pertencem à categoria alfanumérica e podem ser chamados literais alfanuméricos.

Exemplo:

'PROGRAMADOR COBOL'

O literal não numérico acima possui 17 caracteres, porque o branco também faz parte dos caracteres.

'55555'
'/*.;$'
'1234567890'
'.13E-13'
'FUNDO DE GARANTIA DE TEMPO DE SERVIÇO'

6.5.2. Constantes figurativas

São palavras reservadas que exprimem valores específicos; são de grande utilidade prática para estabelecermos valores iniciais para os campos ou para movimentarmos valores expressos por essas constantes. São elas:

ZERO
ZEROS — representam o(s) caracter(es) zero(s)
ZEROES

Como exemplo, lançaremos mão de um comando que movimenta valores de uma área para outra. Seja:

```
MOVE ZEROS TO TOTAL-SALARIO.
```

Com esse comando, o campo TOTAL-SALÁRIO será preenchido com zeros, que é o valor expresso pela constante figurativa; o mesmo aconteceria se utilizássemos outras constantes, tais como:

SPACE
SPACES — que representam caracteres em branco

HIGH-VALUE
HIGH-VALUES — que representam o maior valor que o campo pode assumir

LOW-VALUE
LOW-VALUES — que representam o menor valor que o campo pode assumir

```
MOVE SPACES TO CONTROLE.
```

Serão movimentados brancos para o campo chamado "CONTROLE".

```
MOVE HIGH-VALUES TO CONTROLE.
```

Será movimentado para cada *byte* do campo CONTROLE o valor decimal FF que representa o maior valor possível que podemos ter.

```
MOVE LOW-VALUES TO CONTROLE.
```

Será movimentado para cada *byte* do campo o valor em hexadecimal ØØ que representa o menor valor possível que um campo pode assumir.

É importante saber que, usando tanto o plural como o sigular das constantes figurativas, teremos o mesmo efeito. Excetuando as constantes ZERO, ZEROS, ZEROES, as outras constantes só podem ser aplicadas a itens alfanuméricos.

Outra constante de grande utilidade que pode ser aplicada a itens alfanuméricos é a constante

ALL literal

na qual podemos estabelecer como valor inicial de um campo, ou movimentarmos para um campo, o conjunto de caracteres que compõe o literal que será repetido por todo o campo.

Exemplo:

Suponha que tenhamos um campo de 150 posições e necessitemos movimentar para todo o campo CAMPOTEST o caracter asterisco (*) .

Naturalmente, não podemos utilizar um literal, porque o campo é maior que 120 posições, e, mesmo que fosse menor, necessitaríamos de literais gigantes, se tivéssemos uma grande quantidade de movimentos; isso, porém, não acontece, porque fazemos assim:

```
MOVE ALL '*' TO CAMPOTEST.
```

A área CAMPOTEST será totalmente preenchida com asteriscos.

Outros exemplos:

```
MOVE ALL ' / ' TO CAMPOTEST2.
MOVE ALL ' 9 ' TO CAMPOTEST3.
```

Uma outra constante de menor utilidade é a constante figurativa QUOTE.

Sabemos que, em um literal, não podemos ter como caracter o apóstrofo, mas suponha que necessitemos ter como impressão um literal não numérico com apóstrofos; então faremos assim:

```
QUOTE ' ESTUDE COBOL ' QUOTE
```

Teremos então como saída impressa:

' ESTUDE COBOL '

O que acontece é que os apóstrofos do literal funcionam como delimitadores. QUOTE é a constante figurativa que é substituída pelo caracter apóstrofo (').

Outro exemplo:

' LITROS D' QUOTE ' AGUA '

Fazemos:

LITROS D'AGUA

Os apóstrofos delimitadores desaparecem e a constante figurativa é substituída.

6.6. CLÁUSULA VALUE

Usa se esta cláusula para definir o valor inicial de um campo; ela só pode ser utilizada na WORKING-STORAGE SECTION, exceto com o nível 88.

O formato da opção é o seguinte:

VALUE IS literal

Esta cláusula não pode ser usada em itens que contenham cláusulas OCCURS e/ou REDEFINES, nem em itens subordinados a outros que tenham em suas definições tais cláusulas.

Exemplo:

```
77 CONT-LIN PICTURE 99 VALUE ZERO.
```

o que significa que o valor do campo CONT-LIN será iniciado com zero. No exemplo utilizamos uma constante figurativa, mas poderia ser um literal numérico:

```
77 CONT-LIN PICTURE 99 VALUE Ø.
```

No caso do campo ser alfanumérico, o literal virá entre aspas:

```
03 FILLER PICTURE X(7) VALUE ' SALARIO '.
```

Isso significa que uma determinada área com o tamanho de sete posições tem como conteúdo a palavra SALÁRIO.

Para se definir uma condição, temos o formato:

```
{VALUE  IS  }        literal-1        [THRU  literal-2]
{VALUES ARE }

[literal-3          [THRU  literal-4]]  ...
```

onde, para nome de condição, teremos uma cláusula VALUE.

Fazendo uma revisão: o nome-de-condição é um nome dado pelo usuário para definir que uma variável condicional pode assumir, e cada condição é precedida pelo número de nível 88, o qual, por sua vez, deve ser precedido por outros números de nível 88 ou pela própria variável condicional.

Exemplo:

```
03 ESTADO-CIVIL    PICTURE 9.
   88 SOLTEIRO     VALUE   1.
   88 CASADO       VALUE   2.
   88 VIUVO        VALUE   3.
   88 DESQUITADO   VALUE   4.
```

onde o ESTADO-CIVIL é a variável condicional; SOLTEIRO, CASADO, VIÚVO e DESQUITADO são nomes-de-condição para os valores 1, 2, 3 e 4, respectivamente, que o campo ESTADO-CIVIL pode assumir.

A opção *THRU* especifica o intervalo de valores que o nome-de-condição pode assumir e, quando usada, o *literal-1* precisa ser menor que o *literal-2*, o *literal-3* menor que o *literal-4* etc.

Exemplo:

```
04 NOTAS PICTURE 999.
   88 PESSIMO VALUE 0 THRU 30.
```

88 SOFRIVEL VALUE 31 THRU 49.
88 REGULAR VALUE 50 THRU 59.
88 BOM VALUE 60 THRU 79.
88 OTIMO VALUE 80 THRU 100.

Outros exemplos:

a) Ø4 CAMPOA PICTURE S9 (5) COMP-3 VALUE ZEROS.

b) Ø3 CAMPOB PICTURE S9 (4) COMP VALUE 1ØØ.

c) Ø5 CAMPOC PICTURE X (9) VALUE 'FEVEREIRO'.

d) Ø2 CAMPOD PICTURE 99 VALUE 1.

e) Ø3 CAMPOE PICTURE S9 (5) VALUE –53763.

f) Ø5 CAMPOF PICTURE S999 VALUE + 1.

g) Ø6 CAMPOG PICTURE S99.

88 LIMITE-INFERIOR VALUE – 99.

88 LIMITE-SUPERIOR VALUE + 99.

h) Ø2 CAMPOH PICTURE X (120) VALUE SPACES.

6.7. CLÁUSULA JUSTIFIED

Esta cláusula é usada para inverter a forma de um item alfabético ou alfanumérico.

$$\left\{\begin{matrix}\underline{JUSTIFIED} \\ \underline{JUST}\end{matrix}\right\} \quad \text{RIGHT}$$

Um campo alfabético ou alfanumérico é normalmente alinhado da esquerda para a direita, preenchendo-se com espaços em branco as posições à direita não usadas, ou truncando-as no caso do dado ser maior do que o campo receptor.

Exemplo:

Se movermos a palavra LINGUAGEM para os campos *A* e *B* com tamanhos de sete e onze posições, respectivamente:

```
03  A  PICTURE X(7).
03  B  PICTURE X(11).
```

teremos:

A com o conteúdo | L | I | N | G | U | A | G |

e *B* com o conteúdo | L | I | N | G | U | A | G | E | M | ␣ | ␣ |

O campo *A* foi truncado à direita e o campo *B*, preenchido com dois espaços em branco (␣) à direita.

Com o uso da cláusula JUSTIFIED RIGHT, o alinhamento do campo é feito da direita para a esquerda.

No mesmo exemplo anterior, só que agora usando a cláusula:

```
03  A  PICTURE  X(7)  JUSTIFIED RIGHT.
03  B  PICTURE  X(11) JUSTIFIED RIGHT.
```

teremos:

A com o conteúdo | G | U | A | G | U | E | M |

e *B* com o conteúdo | ␣ | ␣ | L | I | N | G | U | A | G | E | M |

O campo *A*, agora, foi truncado à esquerda e o campo *B* preenchido com dois espaços em branco (␣) à esquerda.

A restrição a esta cláusula é que só pode ser usada em itens elementares e não pode ser especificada para os níveis 66 e 88.

6.8. CLÁUSULA BLANK

Esta cláusula é utilizada quando é preciso que o campo seja impresso com brancos, no caso de seu conteúdo ser todo composto por zeros:

```
BLANK WHEN ZERO.
```

Exemplo:

```
Ø3  COMISSÃO PICTURE 9.999,99
        BLANK WHEN ZERO.
```

Caso o valor de comissão seja igual a zeros, esse conteúdo será trocado para brancos, ou seja, nada será impresso.

6.9. CLÁUSULA REDEFINES

É usada para permitir que diferentes tipos de dados, ou seja, dados com diferentes USAGES ou PICTURES, sejam armazenados numa mesma área de memória, facilitando a manipulação do conteúdo da área para diversos usos.

número de nível nome-de-dado-1 *REDEFINES* nome-de-dado-2

Os números de níveis de nome-de-dado-1 e nome-de-dado-2 devem ser os mesmos e não podem ser 66 ou 88.

Nome-de-dado-1 será considerado como um outro nome-de-dado para a mesma área de memória.

A cláusula REDEFINES nunca deve ser empregada em níveis Ø1 da FILE SECTION, já que o compilador assume que todos os níveis Ø1 desta seção são redefinições um do outro.

Exemplo de uma redefinição:

```
Ø2  GRUPOA.

    Ø3  CAMPOA PICTURE X(5).
    Ø3  CAMPOB PICTURE X(6).
    Ø3  CAMPOC PICTURE XX.

Ø2  GRUPOB REDEFINES GRUPOA PICTURE X(13).
```

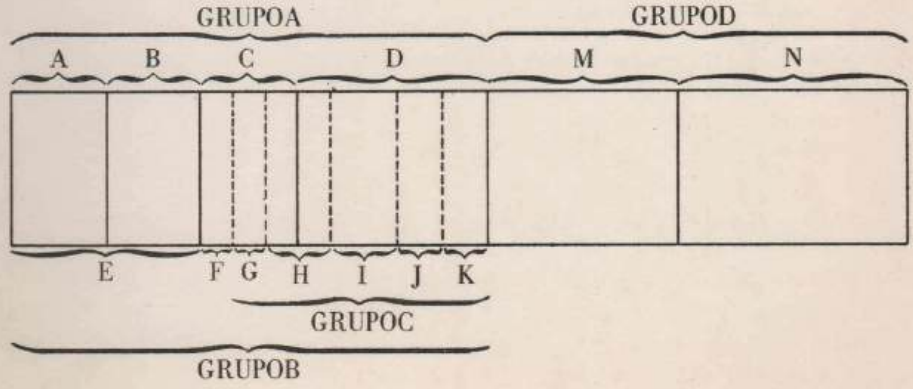
Note que os números de nível de GRUPOA e GRUPOB são iguais, assim como o tamanho da área ocupada pelos dois grupos. No caso, temos a necessidade, por exemplo, de no GRUPOA manipularmos 3 itens elementares e em determinada situação manipularmos o conteúdo desses 3 itens em conjunto; para isso, usaremos o

GRUPOB com os 13 *bytes* que contêm o conteúdo dos 3 campos. GRUPOB, portanto, é uma redefinição de GRUPOA, mas ambos representam *a mesma área* de memória, com nomes e estruturação diferentes.

