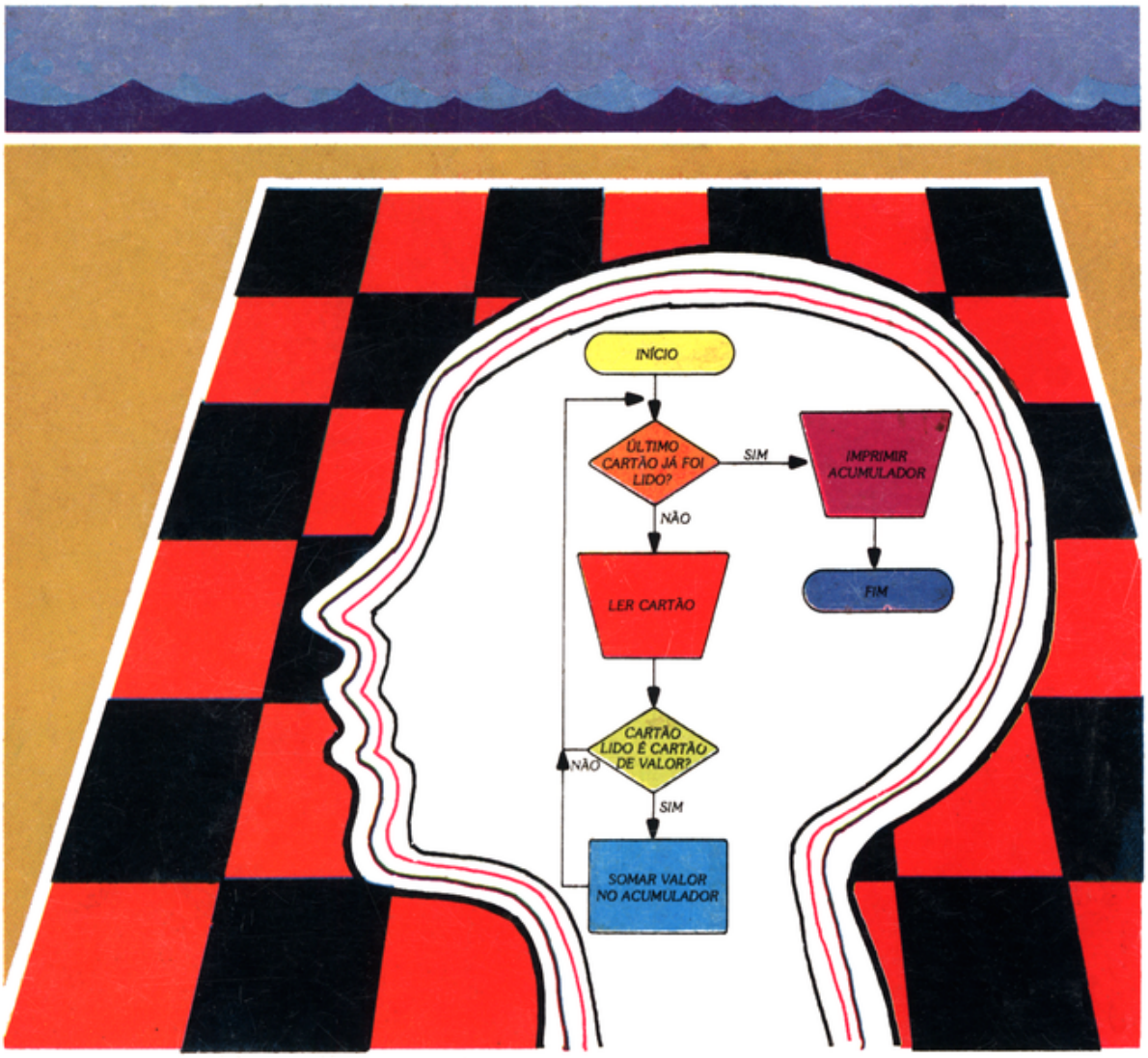


Programação Fortran

Eli Rozendo Moreira dos Santos
Professor de Processamento de Dados
Mestrado de Comunicação-Informática
e Cibernética Pela UFRJ

EDIOURO/10327

Programação Fortran

Eli Rozendo Moreira dos Santos

Professor de Processamento de Dados
Mestrado de Comunicação-Informática
e Cibernética Pela UFRJ

Programação Fortran

As nossas edições reproduzem
integralmente os textos originais

EDITORA TECNOPRINT

Para a minha querida
Neli

Índice

9

Dados de Entrada e de Saída 109

10

Desvios 151

Programação Fortran

1

Generalidades

O que É um Programa

□□ O computador eletrônico é capaz de efetuar operações aritméticas, comparações de valores e tarefas necessárias à execução de um serviço, sem a intervenção do homem. Para isso, no entanto, o computador precisa ser "instruído" previamente. O conjunto de todas as instruções fornecidas ao computador para que ele possa resolver um determinado problema é chamado de PROGRAMA.

O que É um Programador

□□ Alguém precisa indicar ao computador, por meio de um programa, quais são todas as etapas necessárias para resolver um problema. A pessoa que cria o programa, isto é, a pessoa que instrui a máquina, é conhecida como PROGRAMADOR.

O que É Linguagem de Programação

□□ O computador identifica as instruções que lhe são dadas, por meio de códigos. As regras para utilização desses códigos, conhecidos como códigos de máquina, são muito complexas, ficando difícil para o programador fornecer as instruções por meio desses códigos de máquina. Para tornar mais fácil, rápida e eficiente a tarefa do programador, as instruções não são dadas ao computador diretamente através desses códigos de máquina. Foram criadas várias formas de instruir o computador, por meio de códigos e regras mais simples. O conjunto destes códigos e regras simplificadas é conhecido como linguagem de programação.

Linguagem de Programação

□□ Foram criadas várias linguagens de programação. As mais empregadas no Brasil são COBOL, FORTRAN e ASSEMBLER.

□□ O FORTRAN, abreviação de Formula Translation, é muito empregado em processamento científico. O Assembler tem seu emprego restringido ao tipo de computador para o qual foi feito o programa. Por exemplo, se foi feito um programa em Assembler para um computador IBM, este programa não pode ser utilizado em computadores Burroughs.

□□ O COBOL, abreviação de Common Business Oriented Language, é uma linguagem de programação dirigida, principalmente, para o processamento de problemas comerciais.

O que É FORTRAN

□□ FORTRAN é uma abreviação de FORmula TRANslation. O Fortran é uma linguagem desenvolvida inicialmente pela IBM entre 1954 e 1957. O seu objetivo é facilitar a programação dos problemas científicos e técnicos das mais diversas áreas tais como Matemática, Engenharia, Física, Estatística, Economia, etc. Embora tenha sido desenvolvido inicialmente pela IBM, ele é aceito pela maioria dos computadores atuais. Às vezes um programa feito para um determinado tipo de computador necessita de pequenas adaptações para ser processado em outro computador. Mas nem sempre isso acontece. Um programa FORTRAN escrito, por exemplo, para um computador IBM-/360 pode rodar sem nenhuma alteração num computador IBM-/370, num FACOM-M160 ou num FACOM-M200.

□□ Como os códigos de máquina dos diversos computadores são completamente diferentes uns dos outros, o programa escrito pelo programador terá de ser processado por um programa especial chamado "compilador" (explicado logo a seguir) que transformará a linguagem usada pelo programador, em códigos de máquinas usados pelo computador no qual o problema será resolvido.

□□ O FORTRAN, desde a sua versão original, vem sendo periodicamente melhorado e cada versão nova que surge recebe um nome especial. Atualmente o FORTRAN mais usado é o FORTRAN IV, motivo pelo qual o texto deste livro nele se baseia.

O que É um Compilador

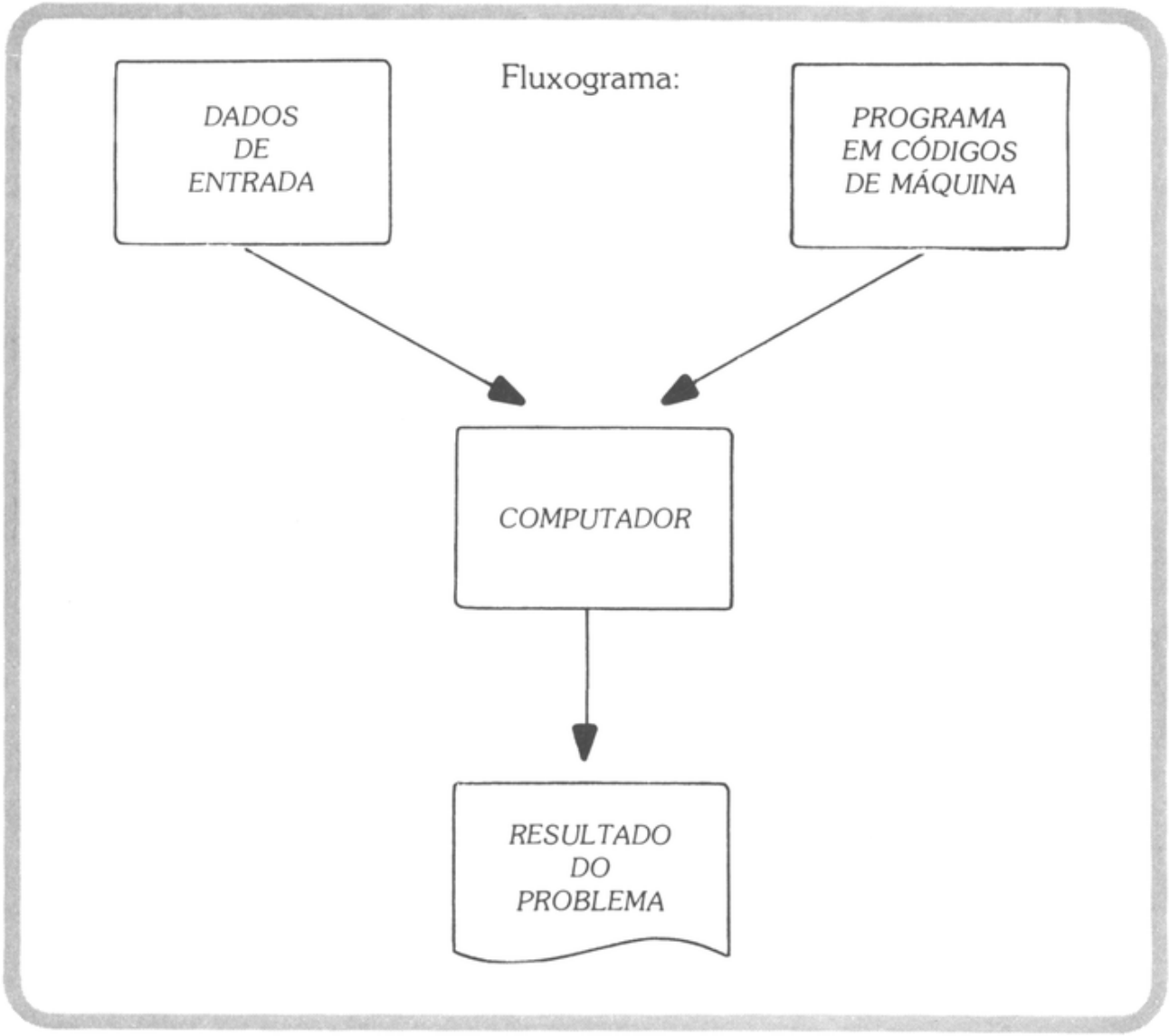
□□ A máquina para resolver um problema, como vimos, precisa ser intruída por meio de códigos de máquina.

□□ O programador, para sua facilidade, escreve os programas em linguagens tais como ASSEMBLER, FORTRAN e COBOL, linguagens simbólicas, as quais não empregam os códigos de máquina. Há necessidade, portanto, de se fazer uma tradução do programa escrito numa linguagem simbólica para uma lin-

guagem de máquina, isto é, há necessidade de transformar o programa escrito pelo programador para uma forma que empregue os códigos de máquina.

□□ O programador não precisa se preocupar com essa tradução. O computador faz essa tradução sozinho, pois existem programas cuja função é, exatamente, fazer esse tipo de transformação. Esses programas são os "compiladores". Um compilador FORTRAN, por exemplo, é um programa que tem por função transformar um programa escrito na linguagem FORTRAN para linguagem de máquina.

Fluxograma:



Fluxogramas

□□ Se você está familiarizado com fluxogramas passe para o Capítulo 3. Caso contrário, acompanhe as explicações dadas a seguir.

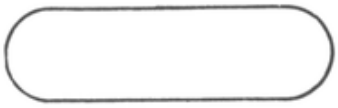
□□ Um programa é composto de muitas instruções. Estas instruções têm de ser fornecidas ao computador numa seqüência lógica rigorosa. Basta, às vezes, uma única instrução, entre milhares, ficar fora de seqüência para que o resultado do programa não seja o desejado pelo programador. Para facilitar o trabalho de colocação das instruções na ordem desejada, foi criada uma linguagem gráfica, chamada "fluxograma".

Os Símbolos Gráficos do Fluxograma

□□ O fluxograma tem símbolos próprios a fim de evitar que cada programador crie os seus próprios símbolos e convenções. Com a padronização dos símbolos gráficos usados nos fluxogramas, um programador pode facilmente analisar um fluxograma feito por outro programador. Os fluxogramas não são usados apenas em programação de computador. Eles são também muito empregados em organização e métodos, análise de sistemas, etc.

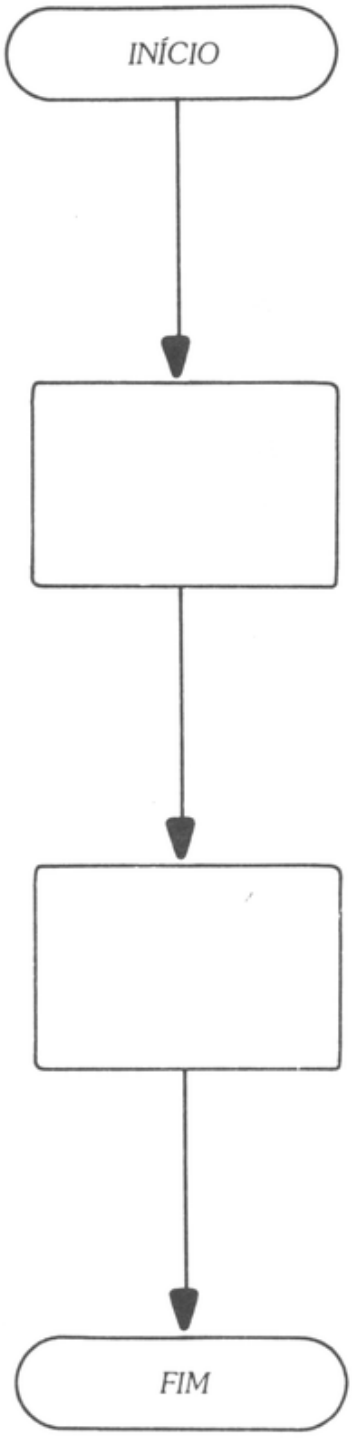
□□ Os fluxogramas usados em programação de computador também são conhecidos como "diagrama de blocos".

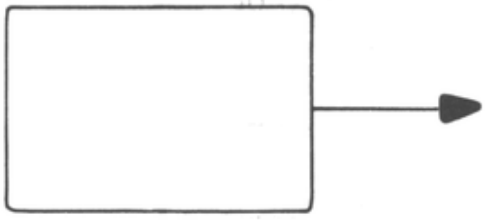
□□ Indicaremos a seguir os símbolos usados nos fluxogramas de programação de computador.



Este símbolo indica início ou fim do fluxograma.

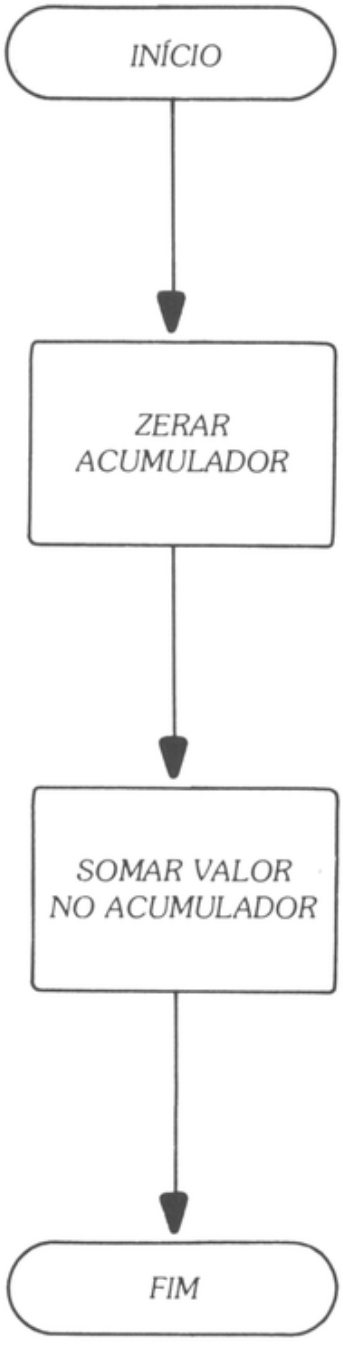
Exemplo 1

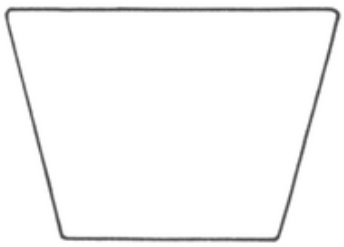




Este símbolo indica um processamento

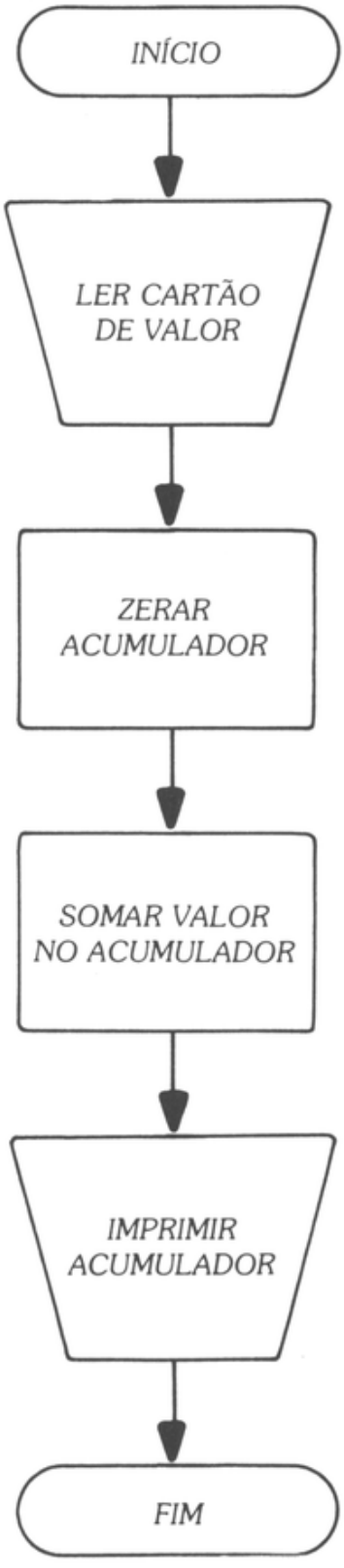
Exemplo 2

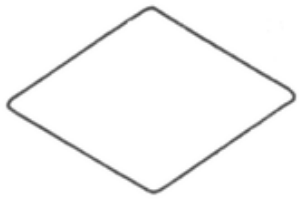




Este símbolo indica uma função de entrada/saída.

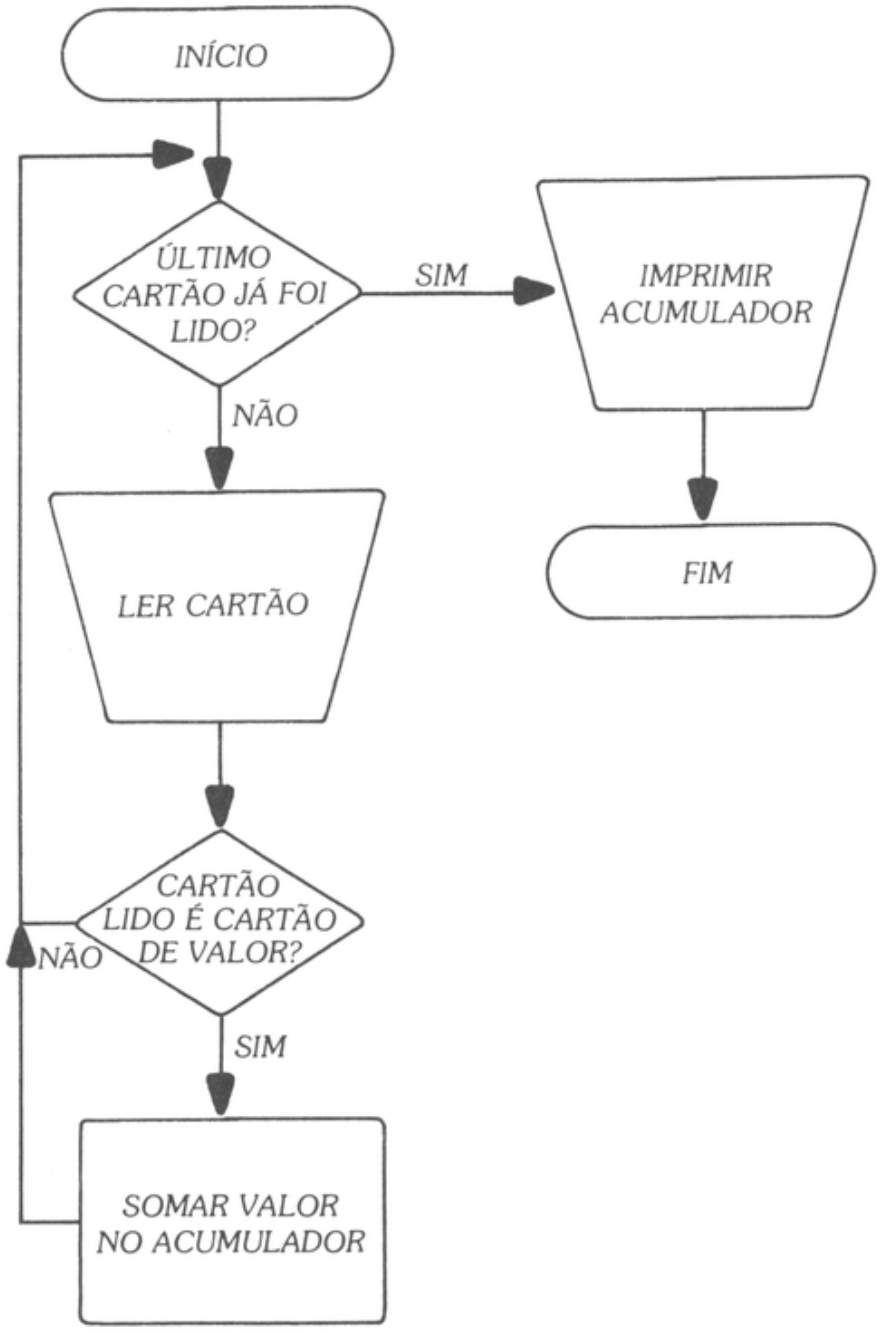
Exemplo 3

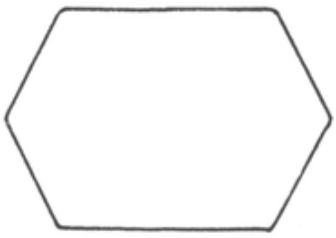




Este símbolo indica uma tomada de decisão.

Exemplo 4

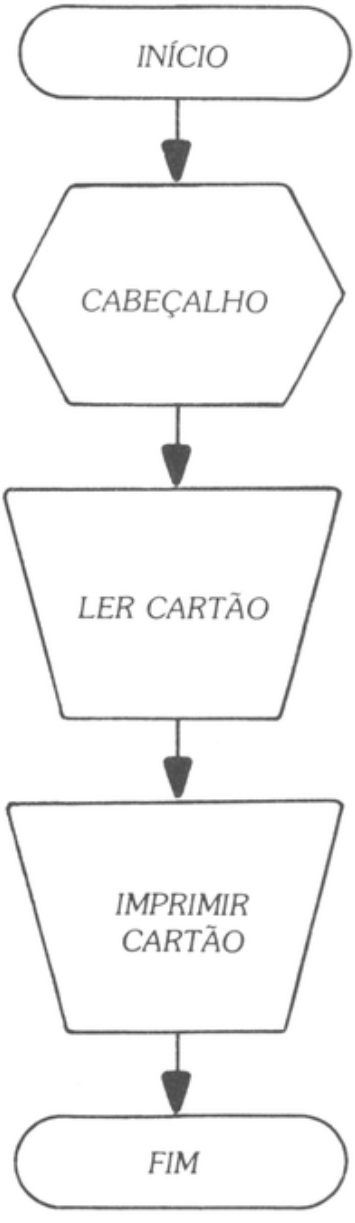


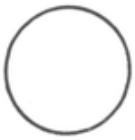


Este símbolo indica uma rotina completa do programa que não foi detalhada no fluxograma. Por exemplo, no fluxograma apresentado aqui, a rotina de cabeçalho não foi detalhada, mas sabemos que o cabeçalho geralmente não pode ser feito com apenas uma instrução. O cabeçalho implica em numeração de folhas, impressão de datas, nomes de relatórios, etc.

Exemplo 5

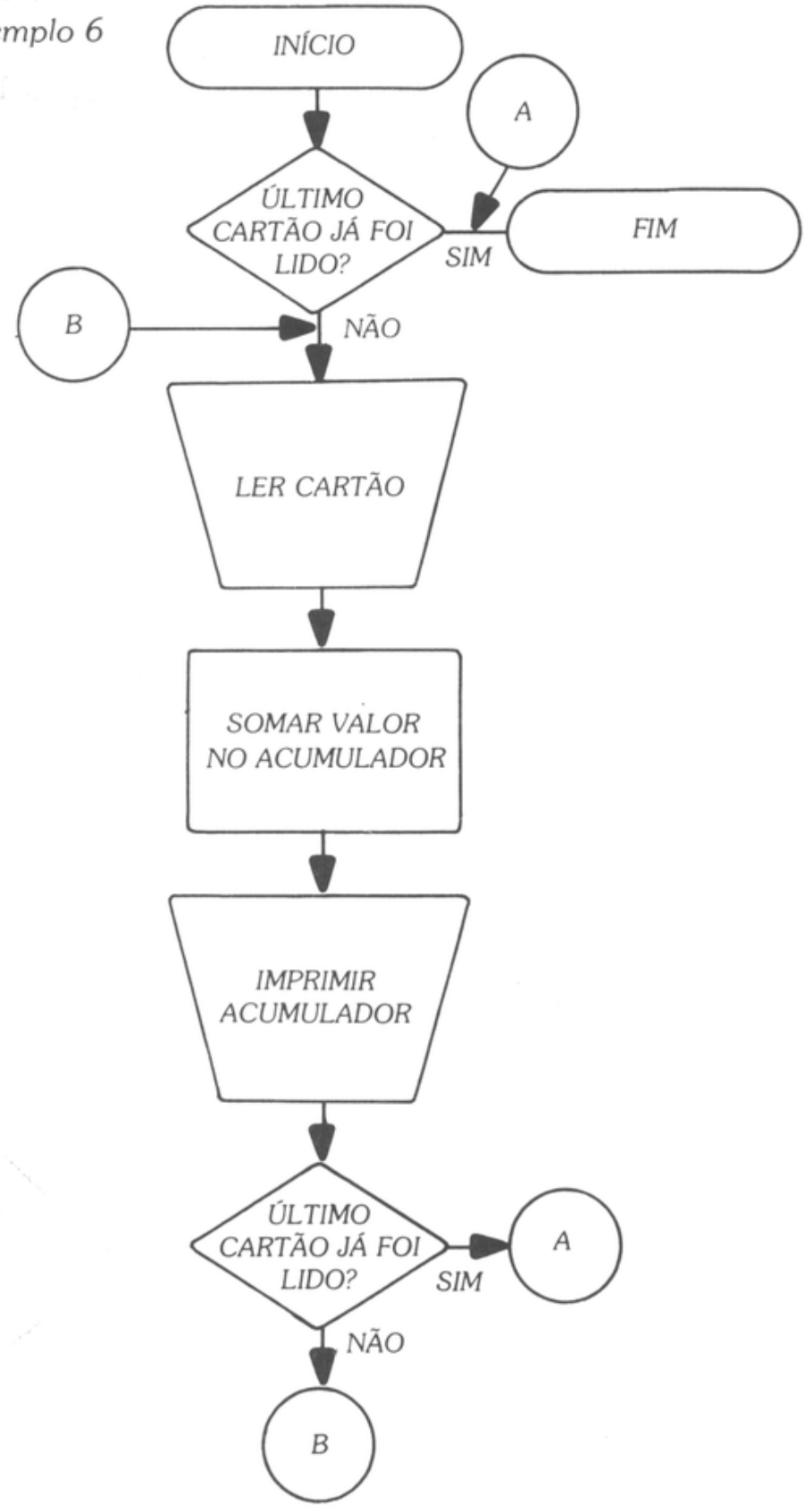
O símbolo subentende todas estas etapas.

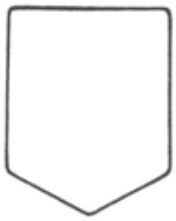




Este símbolo indica uma conexão.

Exemplo 6

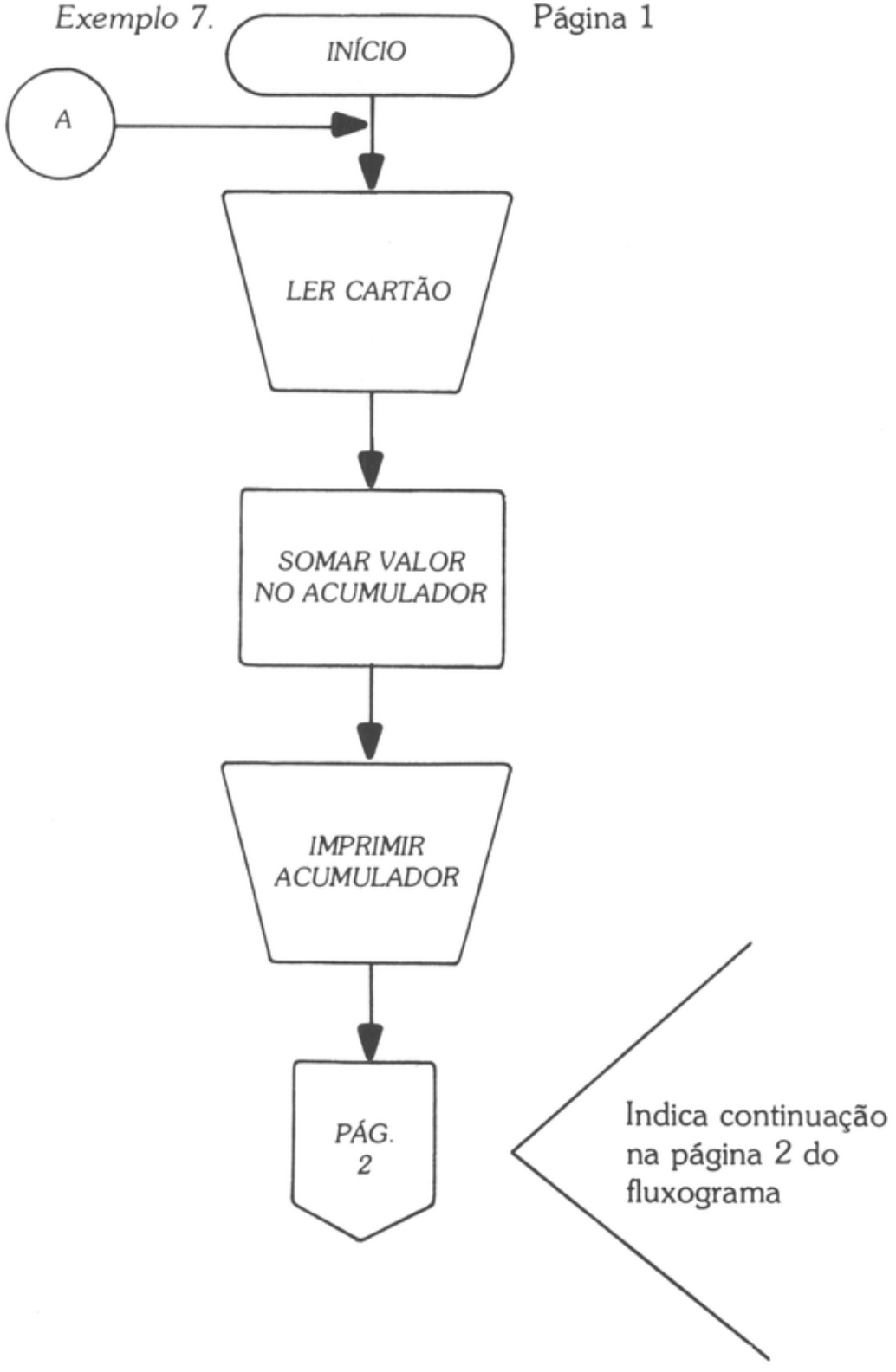




Este símbolo indica saída de uma página ou entrada em uma página.