Exemplos:

```
Ø2 GRUPOA.
    Ø3 CAMPOA PIC 999.
    Ø3 CAMPOB PIC XXXX.
    Ø3 CAMPOC PIC X(5).
    Ø3 CAMPOD PIC X(11).

Ø2 GRUPOB REDEFINES GRUPOA.
    Ø3 CAMPOE PIC X(7).
    Ø3 CAMPOF PIC 9.
    Ø3 CAMPOC.
        Ø4 CAMPOG PIC XXX.
        Ø4 CAMPOH PIC XXX.
        Ø4 CAMPOI PIC XXX.
        Ø4 CAMPOJ PIC XXXX.
        Ø4 CAMPOK PIC XX.

Ø2 GRUPOD.
    Ø3 CAMPOM PIC X(20).
    Ø3 CAMPON PIC X(50).
```

Estariam assim organizados na memória:



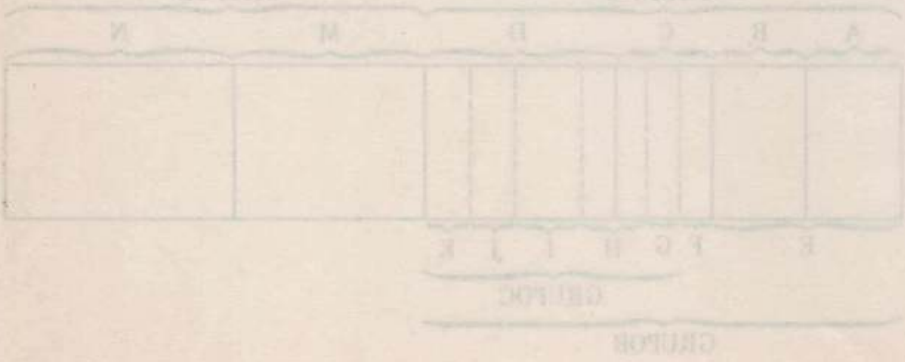
Vejamos outro exemplo:

```
Ø2 A  PIC S99V99.
Ø2 B  REDEFINES A PIC S9V9999.
Ø2 C  REDEFINES A PIC S9(5).
```

Neste exemplo vemos que podemos ter mais de uma redefinição para uma mesma área.

Observe também que o nome-de-dado-2 não precisa ser sempre o que fez a definição original; pode ser o da última redefinição.

```
Ø2 A  PIC XXXXXX.
Ø2 B  REDEFINES A PIC 99.999.
Ø2 C  REDEFINES B PIC ZZ.ZZZ.
Ø2 D  REDEFINES C PIC XXBXXX.
```

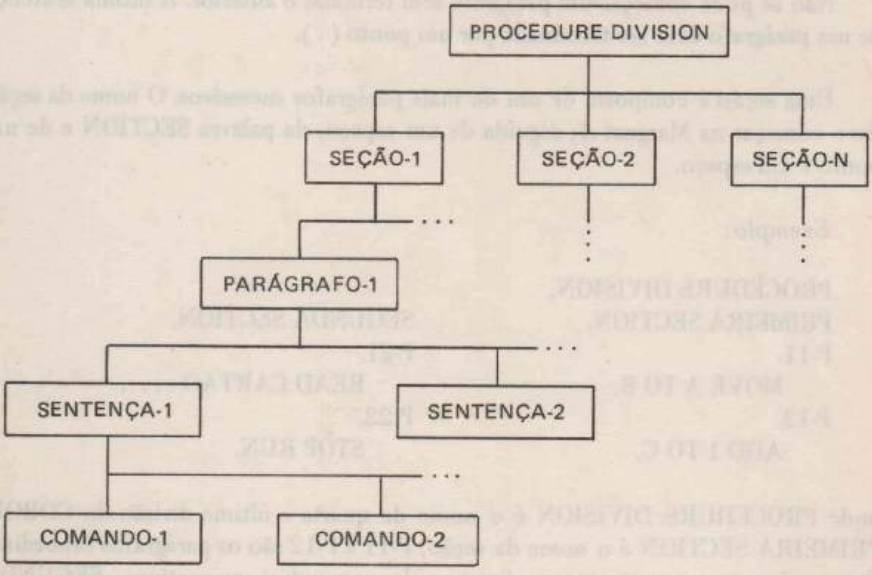


7

Divisão de Procedimentos

É nesta divisão que o programador vai codificar passo a passo o caminho que o programa deve seguir, ou seja: a lógica do programa. Para isso é necessário utilizar comandos aceitos pela linguagem.

O *comando* é a unidade básica da PROCEDURE DIVISION (divisão de procedimentos), e faz parte de uma hierarquia organizacional, como se pode ver na figura:



Como se vê, os comandos são agrupados em sentenças, as sentenças em parágrafos e os parágrafos em seções.

Cada *comando* consiste, no mínimo, em um verbo COBOL seguido de palavras (nomes-de-dados, de arquivos, ou literais); estão divididos em três categorias:

1. *Comando de Decisão* – contém uma condição a ser testado para determinar qual a alternativa do fluxo que o programa deve seguir.

2. *Comandos Imperativos* – especificam uma ação incondicional a ser seguida pelo programa-objeto.

3. *Comandos Especiais para o Compilador* – que o instruem a fazer certas ações durante o tempo de compilação.

Cada *sentença* é composta de um ou mais comandos, que podem ou não estar separados por ponto e vírgula (;) ou vírgula (,), mas devem ser separados no mínimo por *um* espaço.

Várias sentenças que exprimam uma mesma idéia ou procedimento podem ser agrupadas para formar um *parágrafo*. Este deve começar na Margem *A* com um nome-de-parágrafo (paragraph-name) seguido por um ponto e um espaço.

Não se pode começar um parágrafo sem terminar o anterior. A última sentença de um parágrafo deve ser terminada por um ponto (.).

Uma seção é composta de um ou mais parágrafos sucessivos. O nome da seção deve começar na Margem *A*, seguida de um espaço, da palavra SECTION e de um ponto e um espaço.

Exemplo:

```
PROCEDURE DIVISION.
PRIMEIRA SECTION.                    SEGUNDA SECTION.
P-11.                                P-21.
    MOVE A TO B.                         READ CARTAO . . . .
P-12.                                P-22.
    ADD 1 TO C.                          STOP RUN.
```

onde PROCEDURE DIVISION é o nome da quarta e última divisão do COBOL, PRIMEIRA SECTION é o nome da seção, P-11 e P-12 são os parágrafos subordinados a ela com suas sentenças (grupos de comandos) respectivos. SEGUNDA SECTION é o nome da outra seção, e P-21 e P-22 são os parágrafos subordinados a esta, com suas sentenças.

7.1. COMANDOS ARITMÉTICOS

Os comandos aritméticos são usados para cálculos. As quatro operações são realizadas pelos comandos *ADD* (somar), *SUBTRACT* (subtrair), *MULTIPLY* (multiplicar) e *DIVIDE* (dividir), e podem ser combinadas como numa fórmula. Como existem opções comuns aos comandos aritméticos, discuti-las-emos antes das definições de cada comando individualmente.

7.1.1. Opção ROUNDED

Depois do alinhamento do ponto decimal, o número de casas decimais do resultado da operação aritmética é comparado com o número de casas decimais providenciadas para o campo de resultados. Caso exceda, haverá um truncamento, a não ser que *ROUNDED* seja especificado.

No arredondamento, o último dígito do resultado tem seu valor aumentado de 1, sendo o dígito seguinte maior ou igual a 5; caso contrário, todo o excesso é desprezado.

7.1.2. Opção SIZE ERROR

Se, depois do alinhamento do ponto decimal, o valor do resultado excede o maior valor que pode ser contido no campo, ocorre uma condição de "erro de tamanho" (SIZE ERROR).

Divisões por *zero* sempre causam condições de erros de tamanho.

Essa condição se aplica somente ao resultado final, não aos resultados intermediários.

Se a opção *ROUNDED* for especificada, os arredondamentos serão feitos antes da verificação da condição de erro de tamanho.

Quando uma condição de erro de tamanho acontece, a providência a ser tomada varia, conforme a opção *SIZE ERROR* tenha ou não sido especificada. Se o tiver sido, e acontecer uma condição de erro de tamanho, o valor do campo de resultado não será afetado. Se, porém, não for especificada e acontecer uma condição de erro de tamanho, o resultado será imprevisível.

Após ter sido executada a operação aritmética, executa-se o comando imperativo da opção *SIZE ERROR*.

7.2. COMANDO ADD

É utilizado para obter o resultado da soma de dois ou mais itens de dados numéricos, e se apresenta em três formatos.

FORMATO 1

$$\underline{ADD}\ \begin{Bmatrix}\text{nome-de-dado-1}\\ \text{literal-1}\end{Bmatrix}\left[\begin{matrix}\text{nome-de-dado-2}\\ \text{literal-2}\end{matrix}\right]\ldots\ \underline{TO}\ \text{nome-de-dado-m}\left[\underline{ROUNDED}\right]$$

$$\left[\text{nome-de-dado-n}\left[\underline{ROUNDED}\right]\right]\ \ldots\ \left[\text{ON}\ \underline{SIZE\ ERROR}\ \text{comando imperativo}\right]$$

Os valores dos operandos que seguem a palavra ADD (nome-de-dado-1, nome-de-dado-2 etc.) são somados, o resultado desta soma é adicionado aos valores nome-de-dado-m, nome-de-dado-n etc., e o resultado final fica armazenado em nome-de-dado-1, nome-de-dado-n etc.

Exemplos:

1. ADD CAMPO-1 TO CAMPO-2

ANTES (da operação)

CAMPO-1	CAMPO-2
10	10

DEPOIS (da operação)

CAMPO-1	CAMPO-2
10	20
	10 + 10

2. ADD CAMPO-1 CAMPO-2 CAMPO-3 TO CAMPO-4 CAMPO-5 CAMPO-6

Suponha que o valores dos campos sejam:

ANTES (da operação)

CAMPO-1	CAMPO-2	CAMPO-3	CAMPO-4	CAMPO-5	CAMPO-6
3	4	5	2	7	10

DEPOIS (da operação)

3	4	5	14	19	22
			3 + 4 + 5 + 2	3 + 4 + 5 + 7	3 + 4 + 5 + 10

FORMATO 2

```
ADD   { nome-de-dado-1 }   { nome-de-dado-2 }   [ nome-de-dado-3 ]
      { literal-1      }   { literal-2      }   [ literal-3      ] ...
GIVING nome-de-dado-m [ROUNDED] [ON SIZE ERROR comando imperativo]
```

Quando se utiliza a opção GIVING, deve haver no mínimo *dois operandos* precedendo a palavra GIVING.

Os valores desses operandos são somados, e o resultado da soma é armazenado no nome-de-dado-m "destruindo" o valor inicial desse campo.

Exemplo:

```
ADD CAMPO-1 CAMPO-2 CAMPO-3 GIVING CAMPO-4
ANTES (da operação)
CAMPO-1  CAMPO-2  CAMPO-3  CAMPO-4
  30       35        7        100
DEPOIS (da operação)
  30       35        7         72
                             30 + 35 + 7
```

Observe a diferença entre o Formato 1 e o Formato 2.

No Formato 1, os valores dos operandos após a palavra *TO* são usados como parcelas, e o resultado é armazenado nesses operandos, destruindo seus valores. No Formato 2, o valor do operando após a palavra GIVING é também destruído, mas seu valor *não é usado* como parcela.

Nos Formatos 1 e 2, cada nome-de-dado deve referir-se a itens elementares numéricos, excetuando os que aparecem à direita da palavra GIVING; estes podem referir-se a itens numéricos editados.

Exemplo:

```
ADD CAMPO-1 CAMPO-2 GIVING CAMPO-3
ANTES (da operação)
        CAMPO-1          CAMPO-2          CAMPO-3
          10               10          0,00 (PICTURE Z9,99)
DEPOIS (da operação)
          10               10          20,00
```

Nota:

a) Os literais devem ser definidos como literais numéricos.

b) O tamanho máximo de cada operando deve ser de 18 dígitos.

c) O tamanho máximo dos resultados de soma deve ser de 18 dígitos.

FORMATO 3

```
ADD { CORR          } nome-de-dado-1 TO nome-de-dado-2
    { CORRESPONDING }

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

Quando a opção CORRESPONDING é usada, os itens elementares que estão dentro do nome-de-dado-1 são somados e armazenados nos itens elementares correspondentes que estão dentro do nome-de-dado-2 (nome-de-dado-1 e nome-de-dado-2 deverão ser itens de grupo).

Quando ON SIZE ERROR é usada em conjunto com CORRESPONDING, o teste de erro de tamanho é feito somente depois de todas as operações de soma terem sido completadas. Se qualquer uma das somas produzir uma condição de erro de tamanho, o campo de resultado da adição não será alterado e o comando imperativo da operação SIZE ERROR será executado.

7.3. COMANDO SUBTRACT

O comando SUBTRACT é usado para subtrair um ou a soma de dois ou mais dados de outro item-de-dado.

FORMATO 1

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

... FROM nome-de-dado-m [ROUNDED]

[nome-de-dado-n [ROUNDED]] ... [ON SIZE ERROR comando imperativo]
```

No Formato 1, os valores dos literais ou dos itens-de-dado que seguem a palavra SUBTRACT são somados, e o resultado é subtraído do nome-de-dado-m, nome-de-dado-n (se especificado) etc., onde o resultado final é armazenado.

Exemplo:

```
SUBTRACT CAMPO-1 CAMPO-2 CAMPO-3 FROM CAMPO-4

        CAMPO-1   CAMPO-2   CAMPO-3   CAMPO-4

ANTES (da operação)
          10        10        10        40

DEPOIS (da operação)
          10        10        10        10
                                        ------------
                                        40 - (10 + 10 + 10)
```

FORMATO 2
<u>*SUBTRACT*</u> { nome-de-dado-1 / literal-1 } [nome-de-dado-2 / literal-2] . . . <u>*FROM*</u> [nome-de-dado-m / literal-m]
<u>*GIVING*</u> nome-de-dado-m [<u>*ROUNDED*</u>] [ON <u>*SIZE ERROR*</u> comando imperativo]

No Formato 2, os valores dos literais ou dos itens-de-dado que seguem a palavra SUBTRACT são somados, o resultado é subtraído do nome-de-dado-m, ou literal-m, e o resultado final é armazenado no nome-de-dado-n.

Exemplo:

SUBTRACT CAMPO-1 CAMPO-2 CAMPO-3 CAMPO-4 GIVING CAMPO-5

	CAMPO-1	CAMPO-2	CAMPO-3	CAMPO-4	CAMPO-5
ANTES (da operação)					
	20	20	10	100	qualquer
DEPOIS (da operação)					
	20	20	10	100	50

$$\overbrace{100 - (20 + 20 + 10)}$$

FORMATO 3
<u>SUBTRACT</u> { <u>*CORR*</u> / <u>*CORRESPONDING*</u> } nome-de-dado-1 <u>*FROM*</u> nome-de-dado-2
[<u>*ROUNDED*</u>] [ON <u>*SIZE ERROR*</u> comando imperativo]

Quando a opção CORRESPONDING é usada, os itens elementares que estão dentro do nome-de-dado-1 são subtraídos e armazenados nos itens elementares correspondentes que estão dentro do nome-de-dado-2 (os itens-de-dado-1 e 2 deverão ser itens de grupo).

Quando a opção ON SIZE ERROR é usada em conjunto com a opção CORRESPONDING, o funcionamento é análogo ao comando ADD.

Cada item-de-dado deve referir-se a um item numérico, com exceção do item que segue a palavra GIVING, que pode ser definido como item numérico editado (todo literal deve ser definido como literal numérico). O tamanho máximo de cada operando deve ser de 18 dígitos.

Depois do alinhamento decimal, o resultado da subtração terá o tamanho máximo de 18 dígitos.

Exemplo:

Suponha que tenhamos dois itens de grupo definidos da seguinte maneira:

```
Ø2 GRUPO-1.                    Ø3 GRUPO-2.

   Ø3 CAMPO-1 PIC...              Ø4 CAMPO-1 PIC...
   Ø3 CAMPO-2 PIC...              Ø4 CAMPO-2 PIC...
   Ø3 CAMPO-3 PIC...              Ø4 CAMPO-3 PIC...
```

Note que os itens correspondentes deverão ter nomes iguais.

Daremos comandos:

```
ADD CORRESPONDING GRUPO-1 TO GRUPO-2
```

GRUPO-1

CAMPO-1	CAMPO-2	CAMPO-3
ANTES (da operação)		
20	30	40
DEPOIS (da operação)		
20	30	40

GRUPO-2

CAMPO-1	CAMPO-2	CAMPO-3
ANTES (da operação)		
60	70	80
DEPOIS (da operação)		
80	100	120
20 + 60	30 + 70	40 + 80

SUBTRACT CORRESPONDING GRUPO-1 FROM GRUPO-2

GRUPO-2

CAMPO-1	CAMPO-2	CAMPO-3
ANTES (da operação)		
80	100	120
DEPOIS (da operação)		
60	70	80
80 – 20	100 – 30	120 – 40

7.4. COMANDO MULTIPLY

Utiliza-se este comando para multiplicar um item-de-dado por outro.

FORMATO 1

MULTIPLY { nome-de-dado-1 / literal-1 } *BY* nome-de-dado-2 [*ROUNDED*]

[ON *SIZE ERROR* comando imperativo]

Quando o Formato 1 é usado, o valor do nome-de-dado-1 ou literal-1 é multiplicado pelo valor do nome-de-dado-2, e o resultado desse produto é armazenado no nome-de-dado-2.

Exemplo:

```
MULTIPLY        CAMPO-1        BY    CAMPO-2
    CAMPO-1                           CAMPO-2

ANTES (da operação)
  10                                    20

DEPOIS (da operação)
  10                                   200
```

FORMATO 2

MULTIPLY { nome-de-dado-1 / literal-1 } *BY* { nome-de-dado-2 / literal-2 } *GIVING* nome-de-dado-3

[*ROUNDED*] [ON *SIZE ERROR* comando imperativo]

Quando o Formato 2 é usado, o valor do nome-de-dado-1 (literal-1) é multiplicado pelo valor do nome-de-dado-2 (literal-2) e o resultado da multiplicação é armazenado no nome-de-dado-3.

Cada item-de-dado deve referir-se a um item numérico, com exceção do item que segue a palavra GIVING, que pode ser definido como item numérico editado (todo literal deve ser definido como literal numérico).

Depois do alinhamento decimal, o resultado do produto terá o tamanho máximo de 18 dígitos, e o tamanho máximo de cada operando deve ser de 18 dígitos.

Exemplo:

```
MULTIPLY  CAMPO-1  BY  CAMPO-2  GIVING  CAMPO-3
          CAMPO-1          CAMPO-2        CAMPO-3

ANTES (da operação)
   20                      20             qualquer

DEPOIS (da operação)
   20                      20             400
```

7.5. COMANDO DIVIDE

O comando DIVIDE é usado para achar o quociente da divisão de um item-de-dado por outro.

FORMATO 1

```
DIVIDE { nome-de-dado-1 } INTO nome-de-dado-2 [ROUNDED]
       { literal-1      }

[ON SIZE ERROR comando imperativo]
```

Quando o Formato 1 é usado, o valor do nome-de-dado-2 (dividendo) é dividido pelo valor do nome-de-dado-1 ou literal-1 (divisor).

O valor do dividendo nome-de-dado-2 é trocado pelo valor do quociente.

Exemplo:

```
DIVIDE  CAMPO-1  INTO  CAMPO-2  ROUNDED
        CAMPO-1        CAMPO-2

ANTES (da operação)
   5                   30

DEPOIS (da operação)
   5                   6
```

FORMATO 2

```
DIVIDE { nome-de-dado-1 } { INTO } { nome-de-dado-2 } GIVING nome-de-dado-3
       { literal-1      } { BY   } { literal-2      }

[ROUNDED] [REMAINDER nome-de-dado-4]

[ON SIZE ERROR comando imperativo]
```

Quando o Formato 2 é usado, o valor do nome-de-dado-1 (literal-1) funciona como divisor ou dividendo, dependendo do uso das palavras INTO ou BY, respectivamente, e o quociente é armazenado no nome-de-dado-3, podendo o resto, opcionalmente, ser armazenado no nome-de-dado-4.

Encontra-se o resto fazendo a operação de subtração, do dividendo, do produto do divisor com o quociente:

RESTO = DIVIDENDO – DIVISOR X QUOCIENTE

Quando a opção ROUNDED for especificada, o quociente será arredondado e somente depois o resto será calculado.

Cada item-de-dado deve referir-se a um item numérico, com exceção do item que segue a palavra GIVING, que pode ser definido como item numérico editado (todo literal deve ser definido como literal numérico).

Depois do alinhamento decimal, o resultado do quociente e do resto (se especificado) terá o tamanho máximo de 18 dígitos.

Divisão por zero sempre causará uma condição de erro de tamanho.

Exemplo:

```
DIVIDE   CAMPO-1   BY   CAMPO-2   GIVING   CAMPO-3
    REMAINDER   CAMPO-4
```

	CAMPO-1	CAMPO-2	CAMPO-3	CAMPO-4
ANTES (da operação)	43	6	qualquer	qualquer
DEPOIS (da operação)	43	6	7	1

$\overbrace{43 - (7 \times 6)}$

7.6. COMANDO COMPUTE

Este comando permite que valores de itens-de-dados, literais e expressões aritméticas sejam transportados para outro item-de-dado.