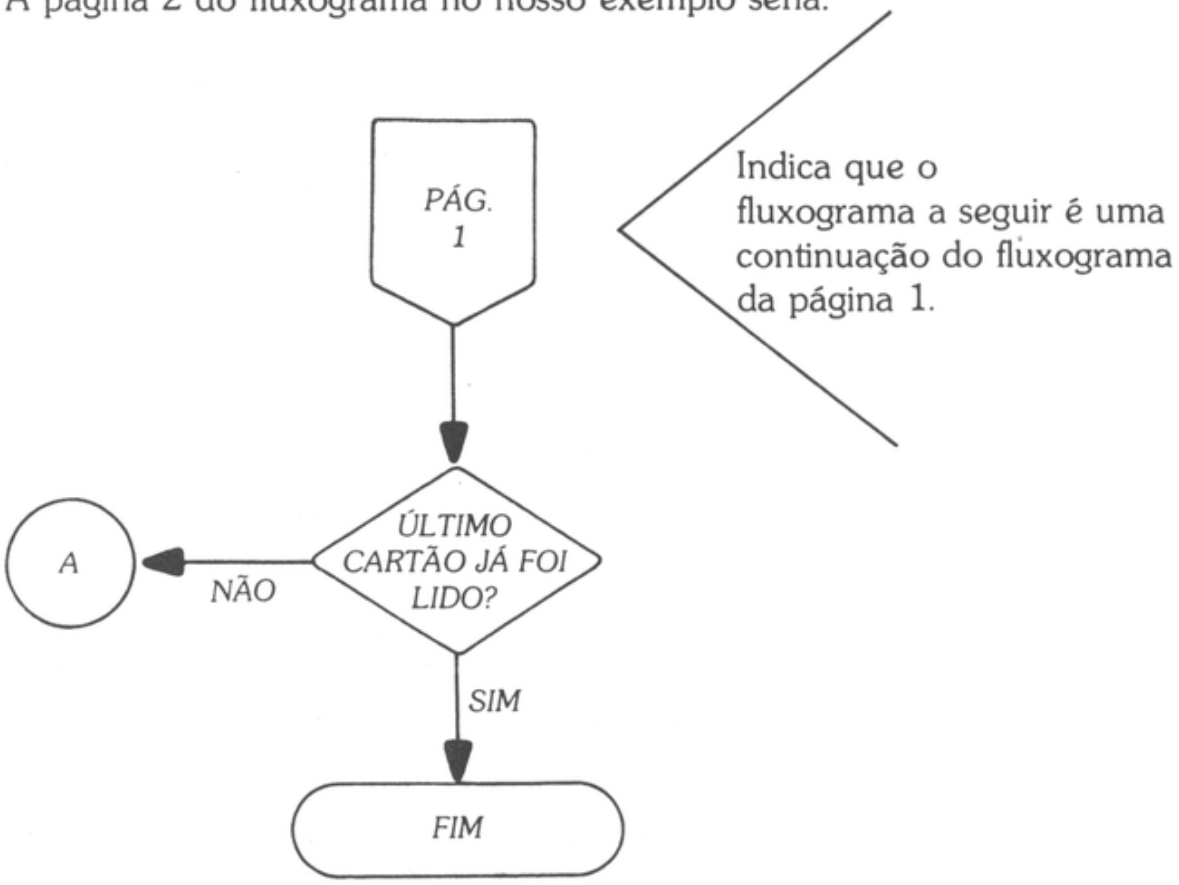
Suponhamos que a primeira página de um fluxograma fosse a seguinte:

Exemplo 7.



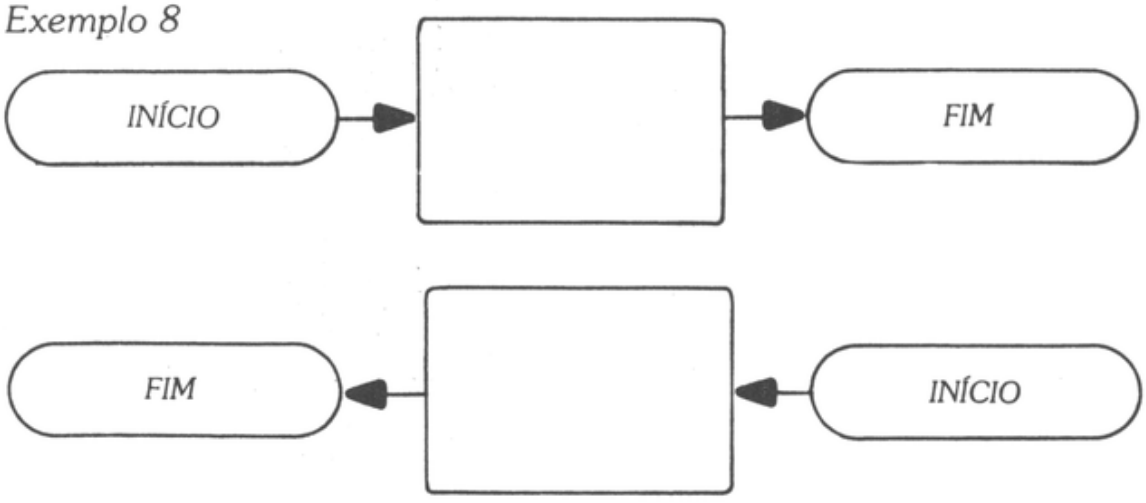
Exemplo 7 (continuação)

A página 2 do fluxograma no nosso exemplo seria:

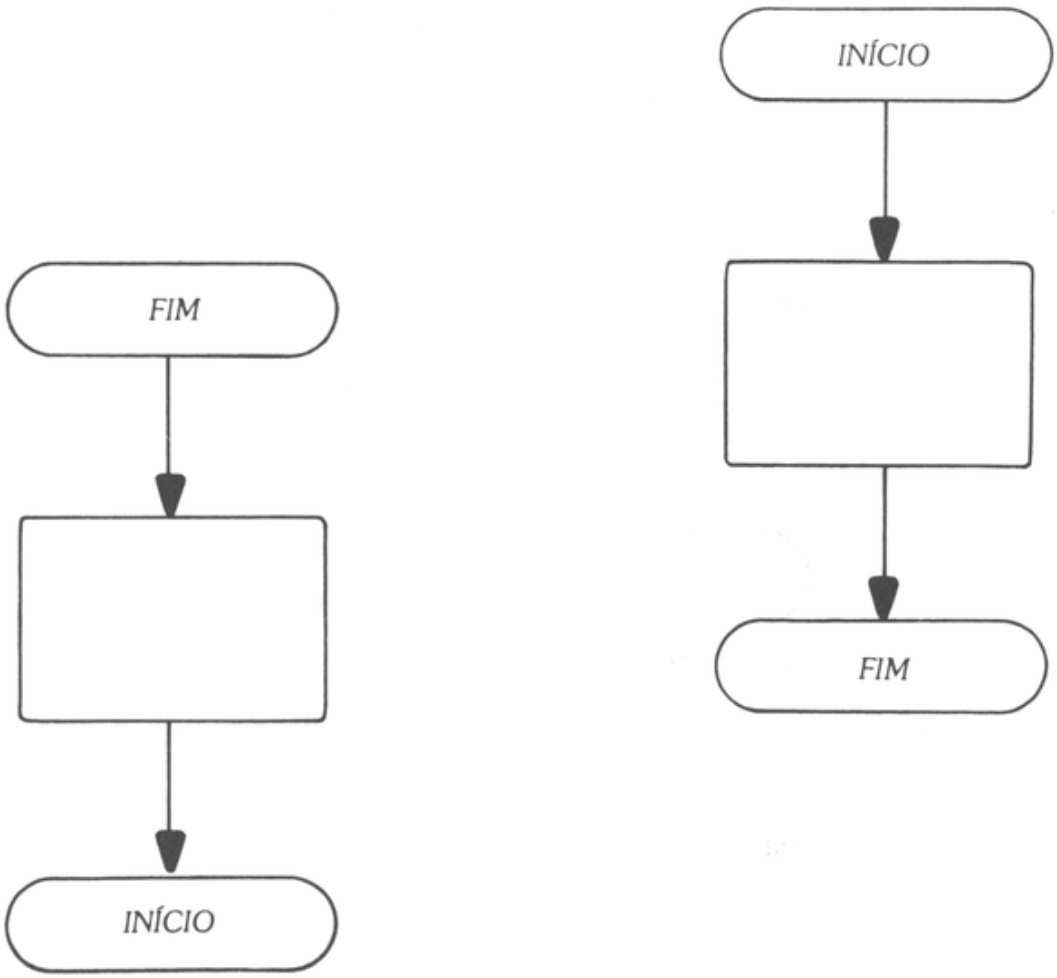


Estes símbolos indicam a direção do fluxo.

Exemplo 8

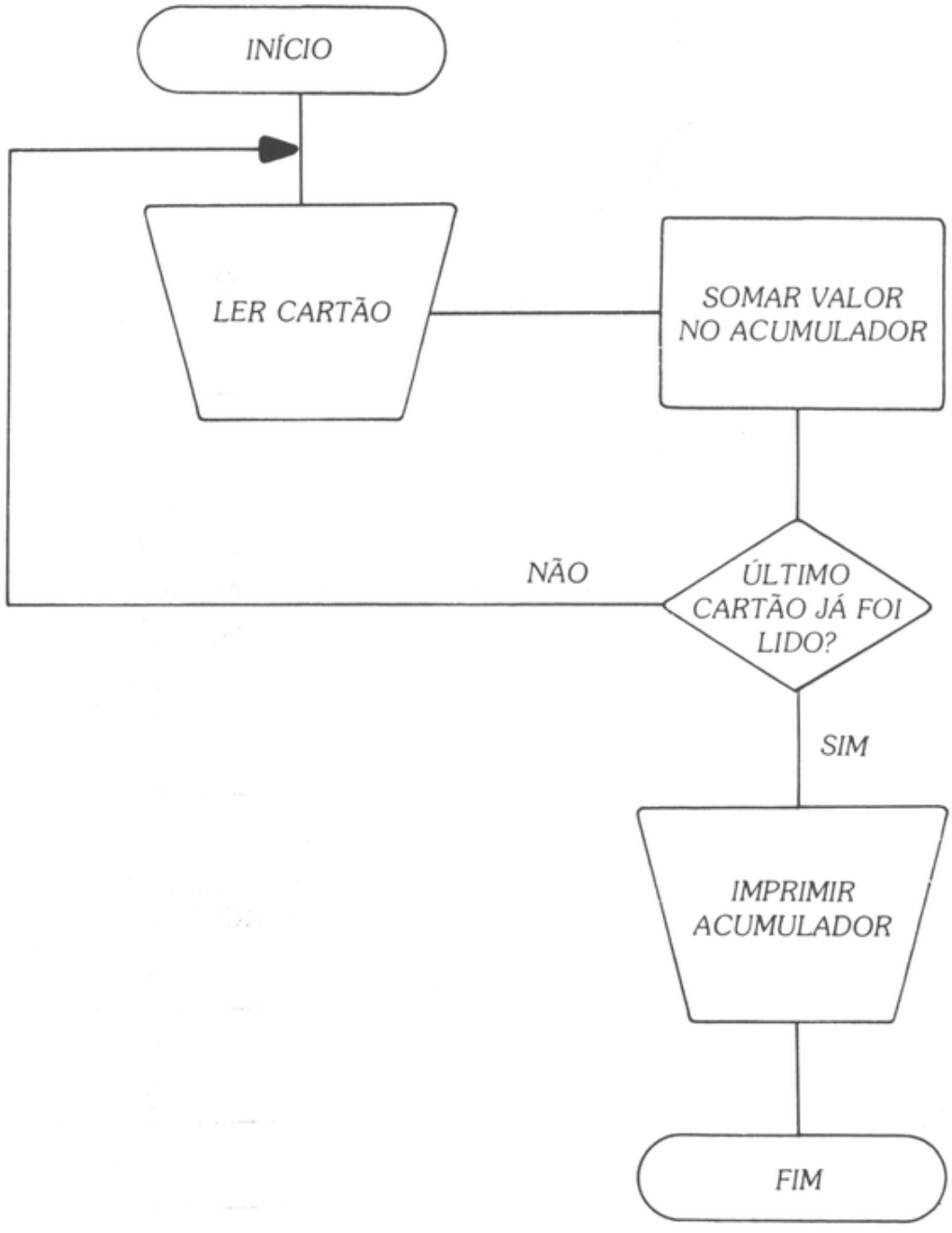


Exemplo 8 (continuação)

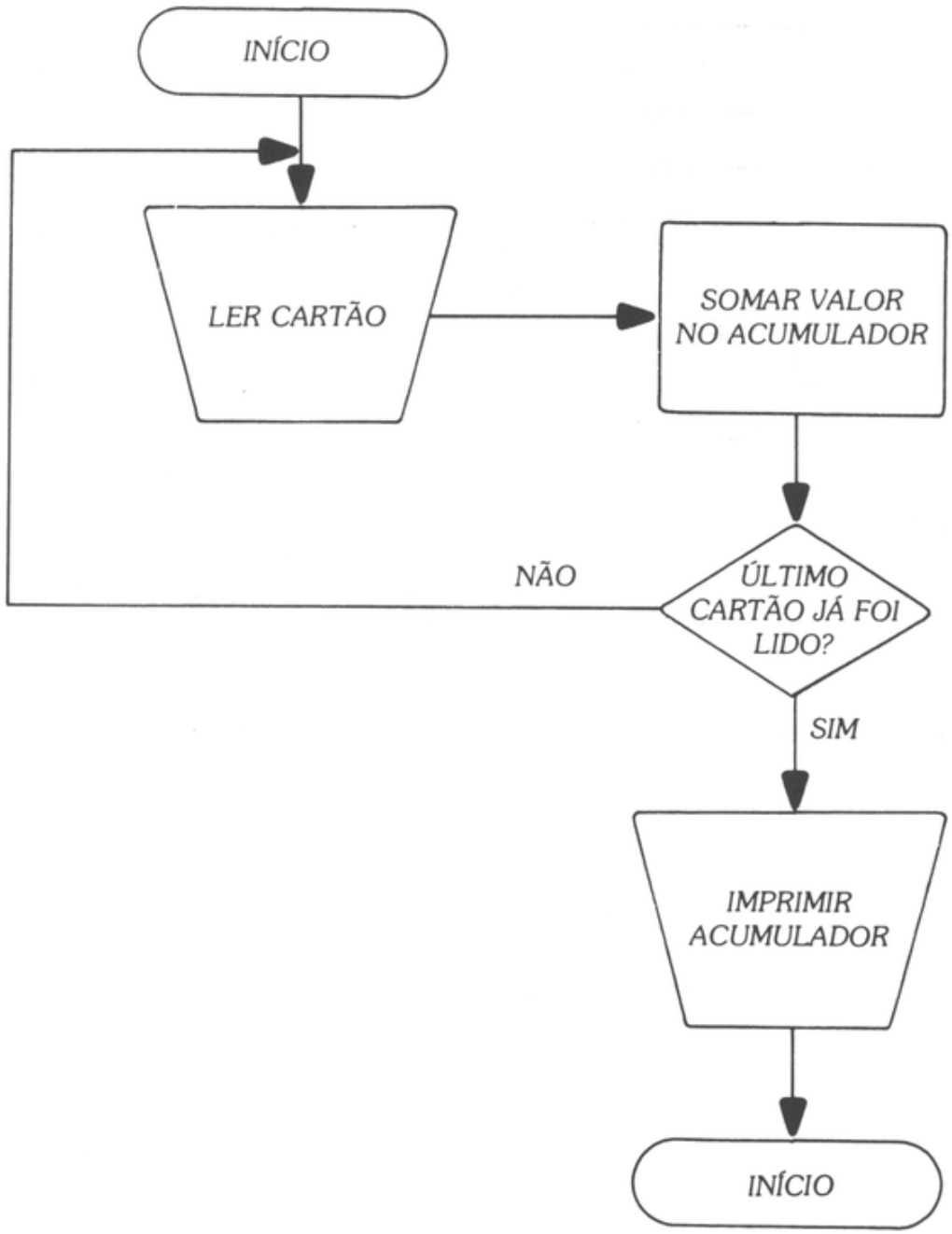


□□ A direção normal do fluxo no fluxograma é da esquerda para a direita ou de cima para baixo. Quando a direção for normal, isto é, da esquerda para a direita ou de cima para baixo, não há necessidade dos símbolos ► ou ▼ Assim, os fluxogramas dos exemplos 9 e 10 se equivalem.

Exemplo 9


INÍCIO
LER CARTÃO
SOMAR VALOR NO ACUMULADOR
NÃO
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
IMPRIMIR ACUMULADOR
FIM

Exemplo 10


INÍCIO
LER CARTÃO
SOMAR VALOR NO ACUMULADOR
ÚLTIMO CARTÃO JÁ FOI LIDO?
NÃO
SIM
IMPRIMIR ACUMULADOR
INÍCIO

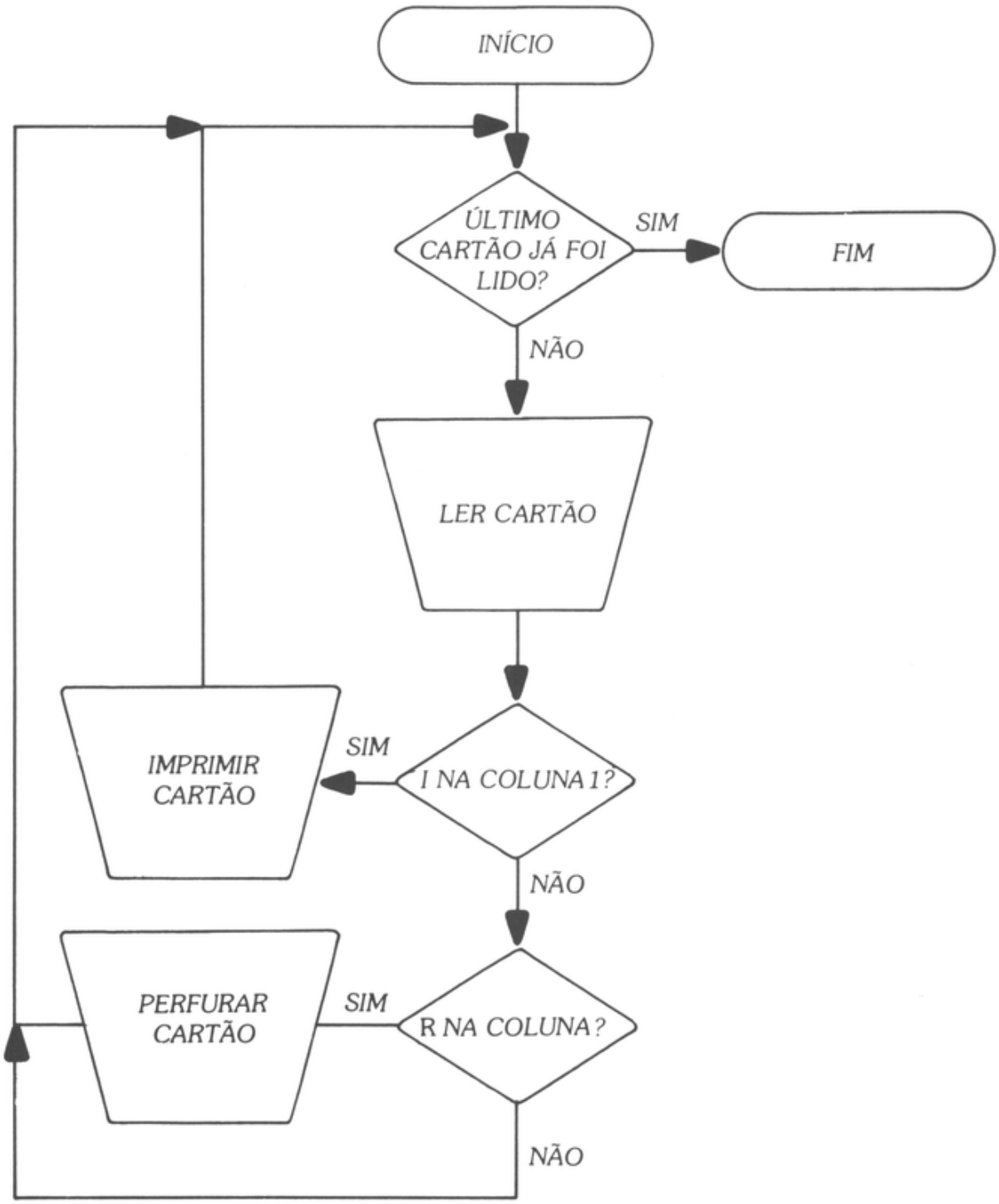
2

Fluxogramas Para Programas Escritos em FORTRAN

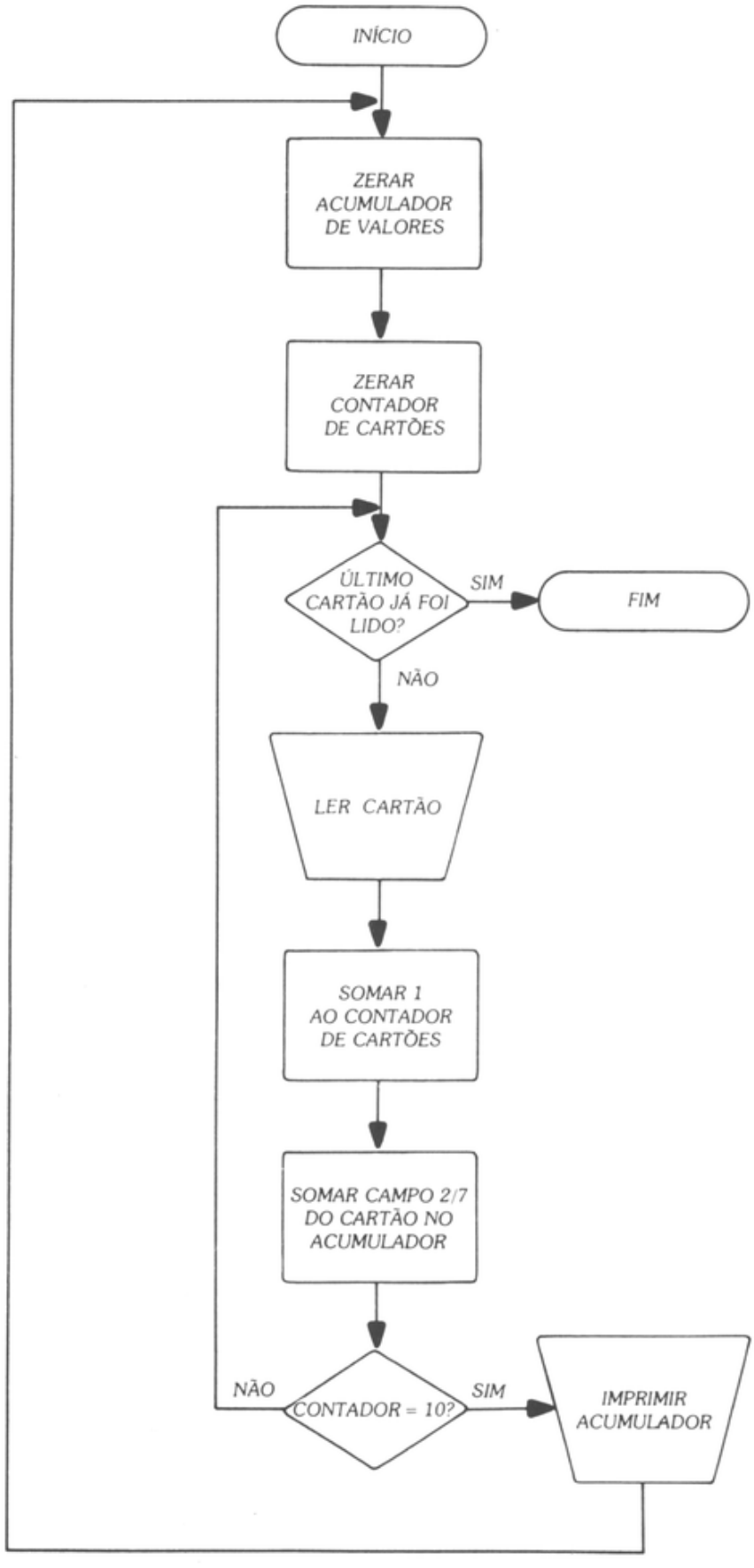
* * *

Exercícios

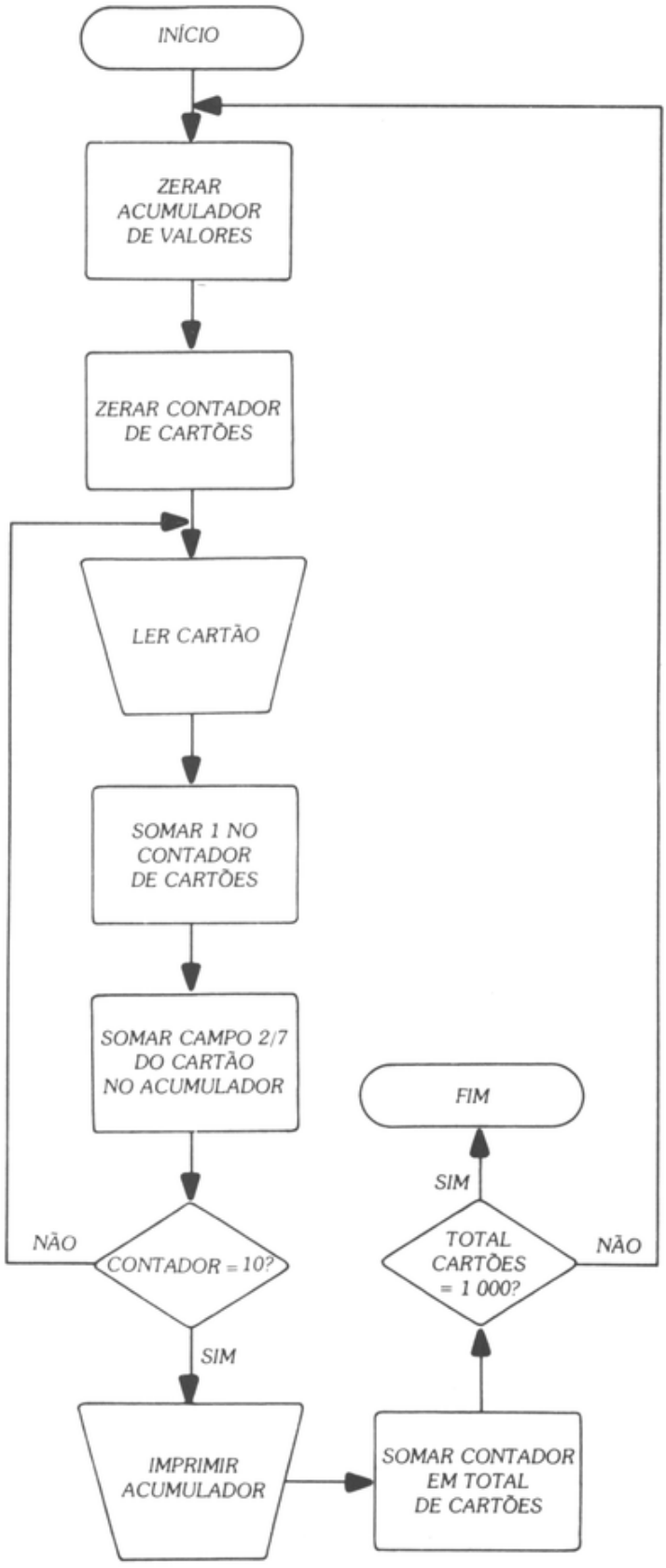
1) Fazer fluxograma do programa abaixo:
Ler uma massa de cartões. Todo cartão é identificado por uma letra perfurada na coluna 1. Quando o cartão tiver uma letra "I" perfurada na coluna 1, imprimir o conteúdo do cartão. Quando a perfuração da coluna 1 for a letra *R*, perfurar outro cartão idêntico ao cartão lido. Não haverá nenhum processamento para os cartões cujas letras perfuradas na coluna 1 forem diferentes de *I* ou *R*.
Compare o seu fluxograma com o apresentado na página seguinte.


INÍCIO
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
FIM
NÃO
LER CARTÃO
IMPRIMIR CARTÃO
SIM
I NA COLUNA 1?
NÃO
PERFURAR CARTÃO
SIM
R NA COLUNA?
NÃO

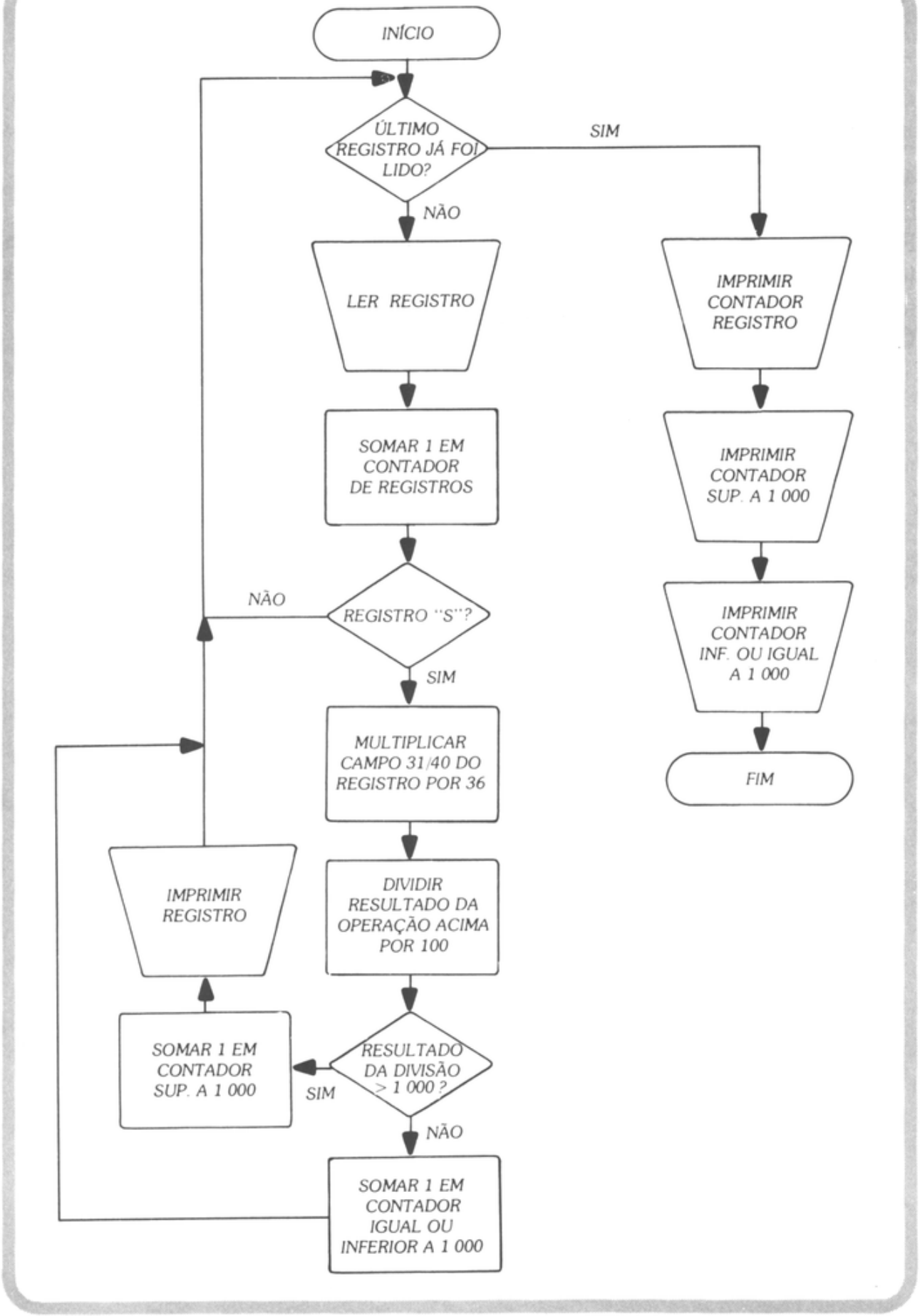
2) Fazer o fluxograma do programa abaixo:
Ler uma massa de 1 000 cartões. A cada 10 cartões lidos, imprimir o total da soma dos valores do campo 2/7 de todos os 10 cartões.
Compare o seu fluxograma com o apresentado na página seguinte.


INÍCIO
ZERAR ACUMULADOR DE VALORES
ZERAR CONTADOR DE CARTÕES
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
FIM
NÃO
LER CARTÃO
SOMAR 1 AO CONTADOR DE CARTÕES
SOMAR CAMPO 2/7 DO CARTÃO NO ACUMULADOR
NÃO
CONTADOR = 10?
SIM
IMPRIMIR ACUMULADOR

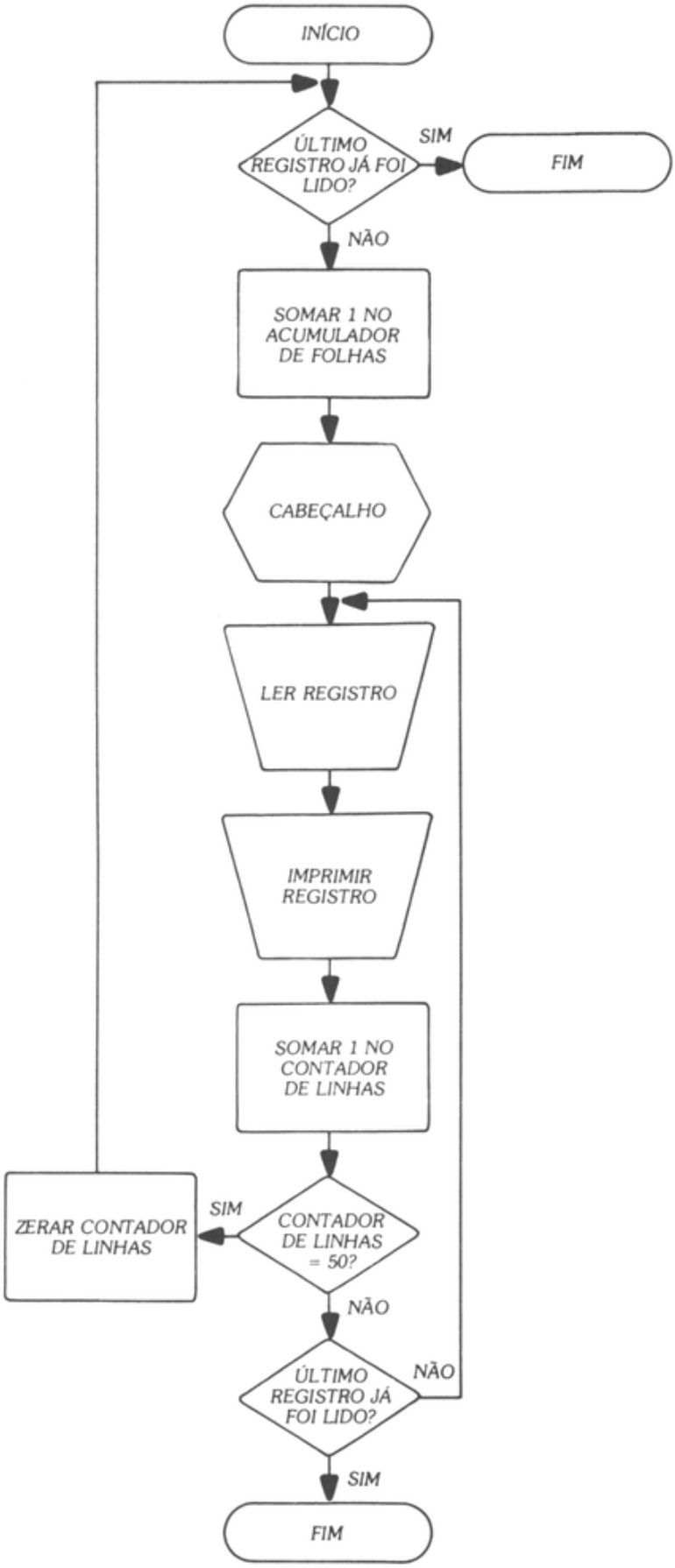
Veja outra solução para o mesmo problema.