FORMATO

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

[ON SIZE ERROR  comando imperativo]
```

O literal-1 deve ser definido como literal numérico, o nome-de-dado-2 deve referir-se a um item numérico elementar, e o nome-de-dado-1 pode ser descrito como um item numérico editado.

A opção "expressão aritmética" permite o uso de uma combinação de itens-de-dados, literais numéricos e operadores aritméticos. Em conseqüência, podem-se combinar operações aritméticas sem as restrições impostas pelos comandos aritméticos ADD, SUBTRACT, MULTIPLY e DIVIDE. Para essas operações usam-se os seguintes sinais:

+	para adição
−	para subtração
*	para multiplicação
/	para divisão
**	para exponenciação
( )	para prioridade de execução das operações

Entre os sinais operadores deve existir pelo menos 1 (um) espaço.

Em todos os comandos aritméticos, o tamanho máximo de cada operando é de 18 dígitos.

Exemplos:

1. COMPUTE CAMPO-1 = CAMPO-2
   ANTES (da operação)
   CAMPO-1 CAMPO-2
   20 30
   DEPOIS (da operação)
   CAMPO-1 CAMPO-2
   30 30

2. COMPUTE CAMPO-1 = 50
   ANTES (da operação)
   CAMPO-1 LITERAL
   100 50
   DEPOIS (da operação)
   50 50

3. COMPUTE CAMPO-1 = CAMPO-2 + CAMPO-3 * CAMPO-4/CAMPO-5

À direita do sinal de igual seguem-se as regras de aritmética elementar.

As operações se realizam da esquerda para a direita, dando prioridade à potenciação, depois à multiplicação e finalmente à divisão, salvo se na expressão existir parênteses.

4. COMPUTE CAMPO-1 = (CAMPO-2 + CAMPO-3) / (CAMPO-4 + CAMPO-5)

CAMPO-1	CAMPO-2	CAMPO-3	CAMPO-4	CAMPO-5
ANTES (da operação)				
qualquer	15	5	4	1
DEPOIS (da operação)				
4	15	5	4	1

Com o uso dos parênteses, primeiramente são somados os conteúdos de CAMPO-2 e CAMPO-3, dando como resultado (*20*), depois se somam os conteúdos de CAMPO-4 com CAMPO-5, e temos como resultado (*5*). Só então se procede à divisão, da esquerda para a direita (*20 ÷ 5*), obtendo-se um total de (*4*).

Os resultados intermediários são armazenados em áreas definidas pelo compilador, as quais são geradas automaticamente pelo comando COMPUTE, sem que haja necessidade de preocupação do programador.

8

Comando de Decisão e Desvio

A cada instante precisamos questionar a máquina: "é fim de arquivo?", "número é maior que anterior?", "contador de linhas é igual a 40?", "Valor é positivo?" etc.

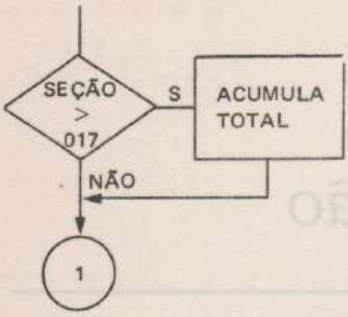
8.1. COMANDO IF

O COBOL nos permite essas perguntas pelo comando IF, usado para os testes de condição:

FORMATO

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

                     { Comando-2      }
                     { NEXT SENTENCE }
```

Assim, por exemplo, se quiséssemos saber se o conteúdo de campo chamado SEÇÃO era maior do que 017, faríamos:



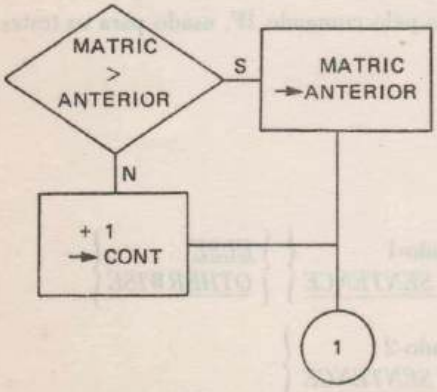

```
IF SECAO > 017
ADD TOTAL TO TOT-TOTAL.
GO TO 1.
```

Isso significa que, se a condição for verdadeira (SECAO > 017), o programa executará a instrução, "ADD TOTAL TO TOT-TOTAL", e prosseguirá normalmente; mas, se a condição for falsa, o programa "saltará" para a primeira instrução a partir do primeiro ponto no caso, "GO TO 1".

Observe que o ponto para o comando IF é de grande importância, pois limita a condição no caso de não ser usada a opção ELSE (senão) ou OTHERWISE (de outra forma).

Exemplo:

Seja o seguinte trecho de fluxo:




```
IF MATRIC > ANTERIOR
MOVE MATRIC TO ANTERIOR
ELSE ADD 1 TO CONT.
GO TO 1.
```

Isso significa que, se o teste MATRIC > ANTERIOR for verdadeiro, o programa executará a instrução MOVE MATRIC TO ANTERIOR, desviando-se para a instrução a seguir do ponto depois do ELSE, no caso GO TO 1; se o teste não for verdadeiro, executará a instrução a seguir do ELSE – ADD 1 TO CONT, e passará normalmente à instrução GO TO 1.

São vários os tipos de testes que se podem realizar, dos quais os principais são:

teste de classe;
teste de relação;
teste de condição;
teste de sinal.

8.1.1. Teste de classe

Testa se o conteúdo de um determinado campo é, ou não, numérico (NUMERIC) ou alfabético (ALPHABETIC). Assim:

$$\underline{IF}\ \text{nome-de-dado}\ \text{IS}\ \left[\underline{NOT}\right]\ \left\{\begin{matrix}\underline{NUMERIC}\\ \underline{ALPHABETIC}\end{matrix}\right\}$$

A opção de NUMERIC é verdadeira quando o conteúdo (em formato DISPLAY ou COMPUTATIONAL-3) do "nome-de-dado" é composto pelos dígitos de 0 a 9, positivos e negativos.

Na opção ALPHABETIC, só são válidos os caracteres de *A* a *Z* e os espaços.

Exemplos:

Se quisermos saber se o conteúdo do campo de nome de aluno (NALUNO) é alfabético, faremos:

```
IF NALUNO ALPHABETIC
      GO  TO   CERTO
ELSE  GO  TO  ERRADO.
```

Isso significa que, se dentro do campo de NALUNO só forem encontradas letras e/ ou espaços, a condição será verdadeira e o programa executará a instrução GO TO CERTO; caso contrário, executará GO TO ERRADO.

No campo de valor (VALOR) não pode haver letras, espaços nem caracteres especiais; então faremos:

```
IF VALOR NOT NUMERIC
      GO TO ERRO
ELSE NEXT SENTENCE.
```

Se o conteúdo não for numérico, satisfará a condição, e o computador executará a instrução GO TO ERRO; caso contrário, passará à sentença seguinte (NEXT SENTENCE).

8.1.2. Teste de relação

Utiliza-se este teste para fazer a comparação entre dois operandos.

FORMATO

$$\underline{IF} \begin{Bmatrix} \text{nome-de-dado-1} \\ \text{literal-1} \\ \text{expressão-aritmética} \end{Bmatrix} \quad \text{operador-de-relação} \quad \begin{Bmatrix} \text{nome-de-dado-2} \\ \text{literal-2} \\ \text{expressão-aritmética-2} \end{Bmatrix}$$

Os dois operadores não podem ser, ao mesmo tempo, do tipo literal; por exemplo:

IF 2 = 3

O "operador-de-relação" especifica o tipo de comparação que pode ser feita, sendo eles:

$$\begin{Bmatrix} \text{IS } [\underline{NOT}] \ \underline{GREATER} \text{ THAN} \\ \text{IS } [\underline{NOT}] \ > \end{Bmatrix}$$ maior que ou não maior que

$$\begin{Bmatrix} \text{IS } [\underline{NOT}] \ \underline{EQUAL\ TO} \\ \text{IS } [\underline{NOT}] \ = \end{Bmatrix}$$ igual a ou não igual a

$$\begin{Bmatrix} \text{IS } [\underline{NOT}] \ \underline{LESS} \text{ THAN} \\ \text{IS } [\underline{NOT}] \ < \end{Bmatrix}$$ menor que ou não menor que

Exemplos:

a) Se $A = B^2 - 4AC$

IF A EQUAL TO (B ** 2 – 4 * A * C) . . .

b) Se B é maior do que C
IF B > C . . .

c) Se $A^2 + 2AB + B^2 = (A + B)(A + B)$
IF (A ** 2 + 2 * A * B + B ** 2) = (A + B) * (A + B)

Observe que os operadores de relação devem estar precedidos e seguidos por pelo menos um espaço.

8.1.3. Teste de condição

Este tipo de teste está combinado com o nome-de-condição definido para o nível 88, ou seja, testa se é verdadeira a condição determinada por este nível.

FORMATO

IF [*NOT*] nome-de-condição

Exemplo:

Se, na Data Division, tivermos a seguinte determinação:

```
03 GRAU-ESCOLARIDADE PICTURE 9.
   88 PRIMARIO VALUE 1.
   88 SECUNDARIO VALUE 2.
   88 SUPERIOR VALUE 3.
```

e quisermos o total de grau de escolaridade por tipo, testaremos da seguinte forma:

```
IF PRIMARIO
    ADD 1 TO CONT-PRIMARIO
    GO TO FORA.

IF SECUNDARIO
    ADD 1 TO CONT-SECUNDARIO
    GO TO FORA.

IF SUPERIOR
    ADD 1 TO CONT-SUPERIOR
```

ou seja, se o conteúdo do campo GRAU-ESCOLARIDADE for igual a 1, significará PRIMÁRIO, e então o teste será "se primário" . . ., pois, se o nível 88 não tivesse sido descrito, faríamos:

```
IF  GRAU-ESCOLARIDADE = 1
        ADD 1 TO CONT-PRIMARIO
        GO TO FORA.

IF  GRAU-ESCOLARIDADE = 2
        ADD 1 TO CONT-SECUNDARIO
        GO TO FORA.

IF  GRAU-ESCOLARIDADE = 3
        ADD 1 TO CONT-SUPERIOR
        GO TO FORA.
```

8.1.4. Teste de sinal

Este testa se o conteúdo de um campo numérico é positivo, negativo ou igual a zero.

FORMATO

IF { nome-de-dado / expressão-aritmética } IS [*NOT*] { *POSITIVE* / *NEGATIVE* / *ZERO* }

Exemplo:

```
IF  NUMERO  POSITIVE . . .
IF  VALOR  NOT  ZERO . . .
IF  TOTAL  NEGATIVE . . .
```

8.1.5. Condições compostas

Podemos combinar duas ou mais condições, formando, assim, as condições compostas, que são ligadas uma a outra pelos operadores lógicos AND (e) e OR (ou):

AND – significa que todas devem ser verdadeiras;
OR – significa que uma ou todas devem ser verdadeiras.

Exemplos:

1. Se A = B *e* C > D, vá para CERTO

```
IF A = B
AND C > D
    GO TO CERTO
```

Aqui, as *duas* condições têm que ser satisfeitas.

2. Se A = B *ou* C > D, vá para CERTO

```
IF A = B
OR C > D
    GO TO CERTO
```

Aqui, basta que uma condição seja satisfeita.

3. Se A NOT EQUAL TO 5 *e* B NOT GREATER THAN C

```
IF A NOT EQUAL TO 5
AND B NOT GREATER THAN C
    GO TO CERTO.
```

8.2. COMANDOS DE DESVIO

Estes comandos modificam o fluxo normal do programa, forçando um desvio para uma parte deste, de acordo com as necessidades do programador.

8.2.1. Comando GO TO

Altera a seqüência do programa, desviando-o para a área especificada.

FORMATO 1

GO TO nome-de-procedimento

Se um comando *GO TO* aparecer em qualquer sentença imperativa, deverá ser localizado como último comando na seqüência; isso é natural e lógico, porque, sendo o *GO TO* um procedimento de desvio incondicional, os comandos que estiverem após ele não serão executados.

Exemplo:

```
PARAGRAFO-1.
    (comandos executáveis do parágrafo)

PARAGRAFO-2.
    (comandos executáveis do parágrafo)

PARAGRAFO-3.
    (comandos executáveis do parágrafo)
        GO TO PARAGRAFO-1.
```

o que significa que o controle retornará ao PARAGRAFO-1 e a seqüência normal será seguida daí em diante a partir do primeiro comando do PARAGRAFO-1.