3) Fazer o fluxograma do programa abaixo:
Ler um arquivo gravado em fita magnética. O arquivo contém vários tipos de registros, todos identificados por uma letra na posição 30 do registro. Para este programa só interessam os registros que tiverem a letra *S* na posição 30 do registro. Ao encontrar um registro "S", calcular 36% do campo 31/40 do registro. Se o resultado for superior a 1 000 imprimir o conteúdo do registro. No final do processamento imprimir a quantidade total de registros lidos, a quantidade de registros "S" cujo resultado do cálculo dos 36% foi superior a 1 000 e a quantidade de registros "S" cujo resultado do cálculo dos 36% foi igual ou inferior a 1 000.
Compare o seu fluxograma com o da página seguinte.

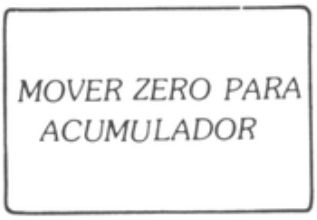

INÍCIO
ÚLTIMO REGISTRO JÁ FOI LIDO?
SIM
NÃO
LER REGISTRO
SOMAR 1 EM CONTADOR DE REGISTROS
REGISTRO "S"?
NÃO
SIM
MULTIPLICAR CAMPO 31/40 DO REGISTRO POR 36
DIVIDIR RESULTADO DA OPERAÇÃO ACIMA POR 100
RESULTADO DA DIVISÃO > 1 000 ?
SIM
NÃO
SOMAR 1 EM CONTADOR SUP. A 1 000
IMPRIMIR REGISTRO
SOMAR 1 EM CONTADOR IGUAL OU INFERIOR A 1 000
IMPRIMIR CONTADOR REGISTRO
IMPRIMIR CONTADOR SUP. A 1 000
IMPRIMIR CONTADOR INF. OU IGUAL A 1 000
FIM

4) Fazer o fluxograma do programa abaixo:
Ler um arquivo gravado em fita magnética.
Para cada registro da fita magnética imprimir uma linha no relatório. Cada folha do relatório deverá ter 50 linhas impressas. Imprimir, no início de cada folha, um cabeçalho onde deverá constar o número da folha.
Compare o seu fluxograma com o apresentado na página seguinte.

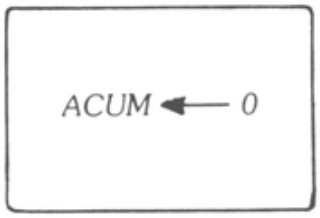

INÍCIO
ÚLTIMO REGISTRO JÁ FOI LIDO?
SIM
FIM
NÃO
SOMAR 1 NO ACUMULADOR DE FOLHAS
CABEÇALHO
LER REGISTRO
IMPRIMIR REGISTRO
SOMAR 1 NO CONTADOR DE LINHAS
CONTADOR DE LINHAS = 50?
SIM
ZERAR CONTADOR DE LINHAS
NÃO
ÚLTIMO REGISTRO JÁ FOI LIDO?
NÃO
SIM
FIM

□□ Os exemplos e exercícios apresentados até agora podem ser usados para escrever programas em FORTRAN. Eles, no entanto, são mais usados para escrever programas cujo objetivo é a resolução de problemas comerciais. Os diagramas de bloco dirigidos para a linguagem FORTRAN são, geralmente, mais compactos, empregando convenções que lembram mais a linguagem do próprio FORTRAN.

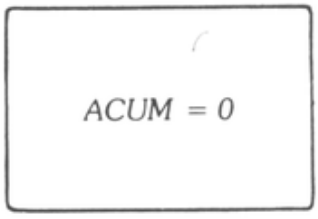
Por exemplo, em vez de usar



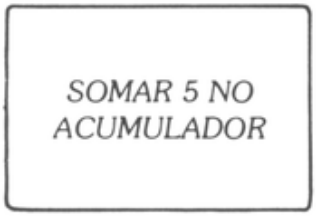
O programador FORTRAN usaria



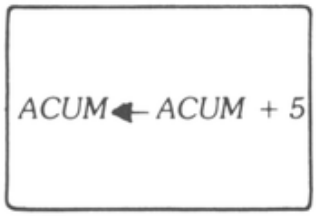
ou



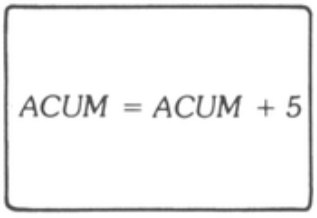
Em vez de



O programador FORTRAN usaria

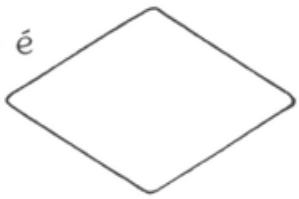


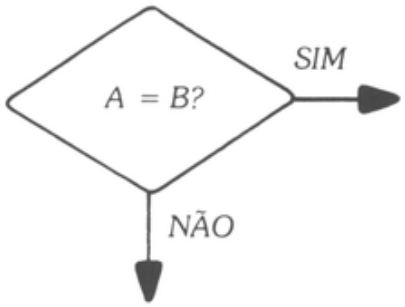
ou



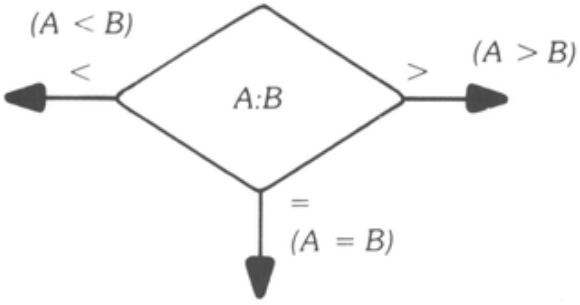
□□ Note que a expressão acima (ACUM = ACUM + 5) não tem validade matemática. O sinal "=" não representa uma igualdade, mas apenas indica que o conteúdo do acumulador (após a operação de soma ser efetuada) ficará acrescido de 5 unidades.

Bloco de Decisões

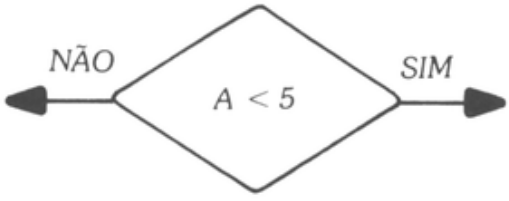
□□ Em Fortran o bloco de decisões lógicas, cujo símbolo é  pode ser apresentado de diversas formas, como apresentadas a seguir:



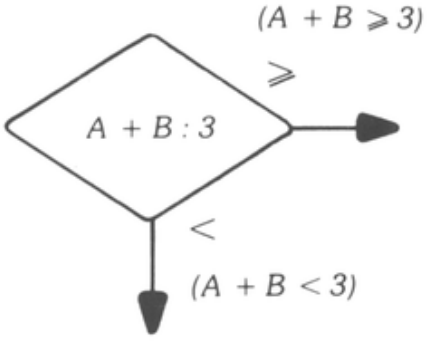
(A é igual a B?)



(Compare A com B)

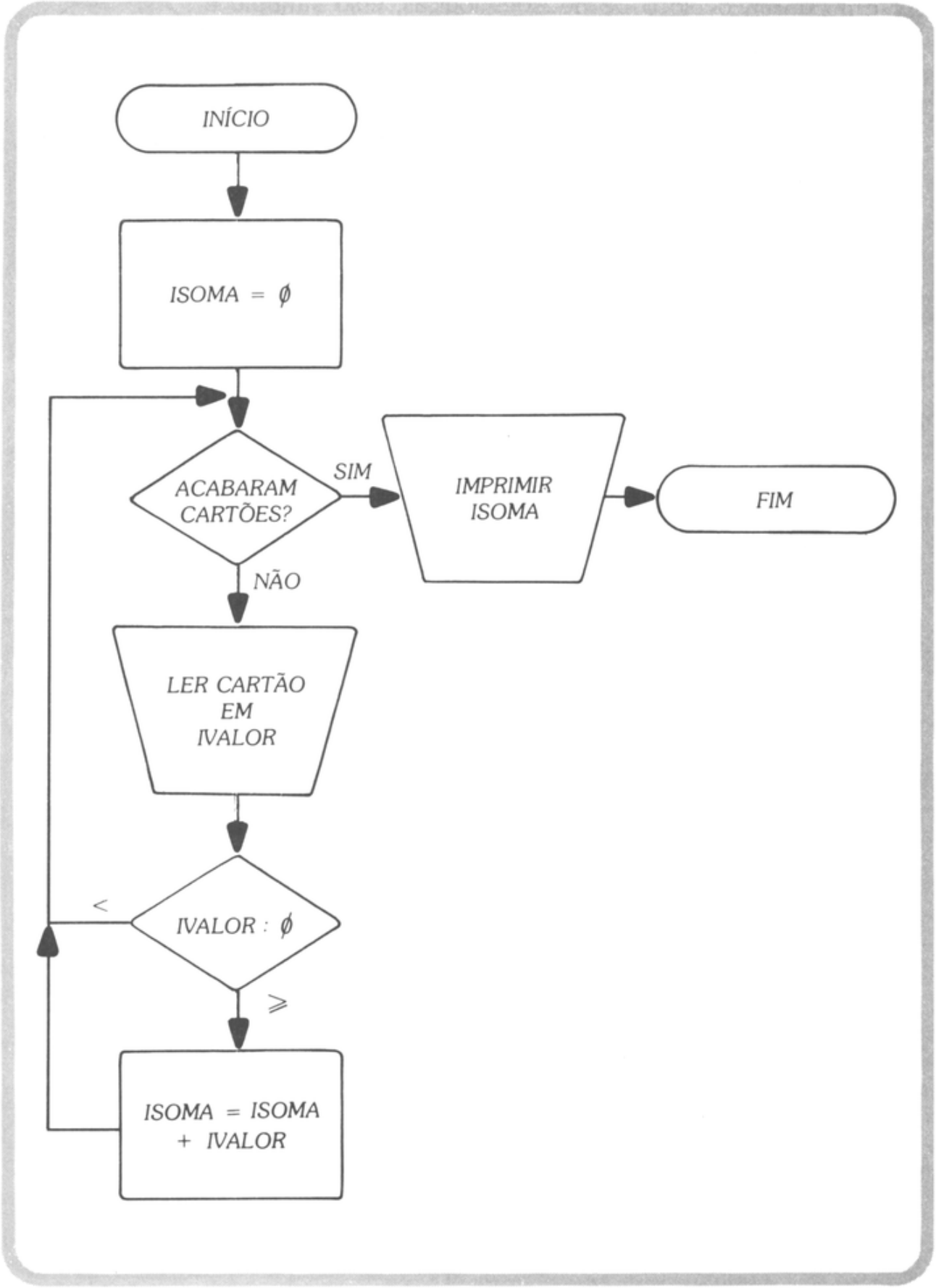


(A é menor do que 5?)



(Compare A + B com 3)

□□ O exemplo 4 apresentado na página 22 poderia ser construído assim:



Teste de Fim de Cartões

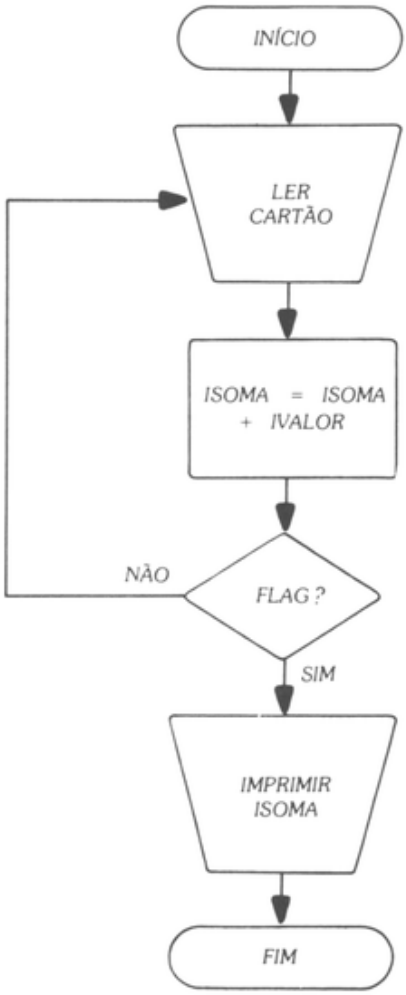
□□ Nos exemplos e exercícios apresentados até agora os diagramas têm um bloco de decisão indicando que deve ocorrer um desvio ao terminar a leitura dos cartões de dados. A linguagem Fortran, no entanto, não faz tal desvio automaticamente. Há necessidade de um artifício qualquer que indique ao programa que o último cartão de dados já foi lido. Os artifícios mais usados são os seguintes:

1. *Perfuração de uma indicação qualquer num campo do cartão (FLAG).*
   Neste caso, perfura-se um código qualquer num campo do último cartão da massa. O programa testa tal campo em todos os cartões que lê. Quando encontrar o código predeterminado num cartão, o programa já sabe que aquele é o último cartão a ser lido. O código a ser perfurado pode ser um número, uma palavra, etc. Veja página 46, exemplo 1.

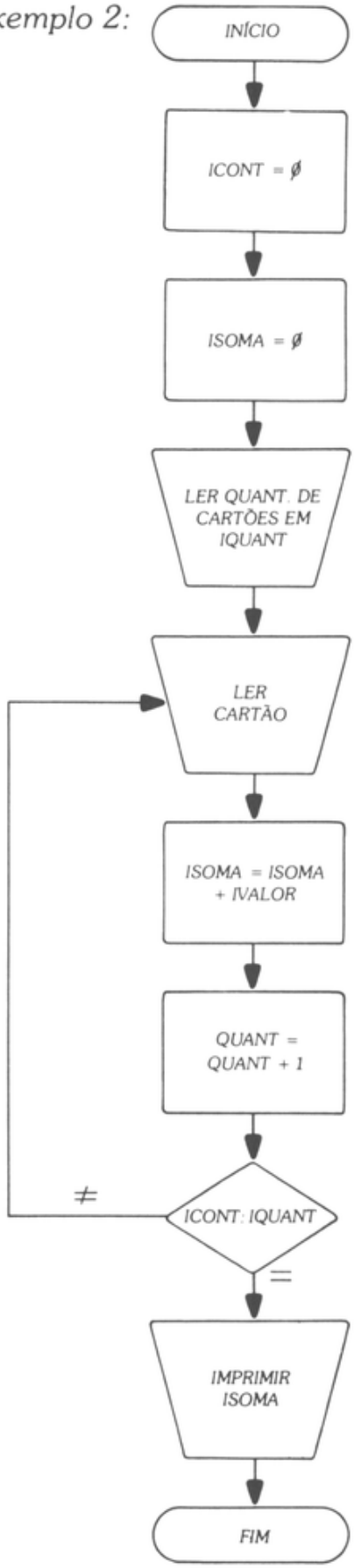
2. *Perfuração, num campo do 1.º cartão, da quantidade de cartões que deve ser lida.*
   Se o programador tiver certeza que será possível determinar, na hora de cada execução do programa, qual a quantidade de cartões de dados que vai ser processada, ele poderá perfurar tal quantidade num campo qualquer do 1.º cartão da massa. Para cada cartão que o programa ler, haverá um teste. Quando o cartão lido corresponder ao número perfurado no campo de "quantidade de cartões" do 1.º cartão, o programa já sabe que aquele é o último cartão que deve ser lido. Veja página 46, exemplo 2.

3. *Acumulador de cartões lidos.*
   Se a quantidade de cartões a serem lidos for sempre a mesma em todas as execuções do programa, não há necessidade de nenhuma informação para o programa na hora da execução. O programador, neste caso, acumulará "1" num contador qualquer para cada cartão lido. Quando o contador for igual à quantidade desejada, o programa já sabe que não deve ler mais nenhum cartão. Veja página 47.

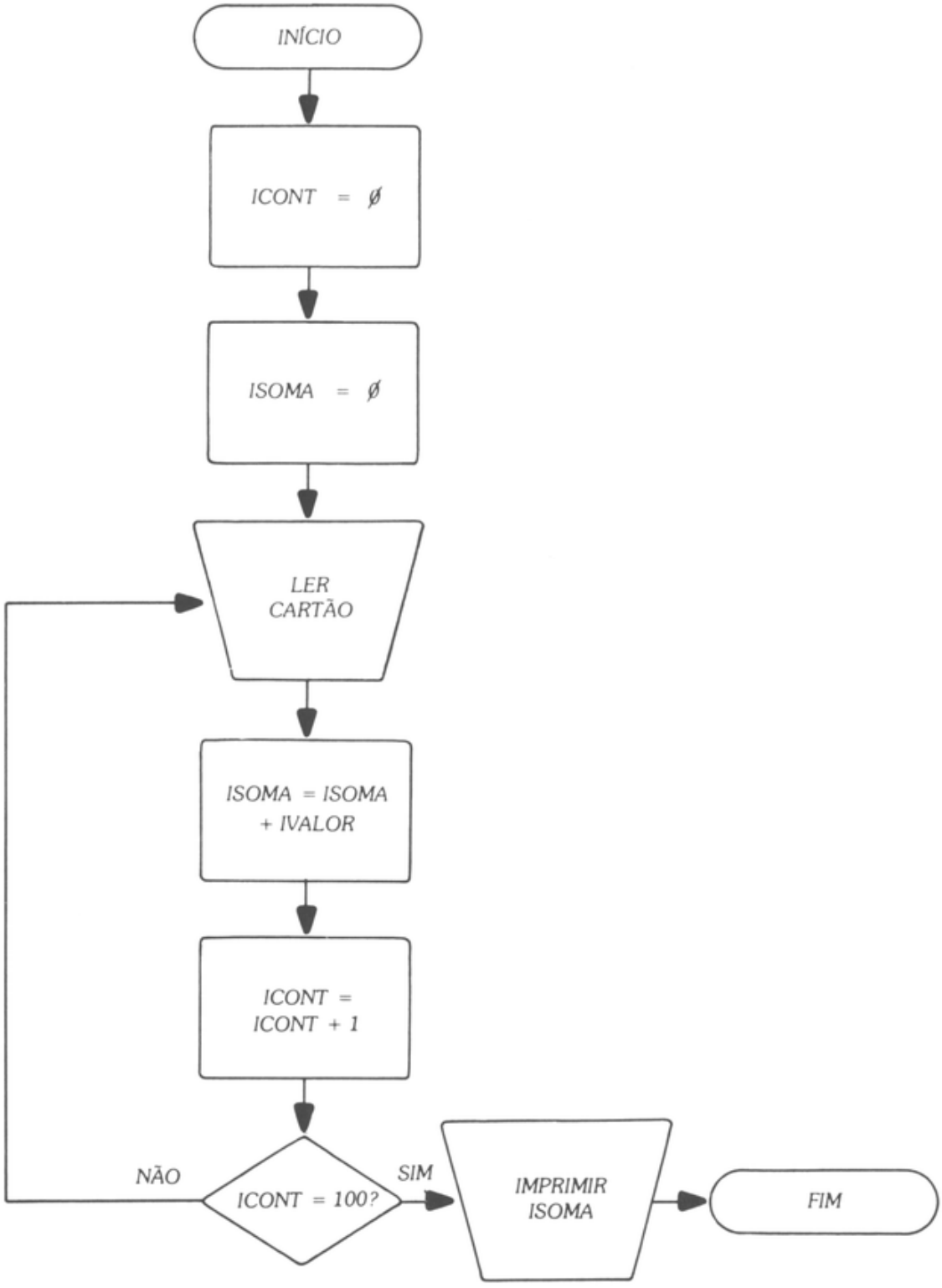
Exemplo 1:



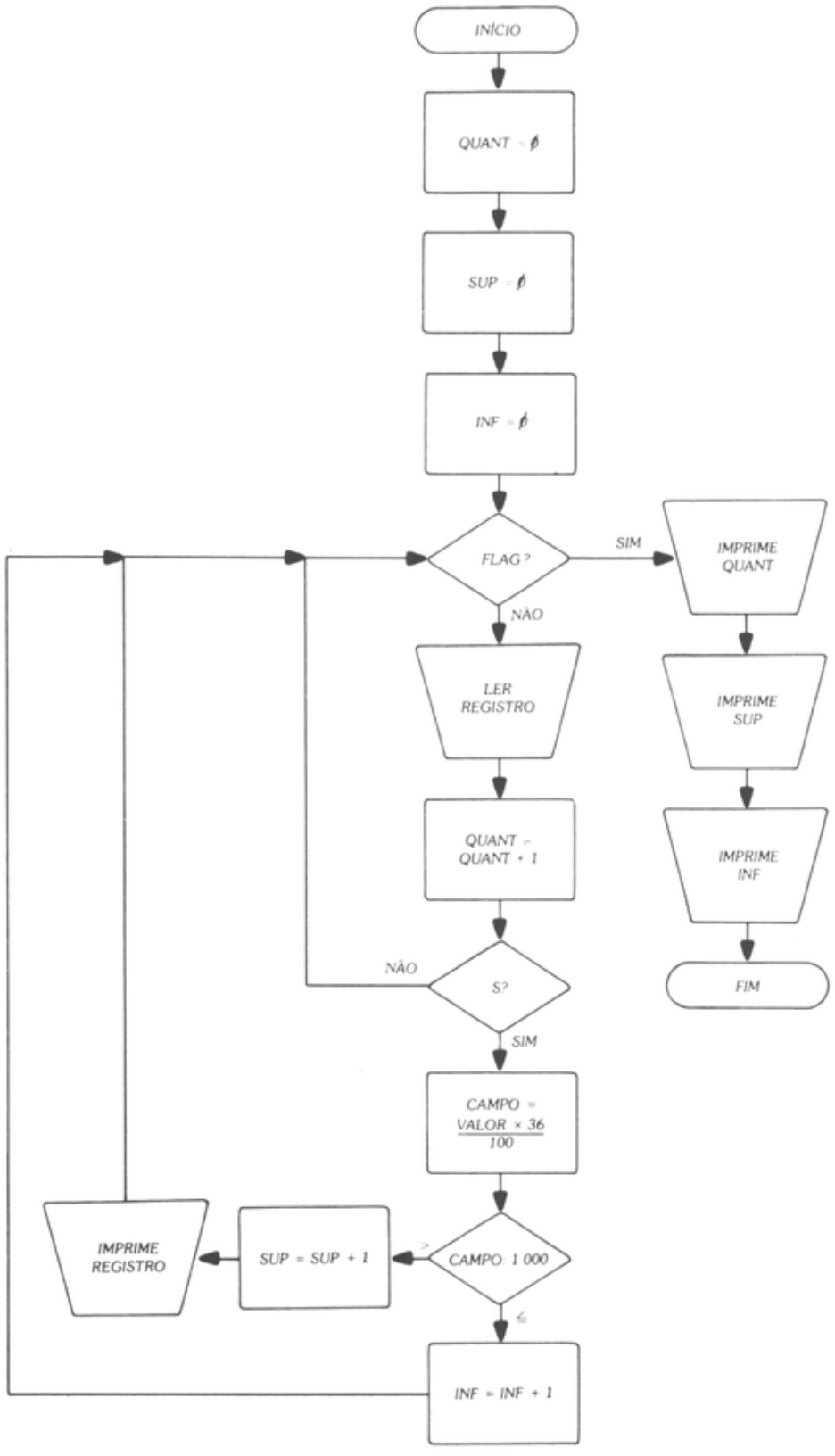
Exemplo 2:



Exemplo 3:


INÍCIO
ICONT = ø
ISOMA = ø
LER CARTÃO
ISOMA = ISOMA + IVALOR
ICONT = ICONT + 1
ICONT = 100?
NÃO
SIM
IMPRIMIR ISOMA
FIM

□□ Veja agora como ficaria o diagrama de blocos do exercício 3 para facilitar a codificação do programa em Fortran.



3

Conceitos Básicos

*

* *

□□ A linguagem Fortran, como qualquer outra linguagem, possui regras que devem ser seguidas rigorosamente. A não obediência a essas regras pode provocar erros nos resultados do programa.

□□ Alguns erros são apontados pelo compilador e são facilmente corrigíveis. Outros, no entanto, podem ser aceitos pelo compilador e só na hora da execução é que o programador tomará conhecimento deles. Outros, ainda, ocasionam uma interrupção do programa e o computador só processará tal programa depois que os erros forem corrigidos. Daí a importância de o estudante de Fortran procurar compreender perfeitamente as regras básicas. Não é preciso decorá-las, basta entendê-las. Sempre que tiver dúvidas, o programador deve consultar os livros e os manuais fornecidos pelos fabricantes dos computadores.

□□ A linguagem Fortran é constituída por uma série de ordens conhecidas como "COMANDOS", "INSTRUÇÕES" ou "DECLARAÇÕES", os quais são os seguintes:

- Comandos de Entrada e Saída (estudados no Capítulo 8);
- Comandos de Controle (estudados no Capítulo 10).
- Declarações Aritméticas (estudadas no Capítulo 7);
- Declarações de Especificação (estudadas no Capítulo 4);
- Declarações de Função (estudadas no Capítulo 6);

A Folha de Codificação FORTRAN

□□ O programador ao codificar o programa em Fortran, faz uso de uma folha própria para codificação. Esta folha é conhecida como "folha de codificação Fortran". A seguir apresentamos um modelo da folha de codificação.

Modelo da Folha de Codificação FORTRAN

SISTEMA		
PROGRAMA		
PROGRAMADOR	DATA	

COMM					CONT																															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37

INSTRUÇÕES DE PERFURAÇÃO

FOLHA DE

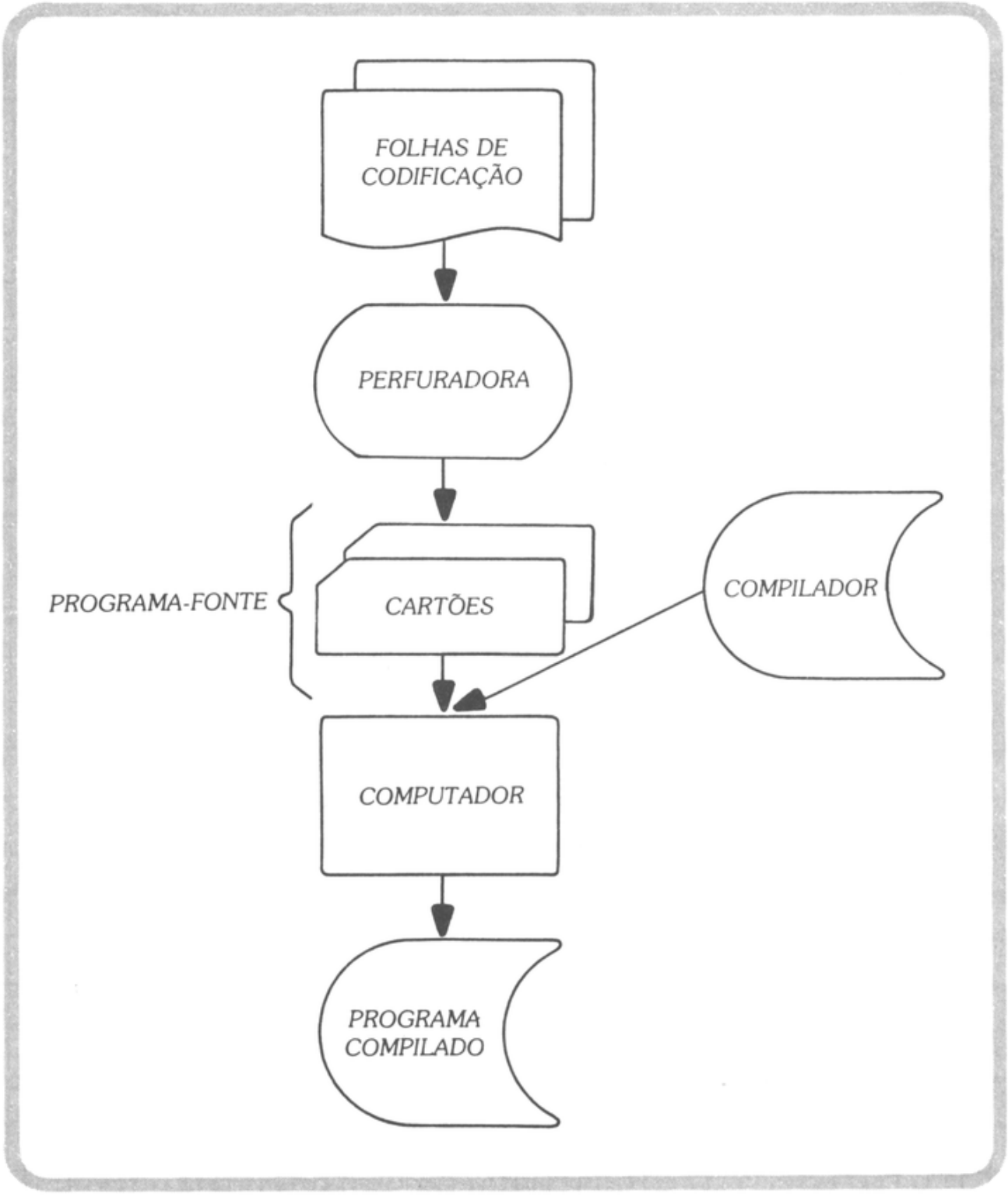
IDENTIFICAÇÃO

73 80

36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

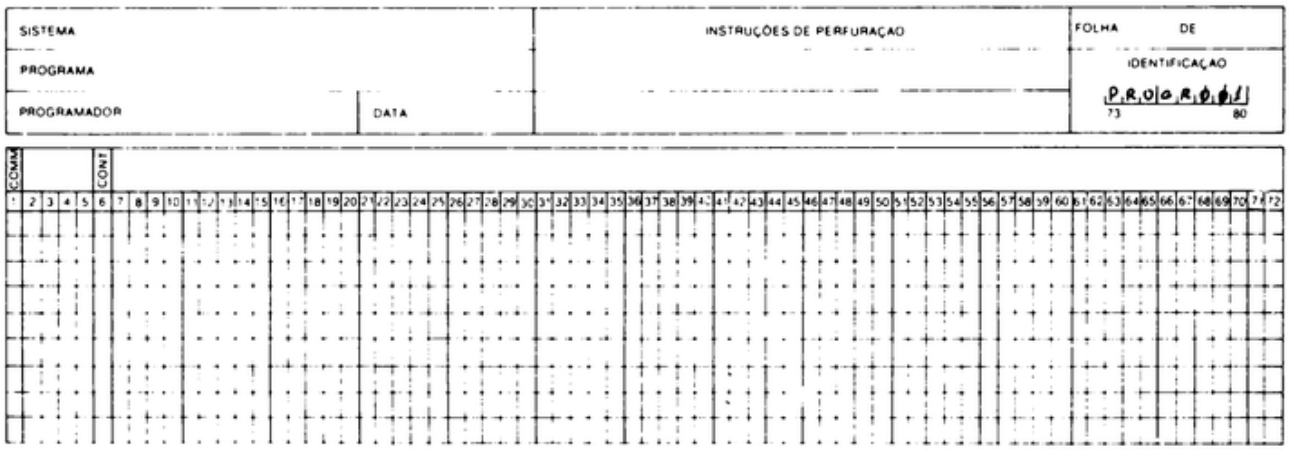
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

□□ A folha de codificação é dividida em várias linhas e cada linha é dividida em 80 retângulos. O motivo da divisão da linha em 80 retângulos é que será perfurado um cartão para cada linha do formulário. Assim, dividindo a linha em 80 retângulos, cada retângulo corresponderá, exatamente a cada uma das 80 colunas do cartão. Os cartões serão usados como entrada para o computador fazer a compilação. A massa de cartões perfurados é conhecida como programa-fonte Fortran.



□□ Examine o modelo da folha de codificação. Repare que os retângulos 73 a 80 estão no canto superior à direita do formulário. Não há necessidade de repetir os retângulos 73/80 em todas as linhas do formulário, porque os dados a serem perfurados nas colunas 73/80 dos cartões serão sempre os mesmos em todos os cartões. Isto porque as colunas 73/80 serão usadas para a identificação do programa. Logo, a identificação será sempre a mesma em todos os cartões.

□□ Por exemplo, se for desejado identificar o programa pela sigla PROGRØØ1, devemos escrever esta sigla no campo 73/80 do formulário.