O comando GO TO é usado para desviar a seqüência normal do programa, em dois casos, como: voltar ao início do programa, voltar a uma rotina que precisa ser repetida, saltar uma série de procedimentos que não precisam ser executados sob uma determinada condição.

FORMATO 2

GO TO nome-de-procedimento-1 [nome-de-procedimento-2]

[nome-de-procedimento-n] ... *DEPENDING* ON nome-de-dado

Nesta forma, o GO TO é um comando condicional, isto é, o desvio ocorre para as referidas áreas de acordo com a variação do valor do nome-de-dado. Assim, se este é igual a 1, o programa se desloca para nome-de-procedimento-1 etc.

Exemplo:

Suponhamos que o valor do campo CONTROLE (nome-de-dado) esteja *no momento* com o valor 3:

A	B
P1.	GO TO P2 P3 P4 P5 DEPENDING ON CONTROLE
P2.	
P3.	(comandos)
P4.	(comandos)
P5.	(comandos)
	(comandos)

Isso significa que, quando o sistema executar a instrução *GO TO*, se deslocará para o nome-de-procedimento P4.

O campo "nome-de-dado" deve ter no máximo 4 *bytes* e não pode exceder o valor 2031.

Caso o valor do campo não esteja no intervalo de 1 a *n*, inclusive, o comando GO TO é ignorado e o controle passa para a próxima instrução.

8.2.2. Comando PERFORM

É utilizado para deslocar o programa do fluxo normal, fazendo-o executar subrotinas especificadas pelo programa e, ao término destas, *retornar* ao ponto de deslocamento para seguir seqüencialmente o fluxo do programa.

FORMATO 1

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

Nesta forma, o programa se desvia para nome-de-procedimento-1, e executa todas as instruções contidas neste parágrafo. Se a opção THRU é especificada, ele executa todos os parágrafos entre os dois nomes-de-procedimentos, inclusive.

Exemplo:

```
PERFORM A THRU D.
GO  TO  F.

A.
B.
C.
D.
E.
F.
```

```
(comandos do parágrafo A)
(comandos do parágrafo B)
(comandos do parágrafo C)
(comandos do parágrafo D).
```

Quando a instrução de "PERFORM A THRU D" é encontrada, o programa *não* executa a instrução seguinte (GO TO F.) até que tenha executado todos os comandos de A a D (inclusive). Só então retorna para a instrução de GO TO F.

FORMATO 2

PERFORM nome-de-procedimento-1

[*THRU* nome-de-procedimento-2] { nome-de-dado / inteiro } *TIMES*

Este formato se comporta exatamente como o anterior, sendo que a execução dos parágrafos ocorre tantas vezes quantas indicadas pelo conteúdo de "nome-de-dado" ou pelo valor de "inteiro", isto é, só retorna à instrução seguinte quando a opção do TIMES (vezes) é satisfeita.

Exemplo:

```
PERFORM A THRU B 7 TIMES
GO   TO   ACABOU.
A.
    ADD 1 TO SOMA.
B.
    EXIT.
```

O programa executará a instrução de "ADD 1 TO SOMA" sete vezes, depois a instrução de GO TO ACABOU.

8.2.3. Comando EXIT

No exemplo anterior foi introduzido um comando novo – EXIT (saída) – que é usado como ponto de retorno de uma sub-rotina.

FORMATO

Nome-de-procedimento. *EXIT*.

Quando usado, deve ser o único comando dentro do parágrafo.

Voltando ao comando PERFORM:

FORMATO 3

PERFORM nome-de-procedimento-1

[*THRU* nome-de-procedimento-2] *UNTIL* condição

Este também se comporta como o Formato 1, sendo que o programa fica executando as instruções que compõem os parágrafos até que (UNTIL) a condição especificada seja satisfeita.

Exemplo:

```
PERFORM A THRU B UNTIL SOMA > 7
GO TO ACABOU.
A.
 ADD 1 TO SOMA.
B.
 EXIT.
```

Quando o campo SOMA for igual a *8*, o programa terá satisfeito a condição e retornará para a instrução seguinte (GO TO ACABOU).

FORMATO 4

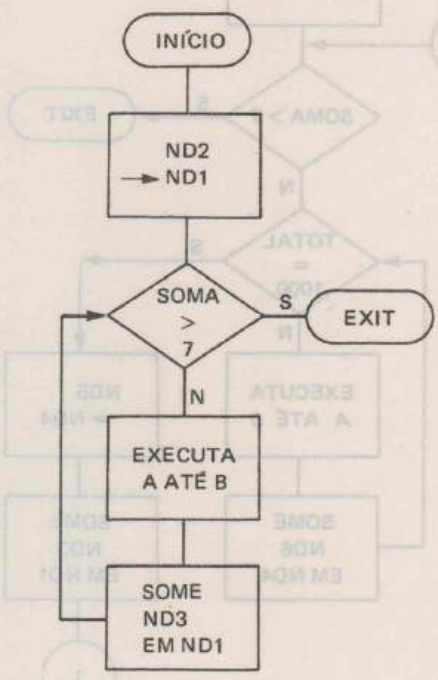
```
PERFORM nome-de-procedimento-1 [THRU nome-de-procedimento-2]

VARYING nome-de-dado-1          FROM { nome-de-dado-2 }
                                     { literal-2      }

BY { nome-de-dado-3 }           UNTIL condição-1
   { literal-3      }

[AFTER nome-de-dado-4           FROM { nome-de-dado-5 }
                                     { literal-5      }

BY { nome-de-dado-6 }           UNTIL condição-2
   { literal-6      }

[AFTER nome-de-dado-7           FROM { nome-de-dado-8 }
                                     { literal-8      }

BY { nome-de-dado-9 }           UNTIL condição-3]]
   { literal-9      }
```

A despeito do tamanho da instrução, não é difícil entendê-la.

Com a variação de *um* nome-de-dado:

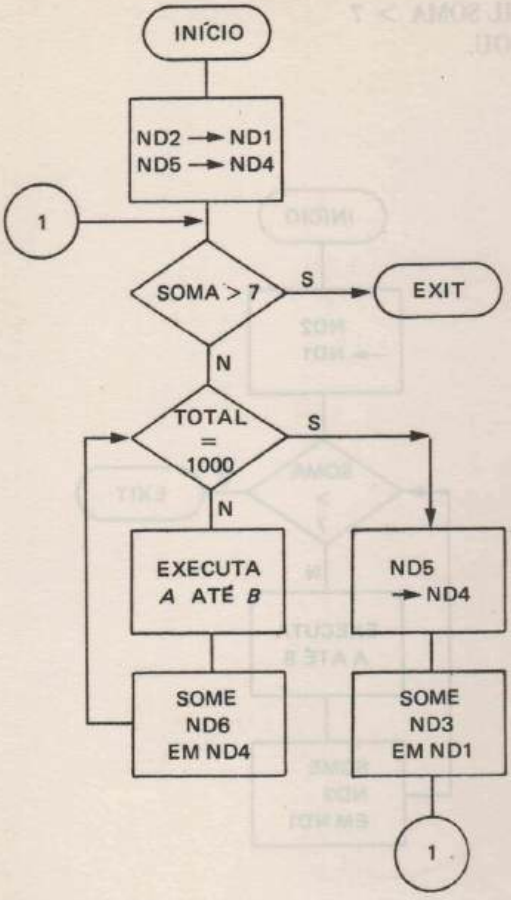
```
PERFORM A THRU B
VARYING ND1 FROM ND2
```

```
BY ND3 UNTIL SOMA > 7
GO TO ACABOU.
```



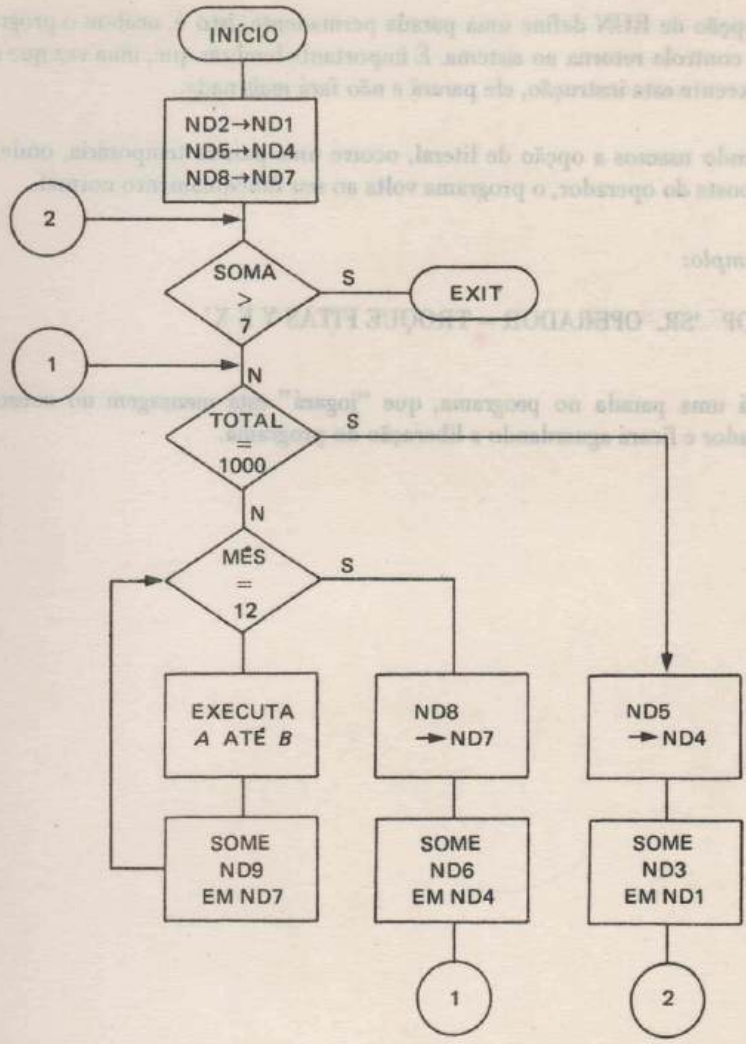
Com a variação de *dois* nomes-de-dado:

```
PERFORM A THRU B
VARYING ND1 FROM ND2
BY ND3 UNTIL SOMA 7
AFTER ND4 FROM ND 5
BY ND6 UNTIL TOTAL = 1000
GO TO ACABOU.
```



Com a variação de *três* nomes de dado:

```
PERFORM A THRU B
VARYING ND1 FROM ND2
BY ND3 UNTIL SOMA  7
AFTER ND4 FROM ND5
BY ND6 UNTIL TOTAL  =  1000
AFTER ND7 FROM ND8
BY ND9 UNTIL MES  =  12
GO TO ACABOU.
```



8.2.4. Comando STOP

Este comando ocasiona uma parada temporária ou permanente no programa-objeto.

$$\underline{STOP} \left\{ \begin{matrix} \underline{RUN} \\ \text{literal} \end{matrix} \right\}$$

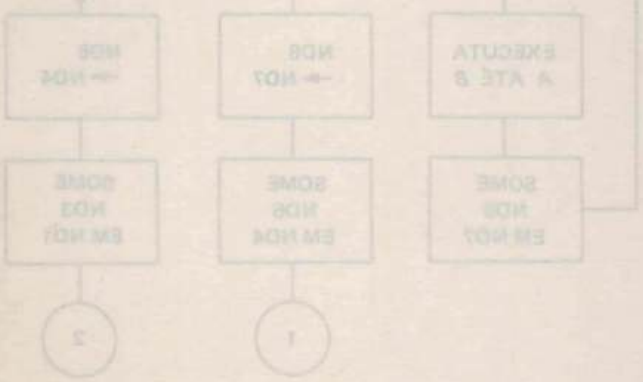
A opção de RUN define uma parada permanente, isto é, acabou o programa e então o controle retorna ao sistema. É importante lembrar que, uma vez que o programa execute esta instrução, ele parará e não fará mais nada.

Quando usamos a opção de literal, ocorre uma parada temporária, onde, com uma resposta do operador, o programa volta ao seu funcionamento normal.

Exemplo:

```
STOP 'SR. OPERADOR – TROQUE FITAS Y E X'
```

Ocorrerá uma parada no programa, que "jogará" esta mensagem no console do computador e ficará aguardando a liberação do programa.



9

Comando de Manipulação, Entrada e Saída de Dados

O comando de manipulação movimenta os dados de uma área para outra, altera o conteúdo dos campos.

9.1. COMANDO MOVE

Usado para "mover" o conteúdo de uma determinada área para outra(s). O "mover" foi colocado entre aspas porque, na realidade, não é feito um movimento, e sim uma repetição dos dados de uma área em outra.

FORMATO 1

$$\underline{MOVE}\left\{\begin{array}{l}\text{nome-de-dado-1}\\ \text{literal}\end{array}\right\}\left\{\underline{TO}\ \text{nome-de-dado-2}\ [\ \text{nome-de-dado-3}\ ]\ \ldots\right.$$

O "nome-de-dado-1" ou literal é "movido" para "nome-de-dado-2", nome-de-dado-3" etc.

O alinhamento do campo receptor (nome-de-dado-2) é feito assim:

— Se numérico – da direita para a esquerda, completando com zeros as posições que sobrarem à esquerda, ou suprimindo, se o campo for menor, as posições à esquerda.

Exemplo:

MOVE 1 TO NUMERO

Suponha que NUMERO seja um campo de 5 posições numéricas e que o seu conteúdo no momento da operação seja 17000.

– ANTES DO MOVIMENTO NUMERO	– DEPOIS DO MOVIMENTO NUMERO
17000	00001

– Se alfanumérico – da esquerda para a direita, completando com espaços as posições à direita que faltarem para preencher a área, ou suprimindo as posições em excesso à direita.

Exemplo:

MOVE A TO B

	ANTES	DEPOIS
A	GAROTA	GAROTA
B	LUA	GAROTA

Os literais podem ser numéricos, não numéricos ou constantes figurativas.

Exemplo:

MOVE ZEROS TO NUMERO

	ANTES	DEPOIS
NUMERO	12345	00000

Nos movimentos numéricos, o alinhamento do campo é feito pelo ponto decimal.

Tabela de Movimentos Permitidos

CAMPO EMISSOR \ CAMPO RECEPTOR	NUMÉRICOS	ALFABÉTICOS	ALFANUMÉRICOS	NUMÉRICO EDITADO
NUMÉRICOS	SIM	NÃO	SIM	SIM
ALFABÉTICOS	NÃO	SIM	SIM	NÃO
ALFANUMÉRICOS	NÃO	NÃO	SIM	NÃO
NUMÉRICO EDITADO	NÃO	NÃO	SIM	NÃO

FORMATO 2

$$\underline{MOVE} \left\{ \begin{matrix} \underline{CORRESPONDING} \\ \underline{CORR} \end{matrix} \right\} \text{nome-de-dado-1 } \underline{TO} \text{ nome-de-dado-2}$$

Nesta forma, os campos têm seus subitens com mesmo nome; então, o conteúdo de "nome-de-dado-1" é movido para *seu* correspondente em "nome-de-dado-2".

Exemplo:

Sejam os campos abaixo definidos:

```
3 DATA-INGLES.
  5 MES PIC 99.
  5 DIA PIC 99.
  5 ANO PIC 99.
```

```
3 DATA-PORT.
  5 DIA PIC 99.
  5 MES PIC 99.
  5 ANO PIC 99.
```

e a instrução:

```
MOVE CORR DATA-INGLÊS TO DATA-PORT.
```

Com isso, o campo de MES de DATA-INGLES será movido para MES de DATA-PORT, e assim respectivamente com cada campo.

Exemplos de movimentos:

CAMPO EMISSOR		CAMPO RECEPTOR	RESULTADO FINAL
PICTURE	CONTEÚDO	PICTURE	CONTEÚDO
X(10)	ABCDEFGHIJ	X(7)	ABCDEFG
9(5)	12345	9(7)	0012345
9V99	100	99V9	01,0
999	123	V999	,000
X(5)	12345	9(5)	—
9(4)	1234	X(8)	1234bbbb
9(5)	00123	ZZZZ9	123
9(4)V99	130000	Z.ZZZ,ZZ	1.300,00

9.2. COMANDOS DE ENTRADA E SAÍDA

Sabemos que as funções básicas de um computador pode ser agrupadas em entrada de dados, processamento de dados e saída de dados.

Para o processamento, temos os comandos de manipulação de dados, comandos aritméticos, comandos de decisão, enfim comandos que "mexem" com os dados definidos na DATA DIVISION. Esses dados podem ser de arquivos, ou áreas de memória com valores pré-definidos ou armazenados, durante os procedimentos do programa.

Para termos dados de um arquivo, que naturalmente está em alguma unidade de memória auxiliar (disco, fita, cartões etc.), deveremos ter também comandos que "apanhem" estes dados e os transfiram para a memória do computador; são os comandos de entrada. Naturalmente, depois do processamento deveremos ter também comandos que "coloquem" os dados resultantes em alguma unidade de memória auxiliar; são os comandos de saída.

Como primeiro comando de entrada/saída temos:

9.2.1. Comando OPEN

Por intermédio deste comando, especificamos quais são os arquivos de entrada e saída que estamos utilizando.

Este comando (como os demais comandos do COBOL) engloba vários comandos de várias sub-rotinas, gerados automaticamente pelo compilador, conforme as definições dos arquivos.

FORMATO

```
OPEN  [ INPUT  { nome-do-arquivo  [ REVERSED          ] } ... ]
                                  [ WITH NO REWIND    ]
      [ OUTPUT { nome-do-arquivo  [ WITH NO REWIND ] } ... ]
      [ I-O    { nome-do-arquivo } ... ]
```

Quando temos um arquivo de entrada, usamos a opção de INPUT, que apresenta duas opções:

a) REVERSED – usada quando estamos com um arquivo em fita magnética que queremos ler de trás para a frente (do fim para o início). Só pode ser usada para registros de tamanho fixo.

b) WITH NO REWIND – quando não queremos que a fita reenrole, ao chegar ao fim.

Para um arquivo de saída usamos a opção de OUTPUT, que apresenta uma opção – WITH NO REWIND, a qual se comporta de maneira similar à do INPUT.

Temos uma terceira opção para o comando, que é abrir um arquivo de I-O, ou seja, que servirá como entrada e como saída. A opção é muito interessante para o caso de se querer atualizar um arquivo em disco magnético, pois podemos ler o registro, processar e regravar na mesma área que ocupara antes.

Exemplo:

Considere um programa que leia um arquivo de cartões e um de fita magnética (posicionado no final), gere um disco magnético e uma listagem, teremos:

```
OPEN INPUT CARTAO FITA REVERSED
OUTPUT DISCO RELATORIO
```

É importante saber que os arquivos poderiam ter sidos colocados em outra ordem:

```
OPEN INPUT FITA REVERSED CARTAO
OUTPUT RELATORIO DISCO
```

9.2.2. Comando CLOSE

Da mesma forma que abrimos um arquivo, devemos fechá-lo quando terminar o processamento.

FORMATO

```
CLOSE  nome-do-arquivo-1        [WITH { NO REWIND }]
                                      { LOCK      }

[nome-do-arquivo-2        [WITH { NO REWIND }]  ...]
                                { LOCK      }
```

Faz com que as informações dos arquivos fiquem bloqueadas, e só possam ser recuperadas se os arquivos são "abertos" por um comando OPEN.

As opções deste comando só se aplicam a arquivos que se encontram em fita magnética, e sua utilização é:

a) WITH NO REWIND – quando, ao final do processamento, não queremos que a fita seja reenrolada, mas fique posicionada onde está.

b) WITH LOCK – quando, ao final do processamento, queremos que a fita seja reenrolada e a unidade de fita magnética, "desarmada" (se abra sozinha), para que o operador possa pegá-la e guardá-la.

Se nenhuma opção for especificada, a fita, ao final do processamento, será simplesmente reenrolada.

Exemplo:

Para fecharmos os arquivos do exemplo anterior, mas liberando a unidade da fita, teríamos:

CLOSE CARTAO FITA WITH LOCK DISCO RELATORIO

9.2.3. Comando READ

Tem a finalidade de ler os registros de um arquivo que já tenha sido aberto.

A cada READ é lido um registro lógico do arquivo, quer esteja em disco, fita, cartão etc., e transferido para a área especificada, tornando-o assim disponível.

FORMATO

READ nome-do-arquivo RECORD [*INTO* nome-da-área]

AT *END* comando imperativo

A opção INTO transfere as informações para uma outra área que não a de entrada normal (no nível 01 onde está definido o registro do arquivo em questão); isso significa que as informações estão nas duas áreas, pois, mesmo com o uso desta opção, o compilador coloca as informações na área original de leitura para depois transferi-las para a outra área, sem destruir o conteúdo da primeira.

Ao chegar o final do arquivo, o programa se desvia pela cláusula de AT END, indo executar a(s) instrução(ões) que se seguem a esta cláusula. Observe que a instrução que se segue ao AT END não pode ser de teste.

9.2.4. Comando WRITE

É usado para impressão ou gravação de arquivos. Para impressão, apresenta o seguinte formato:

WRITE nome-de-registro [*FROM* nome-de-área-1]

[{*BEFORE* / *AFTER*} ADVANCING { nome-de-área-2 LINES / inteiro LINES / nome-simbólico }]

[AT {*END-OF-FILE* / *EOP*} comando imperativo]

Sendo especificada a opção FROM, o conteúdo de "nome-de-área-1" é movido para "nome-de-registro" e este é impresso; sendo omitida, é necessário que o conteúdo de "nome-de-registro" já tenha sido preenchido de outra forma.

As opções de BEFORE (antes) e AFTER (depois) são para provocar o avanço (ADVANCING) no papel da impressora; por exemplo: quando se quer que o formulário posicione em canal-1, faz-se:

```
WRITE LINHA FROM CABECALHO AFTER ADVANCING CANAL-1.
```

No exemplo, foi usado um nome-simbólico que já deve ter sido definido na cláusula SPECIAL-NAMES da CONFIGURATION SECTION na ENVIRONMENT DIVISION, assim:

```
SPECIAL-NAMES. C01 IS CANAL-1.
```

Para o espacejamento das linhas, usa-se o seguinte código (para a opção de "inteiro"):

1 – uma linha;
2 – duas linhas;
3 – três linhas.

A opção de END-OF-FILE (se usada) testa o fim de folha através do canal-12, que é, por convenção, designado para isso. Se a condição é verdadeira (estando no canal 12), o programa executa o comando imperativo que se segue à opção.

Exemplo:

```
WRITE LINHA FROM DETALHE AFTER 1
AT EOP WRITE LINHA FROM CABECALHO AFTER CANAL-1.
```

Para gravação, apresenta o seguinte formato:

WRITE nome-de-registro [*FROM* nome-de-área].

INVALID KEY comando imperativo

Este comando funciona movendo o conteúdo de "nome-de-área" (se a opção FROM é especificada) para "nome-de-registro" e grava "nome-de-registro", que deve estar definido em nível 01 da FILE SECTION.

O sistema usará a INVALID KEY quando a área para gravação estiver completamente cheia.