□□ A pessoa encarregada de perfurar os cartões, ao perfurá-los, perfurará PROGRØØ1 nas colunas 73/80 de todos os cartões.

□□ Se for encontrado um cartão extraviado, poderemos saber a que programa ele pertence, verificando a identificação perfurada nas colunas 73/80.

□□ Os demais dados do cabeçalho do formulário (sistema, programa, programador, data, instruções de perfuração e folha) não serão perfurados nos cartões. Estes dados do cabeçalho serão usados apenas para documentação do programa e como orientação para a pessoa que for perfurar os cartões.

Coluna 1

□□ Esta coluna pode ser usada para duas finalidades distintas:

a) Como parte do campo formado pelas colunas 1 a 5, conforme explicados logo adiante;
b) para indicar que a linha é apenas um comentário.

Comentário

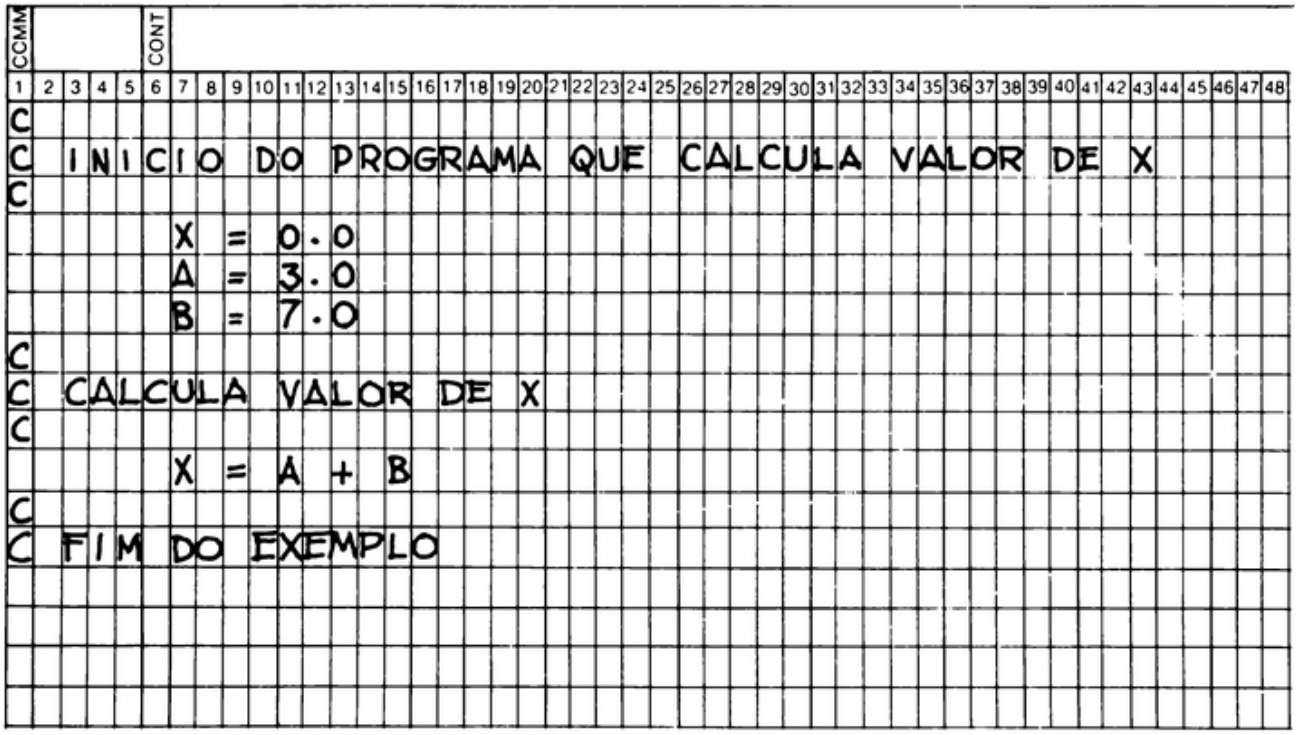
□□ Se for escrita a letra *C* na coluna 1, o compilador considerará toda a linha como sendo uma linha de comentário. As finalidades dos comentários são:

1. Colocar em destaque certas partes do programa;
2. Lembrar ao leitor do programa qual a finalidade de determinados comandos;
3. Dar informações adicionais sobre o programa a quem o lê, como, por exemplo, o nome do programador, a descrição do programa, etc.

□□ É importante notar que a linha de comentário não gerará nenhuma instrução no programa-objeto. Ela serve apenas para efeito de documentação, pois a sua única função é gerar uma linha impressa na listagem do programa fonte.

□□ O comentário só deve ser escrito nas colunas 2 a 72 (além da letra *C* na coluna 1, é óbvio).

Exemplo:



Colunas 1 a 5

□□ As colunas 1 a 5 são usadas para numerar os comandos. A numeração não é obrigatória, não precisa começar na coluna 1 e não precisa terminar na coluna 5.

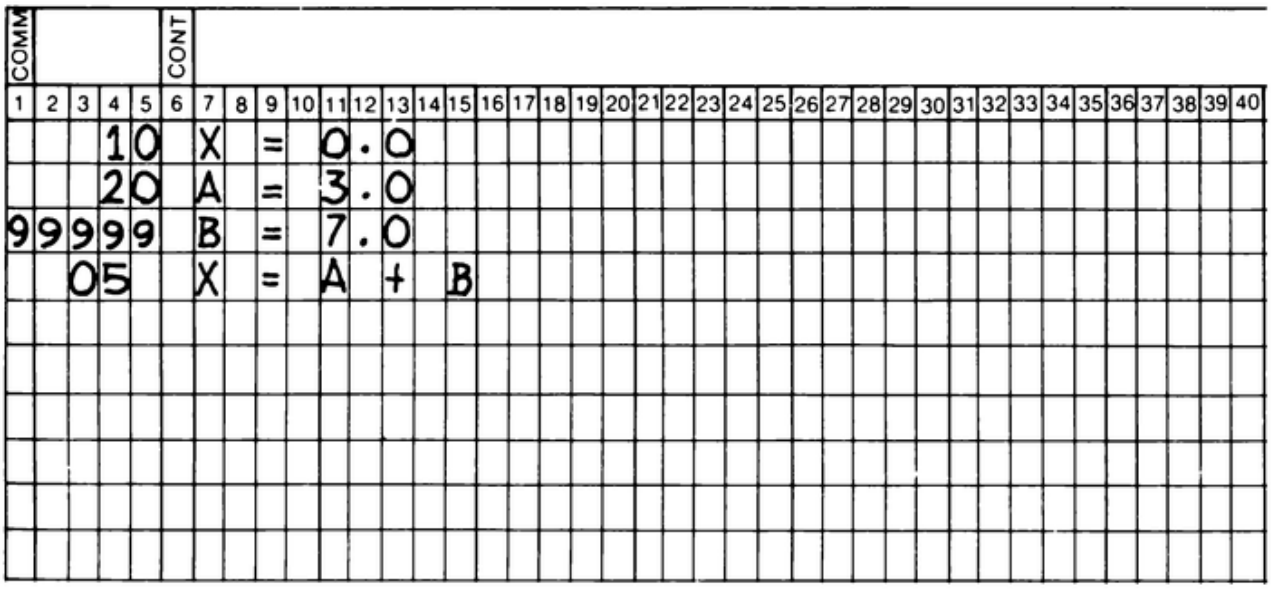
□□ Os programadores, geralmente, só numeram os comandos quando precisam fazer referência a eles em outra parte do programa. A numeração de

todos os comandos em ordem crescente, no entanto, facilita a reorganização de toda a massa, no caso de ela sair fora de ordem.

□□ Para os que preferirem numerar todos os comandos, é conveniente numerá-los de 10 em 10 a fim de possibilitar a inserção de outros cartões sem quebrar a ordem crescente do programa. Se, por exemplo, for necessário incluir um cartão entre os comandos 310 e 320, podemos atribuir-lhe o número 315 e a massa não ficará fora de ordem.

□□ A seqüência de execução dos comandos *não* obedecerá à ordem dos números dos comandos. Enquanto não houver um comando de desvio (estudado no Capítulo 10), o programa executará cada linha seqüencialmente, independentemente de o número do comando estar ou não em ordem crescente.

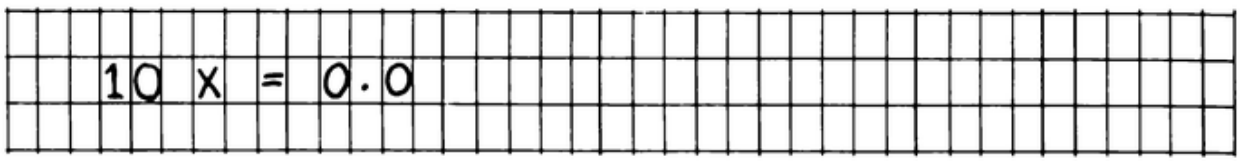
Exemplo de comandos numerados.



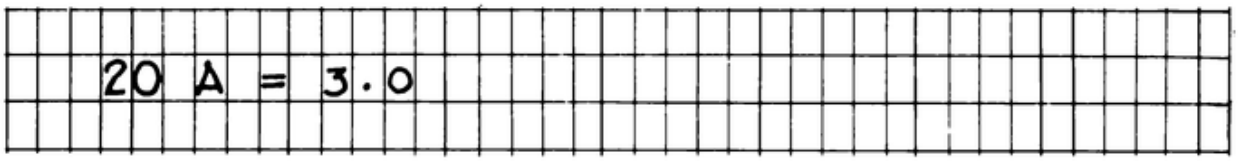
□□ Observe no exemplo anterior que o comando da quarta linha recebeu o número 05, mas este número não começa na coluna 1, nem termina na coluna 5.

□□ Neste *exemplo*, a ordem de execução dos comandos será a seguinte:

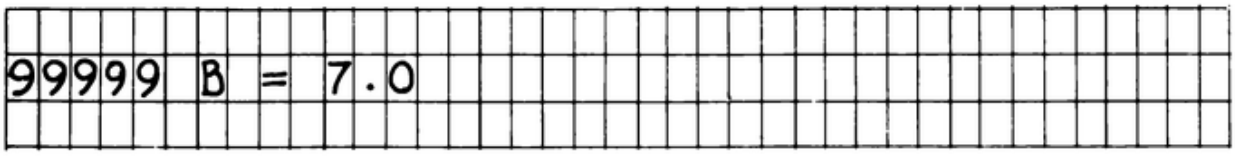
1.º comando a ser executado:



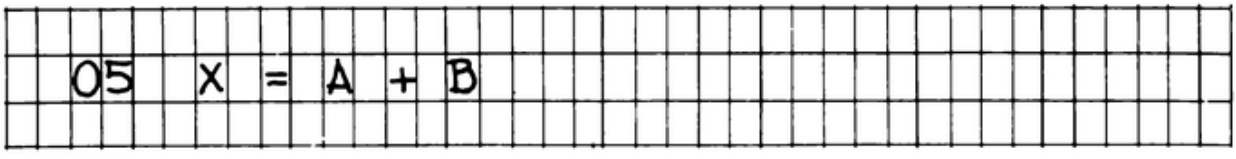
2.º comando a ser executado:



3.º comando a ser executado:



4.º comando a ser executado:

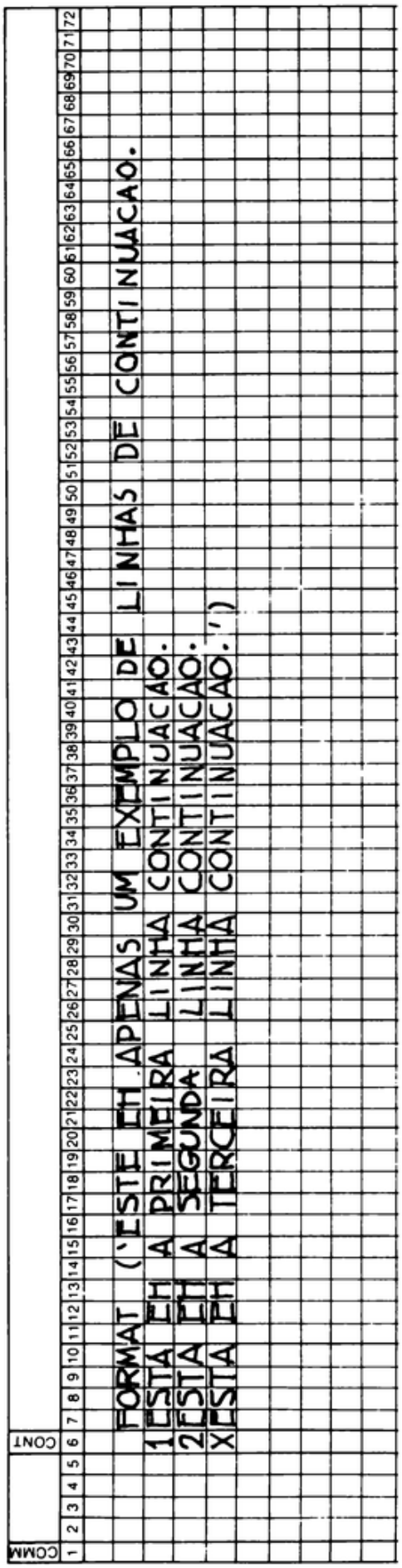


□□ Embora o comando acima tenha um número menor do que o comando 99999, ele será executado depois, porque o computador não levará em conta a numeração dos comandos e sim a seqüência em que eles aparecem no programa.

Coluna 6

□□ Quando um comando não cabe apenas numa linha, podemos continuá-lo na(s) linhas(s) seguintes(s). Para indicar que haverá continuação, basta escrever um caráter qualquer na coluna 6.

□□ Alguns computadores, como o IBM-1130, aceitam até 5 linhas de continuação; outros, como o IBM-/360, o IBM-/370, o FACOM-M160 e o FACOM-M200, aceitam até 19 linhas de continuação.



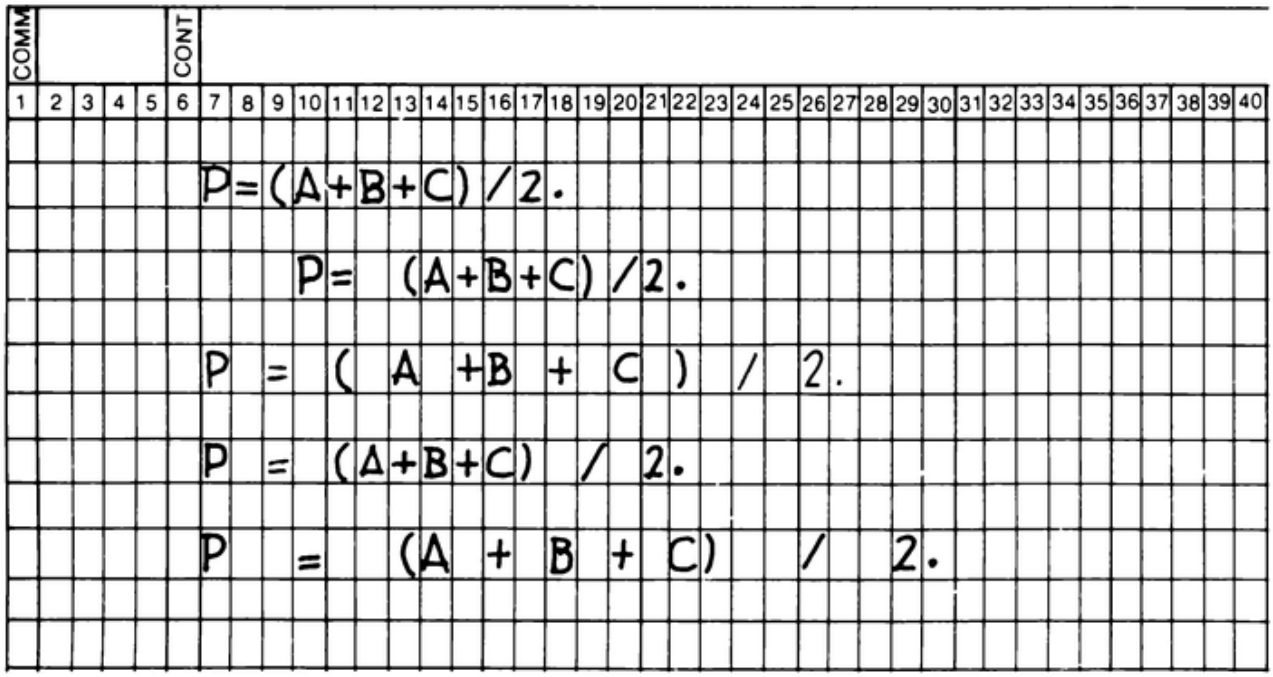
Linhas de continuação:

Colunas 7 a 72

□□ As colunas 7 a 72 são usadas para se escrever os comandos propriamente ditos.

□□ Os comandos *não* precisam começar obrigatoriamente na coluna 7. Eles podem começar em quaisquer das colunas 7 a 72.

□□ Para o compilador não faz diferença se existem ou não espaços em branco no meio dos comandos. Por este motivo, os diversos comandos abaixo estariam corretos.



Colunas 73 a 80

□□ As colunas 73 a 80 são ignoradas pelo compilador. Elas podem ser usadas para a identificação dos programas, conforme já explicamos anteriormente.

□□ Em algumas empresas, no entanto, as colunas 73 a 80 são aproveitadas para a seqüenciação dos cartões, além da identificação dos programas. Neste caso, o modelo da folha de codificação Fortran precisa ser um pouco diferente do modelo que apresentamos, ou seja, o campo 73 a 80 não deve ser impresso no alto da folha à direita; ele precisa ser impresso logo após a coluna 72 de cada linha.

Conjunto dos Caracteres Usados no FORTRAN

□□ Para se escrever um programa Fortran só se pode usar os 48 caracteres relacionados a seguir:

As letras: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Os dígitos: 0 1 2 3 4 5 6 7 8 9

Os caracteres especiais: = * + − / ' $. ' ( )

O caráter "branco" (representado neste livro por ☒).

□□ Os operadores aritméticos no Fortran são assim representados:

Soma: +
Subtração: −
Multiplicação: *
Divisão: /
Potenciação: **

* * *

4

Representação das Informações

* * *

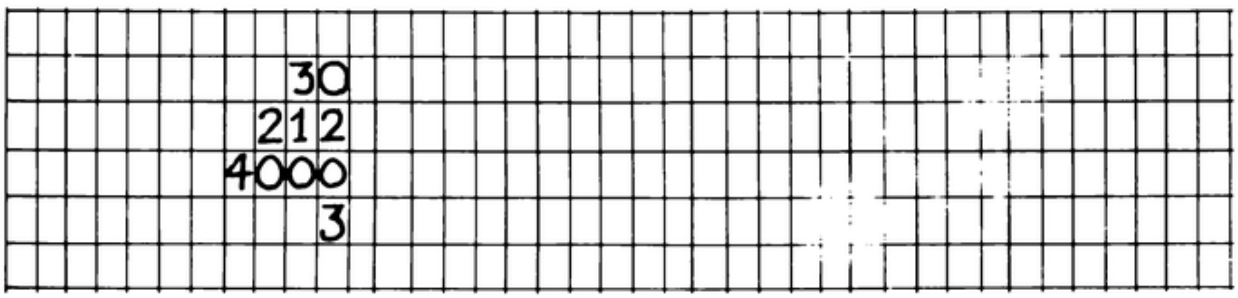
Números

□□ O Fortran aceita, entre outros, dois tipos de números:
- os números inteiros;
- os números reais.

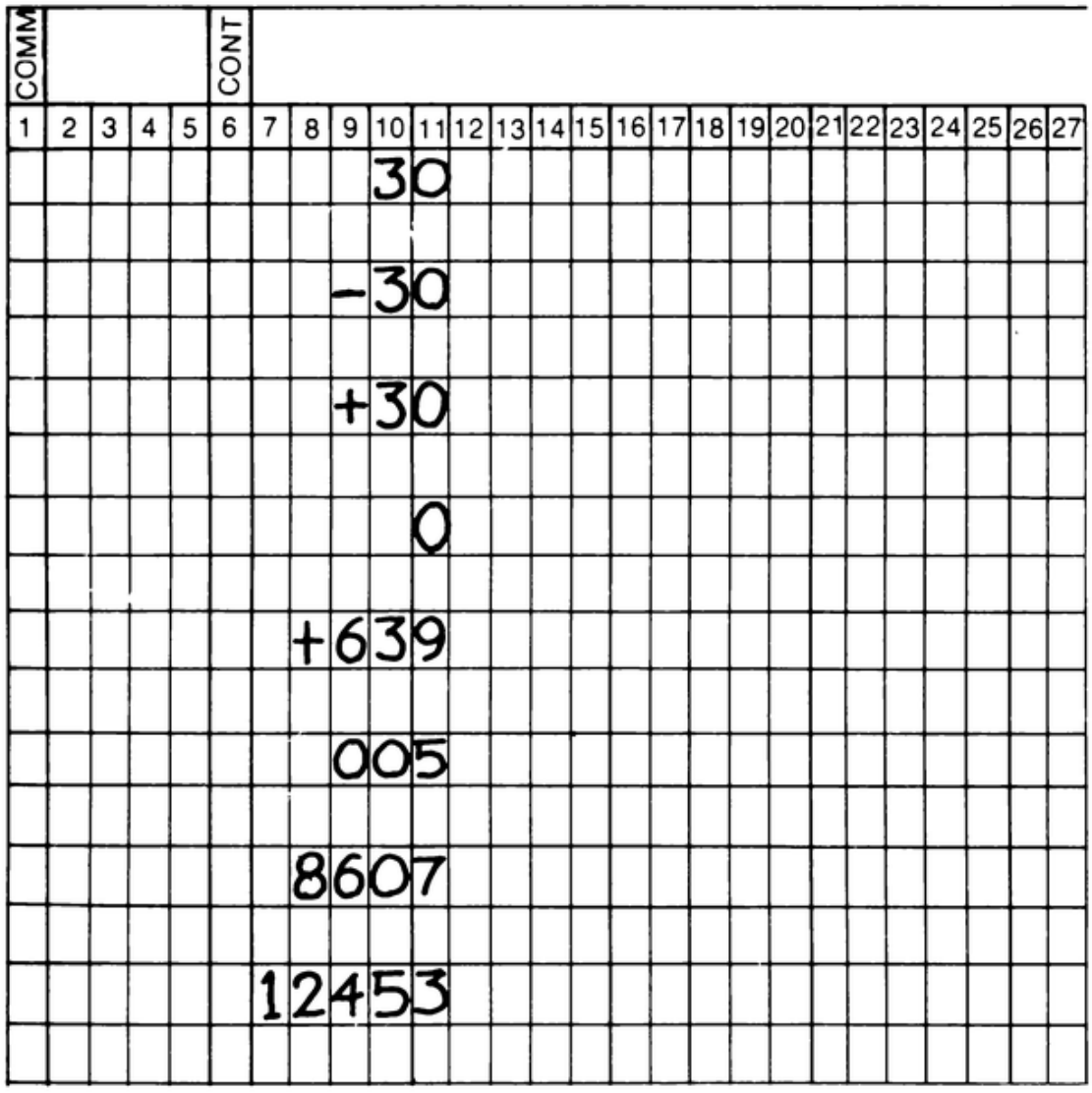
Os Números Inteiros

□□ Os números inteiros são caracterizados pela ausência da vírgula decimal.
□□ Na notação inglesa a vírgula decimal é substituída pelo "ponto decimal". Como o Fortran se baseia na notação inglesa, podemos dizer que, em Fortran, os números inteiros são caracterizados pela ausência do ponto decimal.
□□ Os números inteiros também são conhecidos como números de "ponto fixo", porque admite-se que o ponto decimal fica fixo à direita do número.

Exemplos de números inteiros:



□□ O número inteiro pode ser positivo, negativo ou zero. O sinal positivo é opcional. Quando o número inteiro não contiver sinal é considerado positivo.



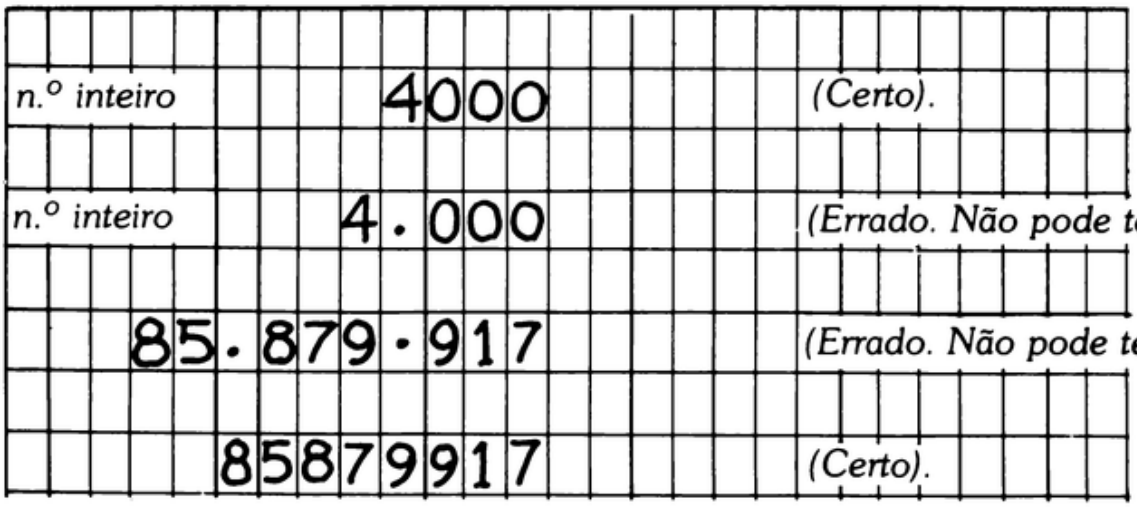
(positivo)

(negativo)

(positivo)

□□ Como você pode observar nos exemplos acima, o número inteiro é representado sem nenhum ponto ou vírgula.

Exemplos:



(Certo).

(Errado. Não pode ter ponto).

(Errado. Não pode ter ponto).

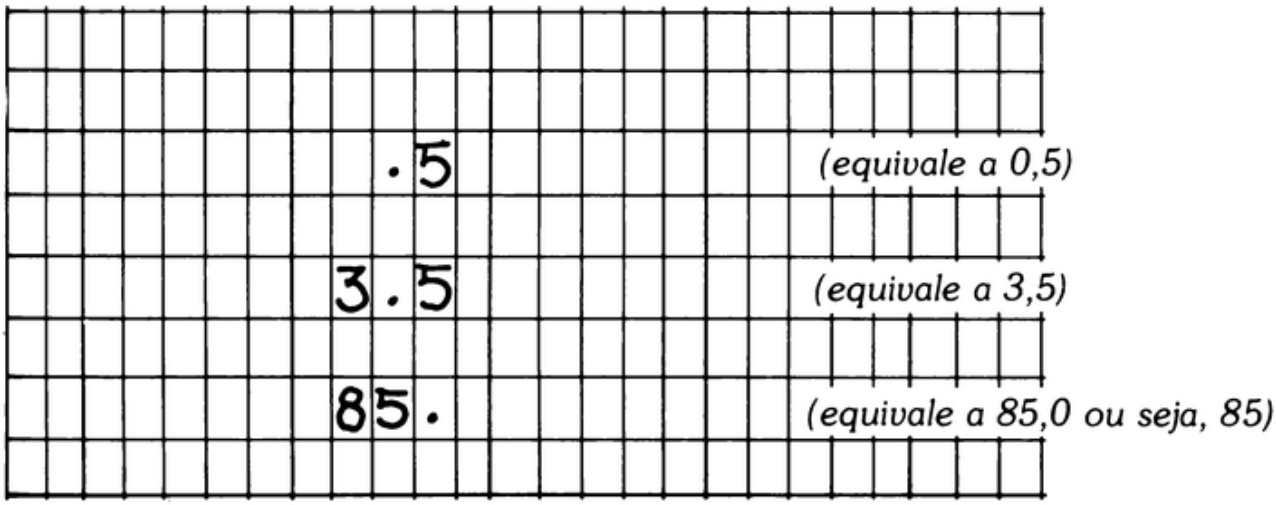
(Certo).

□□ O maior número em valor absoluto que os computadores aceitam varia dependendo do tipo de computador. O IBM-1130, por exemplo, aceita até 32767 ($2^{15} - 1$). Os IBM-/360 e IBM-/370 aceitam até 2147483647 ($2^{31} - 1$).

Os Números Reais

□□ Os números reais são aqueles em que aparece o ponto decimal (lembre-se que o Fortran adota a notação inglesa e, em conseqüência, em vez da vírgula decimal usa o ponto decimal).
□□ Os números reais podem ser positivos, negativos ou zero. Um número real sem sinal é considerado positivo.
□□ O ponto decimal pode ficar no início, no meio ou no fim do número real.

Exemplos:

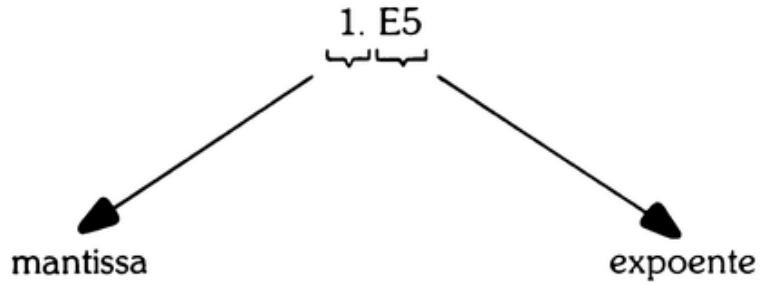


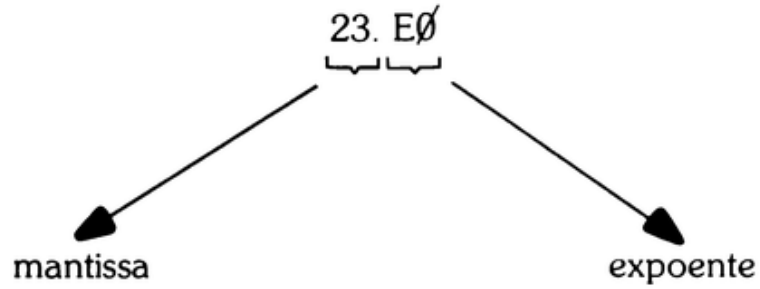
□□ Os números reais podem ser escritos em duas partes: a mantissa e o expoente.

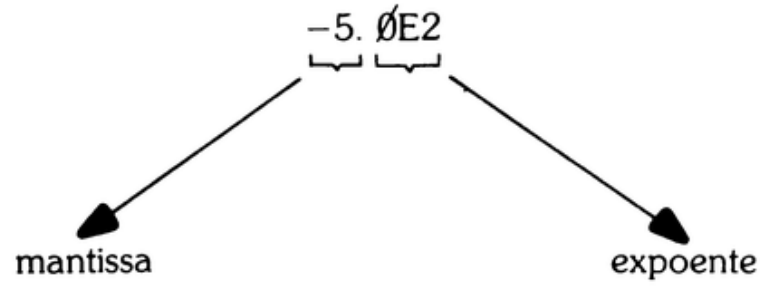
□□ A mantissa é a parte decimal, com ou sem sinal.

□□ O expoente é formado pela letra *E* seguida de um número inteiro, com ou sem sinal, que indica a potência de 10 pela qual deve ser multiplicada a mantissa. Quando o expoente do número for zero, não será necessário escrever a parte exponencial.

Exemplos:







□□ Os valores que os exemplos anteriores estão representando são os seguintes:

1.E5 $(1 \times 10^5 = 1 \times 100.000 = 100.000)$

23.EØ $(23 \times 10^0 = 23 \times 1 = 23)$

− 5.ØE2 $(-5 \times 10^2 = -5 \times 100 = -500)$

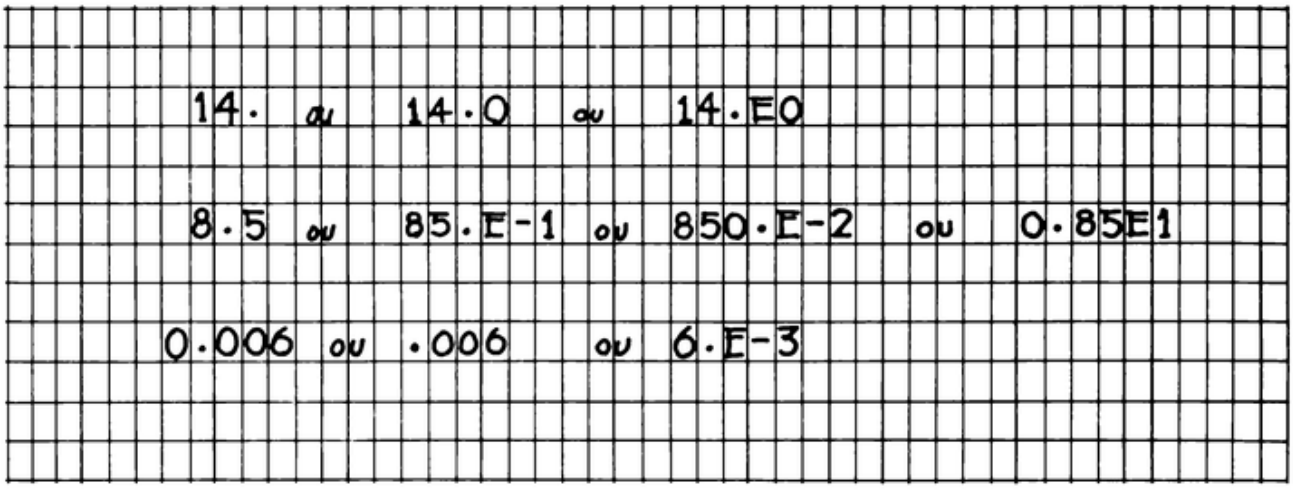
·5 $(0{,}5 \times 10^0 = 0{,}5 \times 1 = 0{,}5)$

3.5 $(3{,}5 \times 10^0 = 3{,}5 \times 1 = 3{,}5)$

85. $(85{,}0 \times 10^0 = 85 \times 1 = 85)$

□□ De acordo com o que já foi explicado, podemos representar um mesmo valor de diversas maneiras.

Exemplos:



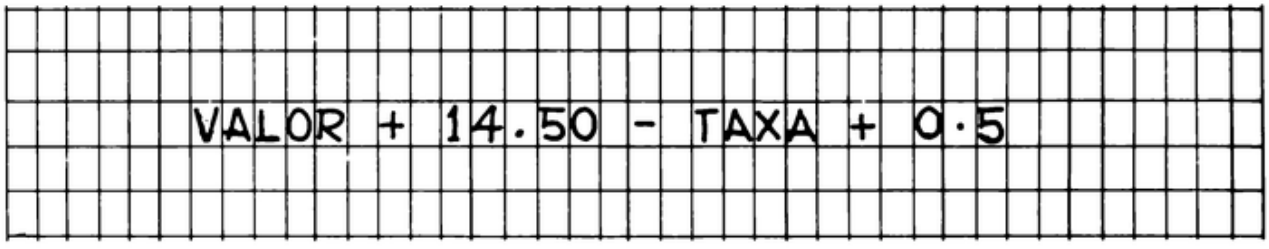
□□ O valor absoluto de um número real também é limitado. O limite varia com o tipo de computador. No IBM-1130, por exemplo, os limites são os seguintes:

Maior valor absoluto: 10^{38}
Menor valor absoluto: 10^{-39}

Nos IBM-/360 e IBM-/370, os limites são:
Maior valor absoluto: 10^{75}
Menor valor absoluto: 10^{-78}

Constantes e Variáveis

□□ Em programação, chamamos de constantes aquelas quantidades que não mudam de valor durante todo o processamento do programa.
□□ As variáveis, ao contrário das constantes, podem assumir diversos valores durante o processamento do programa. As variáveis são representadas por nomes. Por exemplo, na expressão a seguir, escrita em FORTRAN,



14.50 e 0,5 são constantes; VALOR e TAXA são variáveis.
□□ As constantes podem ser escritas como números inteiros ou reais.

Regras Para Formação de Nomes de Variáveis

□□ Cada variável é identificada por uma representação simbólica ou "nome". É importante não confundir o nome de uma variável com o seu conteúdo.

□□ A variável NUMERO, por exemplo, poderá conter o número 10, ou 25 ou 6836 ou 10812 ou qualquer outro, dependendo das instruções do programa. Embora o nome NUMERO permaneça imutável, o conteúdo da variável poderá sofrer alteração.

□□ O conceito de variável fornecido anteriormente é simplesmente didático. Na realidade, o nome da variável (no caso, NUMERO) não fará parte do programa-objeto. O nome da variável apenas representa um local da memória do computador onde serão armazenados os diversos valores durante o processamento do programa. Para o leitor que não deseja se aprofundar no estudo de programação, no entanto, é suficiente ficar com a imagem de que o nome da variável não varia. O que varia é o seu conteúdo.

□□ O nome de uma variável deve ser formado por um caráter ou por um grupo de caracteres, sendo que o primeiro tem de ser, obrigatoriamente, alfabético.

□□ O IBM-1130 aceita nomes com até 5 caracteres. Os IBM-360 e IBM-/370 aceitam nomes com até 6 caracteres.

□□ O primeiro caráter do nome de uma variável tem de ser alfabético, mas os demais caracteres podem ser alfabéticos ou numéricos.

□□ O nome de uma variável pode ser criado à vontade do programador. Aconselhamos, no entanto, a criação de nomes que lembrem o conteúdo da variável. Por exemplo, os nomes a seguir seriam apropriados para representar, respectivamente, o total de uma acumulação, um acumulador qualquer, as diferentes velocidades de um objeto em movimento e os pesos de diversos objetos.

TOTAL
ACUM
VELOC
PESO

□□ Alguns dos nomes abaixo não seriam aceitos pelo compilador FORTRAN, pois não estão de acordo com as indicações anteriores.

A
CALC
ACUMULADOR (Não é válido. Possui mais de 6 caracteres).
C43

43C	(Não é válido. Não começa com uma letra).
TEMPO*	(Não é válido. O * não é letra nem dígito).
ME DIA	(Não é válido. Há um espaço entre o E e o D).

Tipos de Variáveis

□□ As variáveis são, geralmente, inteiras ou reais. As variáveis inteiras representam números inteiros e as variáveis reais representam números reais.
□□ Há duas maneiras de indicar se a variável deve ser inteira ou real: a especificação implícita e a especificação explícita.

Especificação Implícita

□□ Já vimos que a primeira letra do nome de uma variável tem de ser, obrigatoriamente, alfabético. Quando a primeira letra do nome for I, J, K, L, M ou N, a variável será uma variável inteira. Este tipo de especificação é conhecido como especificação implícita.

Exemplos de variáveis do tipo inteiro com especificação implícita:

INDICE
JUNHO
K
KCALC
LETRA
MONEY
NOTA
IFOLHA
IACUM

□□ As variáveis cujo nome não comece com nenhuma das letras I, J, K, L, M ou N são implicitamente do tipo real.

Exemplos: ACUM
CARGA
FOLHA
TOTAL
VALOR
VOLT
X
Y

Especificação Explícita

□□ O tipo de uma variável também pode ser definido por meio de instruções de especificação. Estas instruções servem para indicar se as variáveis deverão ser consideradas como variáveis inteiras ou reais. Este tipo de especificação é conhecido como especificação explícita.

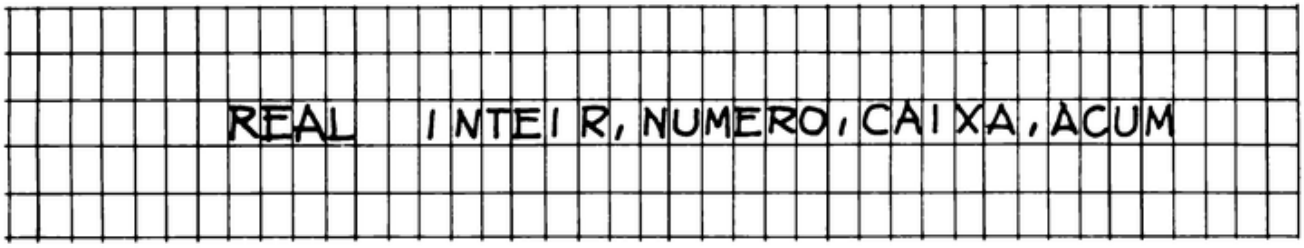
□□ A especificação explícita tem precedência sobre a especificação implícita, ou seja, mesmo que o nome de uma variável comece com a letra I, J, K, L, M ou N, se houver uma especificação explícita para considerá-la real, ela será considerada como variável real, apesar de ela estar implicitamente definida como inteira. Se houver especificação explícita para considerar uma variável como inteira, ela o será, mesmo que o seu nome não comece com nenhuma das letras I, J, K, L, M ou N.

□□ As formas gerais das especificações explícitas são:

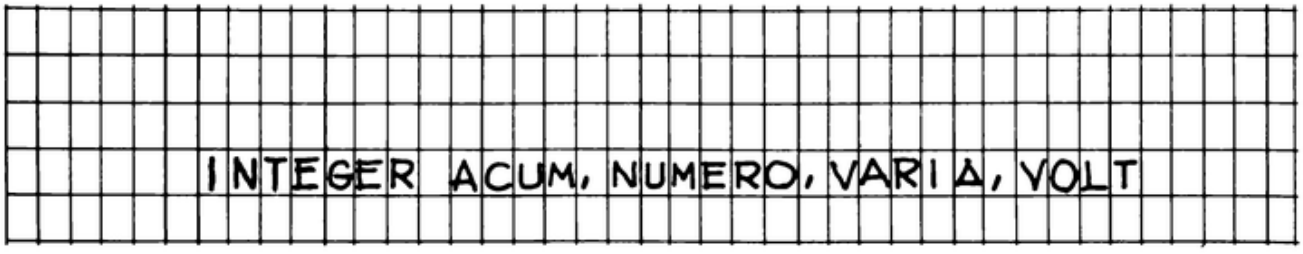
REAL nome1, nome2, nome3, ...
INTEGER nome1, nome2, nome3, ...

onde nome1, nome2, nome3, ... são nomes de variáveis.

Exemplos:



□□ O compilador considerará as variáveis INTEIR e NUMERO como reais, embora elas implicitamente sejam inteiras.



□□ O compilador considerará as variáveis ACUM, VARIA e VOLT como inteiras, embora seja implicitamente reais.

□□ A especificação explícita é muito usada para evitar que tenhamos de usar o artifício de criar nomes tais como IACUM, IFOLHA ou KTOTAL, onde a primeira letra só tem a finalidade de indicar que a variável é inteira.

5

Operações Aritméticas no FORTRAN

□□ As operações aritméticas básicas do FORTRAN são:

adição
subtração
multiplicação
divisão
exponenciação

□□ Os símbolos que representam as operações aritméticas são chamados de "operadores aritméticos". São eles:

\+ (adição)
\- (subtração)
* (multiplicação)
/ (divisão)
** (exponenciação)

□□ O símbolo da exponenciação (**), embora seja constituído de dois caracteres, é considerado como um símbolo único.

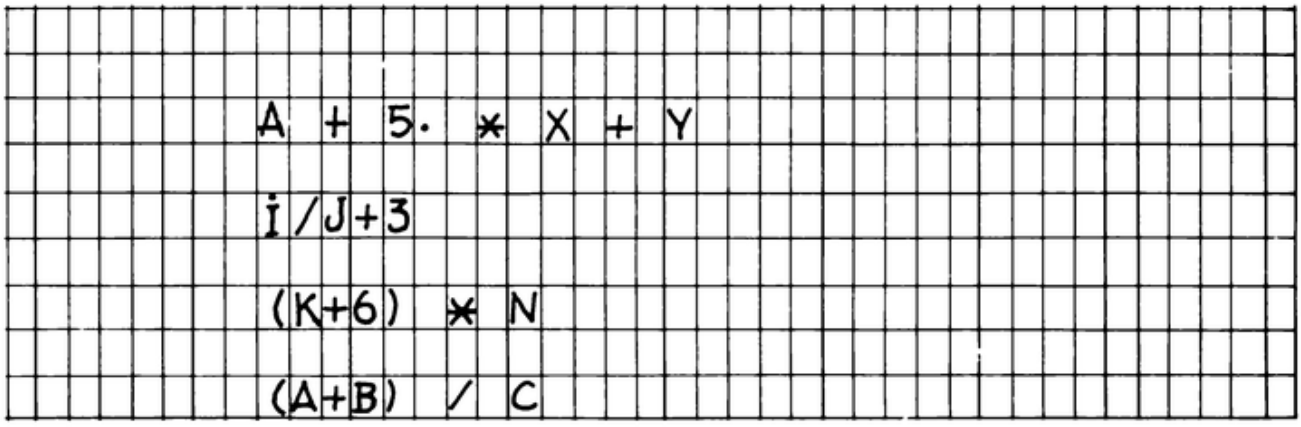
Exemplos:

Operação Aritmética	**Fortran**
A + B	A + B
A − B	A − B
A × B	A * B
A ÷ B	A / B
$\frac{A}{B}$	A / B
A^2	A ** 2
A.B	A * B
AB	A * B
A(B)	A * B

Expressões Aritméticas

□□ O FORTRAN aceita expressões aritméticas formadas pela combinação de constantes, variáveis, operadores aritméticos e parênteses.

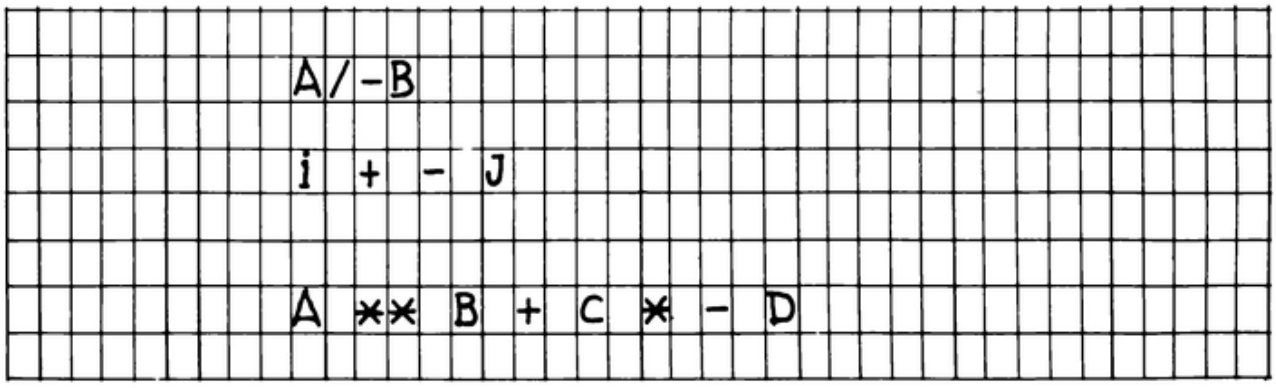
Exemplos de expressões aritméticas:



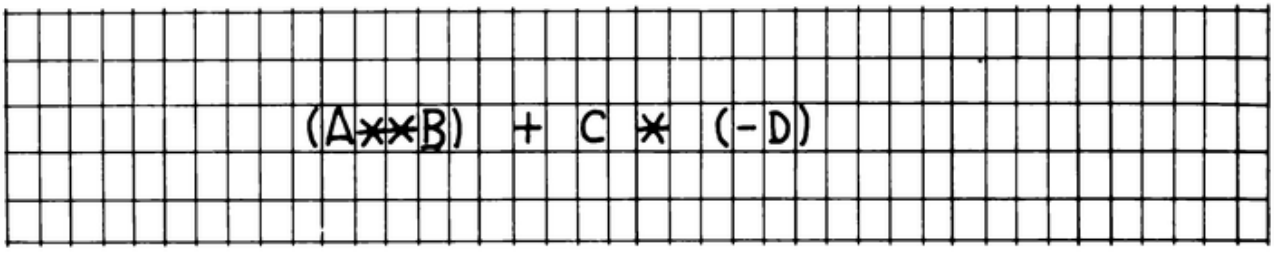
Regras que Regem as Expressões Aritméticas em FORTRAN

1. Não pode haver dois operadores aritméticos juntos. (Não devemos nos esquecer que ** não representa dois operadores de multiplicação e sim o símbolo da exponenciação).

Exemplos de expressões com erro:



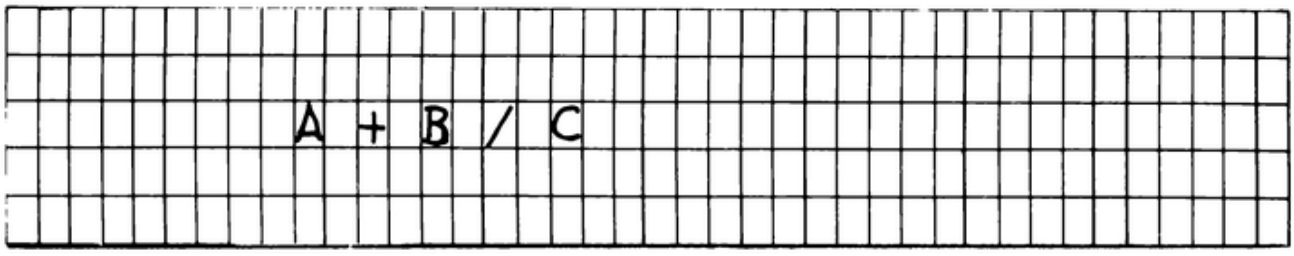
A última expressão deveria ser escrita assim:



2. Quando a expressão aritmética não contém parênteses, as operações são efetuadas na seguinte ordem: em primeiro lugar as exponenciações, em seguida as multiplicações e divisões e finalmente as adições e subtrações.

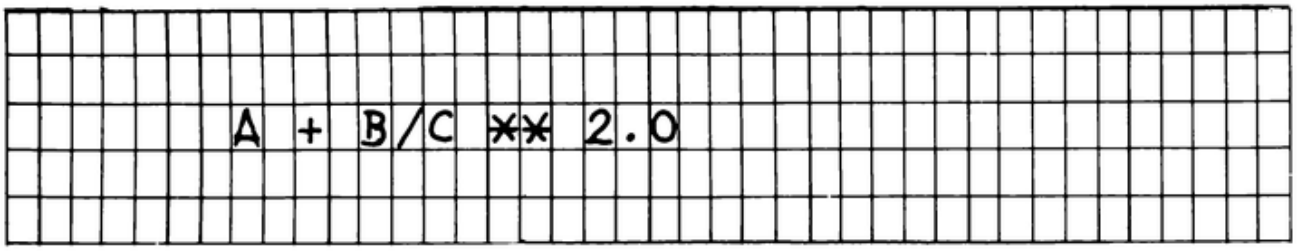
Exemplos:

a) Na expressão



em primeiro lugar é efetuada a operação B/C. O resultado desta operação é somado à variável A.

b) Na expressão

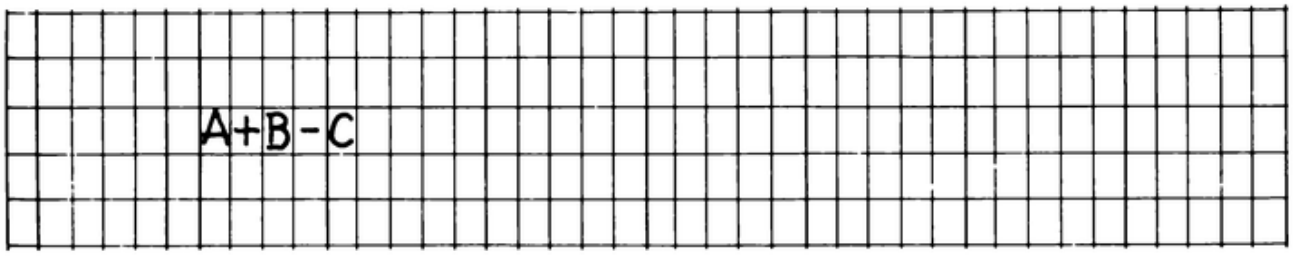


a ordem das operações será a seguinte:
1.ª) C**2.0
2.ª) A divisão de B pelo resultado da operação acima.
3.ª) A soma da variável A com o resultado da operação acima.

3. Dentro do mesmo nível hierárquico as operações são efetuadas da esquerda para a direita, na ordem em que elas aparecem na expressão.

Exemplos:

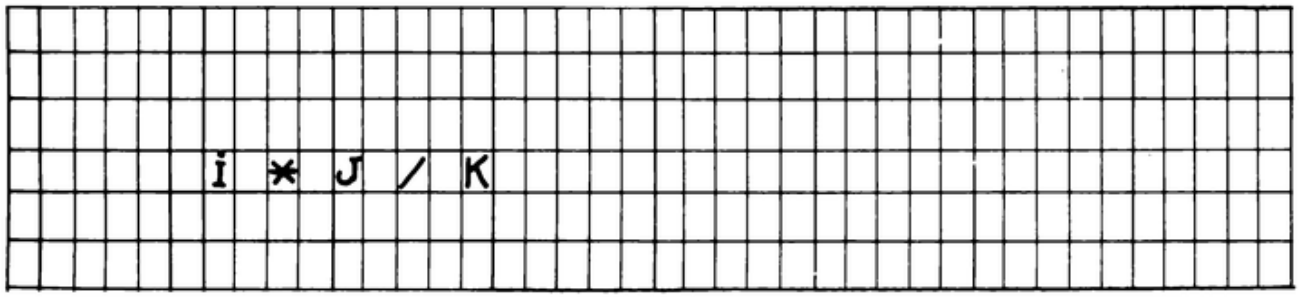
a)



Como a soma e a subtração possuem o mesmo nível hierárquico, a ordem das operações será a seguinte:

1.ª) A + B
2.ª) Ao resultado da operação acima será subtraída a variável C.

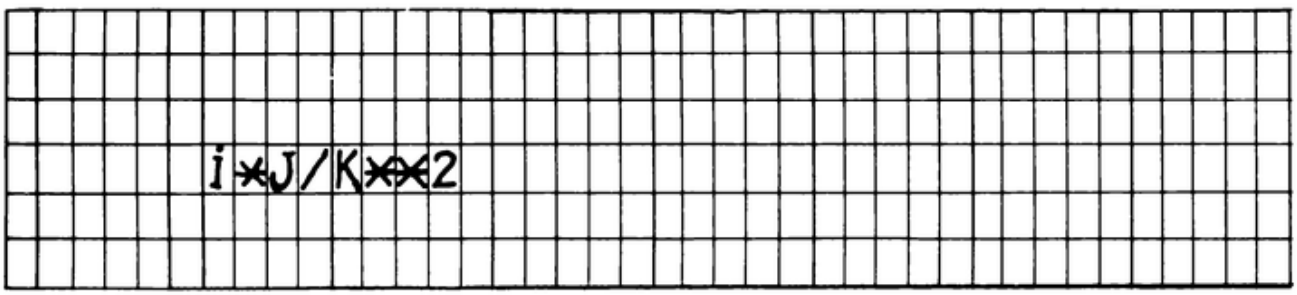
b)



□□ Como a multiplicação e a divisão possuem o mesmo nível hierárquico, a ordem das operações será a seguinte:

1.ª) I * J
2.ª) O resultado da operação acima será dividido por K.

c)



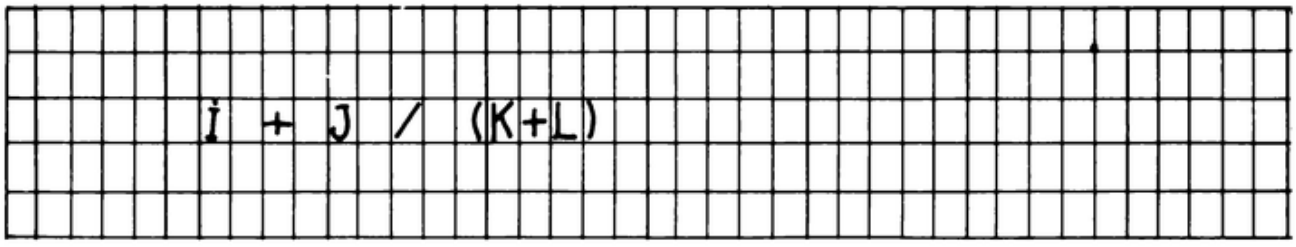
□□ A exponenciação tem procedência sobre a multiplicação e a divisão e, portanto, ela será a primeira operação a ser efetuada. Em seguida, como a multiplicação e a divisão têm o mesmo nível de hierarquia, a multiplicação, por ser a primeira operação da esquerda para a direita, será a próxima a ser executada. Então, a ordem das operações da expressão acima será:

1.ª) K**2
2.ª) I*J
3.ª) O resultado de I*J será dividido pelo resultado de K**2.

4. Quando uma expressão aritmética contém parênteses, as operações indicadas dentro dos parênteses serão executadas em primeiro lugar.

Exemplos:

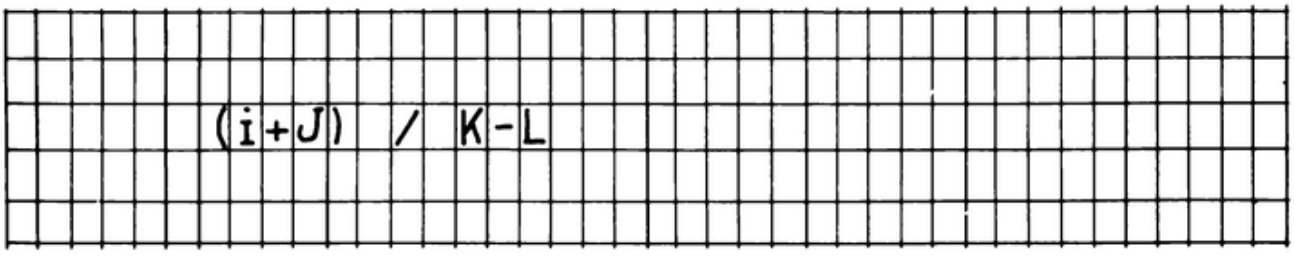
a)



□□ A ordem de execução das operações acima será:

1.ª) K + L
2.ª) A divisão de J pelo resultado da operação acima.
3.ª) A soma de I com o resultado da operação acima.

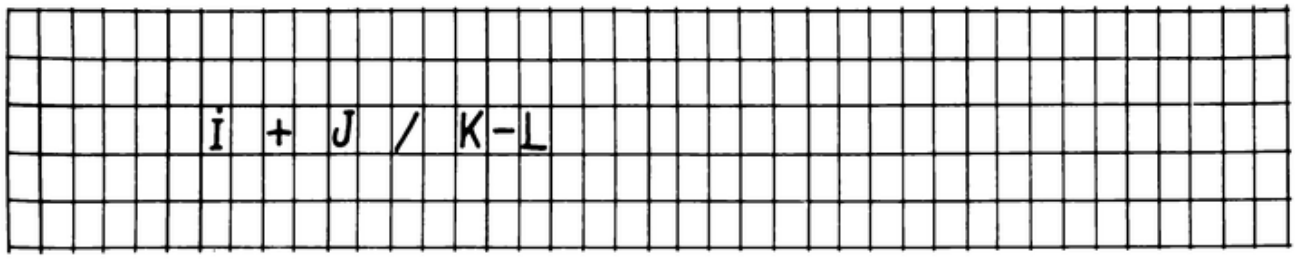
b)



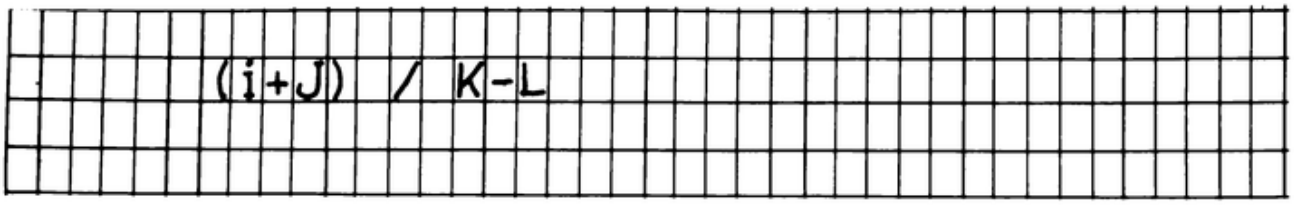
A ordem de execução das operações acima será:

1.ª) I + J
2.ª) A divisão do resultado da operação acima por K.
3.ª) A subtração de L, do resultado da operação acima.

Observação:
Note que a expressão



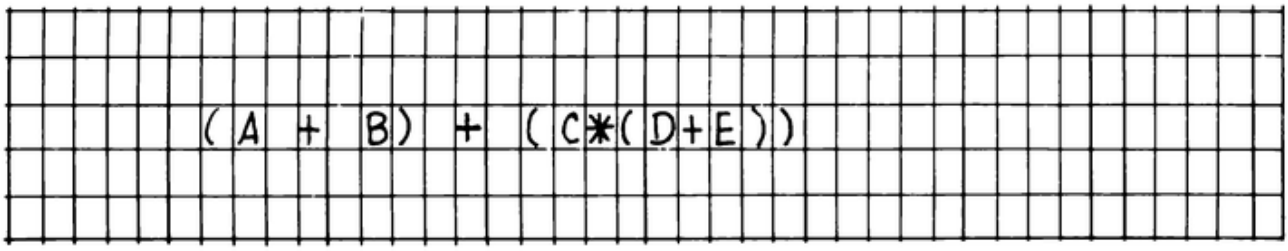
é válida em FORTRAN. A expressão



também é válida. O resultado das duas expressões, no entanto, pode ser completamente diferente, pois a ordem de execução das operações não é a mesma nas duas expressões.

5. O FORTRAN aceita expressões aritméticas com mais de um nível de parênteses. Neste caso, as expressões contidas nos parênteses mais internos são executadas antes das contidas nos parênteses mais externos.

Exemplo:



A ordem de execução das operações acima será:

1.ª) D + E
2.ª) A multiplicação de C pelo resultado da operação acima.
3.ª) A + B
4.ª) A soma do resultado da operação acima, com o resultado da 2.ª operação.

Observação:
O programador deve se certificar que o número de parênteses abertos seja igual ao número de parênteses fechados.

6. O FORTRAN não aceita nenhum operador aritmético implícito. Em conseqüência, as expressões:

   AB
   A (B)
   BA
   B (A)
   (A) B
   (B) A

   não são válidas. Para indicar a multiplicação de A por B precisamos escrever

   A * B

7. A expressão A**B**C é permitida em FORTRAN. O programador, no entanto, deve evitá-la, pois a sua interpretação não é igual em todos os computadores.

   Alguns computadores interpretam a expressão acima assim:

   (A**B) **C

   Outros computadores a interpretam assim:

   A** (B**C)

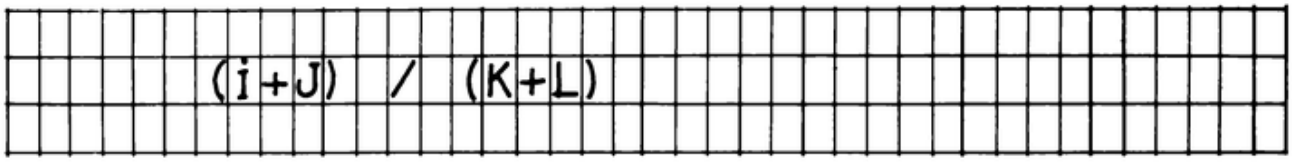
□□ Aconselhamos o programador fazer uso dos parênteses para iniciar a interpretação desejada.