Exemplo:

```
WRITE REG-DISCO FROM REG-TRABALHO
        INVALID KEY STOP 'ACABOU ESPACO'
```

9.2.5. Comando DISPLAY

A função do comando é imprimir dados de pouco volume em uma unidade de saída. Apresenta o seguinte formato:

```
DISPLAY  {literal-1}  {literal-2}  ...
         {área-1  }  {área-2  }

         [ UPON { CONSOLE      }  ]
                { SYSPCH       }
                { SYSPUNCH     }
                { SYSLST       }
                { nome-simbólico }
```

O nome-simbólico precisa ser especificado no parágrafo Special-Names da Environment Division e associado com as palavras reservadas CONSOLE, SYSPUNCH (ou SYSPCH) ou SYSLST; por exemplo:

```
SPECIAL-NAMES.
        SYSLST IS IMPRESSORA.
```

onde IMPRESSORA é nome-simbólico para SYSLST.

Quando for usado o comando DISPLAY, teremos:

```
DISPLAY 'ISTO E SO UM EXEMPLO' UPON IMPRESSORA
```

o que equivale a:

```
DISPLAY 'ISTO E SO UM EXEMPLO' UPON SYSLST.
```

Para cada unidade é assumido um tamanho máximo para o registro lógico; assim, temos:

. CONSOLE – 100 caracteres
. SYSLST – 120 caracteres
. SYSPUNCH – 72 caracteres (as posições de 73 a 80 são usadas para o nome do PROGRAM-ID)

Se o número de caracteres usados no DISPLAY for menor do que o máximo (admitido para cada opção), as posições restantes serão preenchidas com espaços, e se, ao contrário, exceder este tamanho máximo, serão utilizados tantos registros quantos forem necessários.

Se uma constante figurativa for especificada em um operando, somente *uma* representação da mesma será exibida. Assim:

```
DISPLAY ZEROS UPON CONSOLE
```

Ao executar o comando no console, teremos um, e somente um, zero.

No caso de existir mais de um operando, os dados de cada um deles são armazenados, e a saída só é dada depois que todo o registro é preenchido.

Quando a opção UPON for omitida, o sistema assumirá a unidade de SYSLST, que é sempre a unidade de impressão do sistema.

9.2.6. Comando ACCEPT

A função do comando ACCEPT é obter dados de pequeno volume provenientes da unidade lógica de entrada do sistema (SYSIPT), ou do CONSOLE.

$$\underline{ACCEPT}\ \text{nome-de-área}\ \left[\ \underline{FROM}\ \left\{\begin{array}{l}\underline{SYSIPT}\\ \underline{CONSOLE}\\ \text{nome-simbólico}\end{array}\right\}\right]$$

Os dados são lidos e o número de caracteres apropriados (de acordo com o tamanho da área), movido para dentro de "nome-de-área".

Exemplo:

```
ACCEPT DATA-ATUAL FROM CONSOLE
```

Significa que a resposta batida pelo operador no console será movida para o campo chamado DATA-ATUAL.

O "nome-simbólico" pode significar tanto SYSIPT como CONSOLE, isto é, se especificado no parágrafo Special-Names da Environment Division; por exemplo:

Se foi especificado:
CONSOLE IS TECLADO
a instrução fica:
ACCEPT MATRICULA FROM TECLADO

Se for usada a opção FROM SYSIPT (ou um nome-simbólico associado com SYSIPT), será assumido um registro de entrada com o tamanho de 80 posições. Se os dados a serem aceitos forem de tamanho inferior a 80, deverão aparecer no início do registro de entrada. Se forem de tamanho superior a 80 caracteres (e neste caso só poderão ser múltiplos de 80), deverão ser lidos tantos registros de entrada quantos forem necessários, até que a área de memória atribuída para itens de dados esteja preenchida. Caso contrário, os caracteres do último registro necessários ao complemento serão desprezados.

Quando for usada a opção *FROM CONSOLE* (ou se o nome-simbólico for associado a CONSOLE), o tamanho da área não poderá exceder 255 caracteres. Na execução do comando, o sistema gerará a mensagem "AWAITING REPLY", que será automaticamente exibida no console, e suspenderá a execução até que seja dada uma resposta que o controle do programa identifique. Aí retornará à execução do comando ACCEPT, movendo os dados obtidos para área determinada, alinhando da esquerda para a direita e ignorando a categoria da PICTURE. Se a área não for preenchida até suas posições mais à direita, poderão restar nela dados inválidos, pois a área não será "limpa" automaticamente pelo comando.

Se a opção *FROM* não for especificada, o sistema assumirá como SYSIPT, que é sempre a unidade de entrada do sistema, a leitora de cartões.

Apêndice 1

Extrato da Norma da ABNT

1. OBJETIVO

A presente Norma especifica os símbolos gráficos a serem usados nos fluxogramas para sistemas automáticos de processamento de dados, exemplificando a sua utilização.

2. CAMPO DE APLICAÇÃO

Fica entendido que os símbolos gráficos apresentados na presente Norma representam, nos fluxogramas, seja a seqüência das operações, seja a circulação dos dados e dos documentos em sistemas automáticos de processamento da informação. Esta Norma não inclui os fluxogramas do tipo esquemático, que usam figuras ou esquemas para descrever um sistema. As informações que se escrevem no interior ou ao lado de um símbolo para assegurar a identificação, descrição ou explicação deste não são abrangidas por esta Norma, que, no entanto, exemplifica a utilização dos símbolos em conjunção com esse tipo de texto informativo.

3. CONVENÇÕES

3.1. A direção geral das linhas deve ser:
– da esquerda para a direita;
– de cima para baixo.

Quando esta orientação não for respeitada, serão usadas setas. Estas serão ainda utilizadas sempre que disso puder advir uma melhor compreensão.

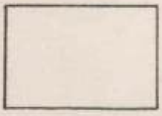
3.2. As linhas de ligação podem cruzar-se, o que significa que não existe relação lógica entre elas.

3.3. Várias linhas de ligação afluentes podem juntar-se a uma linha eferente.

3.4. As dimensões quanto à relação altura/largura devem ser encaradas apenas como sugestões; entretanto, recomenda-se não variar essas relações senão de forma tal que o símbolo seja imediatamente identificável. A unidade empregada nas sugestões de relações altura/largura é equivalente a 3 centímetros.

4. SÍMBOLOS

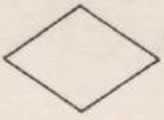
1 PROCESSAMENTO



Este símbolo representa todas as variedades de funções de processamento; por exemplo, a execução de uma operação específica ou de um grupo de operações determinadas, tendo por resultado uma alteração do valor, da forma ou da posição de uma informação.

Relação altura/largura: 2/3 : 1.

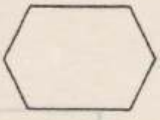
2 DECISÃO



Este símbolo representa uma operação de decisão ou de chaveamento que determina o caminho a seguir entre os vários possíveis.

Relação altura/largura: 2/3 : 1.

3 PREPARAÇÃO

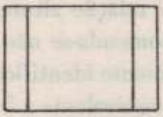


Este símbolo representa a modificação de uma instrução ou de um grupo de instruções que altera o programa em si; por exemplo, o posicionamento de uma cha-

ve, a modificação de um registro indexador e o posicionamento de um programa em seu estado inicial.

Relação altura/largura: 2/3 : 1.

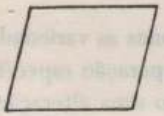
4 SUBPROGRAMA



Este símbolo representa um processamento referenciado, composto de uma ou mais operações ou de seqüências de programa definidas em outra parte, por exemplo, subprograma ou sub-rotina.

Relação altura/largura: 2/3 : 1.

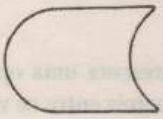
5 ENTRADA/SAÍDA



Este símbolo representa uma função de entrada/saída, por exemplo, a colocação à disposição de uma informação para o processamento (entrada) ou a gravação de uma informação processada (saída).

Relação altura/largura: 2/3 : 1.

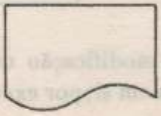
6 MEMÓRIA LIGADA AO SISTEMA – (ON LINE) –



Este símbolo representa uma função de entrada/saída que usa um tipo qualquer de memória interior ao sistema, por exemplo, um tambor magnético, um disco magnético.

Relação altura/largura: 2/3 : 1.

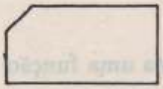
7 DOCUMENTO



Este símbolo representa uma função de entrada/saída para a qual o veículo é um documento.

Relação altura/largura: 2/3 : 1.

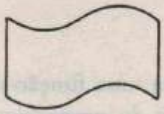
8 CARTÃO PERFURADO



Este símbolo representa uma função de entrada/saída para a qual o veículo é o cartão perfurado, incluindo os cartões para detecção de marcas (*mark sense*), cartões para leitura (ótica), cartões curtos (*stub cards*) etc.

Relação altura/largura: 1/2 : 1.

9 FITA PERFURADA



Este símbolo representa uma função de entrada/saída para a qual o veículo é uma fita de papel perfurada.

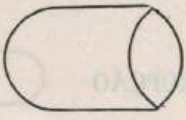
Relação altura/largura: 1/2 : 1.

10 FITA MAGNÉTICA

Este símbolo representa uma função de entrada/saída para a qual o veículo é uma fita magnética.

Diâmetro: 2/3 : 2/3

11 TAMBOR MAGNÉTICO



Este símbolo representa uma função de entrada/saída para a qual o veículo é um tambor magnético.

Relação altura/largura: 2/3 : 1.

12 DISCO MAGNÉTICO

Este símbolo representa uma função de entrada/saída para a qual o veículo é um disco magnético.

Relação altura/largura: 1 : 1/2

13 SAÍDA ILUSTRADA (DISPLAY)

Este símbolo representa uma função de entrada/saída graças à qual a informação é extraída no momento do processamento, sob uma forma ilustrada utilizável pelo homem, por meio de indicadores, telas de televisão, máquinas de escrever, traçadores de curvas (*plotters*), sistemas de áudio-respostas etc., interiores ao sistema.

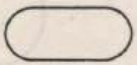
Relação altura/largura: 2/3 : 1.

14 CONECTOR

Este símbolo representa uma saída em direção a, ou uma entrada em, uma outra parte do fluxograma.

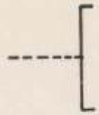
Relação altura/largura: 1/2 : 1/2.

15 INÍCIO, FIM, INTERRUPÇÃO



Este símbolo representa uma etapa no fluxograma, por exemplo, uma partida, uma parada, uma suspensão, uma espera ou uma interrupção.

Relação altura/largura: 1/2 : 1.

16 COMENTÁRIO 

Este símbolo representa a função de anotação, isto é, a inclusão de comentários descritivos ou de notas explicativas, destinadas a esclarecimento.