Exercícios

1. Escreva as expressões FORTRAN que equivalem às expressões matemáticas abaixo.

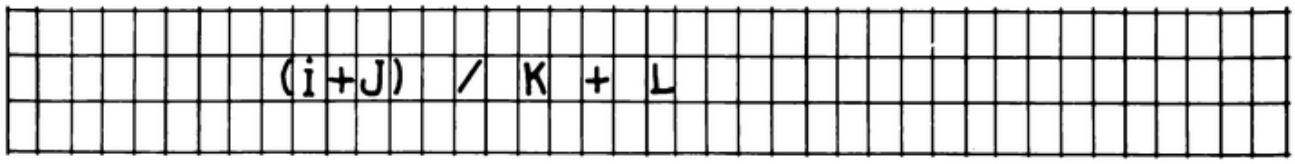
   a) $\frac{I + J}{K + L}$

Resposta:



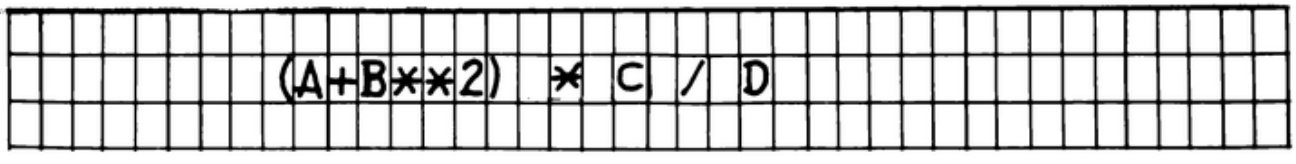
   b) $\frac{I + J}{K} + L$

Resposta:



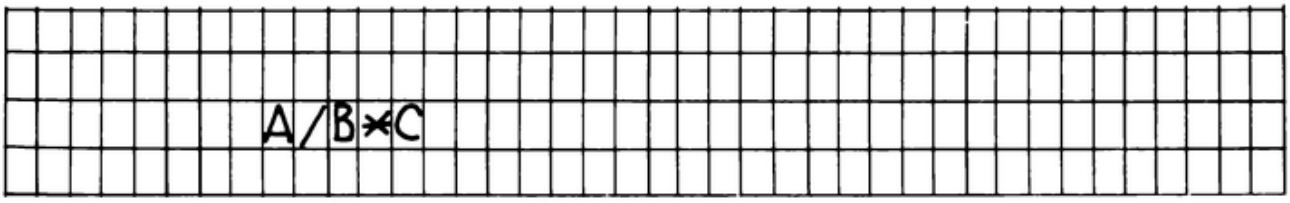
   c) $\frac{(A + B^2)\ C}{D}$

Resposta:



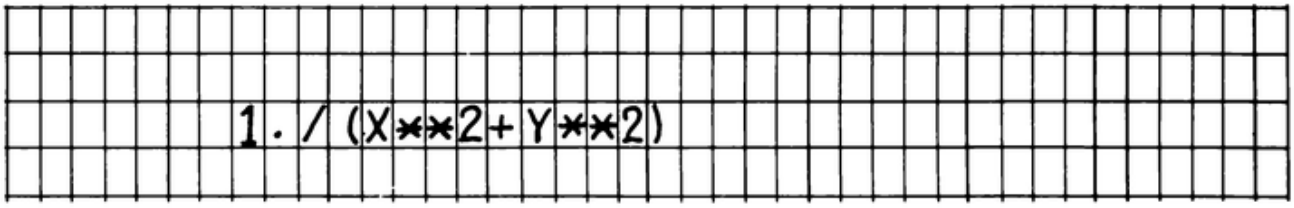
d) $\frac{A}{B} \times C$

Resposta:

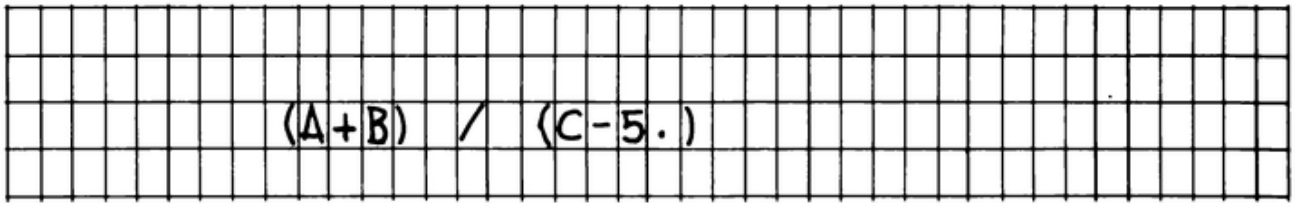


e) $\frac{1}{x^2 + y^2}$

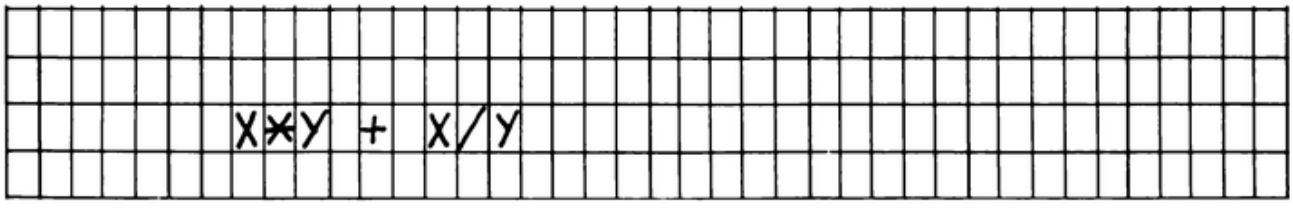
Resposta:



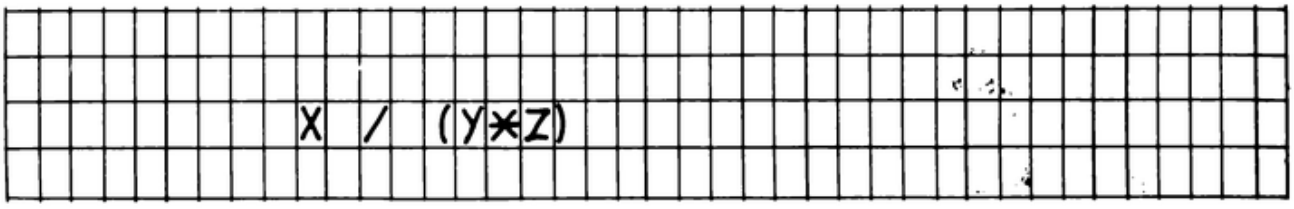
f) $\frac{a + b}{c - 5}$



g) $xy + \frac{x}{y}$



h) $\frac{x}{yz}$



i)

$$\left[\frac{(A + B)\,(A - B)}{2} + C\right]^2$$

Resposta:

```
((A+B) * (A-B) / 2. + C)**2
```

2. Indicar qual o erro nas expressões FORTRAN abaixo.

a)

```
(A+B) (C+D)
```

Resposta: Falta o operador aritmético entre os parênteses. A ausência do operador aritmético NÃO indica multiplicação. Possivelmente o programador desejava escrever

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

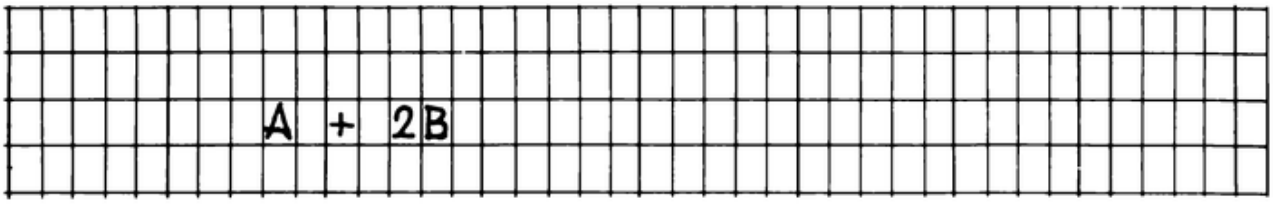
b)

```
A*-B
```

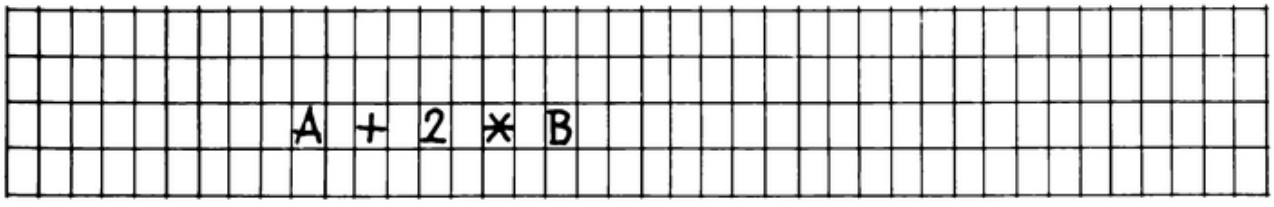
Resposta: Dois operadores aritméticos juntos. Possivelmente o programador desejava escrever

```
A * (-B)
```

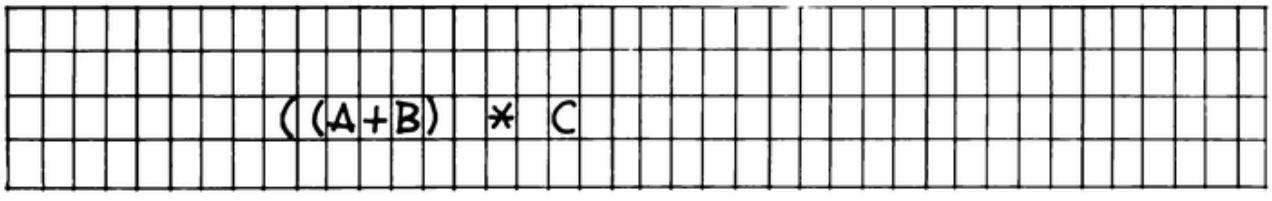
c)



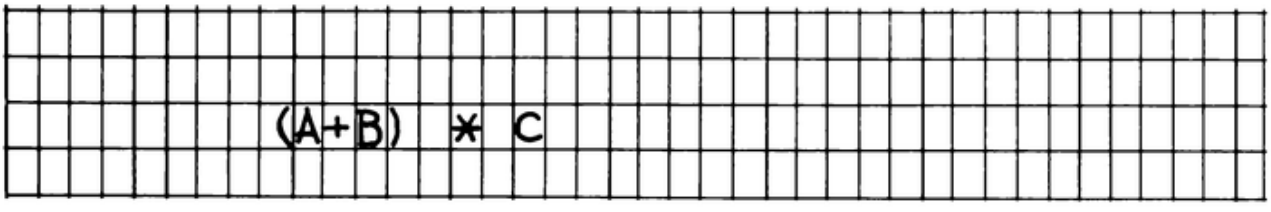
Resposta: Falta operador aritmético. Possivelmente o programador desejava escrever



d)



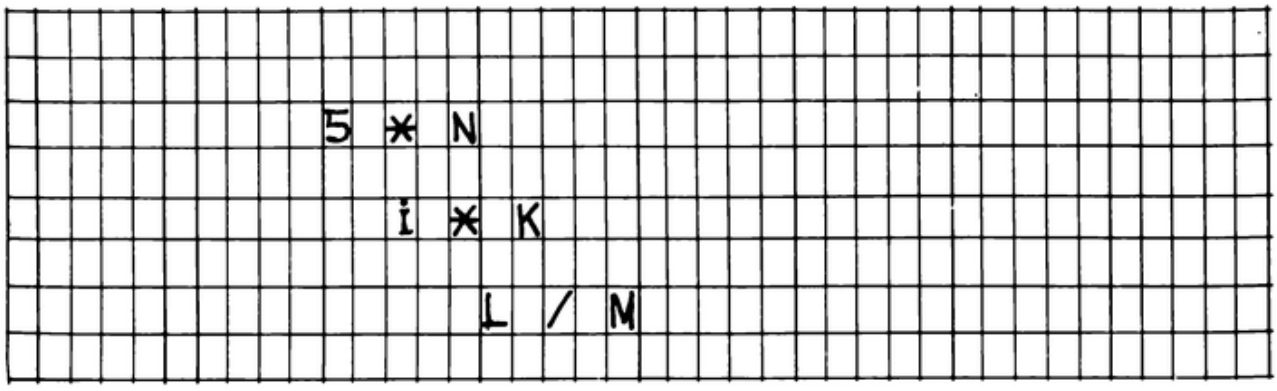
Resposta: Há dois parênteses abertos e apenas um fechado. Possivelmente o programador desejava escrever



Tipos de Expressões Aritméticas

□□ As expressões aritméticas em FORTRAN podem ser do tipo inteiro, do tipo real ou mistas. Elas são do tipo inteiro quando os operandos são do tipo inteiro e o resultado também é um número inteiro.

Exemplos de operações aritméticas inteiras:



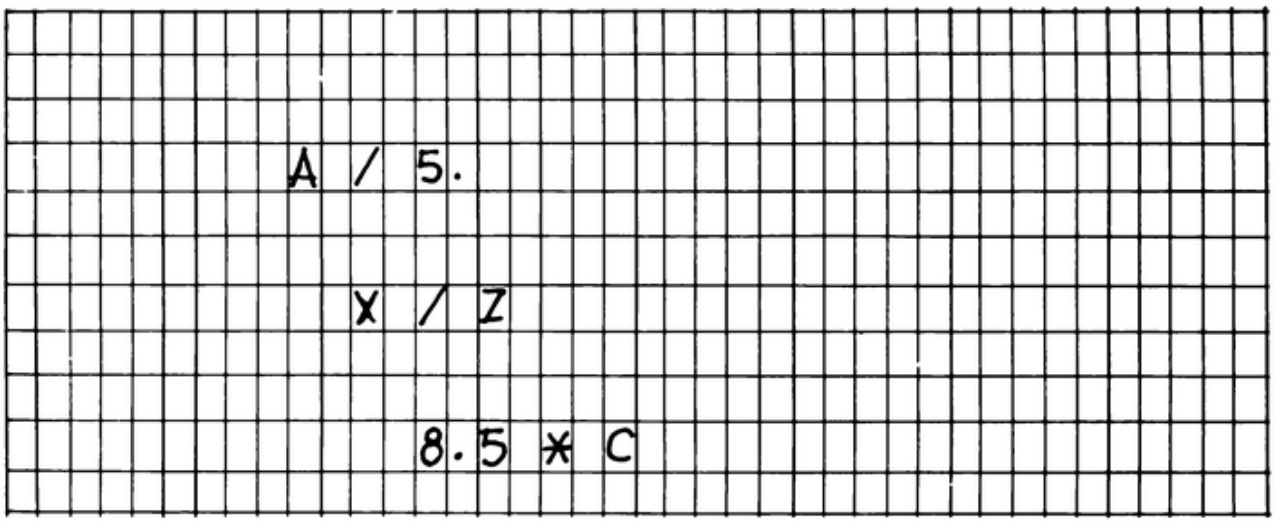
□□ O programador deve se lembrar que o resultado da divisão de uma expressão do tipo inteiro é sempre um número inteiro. Assim, se o resultado da divisão não for exato, ocorrerá um "truncamento", pois o resto será abandonado.

Exemplo: Se na operação L/M, o valor de L for 7 e o valor M for 2, o resultado será

L/M = 7/2 = 3

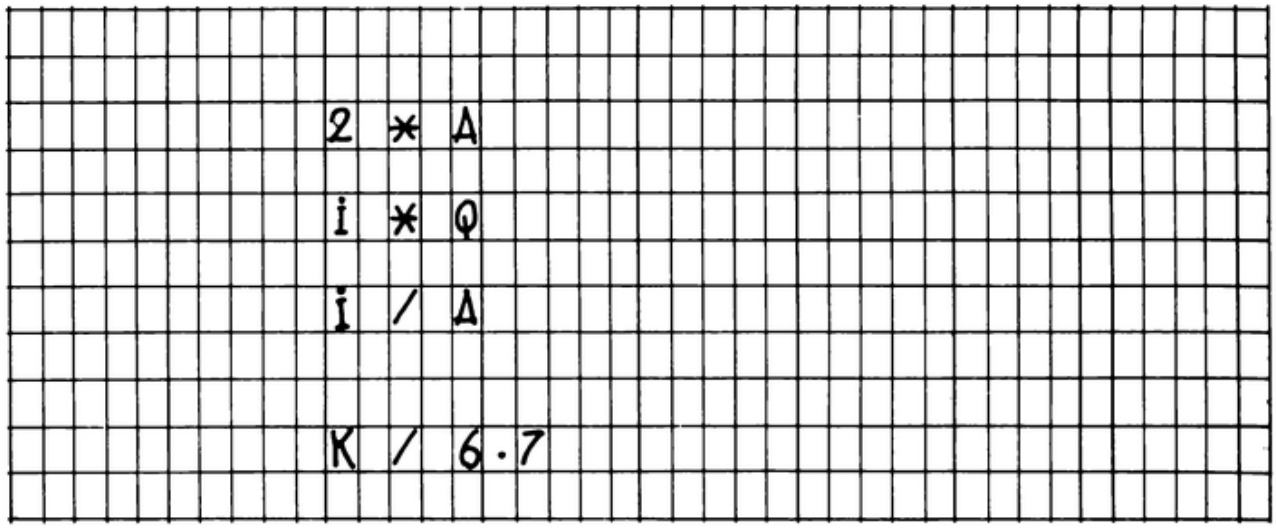
□□ A expressão aritmética é do tipo real quando os seus operandos são do tipo real.

Exemplos de operações aritméticas do tipo real:



□□ A expressão é do tipo misto quando um dos operandos é do tipo real e o outro é do tipo inteiro.

Exemplos de expressões do tipo misto:



□□ As expressões do tipo misto devem ser evitadas, pois elas não são aceitas por todos os tipos de computador.

6

Funções

□□ Além das operações aritméticas apresentadas no Capítulo 5, o FORTRAN aceita determinadas funções escritas dentro das expressões aritméticas.
□□ Para o programador usar uma função, ele deve escrever o nome da função seguido do valor, entre parênteses, para o qual a função será calculada.
□□ O valor para o qual a função será calculada é chamado de "argumento da função" e pode estar representado por uma expressão aritmética.

Exemplos:

a) A expressão matemática:

a.sen θ

expressão FORTRAN:

```
A * SIN (TETA)
```

b) expressão matemática:

$$\text{sen } \frac{b}{2} + \cos b$$

expressão FORTRAN:

```
SIN (B/2.) + COS (B)
```

□□ Uma função pode ter mais de um argumento. Neste caso, os argumentos devem ser escritos dentro dos parênteses e separados por vírgulas.

□□ A avaliação de funções tem precedência sobre todas as outras operações aritméticas e, portanto, nas expressões aritméticas que contêm funções a prioridade de execução é a seguinte:

1. avaliação de funções;
2. exponenciação;
3. multiplicação e divisão;
4. adição e subtração.

As principais funções aceitas pelo FORTRAN são:

- seno;
- co-seno;
- raiz quadrada;
- logaritmo natural;
- tangente;
- arcotangente;
- exponencial;
- valor absoluto;
- resto de uma divisão do tipo inteiro;
- conversão de inteiro para real;
- conversão de real para inteiro.

□□ O quadro a seguir mostra o nome das funções, as funções executadas e o tipo das funções. O programador, no entanto, deve ser cientificado de que nem todas são aceitas por todos os computadores, devendo, antes de escrever o seu programa, verificar quais são as que são aceitas no computador onde seu programa será compilado.

Nome da função	Função executiva	Tipo da função
SIN (R)	Seno	Real
COS (R)	Co-seno	Real
SQRT (R)	Raiz quadrada	Real
ALOG (R)	Logarítmo natural	Real
TAN (R)	Tangente	Real
ATAN (R)	Arcotangente	Real
EXP (R)	Exponencial	Real
ABS (R)	Valor absoluto de R	Real
IABS (I)	Valor absoluto de I	Inteira
MOD (I1, I2)	Resto da divisão de I1 por I2	Inteira
FLOAT (I)	Conversão de inteiro para real	Real
IFIX (R)	Conversão de real para inteiro	Inteira

Convenção: R expressão aritmética do tipo real. I, I1 e I2 expressões aritméticas do tipo inteiro.

Observação: Nenhuma variável pode ter o mesmo nome das funções. Assim, o programador não pode usar os nomes SIN, COS, SORT, ALOG, TAN, ATAN, EXP, ABS, IABS, MOD, FLOAT ou IFIX para identificar uma variável. Alguns computadores aceitam ainda outras funções além das apresentadas.

Exercícios

□□ Escrever a expressão FORTRAN equivalente às seguintes expressões matemáticas:

a) $\sqrt{a^2 + b^2}$

Resposta:

```
SQRT (A**2 + B**2)
```

b) $\text{sen}^2\,(a + b)$

Resposta:

```
SIN (A+B) ** 2.
```

c) $\text{arctg}\ \dfrac{x}{\sqrt{1 - x^2}}$

Resposta:

```
ATAN (X/SQRT(1.X**2))
```

d) $\dfrac{\sqrt{c + d + f}}{\text{sen } p} + \cos(\sqrt{q})$

Resposta:

```
( (SQRT (C+D+F)) / (SIN (P)) ) + COS (SQRT (Q))
```

7

Declaração Aritmética

* * *

□□ A declaração aritmética, considerada como a mais importante do FORTRAN, possui a forma geral

$$V = E$$

que significa o seguinte:

VARIÁVEL = EXPRESSÃO ARITMÉTICA

O sinal "=" não indica uma igualdade como na notação algébrica. Ele indica uma substituição. A expressão A = B + C, por exemplo, indica que o valor da variável A deve ser substituído pelo resultado de B + C. A expressão I = I + 1 indica que o valor da variável I deve ser substituído pelo resultado de I + 1, isto é, ao valor da variável I será somada uma unidade.
□□ Durante a execução de um comando aritmético só será alterado o valor da variável colocada à esquerda do sinal "=", ficando inalterados os valores das variáveis colocadas à direita daquele sinal.

Os comandos do tipo

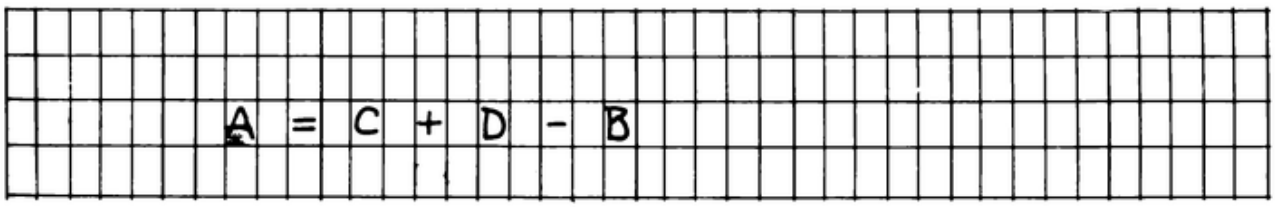
$$A + B = C + D$$

não são permitidos em FORTRAN. O FORTRAN só admite um nome à esquerda do sinal "=".

□□ A expressão acima, no entanto, poderia ser alterada para a forma abaixo, a qual é aceita pelo FORTRAN

$$A = C + D - B$$

□□ Em FORTRAN a expressão acima ficaria assim:



□□ A expressão $4y = 5x + 7$ também não poderia ser escrita em FORTRAN. Bastaria, no entanto, escrevê-la sob a forma

$$y = \frac{5x + 7}{4}$$

para que em FORTRAN ela pudesse ser escrita assim:

```
Y = (5.*X+7.) / 4.
```

□□ Para encontrarmos o valor de y na expressão

$$5x - 3y = 8$$

não podemos transcrever a expressão tal como está para FORTRAN. Devemos nos lembrar que o comando aritmético em FORTRAN só admite uma variável à esquerda do sinal "=" e, portanto, precisamos primeiro transformar a expressão acima para

$$y = \frac{8 - 5x}{3}$$

e só depois então escrever em FORTRAN

```
Y = (8. - 5. * X) / 3.
```

Mudança de Tipos de Números

□□ Num comando aritmético FORTRAN, cuja forma já vimos que é

$$V = E$$

quando V e E são do mesmo tipo, isto é, quando ambos são reais ou quando ambos são inteiros, a execução do comando corresponde a uma simples substituição de valores. Quando, no entanto, V e E são de tipos diferentes, antes da substituição haverá uma mudança de tipo de números (o FORTRAN muda o tipo de E para o tipo de V), conforme indicamos abaixo:

— Se V for inteiro e E for real, haverá truncamento da parte decimal de E. Diz-se, neste caso, que houve uma "fixação".

Exemplo:

$I = 8.16$

O valor de I será substituído por 8.

K = 9.3333

O valor de K será substituído por 9.

— Se V for real e E for inteiro, haverá mudança no tipo de representação sem, contudo, alterar o valor numérico. Diz-se, neste caso, que o número tornou-se real ou foi "flutuado".

Exemplo:

$A = 7/3$

O valor de A será substituído por (2.).

8

Comandos de Entrada e Saída

* * *

□□ Neste capítulo nós vamos estudar como fornecer informações para o computador através de meios de entrada como cartão perfurado, fita magnética, etc., e como imprimir relatórios em impressoras, como fazer o computador gravar fitas magnéticas e como fazê-lo perfurar cartões.

Comando de Entrada (READ)

□□ O comando READ é usado para fazer o computador ler dados de uma unidade de entrada qualquer (leitora de cartões, unidade de fita magnética, console, etc.).

A forma geral do comando READ é

READ (i,n) Lista de variáveis

onde:

i — é uma constante inteira sem sinal ou uma variável inteira cujo conteúdo é um número inteiro sem sinal. O valor de i define a unidade de entrada através da qual o computador lerá os dados de entrada. Há uma correspondência entre o valor de *i* e as diversas unidades de entrada do computador. Geralmente tal correspondência é a indicada no quadro abaixo, mas ela não é obrigatória. Em certos sistemas o valor de i para uma determinada unidade de entrada é atribuído por ocasião da instalação do sistema. Neste livro, adotaremos o valor 2 para indicar a leitora de cartões perfurados.

Unidade de entrada/saída	valor de i
Leitora/perfuradora de cartões IBM-1442	2
Leitora de fita de papel	4
Teclado do console	6
Leitora de cartões IBM-2501	8

n — é o número da declaração FORMAT (estudada logo adiante) que indica como os dados estão dispostos no meio de entrada de dados (cartão perfurado, fita de papel, fita magnética, etc.).

Lista de variáveis — é o nome de uma variável ou os nomes de uma série de variáveis para onde serão transferidos os dados que serão lidos através da unidade de entrada de dados.

Observações sobre as variáveis:

- Uma lista de variáveis pode ser formada por variáveis de tipos diferentes;
- A lista de variáveis pode ser formada por um número bastante grande de nomes de variáveis; (o número exato depende do tipo de computador).
- A disposição em que os valores estão no registro de entrada é indicada no comando FORMAT (estudado logo adiante) referido pelo inteiro *n*.
- O número de variáveis na lista deve ser igual ao número de campos de dados contidos em um cartão. Se o número de variáveis da lista for maior do que o número de campos de dados do registro de entrada, serão lidos tantos cartões sucessivos quantos forem necessários para que o número de campo de dados lidos seja igual ao número de variáveis da lista.

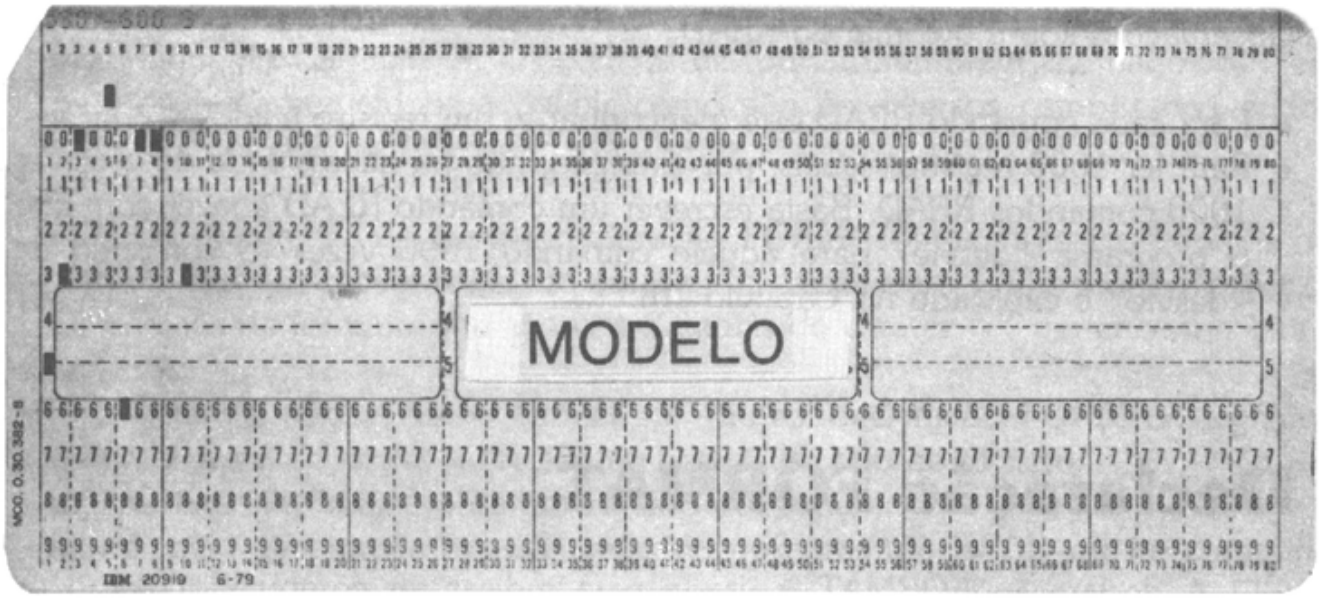
  Se existirem mais campos de dados no registro de entrada do que nomes na lista de variáveis, somente será transmitido o número de campos de dados igual ao número de nomes de variáveis da lista. Os demais campos de dados serão ignorados.

Exemplos:

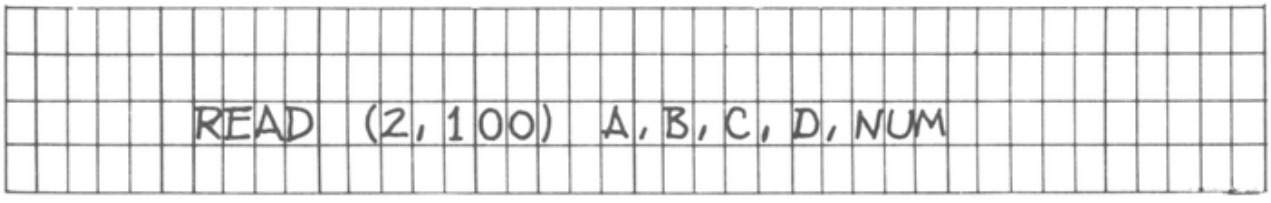
a)

```
READ (2,80) INFO1, INFO2, INFO3
```

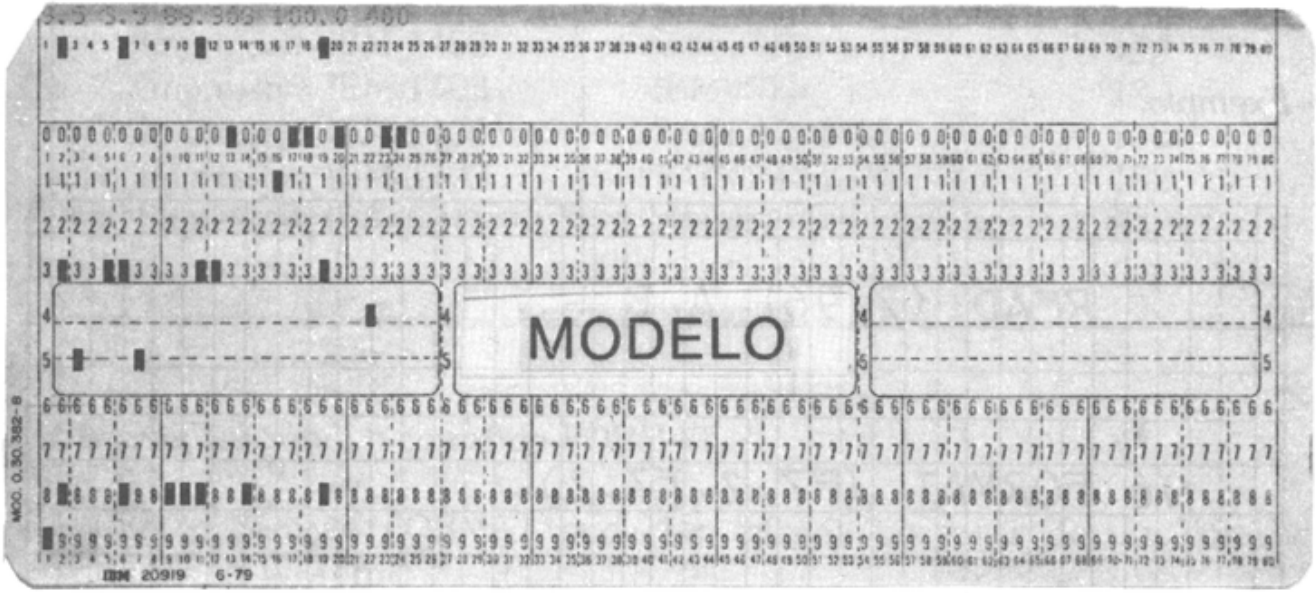
□□ Se, por ocasião da execução do programa, o cartão mostrado a seguir estivesse alimentado na leitora de cartões, o programa o leria e colocaria 530 em INFO1, -600 em INFO2 e 3 em INFO3.



b)



□□ Se por ocasião da execução do programa o cartão abaixo estivesse alimentado na leitora de cartões, o programa o leria e colocaria 9.05 em A, 3.5 em B, 88.308 em C, 100.0 em D e 400 em NUM.



Observações sobre o comando READ:

- Para cada comando READ que é encontrado, um registro é lido;
- Para ler 1000 registros iguais, por exemplo, não há necessidade de escrever 1000 comandos READ. Basta escrever um comando READ e fazer com que o programa "desvie" para aquele comando 1000 vezes (o comando de "desvio" é explicado no Capítulo 10).

Declaração FORMAT

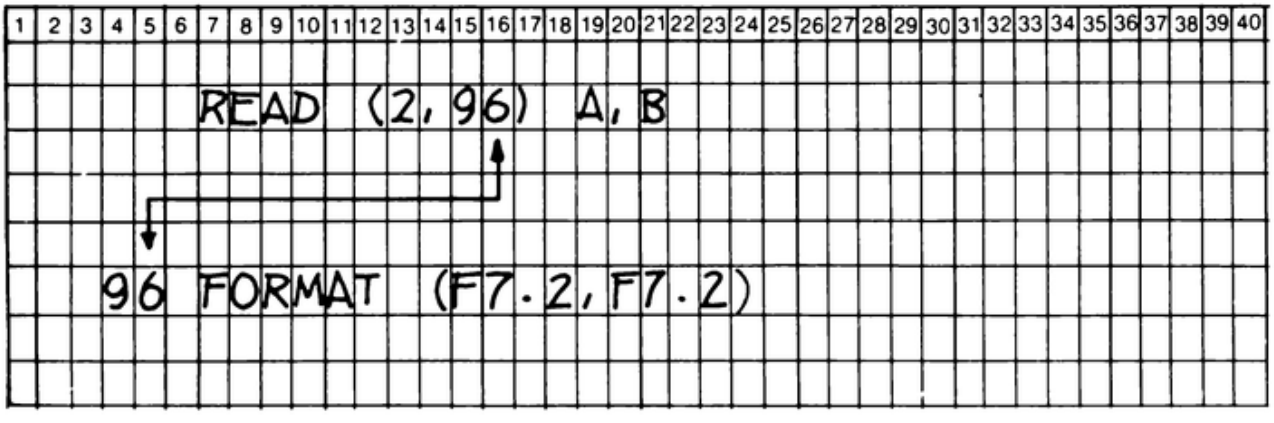
□□ A declaração FORMAT é usada para indicar ao computador como os dados estão dispostos no registro de entrada ou como eles ficarão no registro de saída.
□□ É através da especificação FORMAT que o computador sabe como estão os dados que serão lidos ou gravados.
□□ A declaração FORMAT pode ser escrita em qualquer ponto do programa.
□□ Ela não é executada, pois funciona apenas como uma tabela que os comandos READ e WRITE consultam para saber como estão dispostos os dados que serão processados.
□□ A forma geral da declaração FORMAT é:

n FORMAT (a_1, a_2, a_3, ... an / b_1, b_2, b_3, ... bn / c_1, c_2, c_3 ... / ...)

onde

n — é o número da declaração FORMAT. Este número tem de ser o mesmo que for escrito no comando READ ou no comando WRITE.

Exemplo:



a_1, a_2, a_3, ... an, b_1, b_2, b_3, ... bn, c_1, c_2, c_3 ... cn — são códigos de formato (estudados no Capítulo 9). Os códigos de formato, também chamados de "especificações", servem para definir como são os diversos campos dos dados armazenados nos meios de entrada e saída.

□□ O caráter "/" dentro da declaração FORMAT indica que os códigos de formato que aparecem antes do caráter "/" pertencem a um mesmo registro. Os códigos de formato que aparecem depois do caráter "/" pertencem a outro registro.

□□ Quando não existe o caráter "/" numa declaração FORMAT, significa que o arquivo de dados a ser lido só possui um tipo de registro.

□□ O tamanho máximo de um registro depende:

a) do meio de entrada e saída indicado no comando READ ou no comando WRITE e
b) do computador onde será executado o programa.