Apêndice 2

Resumo das Instruções

1. *Identification Division*

```
{IDENTIFICATION}          DIVISION.
{ID            }

{PROGRAM-ID.        nome-do-programa.}
 AUTHOR.            [ comentários ]...
 INSTALLATION.      [ comentário ]...
 DATE-WRITTEN.      [ comentário ]...
 DATE-COMPILED.     [ comentário ]...
 SECURITY.          [ comentário ]...
 REMARKS.           [ comentário ]...
```

2. *Environment Division*

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
```

SOURCE-COMPUTER. nome-do-computador.

OBJECT-COMPUTER. nome-do-computador.

SPECIAL-NAMES.

[nome-de-função *IS* nome-simbólico]

[*DECIMAL-POINT IS COMMA*].

INPUT-OUTPUT SECTION.

FILE-CONTROL.

{ *SELECT* nome-do-arquivo *ASSIGN* TO identificação }

[*RESERVE* { 1 / *NO* } ALTERNATE [AREA / AREAS]]

[*ACCESS* MODE *IS* { *SEQUENTIAL* / *RANDOM* }]

3. *Data Division*

DATA DIVISION.

FILE SECTION.

FD nome-do-arquivo

BLOCK CONTAINS [inteiro-1 *TO*] inteiro-2 { CHARACTERS / *RECORDS* }

RECORD CONTAINS [inteiro-1 *TO*] inteiro-2 CHARACTERS

RECORDING MODE IS modo

LABEL { *RECORD* IS / *RECORDS* ARE } { *OMITTED* / *STANDARD* }

DATA { *RECORD* IS / *RECORDS* ARE } nome-do-registro-1 [nome-do-registro-2] . . . .

a) Descrição de Registro

01-49 { nome-do-registro / *FILLER* }

{ *PICTURE* / *PIC* } IS formato

{ *JUSTIFIED* / *JUST* } RIGHT

BLANK WHEN *ZERO*

REDEFINES nome-de-dado-2

USAGE IS { *DISPLAY* / { *COMPUTATIONAL* / *COMP* } / { *COMPUTATIONAL-3* / *COMP-3* } }

88 nome-de-condição { *VALUE* IS / *VALUE* ARE } literal-1

[*THRU* literal-2]

b) Áreas Trabalho:

WORKING-STORAGE SECTION.

77 nome-de-dado

aqui repete todo o item *a* acima.

VALUE IS literal

4. *Procedure Division*

PROCEDURE DIVISION

a) Comandos Aritméticos

– Comando ADD

Formato 1:

ADD { nome-de-dado-1 / literal-1 } { nome-de-dado-2 / literal-2 } ... *TO* nome-de-dado-m

[*ROUNDED*] nome-de-dado-n [*ROUNDED*] ... [ON *SIZE ERROR* comando imperativo]

Formato 2:

ADD { nome-de-dado-1 / literal-1 } { nome-de-dado-2 / literal-2 } { nome-de-dado-3 / literal-3 } ... *GIVING*

nome-de-dado-m [*ROUNDED*] [ON *SIZE ERROR* comando imperativo]

Formato 3:

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

– Comando SUBTRACT

Formato 1:

SUBTRACT { nome-de-dado-1 / literal-1 } { nome-de-dado-2 / literal-2 } ...

FROM nome-de-dado-m [*ROUNDED*]

[nome-de-dado-n [*ROUNDED*]]

[ON *SIZE ERROR* comando imperativo]

Formato 2:

```
SUBTRACT { nome-de-dado-1 } { nome-de-dado-2 } ...
         { literal-1      } { literal-2      }

  FROM { nome-de-dado-m } GIVING nome-de-dado-n
       { literal-m      }
[ROUNDED] [ON SIZE ERROR comando imperativo]
```

Formato 3:

```
SUBTRACT { CORR          }
         { CORRESPONDING } nome-de-dado-1

  FROM nome-de-dado-2 [ROUNDED]

[ON SIZE ERROR comando imperativo]
```

– Comando MULTIPLY

Formato 1:

```
MULTIPLY { nome-de-dado-1 } BY nome-de-dado-2
         { literal-1      }

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

Formato 2:

```
MULTIPLY { nome-de-dado-1 } BY { nome-de-dado-2 }
         { literal-1      }    { literal-2      }

GIVING nome-de-dado-3 [ROUNDED]

[ON SIZE ERROR comando imperativo]
```

– Comando DIVIDE

Formato 1:

DIVIDE { nome-de-dado-1 / literal-1 } *INTO* nome-de-dado-2

[*ROUNDED*] [ON *SIZE ERROR* comando imperativo]

Formato 2:

DIVIDE { nome-de-dado-1 / literal-1 } { *INTO* / *BY* } { nome-de-dado-2 / literal-2 }

GIVING nome-de-dado-3 [*ROUNDED*]

[*REMAINDER* nome-de-dado-4]

ON *SIZE ERROR* comando imperativo

– Comando COMPUTE

COMPUTE nome-de-dado-1 [*ROUNDED*] =

{ nome-de-dado-2 / literal-2 / expressão-aritmética }

[ON *SIZE ERROR* comando imperativo]

b) Comando de DECISÃO e DESVIO

– Comando IF

IF condição [THEN] { comando-1 / *NEXT SENTENCE* } { *ELSE* / *OTHERWISE* }

{ comando-2 / *NEXT SENTENCE* }

– Comando GO TO

Formato 1:

GO TO nome-de-procedimento

Formato 2:

GO TO nome-de-procedimento-1

[nome-de-procedimento-2] . . .

DEPENDING ON nome-de-dado

– Comando PERFORM

Formato 1:

PERFORM nome-de-procedimento-1

[*THRU* nome-de-procedimento-2]

Formato 2:

PERFORM nome-de-procedimento-1

[*THRU* nome-de-procedimento-2]

{nome-de-dado / inteiro} *TIMES*

Formato 3:

PERFORM nome-de-procedimento-1

[*THRU* nome-de-procedimento-2]

UNTIL condição

Formato 4:

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

VARYING nome-de-dado-1 *FROM* {nome-de-dado-2 / literal-2}

BY { nome-de-dado-3 / literal-3 } *UNTIL* condição-1

[*AFTER* nome-de-dado-4 *FROM* { nome-de-dado-5 / literal-5 }

BY { nome-de-dado-6 / literal-6 } *UNTIL* condição-2

[*AFTER* nome-de-dado-7 *FROM* { nome-de-dado-8 / literal-8 }

BY { nome-de-dado-9 / literal-9 } *UNTIL* condição-3]]

– Comando EXIT

nome-de-procedimento. *EXIT*.

– Comando STOP

STOP { *RUN* / literal }

c) Comando de Manipulação, Entrada e Saída de Dados

– Comando MOVE

Formato 1:

MOVE { nome-de-dado-1 / literal } *TO* nome-de-dado-2

[nome-de-dado-3] ...

Formato 2:

MOVE { *CORRESPONDING* / *CORR* } nome-de-dado-1 *TO* nome-de-dado-2

– Comando OPEN

```
OPEN [ INPUT { nome-do-arquivo [ REVERSED / WITH NO REWIND ] } ... ]
     [ OUTPUT { nome-do-arquivo [ WITH NO REWIND ] } ... ]
     [ I-O { nome-do-arquivo } ... ]
```

– Comando CLOSE

```
CLOSE nome-do-arquivo-1 [ WITH { NO REWIND / LOCK } ]
[ nome-do-arquivo-2 WITH { NO REWIND / LOCK } ... ]
```

– Comando READ

```
READ nome-do-arquivo RECORD [ INTO nome-de-área ] AT END comando
                                                   imperativo
```

– Comando WRITE

Formato 1:

```
WRITE nome-de-registro FROM nome-de-área-1
[ { BEFORE / AFTER } ADVANCING { nome-de-área-2 LINES
                               { inteiro LINES
                               { nome-simbólico } ]
[ AT { END-OF-FILE / EOP } comando imperativo ]
```

Formato 2:

```
WRITE nome-de-registro [ FROM nome-de-área ]
  INVALID KEY comando imperativo
```

– Comando DISPLAY

```
DISPLAY { literal-1 } { literal-2 } ...
        { área-1    } { área-2    }

[ UPON { CONSOLE      } ]
       { SYSPCH       }
       { SYSPUNCH     }
       { SYSLST       }
       { nome-simbólico }
```

– Comando ACCEPT

```
ACCEPT nome-de-área [ FROM { SYSIPT       } ]
                           { CONSOLE      }
                           { nome-simbólico }
```

Apêndice 3

Lista das Palavras Reservadas

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BASIS
BEFORE
BEGINNING
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
CBL
CD
CF
CH
CHANGED
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COM-REG
COMMA
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORE-INDEX
CORR
CORRESPONDING
COUNT
CSP
CURRENCY
CURRENT-DATE
CYL-INDEX
CYL-OVERFLOW
C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DISABLE
DISP
DISPLAY
DISPLAY-ST
DISPLAY-n
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
EJECT
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENDING
ENTER

ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
ESI
EVERY
EXAMINE
EXCEEDS
EXCEPTION
EXHIBIT
EXIT
EXTEND
EXTENDED-SEARCH

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GO
GOBACK
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSERT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY
LABEL
LABEL-RETURN
LAST
LEADING
LEAVE
LEFT
LENGTH
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MASTER-INDEX
MEMORY
MERGE
MESSAGE
MODE
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOMINAL
NOT
NOTE
NSTD-REELS
NUMBER
NUMERIC
NUMERIC-EDITED

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

PAGE
PAGE-COUNTER
PASSWORD
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONING
POSITIVE
PRINT-SWITCH
PROCEDURE
PROCEDURES
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
READY
RECEIVE
RECORD
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REFERENCES
REEL
RELATIVE
RELEASE
RELOAD
REMAINDER
REMARKS
REMOVAL
RENAMES
REORG-CRITERIA
REPLACING
REPORT
REPORTING
REPORTS
REREAD
RERUN
RESERVE
RESET
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENTIAL
SERVICE
SET
SIGN
SIZE
SKIP1
SKIP2
SKIP3
SORT

SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MERGE
SORT-MESSAGE
SORT-MODE-SIZE
SORT-OPTION
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSOUT
SYSPCH
SYSPUNCH
S01
S02
S03
S04
S05

TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIME-OF-DAY
TIMES
TO
TOP
TOTALED
TOTALING
TRACE
TRACK
TRACK-AREA
TRACK-LIMIT
TRACKS
TRAILING
TRANSFORM
TYPE

UNEQUAL
UNIT
UNSTRING
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
UPSI-0
UPSI-1
UPSI-2
UPSI-3
UPSI-4
UPSI-5
UPSI-6
UPSI-7
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WHEN-COMPILED
WITH
WORDS
WORKING-STORAGE
WRITE
WRITE-ONLY
WRITE-VERIFY

ZERO
ZEROES
ZEROS

Bibliografia

1. HIRSCH, SEYMOUR C. – COBOL: *A Simplified Approach.*
2. FLORES, IVAN – *Data Structure and Management.*
3. CONDON, ROBERT J. – *Data Processing Systems Analysis & Design.*
4. GC28-6394-6 – IBM DOS US, *Full American National Standard COBOL.*
5. McCRACKEN, DANIEL D. – *A Guide to COBOL Programming.*
6. BERTINI, MARIE. THÉRÊSE ET TALLINEAU, YVES – LE COBOL STRUCTURÊ: *Un Modèle de Programmation.*

Impressão e Acabamento
IONELI - Indústrias Gráficas Ltda.
IONELI Rua Viúva Cláudio, 81 - Tel. 201-2644

FLUXOGRAMAS E PROGRAMAÇÃO COBOL

Gusman / Vasconcellos

2ª edição

Este livro tem por finalidade apresentar ao leitor as técnicas necessárias para a produção de programas de computador na linguagem COBOL.

O texto é iniciado pela apresentação de fluxogramas, com suas técnicas básicas, e evolui de modo a permitir uma transição simples e fácil para a linguagem.

SUMÁRIO: Diagramas de Fluxos, A Transição, Introdução à Linguagem, Divisões de Especificações, Estruturas de Arquivos, Estruturas de Dados, Divisão de Procedimentos, Comando de Decisão e Desvio, Comando de Manipulação, Entrada e Saída de Dados.

MAIS UM LANÇAMENTO DA

LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A.

ISBN: 85.216.0427-0