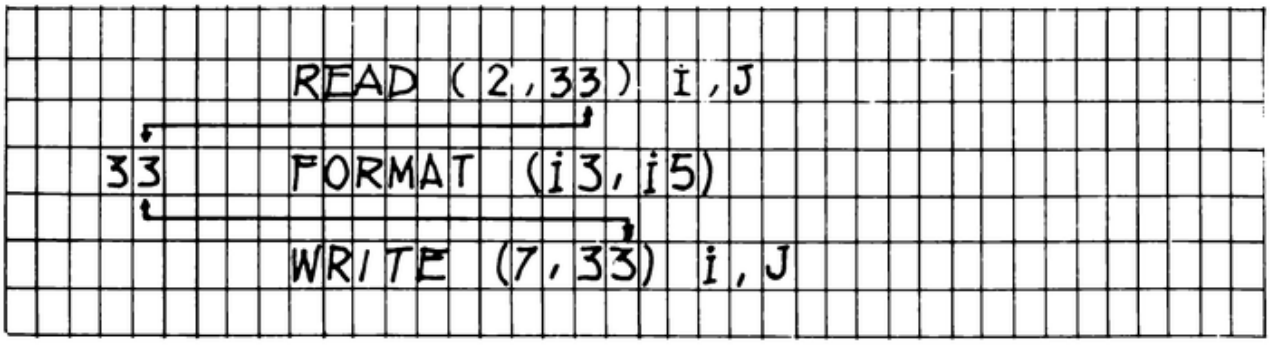
Tamanhos máximos de registro

Unidade de entrada/saída	Computador	N.º máximo de caracteres por registro
Leitora de cartões	diversos	80
Teclado do console	diversos	80
Leitora de fita papel	diversos	80
Impressora IBM-1403	IBM-7044	132
Impressora IBM-1403	IBM-1130	120
Impressora IBM-1403	IBM-/360	132
Impressora IBM-1403	IBM-/370	132
Impressora IBM-1403	FACOM-M160	132
Impressora IBM-1403	FACOM-M200	132
Impressora IBM-3211	diversos	132

Observações sobre o comando FORMAT

- Uma declaração FORMAT pode ser referida por mais de um comando READ ou WRITE.

Exemplos:



□□ Neste capítulo estudamos a função da declaração FORMAT e como ela se liga aos comandos READ e WRITE. No Capítulo 9, após o estudo dos códigos de formato, daremos mais detalhes sobre a declaração FORMAT.

Comando WRITE

□□ O comando WRITE é usado para fazer o computador perfurar dados em cartões IBM, gravar dados em fitas magnéticas ou imprimir relatórios em impressora.

A forma geral do comando WRITE é

WRITE (i,n) Lista de variáveis

onde

i — tem função semelhante à do parâmetro *i* do comando READ. O *i* é uma constante inteira sem sinal ou uma variável inteira cujo conteúdo é um número inteiro sem sinal. O valor de *i* define a unidade de saída através da qual o computador transmitirá os dados de saída. Há uma correspondência entre o valor de *i* e as diversas unidades de saída do computador. Geralmente tal correspondência é a indicada no quadro mostrado a seguir, mas ela não é obrigatória. Em certos sistemas o valor de *i* para uma determinada unidade de saída é atribuído por ocasião da instalação do sistema.

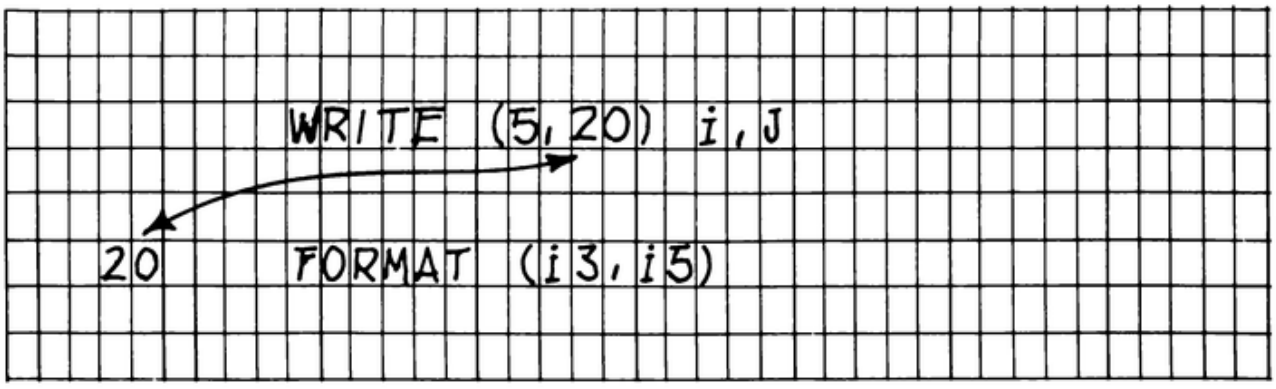
□□ Neste livro adotaremos o valor 5 para indicar a impressora.

Unidades de saída	Valor de i
Impressora do console	1
Perfuradora de cartões IBM-1442	2
Impressora IBM-1132	3
Perfuradora de fita de papel	4
Impressora 1403	5
Plotador	7

n — é o número da declaração FORMAT que indica como os dados deverão ficar no meio de saída (cartão, fita de papel, formulário contínuo, fita magnética, etc.).

Lista de variáveis — é o nome de uma variável ou os nomes de uma série de variáveis de onde serão retirados os dados que deverão ser enviados à unidade de saída.

Exemplo:



9

Dados de Entrada e de Saída

* * *

Entrada, Processamento, Saída

□□ Os serviços executados num computador têm, basicamente, o esquema abaixo:

Entrada (Input)

□□ A "entrada" seria o conjunto de informações fornecidas ao computador para efetuar os cálculos necessários. Por exemplo: para o computador fazer uma relação de clientes com atraso nos pagamentos, a entrada seria o conjunto de:

□□ a) nomes dos clientes;

□□ b) datas dos pagamentos efetuados e seus respectivos vencimentos.

□□ Com base nestas informações o computador poderia, após ser programado, fazer a relação dos clientes em atraso.

Saída (Output)

□□ A "saída" é o resultado desejado. No exemplo anterior, a saída seria a relação de clientes em atraso.

Processamento

□□ A transformação da entrada, com o objetivo de obter a saída desejada, é conhecida como "processamento".

Dados

□□ As informações que entram ou que saem do computador são conhecidas como "dados" em computação eletrônica.
□□ Os conceitos de "dado" e de "informação" são distintos à luz da Teoria da Comunicação. Em processamento de dados, no entanto, "informação" e "dado" têm o mesmo significado.

Dados de Entrada

□□ Os dados que entram para o computador, isto é, as informações que são fornecidas para o computador, recebem a designação de "dados de entrada".

Dados de Saída

□□ As informações geradas pelo computador, ou seja, as informações que o computador nos fornece após o processamento são conhecidas como "dados de saída".

Meios de Entrada

□□ O material usado para levar o dado de entrada até o computador é reconhecido como "meio de entrada". Os principais meios de entrada são: a fita magnética, o disco magnético, a fita de papel, o cartão IBM (ou cartão perfurado).
□□ Por exemplo: Se o cartão perfurado nas apostas da loteria esportiva fosse usado diretamente para fazer a conferência das apostas, o cartão perfurado seria o meio de entrada. Os dados de entrada seriam as apostas e as demais informações perfuradas no cartão.

Meios de Saída

□□ O material usado pelo computador para nos fornecer os dados desejados é o "meio de saída". Os principais meios de saída são:
□□ — o formulário contínuo;
□□ — a fita magnética;
□□ — o disco magnético;
□□ — o cartão IBM (ou cartão perfurado).

□□ Por exemplo: Suponhamos que uma empresa emitisse uma relação em ordem alfabética de todos os seus fornecedores. Os nomes dos seus fornecedores seriam os dados de saída. O formulário contínuo empregado na relação seria o meio de saída.

Unidades de Entrada / UCP / Unidades de Saída

UCP — Unidade Central de Processamento

□□ O que comumente chamamos de "computador eletrônico" não é apenas uma máquina, mas um conjunto de várias máquinas interligadas.
□□ A principal delas é a unidade central de processamento, também conhecida como UCP, responsável pela efetuação dos cálculos e pelo controle das demais máquinas.

Unidades de Entrada/Leitura

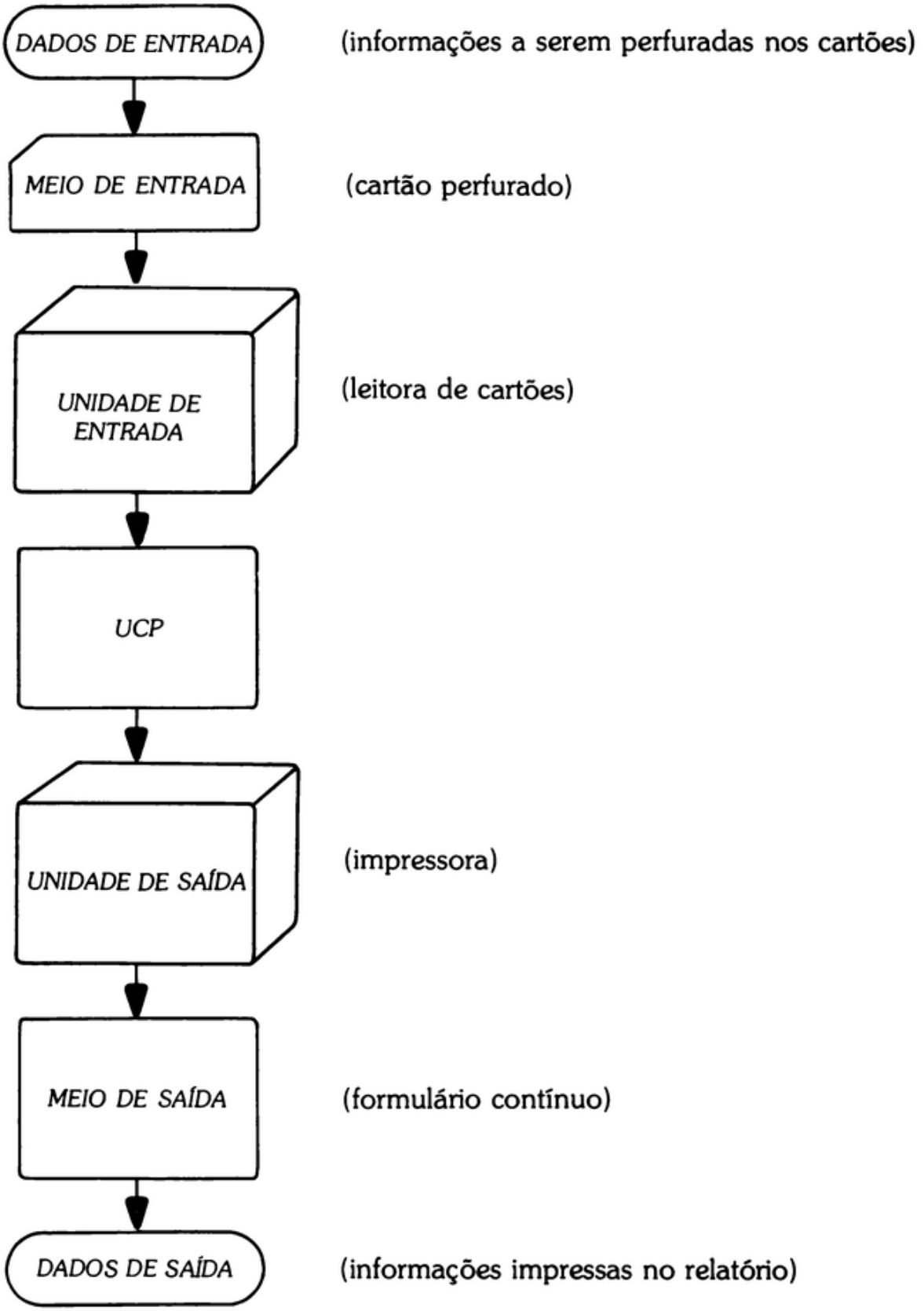
□□ As máquinas cuja função é captar os dados de entrada contidos nos meios de entrada são chamadas de "unidades de entrada". O processo de captar os dados contidos nos meios de entrada é conhecido como "leitura".
□□ Por exemplo: A "leitora de cartões" é capaz de "ler" os cartões perfurados. Isto significa que a máquina conhecida como "leitora de cartões" é capaz de reconhecer as informações codificadas em cartões perfurados e remeter tais informações para a UCP (unidade central de processamento).
□□ As principais unidades de entrada são:
□□ — leitora de cartões;
□□ — leitora de fitas de papel;
□□ — unidade de fita magnética (veremos adiante que a unidade de fita magnética é uma unidade de ENTRADA/SAÍDA porque não só é capaz de ler os dados de entrada como é capaz, também, de gravar dados de saída);
□□ — unidade de disco magnético (veremos mais adiante, que este tipo de máquina é, também, unidade de ENTRADA/SAÍDA, podendo ler os dados de entrada ou gravar dados de saída).

Unidades de Saída

□□ Os dados de saída podem ser fornecidos pelo computador, como vimos, de várias maneiras; em cartão perfurado, impressos em formulários contínuos, gravados em discos magnéticos ou em fitas magnéticas.

□□ As máquinas que têm por função fornecer-nos os dados de saída são as "unidades de saída", tais como:
□□ — a perfuradora de cartões, cuja função é perfurar cartões com os dados de saída;
□□ — a impressora, cuja função é imprimir os dados de saída em formulários contínuos (formulários contínuos são folhas ligadas umas às outras, formando pilhas de formulários. A união entre uma folha e outra é picotada, de modo a facilitar o destacamento manual de uma folha das outras, após o serviço de impressão. No caso de serviços muito grandes há máquinas especiais para cortar os formulários nos locais apropriados);
□□ — a unidade de disco magnético, cuja função como unidade de saída é gravar em discos magnéticos os dados de saída. (As unidades de discos magnéticos são capazes, também, de ler dados de entrada gravados em discos magnéticos. Por esta sua dupla capacidade de ler/gravar dados, a unidade de disco magnético é conhecida como unidade de ENTRADA/SAÍDA.)
□□ — a unidade de fita magnética, cuja função como unidade de saída é gravar em fitas magnéticas os dados de saída. (As unidades de fitas magnéticas são capazes, também, de ler dados de entrada gravados em fitas magnéticas. Por esta sua dupla capacidade de ler/gravar dados, a unidade de fita magnética é conhecida como unidade de ENTRADA/SAÍDA.)

□□ A figura a seguir sintetiza os conceitos estudados até agora.

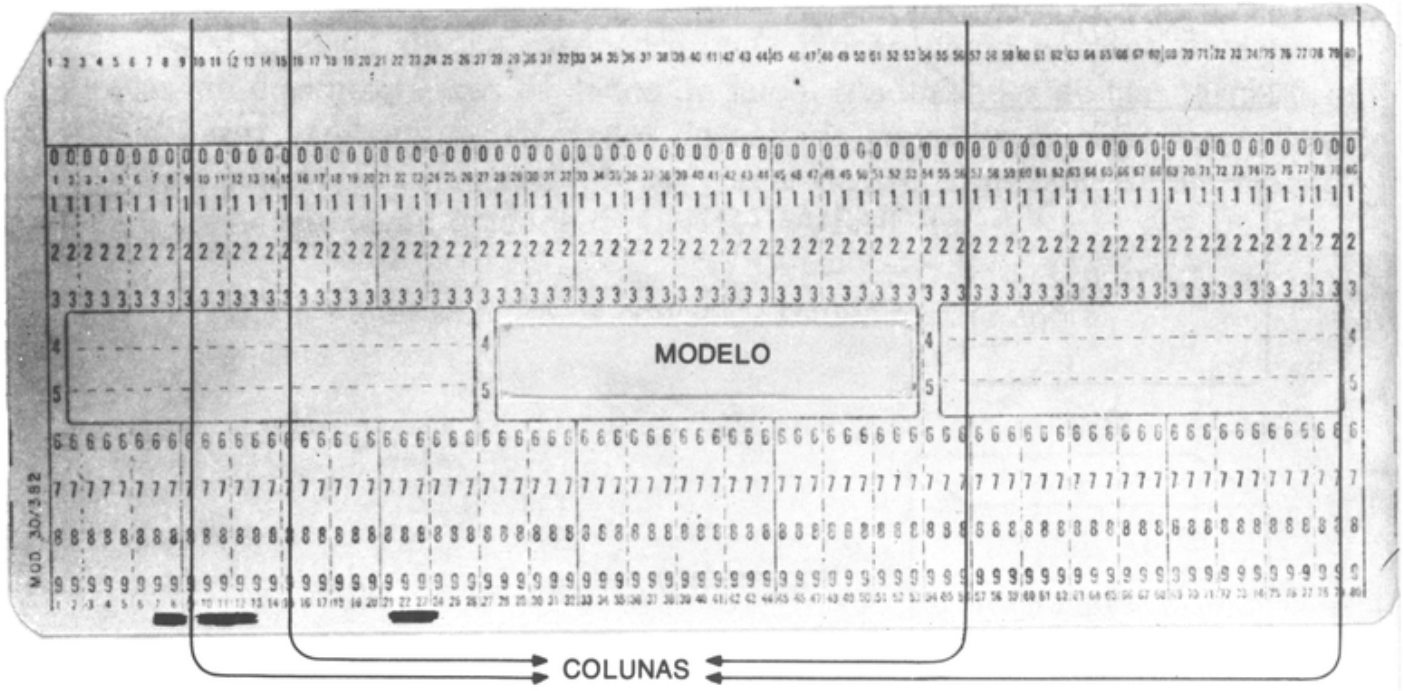

DADOS DE ENTRADA
(informações a serem perfuradas nos cartões)
MEIO DE ENTRADA
(cartão perfurado)
UNIDADE DE ENTRADA
(leitora de cartões)
UCP
UNIDADE DE SAÍDA
(impressora)
MEIO DE SAÍDA
(formulário contínuo)
DADOS DE SAÍDA
(informações impressas no relatório)

O Cartão Perfurado

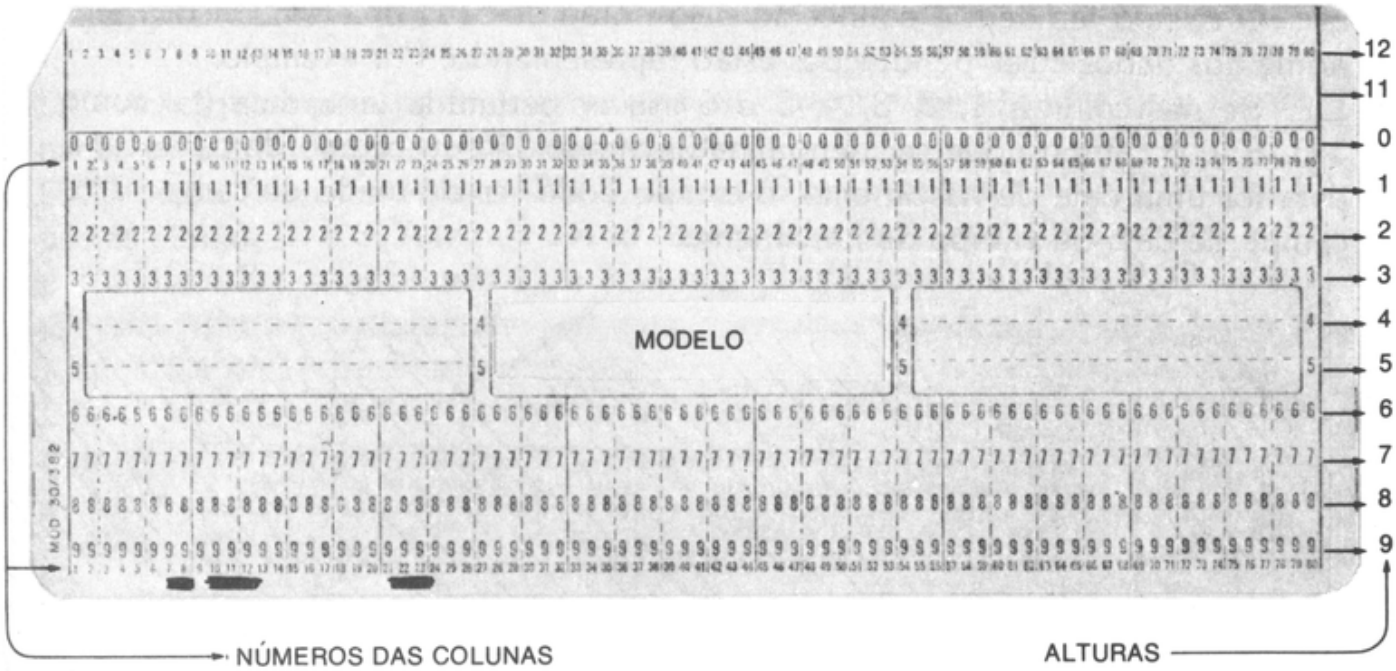
□□ Para que um dado seja gravado na memória principal é preciso que antes ele tenha sido introduzido através de um meio de entrada. Como vimos, os principais meios de entrada são o disco magnético, a fita magnética, a fita de papel e o cartão perfurado (ou cartão IBM).

□□ A leitura dos discos e fitas magnéticas é feita tão-somente pela máquina. O cartão perfurado, no entanto, terá, às vezes, de ser lido por nós. É importante, portanto, que estejamos familiarizados com o cartão perfurado e a codificação empregada em sua perfuração.

□□ Existe mais de um tipo de cartão perfurado, porém o mais usado é o chamado "cartão de 80 colunas". Ele assim é chamado porque possui 80 colunas, sobre cada uma das quais poderá ser perfurado um dígito ou caráter.



□□ As oitenta colunas são numeradas de 1 a 80. Geralmente os números das colunas vêm impressos no cartão, conforme mostrado na figura a seguir.



□□ A figura acima mostra, também, o que chamamos de altura. Cada coluna é dividida em doze alturas. As alturas 12, 11 e 0 são chamadas de altura de zona. As alturas, 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 são chamadas de alturas numéricas. Assim, a altura 0 tanto é altura de zona como é altura numérica.

□□ Os algarismos são perfurados no cartão exatamente na altura de idêntico número. Por exemplo:

□□ — o algarismo 7 é perfurado na altura 7;

□□ — o algarismo 3 é perfurado na altura 3;

□□ — o algarismo 1 é perfurado na altura 1 e assim por diante.

□□ Como só é possível perfurar um algarismo em cada coluna, para perfurar um número de dois algarismos faz-se perfurações em duas colunas. Para perfurar um número de três algarismos, perfura-se três colunas. Para perfurar um número de *n* algarismos perfura-se *n* colunas.

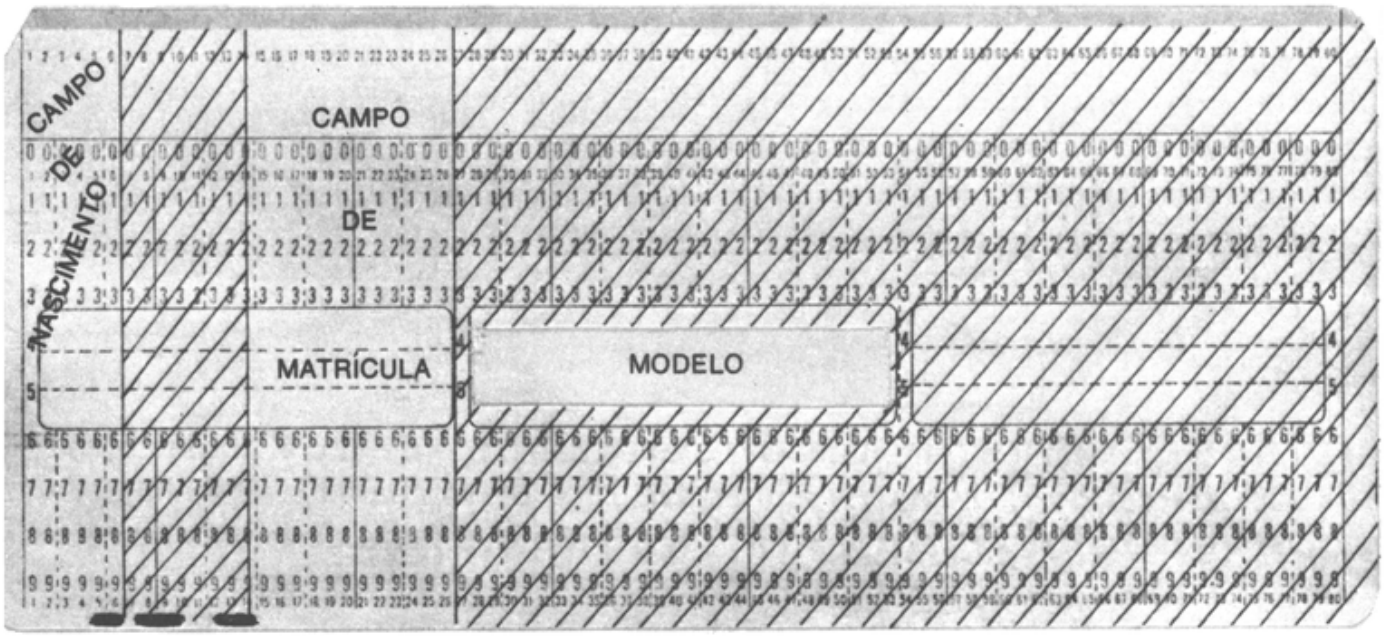
Campo

□□ O conjunto de duas ou mais colunas é conhecido como "campo". Portanto, para se perfurar um número de mais de 1 algarismo utiliza-se um campo.

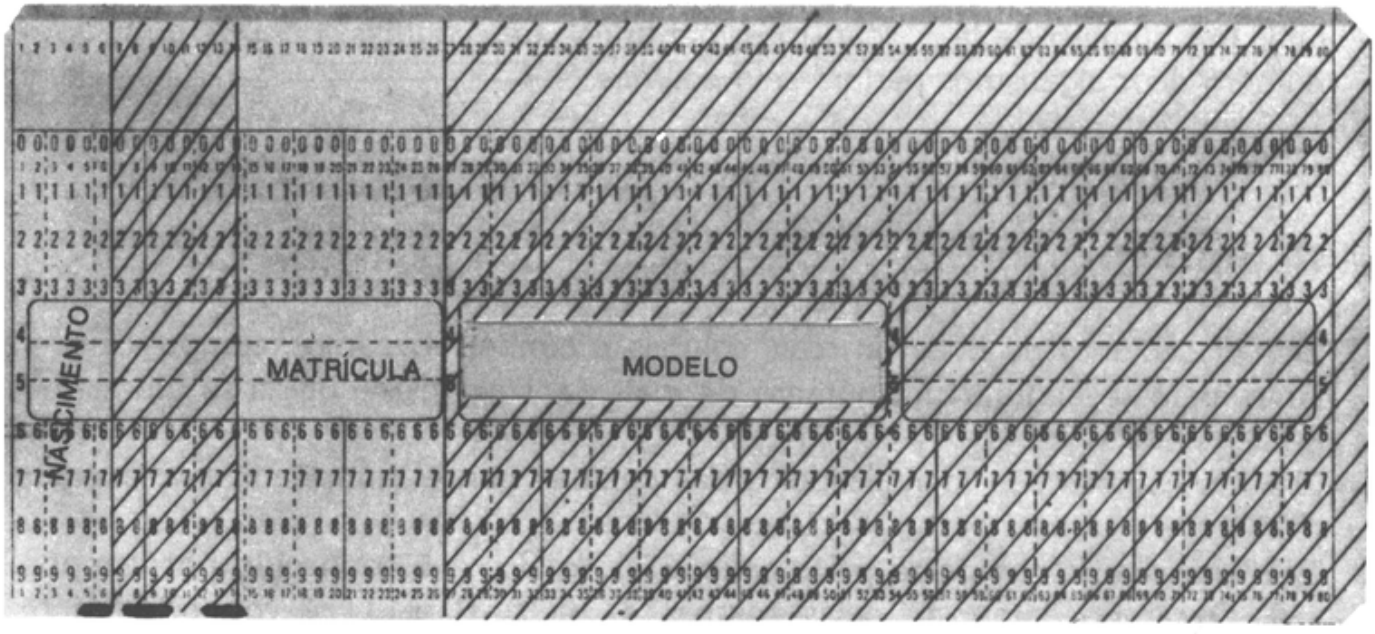
□□ Os campos são identificados pelas colunas que o compõem. Por exemplo:

□□ Um campo composto pelas colunas, 1, 2, 3 e 4 é conhecido por campo 1/4. O campo composto pelas colunas 7, 8, 9, 10 e 11 é conhecido por campo 7/11 (campo 7 a 11).

□□ O campo também é conhecido pelo nome do seu conteúdo, isto é, pelo nome dos dados cujas perfurações estão representando. Por exemplo:
□□ Se nas colunas 1, 2, 3, 4, 5 e 6 estiver perfurada uma data (tal como 141183), o campo pode ser denominado campo 1/6 ou campo de data. Se a data for uma data de nascimento, o campo pode ser chamado de campo 1/6, campo de data ou campo de nascimento.



□□ Geralmente omite-se a palavra "campo" nos *layouts* dos cartões. O *layout* anterior, por exemplo, ficaria assim:



Campo Alfanumérico

□□ A máquina para perfurar uma letra ou caráter especial utiliza uma combinação de altura de zona e altura numérica.

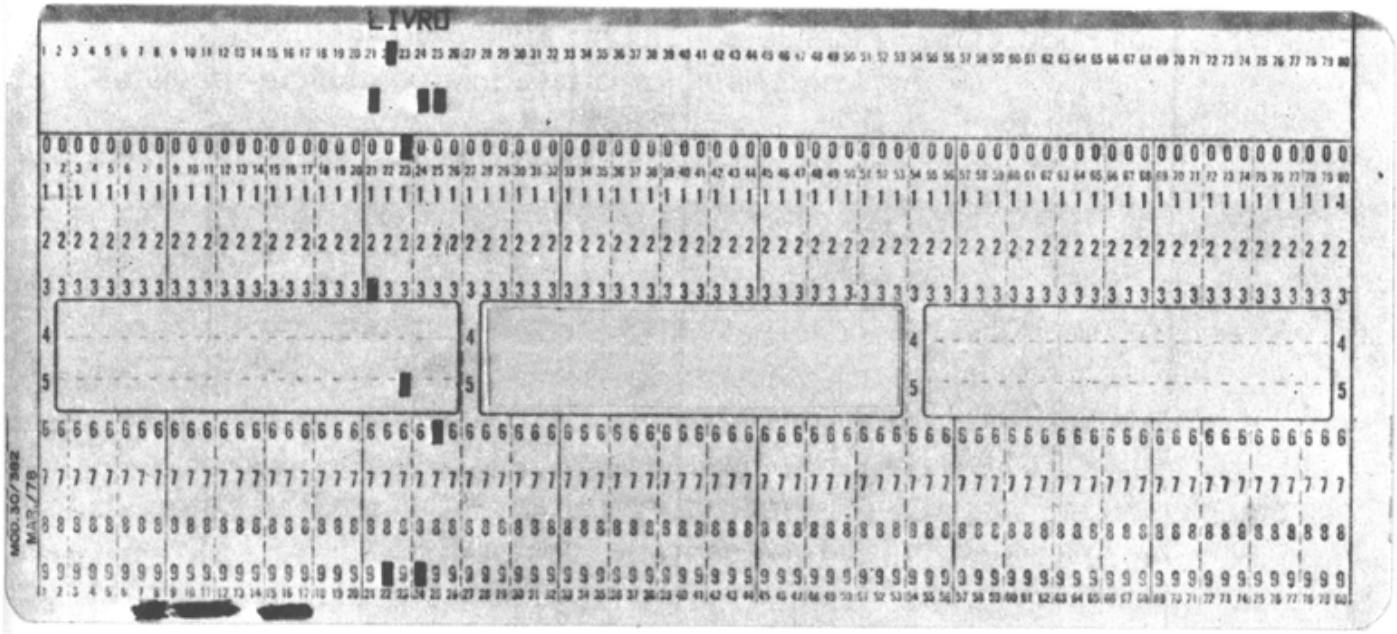
□□ A letra A usa as alturas 12 e 1. A letra B usa as alturas 12 e 2. A letra C usa as alturas 12 e 3. A letra I usa as alturas 12 e 9. A letra J usa as alturas 11 e 1. A letra K usa as alturas 11 e 2. A letra R usa as alturas 11 e 9. O caráter especial "/" usa as alturas 0 e 1. A letra S usa as alturas 0 e 2. A letra T usa as alturas 0 e 3. A letra Z usa as alturas 0 e 9.

□□ Como só pode ser perfurada uma letra em cada coluna, para perfurar uma palavra de duas letras será necessário um campo de 2 colunas; para perfurar uma palavra de 3 letras será necessário um campo de 3 colunas e assim por diante.

□□ Por exemplo:

□□ Para perfurar a palavra LIVRO haverá necessidade de 5 colunas do cartão, porque a palavra "livro" tem 5 letras. O analista poderá escolher 5 das 80 colunas onde será perfurada a palavra "livro". Posteriormente indicará à máquina, por meio de um programa, quais são as colunas que conterão a palavra desejada.

□□ Na figura a seguir foram escolhidas as colunas 21/25 para perfurar a palavra "livro".



□□ Além das perfuradoras ligadas diretamente ao computador, que fazem perfurações automaticamente sem intervenção humana, existem máquinas especiais para fazer perfuração nos cartões. Nestas máquinas, chamadas perfuradoras, ao se acionar a tecla A, a máquina, perfurará automaticamente as

alturas 12 e 1. Ao apertar a tecla R a máquina perfurará automaticamente as alturas 11 e 9. A coluna, onde deverão ocorrer as perfurações correspondentes à letra desejada, pode ser selecionada à vontade de quem estiver trabalhando com a máquina.

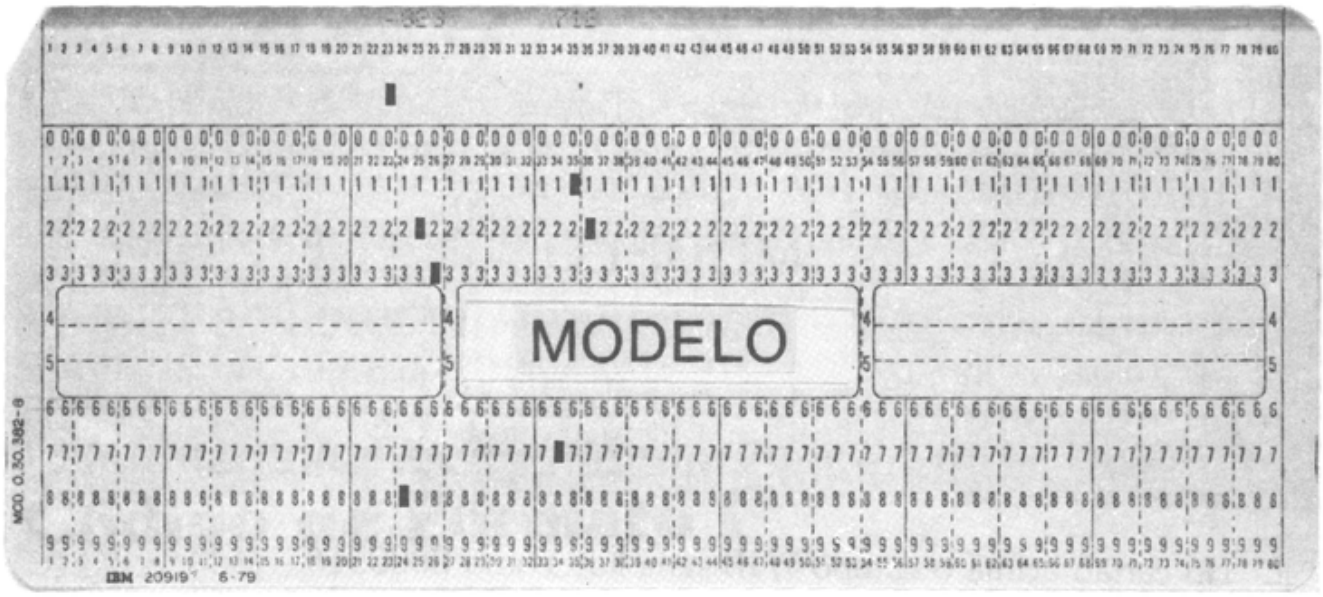
A tabela a seguir fornece as alturas perfuradas em cartão, correspondentes às letras, números e principais caracteres especiais.

Caráter	Alturas
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	12—1
B	12—2
C	12—3
D	12—4
E	12—5
F	12—6
G	12—7
H	12—8
I	12—9
J	11—1
K	11—2
L	11—3
M	11—4
N	11—5
O	11—6
P	11—7
Q	11—8
R	11—9
S	0—2
T	0—3
U	0—4
V	0—5
W	0—6
X	0—7
Y	0—8
Z	0—9
/	0—1
—	11
.	12—3—8
(	12—5—8
)	11—5—8
&	12
?	0—7—8
=	6—8

Tratamento dos Campos num Programa FORTRAN

□□ Nos campos numéricos de entrada o sinal (+) pode ser omitido para os valores positivos.

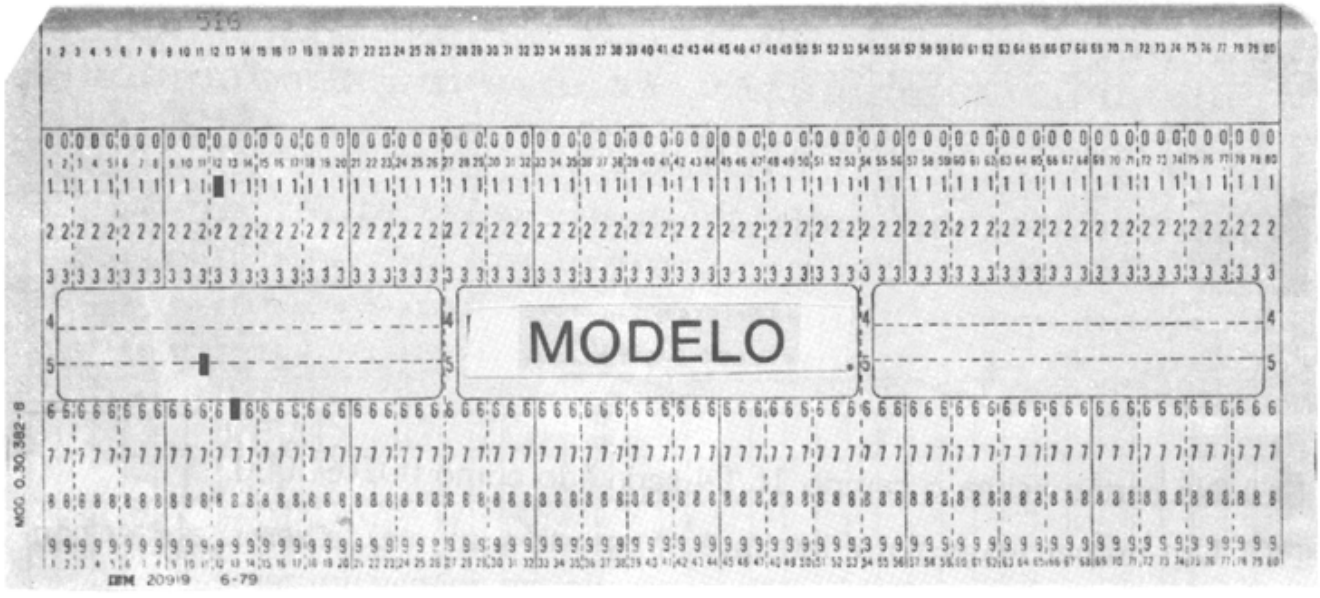
Exemplos:



□□ Os dois campos do cartão acima seriam lidos como + 600.

□□ Nos campos numéricos de entrada o sinal (−) *não* pode ser omitido, pois se ele for omitido o valor será considerado positivo.

Exemplo:

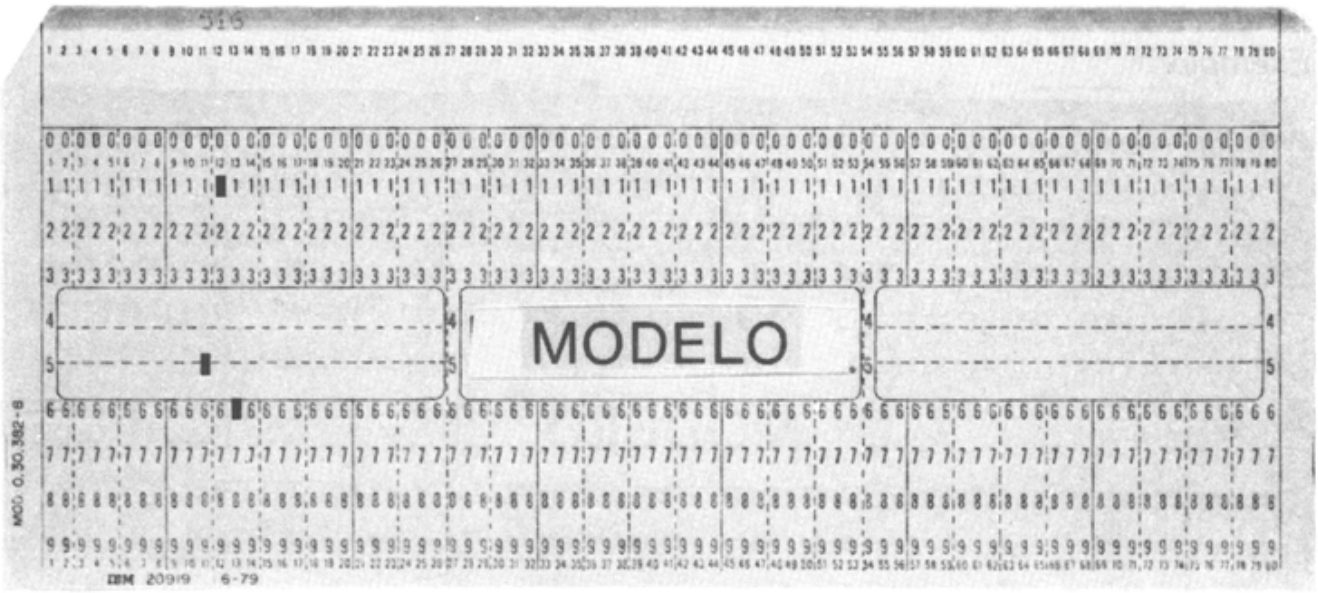


□□ No cartão acima os dois campos seriam lidos como − 823 e + 712.

□□ No campos numéricos de entrada as colunas não perfuradas (em branco) são interpretadas como zeros.

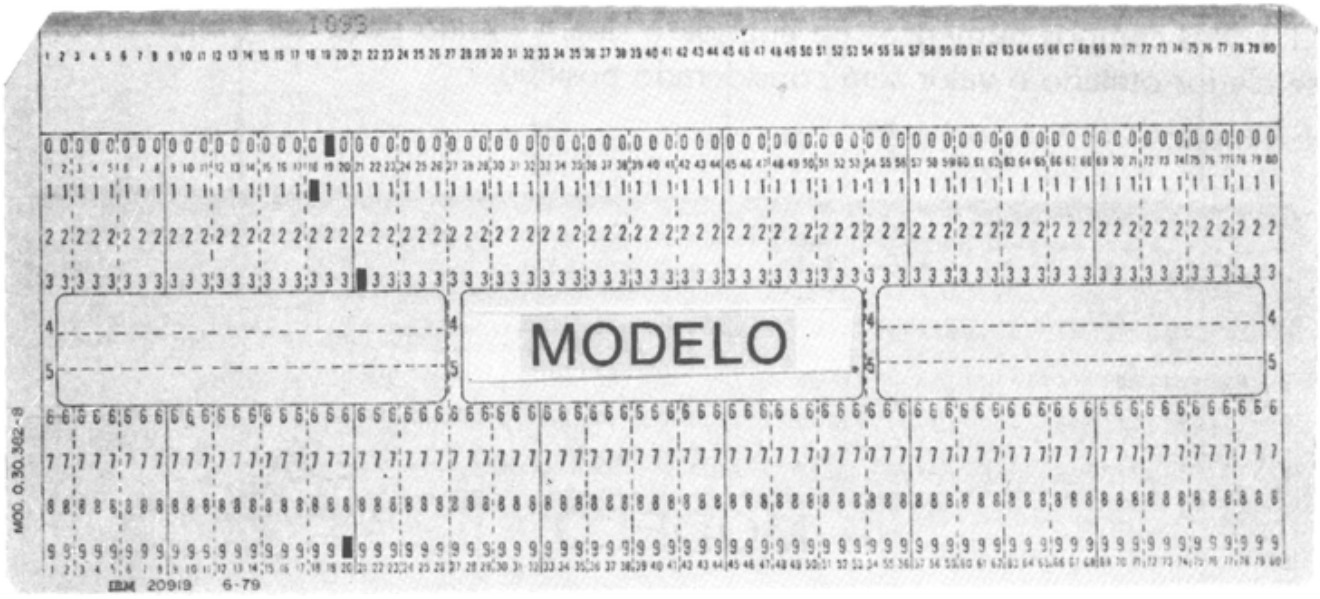
Exemplos:

a)



□□ No cartão acima o campo 7/13 seria lido como 0000516.

b)



□□ No cartão acima o campo 16/24 seria lido como 001093000.

□□ Um número cuja parte inteira é composta somente de zeros, ao ser considerado como dado de saída é transformado em apenas um zero.

Exemplo: o valor 000 seria impresso assim: 0

□□ Ao serem transferidos para um campo de saída, os zeros não significativos da parte inteira de um número são eliminados.

Exemplo: O valor +076 ficaria assim: 76
O valor −00100 ficaria assim: − 100.

Códigos de Formato

□□ Os códigos de formato, também chamados de "especificações", servem para definir como são os diversos campos dos dados armazenados nos meios de entrada e saída.
□□ Os códigos de formato são escritos no comando FORMAT, o qual já foi estudado no Capítulo 8.
□□ Neste capítulo estudaremos os códigos de formato e daremos mais informações sobre o comando FORMAT.

Código de Formato I

□□ O código de formato I indica que a variável de entrada ou de saída é do tipo inteiro.

□□ A sua forma geral é

Iw

onde

w — é uma constante inteira sem sinal que especifique o tamanho do campo utilizado.

Exemplo

a)

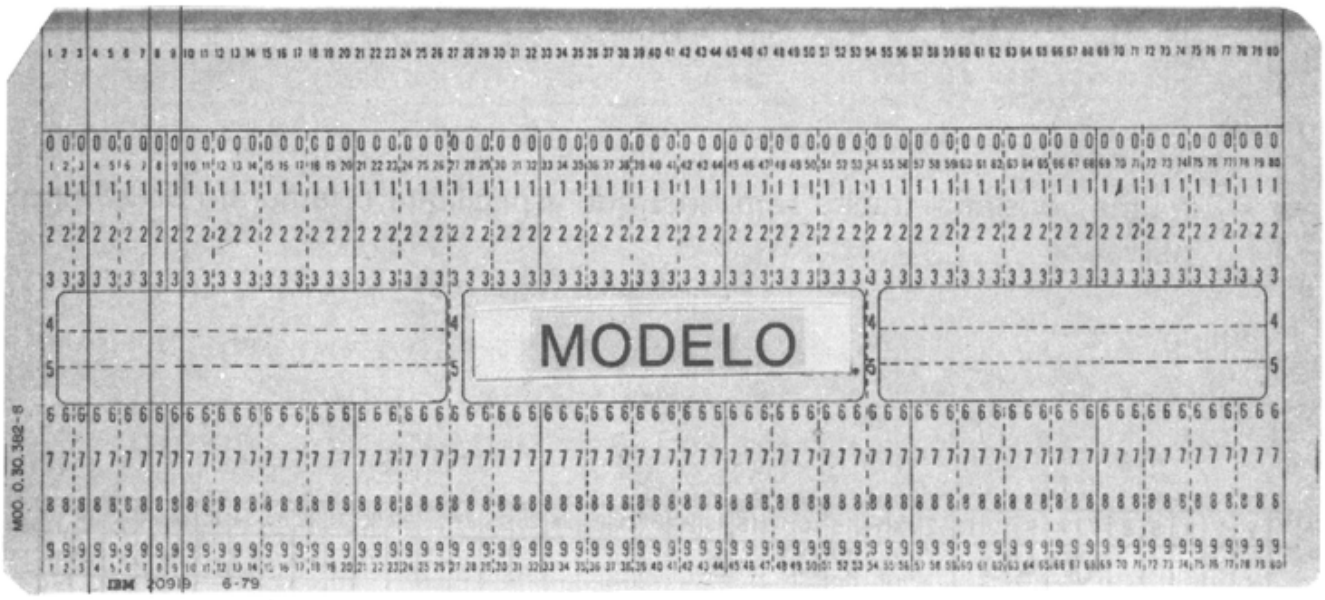
```
      READ (2, 20) K
20    FORMAT (i3)
```

□□ Neste exemplo, após a leitura, a variável K conteria o valor perfurado nas colunas 1, 2 e 3 do cartão, pois o código de formato I3 indica que o tamanho do campo de entrada é de 3 colunas.

b)

```
      READ (2, 20) K, L, M
20    FORMAT (I3, I4, I2)
```

□□ Neste exemplo, após a leitura, a variável K conteria o valor perfurado nas colunas 1, 2 e 3 (por causa do código de formato I3); a variável L conteria o valor perfurado nas colunas 4, 5, 6 e 7 (por causa do código de formato I4); a variável M conteria o valor perfurado nas colunas 8 e 9 (por causa do código de formato I2).



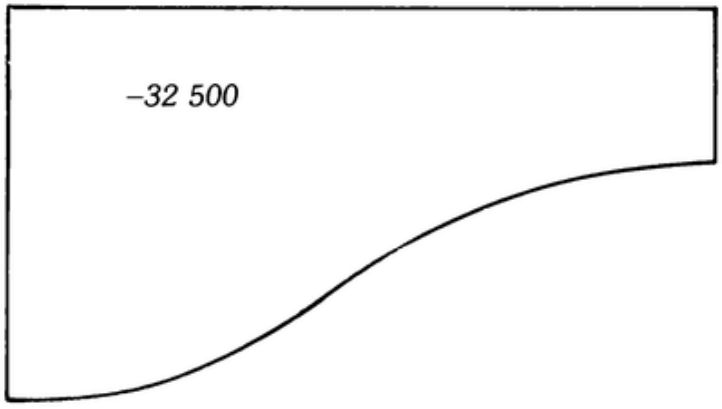
□□ O campo utilizado deve ter uma coluna reservada para o sinal sempre que o valor que ele contiver for um valor negativo. Por medida de segurança, é conveniente especificar os campos numéricos sempre com uma posição a

mais para ser ocupada pelo sinal, independentemente de o valor contido no campo ser positivo ou negativo.

Exemplo:

```
      READ (5, 91) K, L
   91 FORMAT (i3, i4)
```

□□ Se o valor contido em K for −32 e o valor contido em L for +500 a linha impressa será a seguinte:

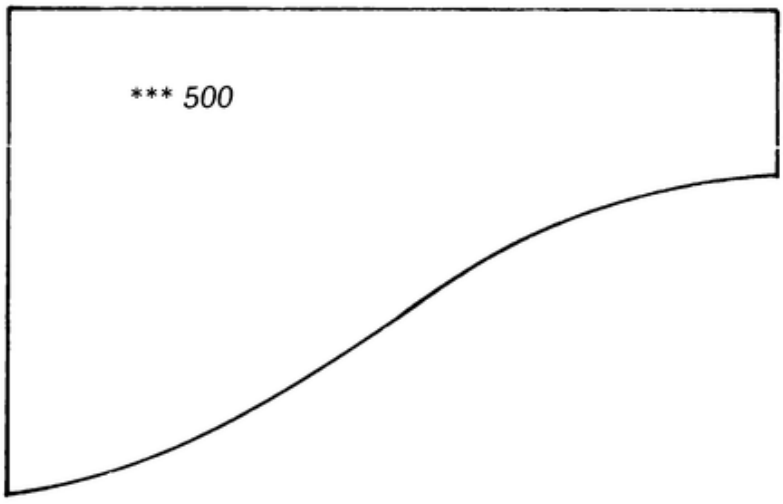


□□ Quando há um erro de dimensionamento e o valor da variável é maior do que o tamanho do campo especificado, ou o campo especificado não comporta o tamanho do valor e mais o sinal, o campo mal dimensionado é preenchido com asteriscos.

□□ Se no exemplo anterior,

```
      WRITE (5, 91) K, L
   91 FORMAT (i3, i4)
```

o valor contido em K fosse −933 e o valor de L fosse +500, a linha impressa seria a seguinte:



Exercício

□□ Escrever o comando WRITE e a declaração FORMAT para imprimir o valor de uma variável inteira de nome INFO, sabendo que ela pode variar de −999 a +999.

Solução:

```
      WRITE (5,23) INFO
   23 FORMAT (I4)
```

Observações:

a) O código de formato é I4 para poder conter o sinal + ou −.
b) O número 23 no comando WRITE acima poderia ser substituído por outro qualquer, desde que o número que o substituísse fosse informado também na declaração FORMAT.

Código de Formato X

□□ O código de formato X indica a quantidade de posições a serem saltadas, isto é, se o registro for de entrada ele indica quantas colunas consecutivas devem ser ignoradas; se o registro for de saída ele indica quantas posições "em branco" (quantos caracteres "branco") devem ser deixadas entre um campo e outro.

□□ O seu formato geral é

wX

onde

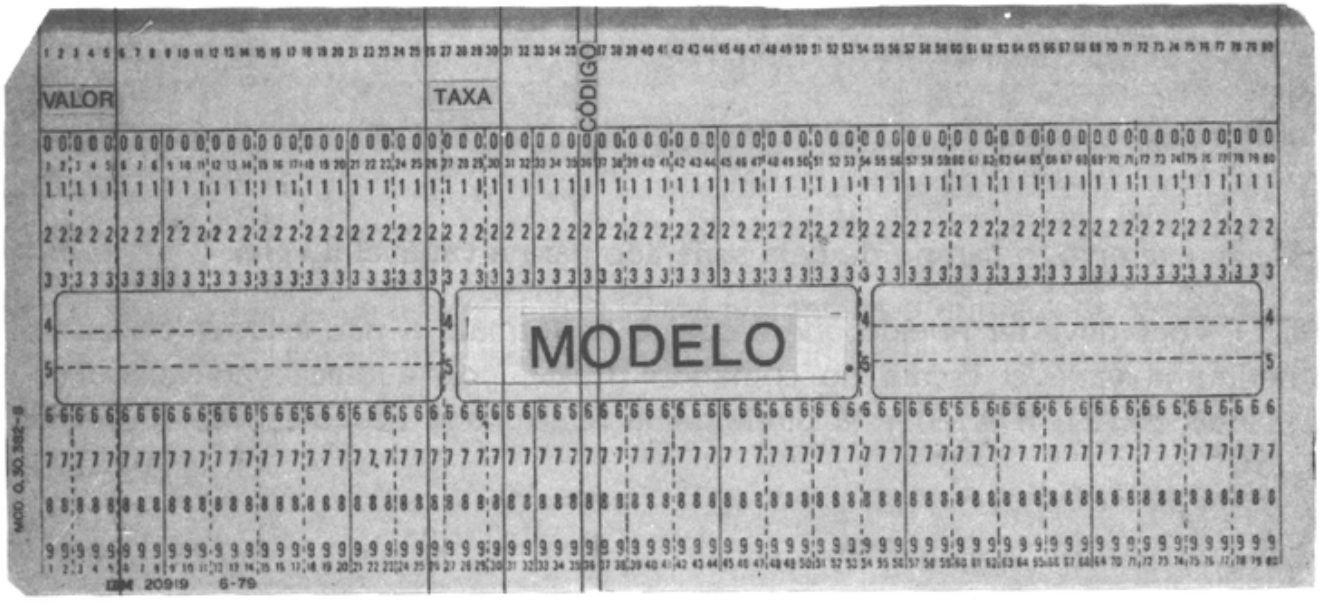
x — é uma constante inteira sem sinal, a qual indica o tamanho do campo que será saltado quando o registro for de entrada; se o registro for de saída, a constante indica o tamanho do campo que deverá ser deixado em branco.

Exemplos:

a)

```
      READ (2,27) IVALOR,ITAXA,ICOD
   27 FORMAT (I5,20X,I5,5X,I1)
```

□□ Os comandos acima seriam usados para ler o cartão

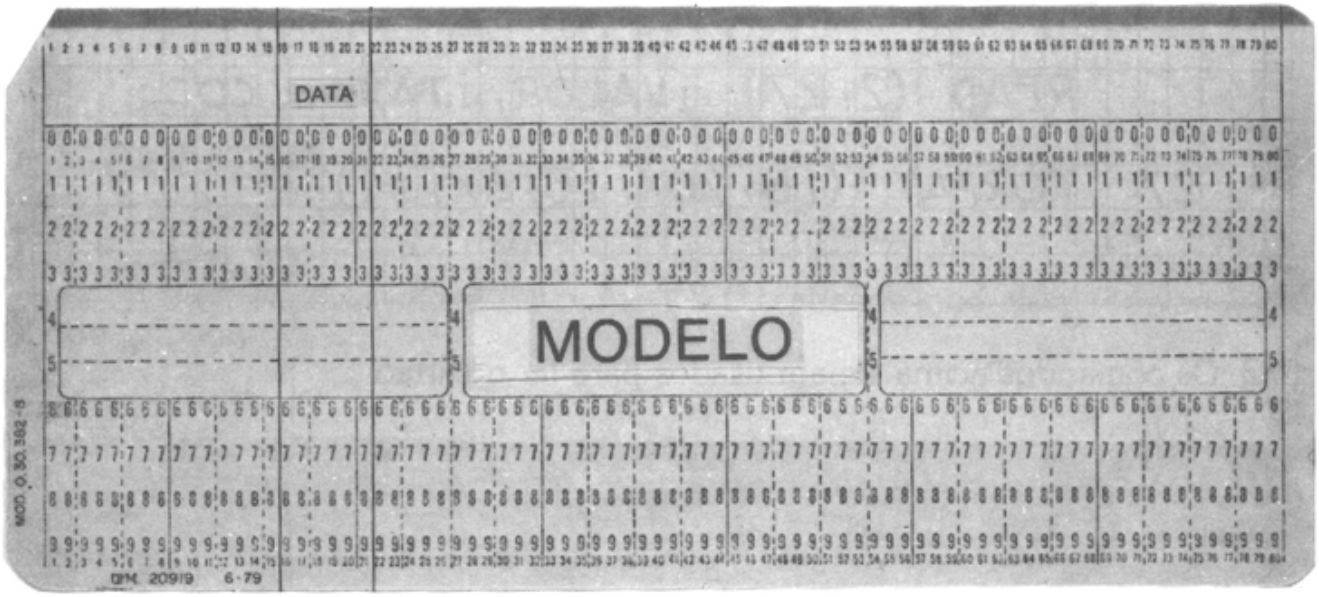


□□ O campo "valor" do cartão (colunas 1 a 5) seria lido para a variável IVALOR; as 20 colunas seguintes, isto é, as colunas 6 a 25 seriam ignoradas por causa da especificação 20X. O campo "TAXA" (colunas 26/30) seria lido para a variável ITAXA; as colunas 31 a 35 seriam ignoradas por causa da especificação 5X e, finalmente, a coluna 36 (campo de "código") seria lida para a variável ICOD.

b)

```
      READ (2,44) IDATA
44    FORMAT (15X,I6)
```

□□ Os comandos acima seriam usados para ler o cartão



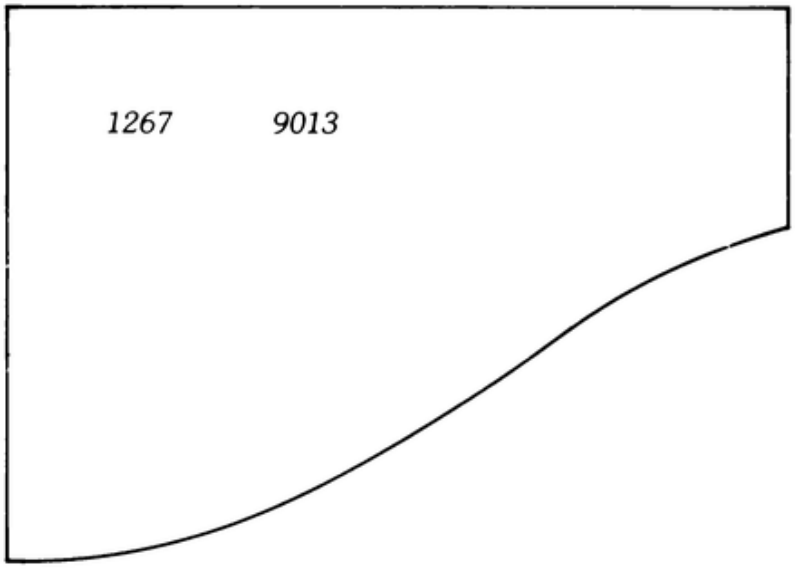
Neste exemplo, o campo de data seria lido para a variável IDATA.

□□ Você deve ter notado que o código de formato X não está relacionado a nenhuma variável escrita no comando READ, pois a única variável do comando READ (IDATA) está diretamente relacionada ao código de formato I (I6).

c)

```
      WRITE (5,16) IMPOST,JUROS
16    (I5,3X,I4)
```

□□ Supondo que o conteúdo da variável IMPOST fosse 1267 e o conteúdo da variável JUROS fosse 9013, a linha impressa seria.



Código de Formato H

□□ O código de formato H indica quais são os campos de entrada ou saída cujos dados alfanuméricos não deverão ser alterados durante a execução do programa.

□□ O seu formato geral é

wH

onde

w — é uma constante inteira sem sinal, a qual indica o número de caracteres que serão lidos, impressos ou perfurados, inclusive as colunas em branco.

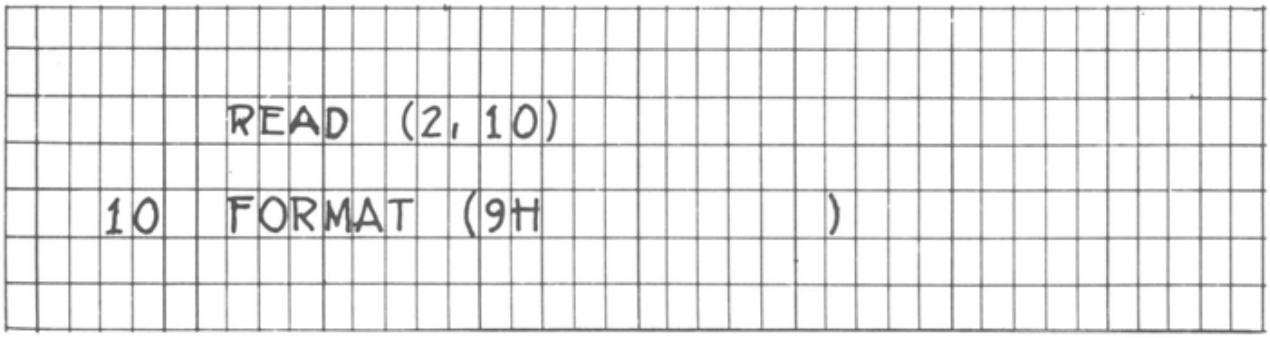
Exemplos:

a)

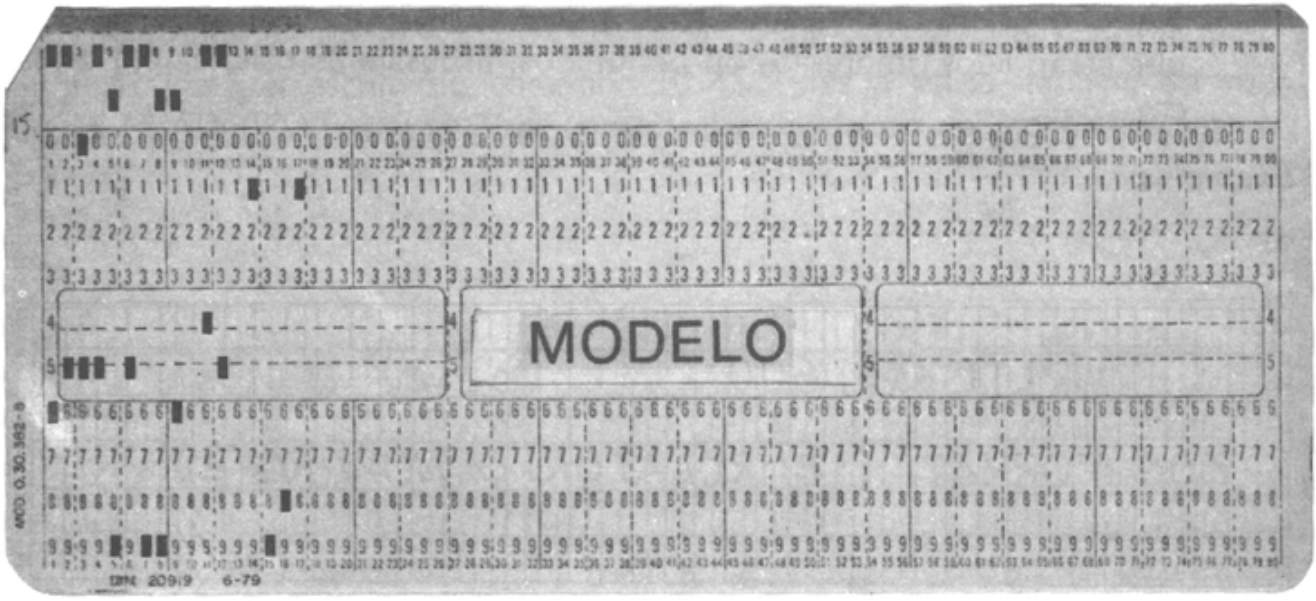
```
      WRITE (1,600)
  600 FORMAT (18HRELATORIO NUMERO 1)
```

□□ Os comandos anteriores provocariam a seguinte impressão na impressora do console:

b)



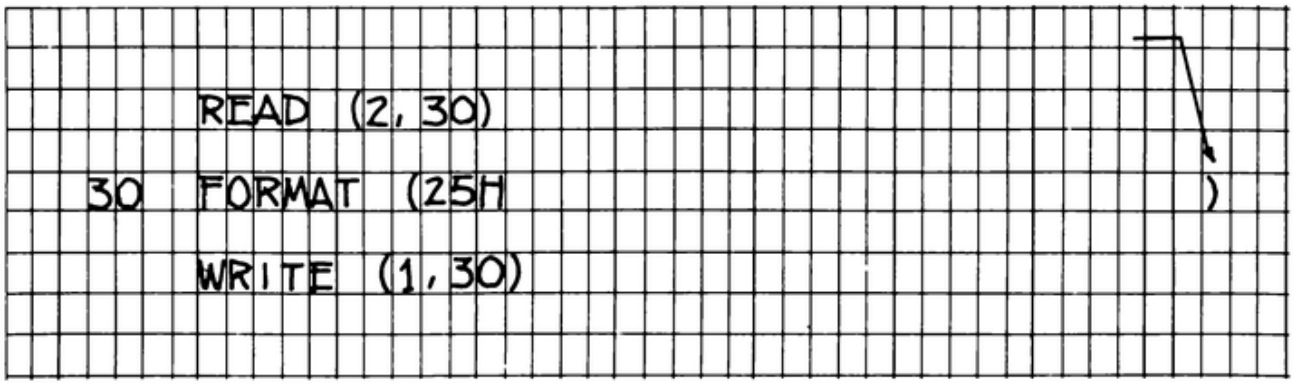
□□ Os comandos acima fariam com que um cartão fosse lido e o conteúdo das colunas 1 a 9 fosse copiado para as 9 posições em branco que aparecem à direita da especificação.
□□ Se o cartão abaixo fosse lido, apenas a palavra FEVEREIRO seria copiada.



□□ A mesma declaração FORMAT pode ser usada para comandos READ e WHITE, desde que as especificações sejam compatíveis com todas as variáveis (se houver) dos dois comandos.

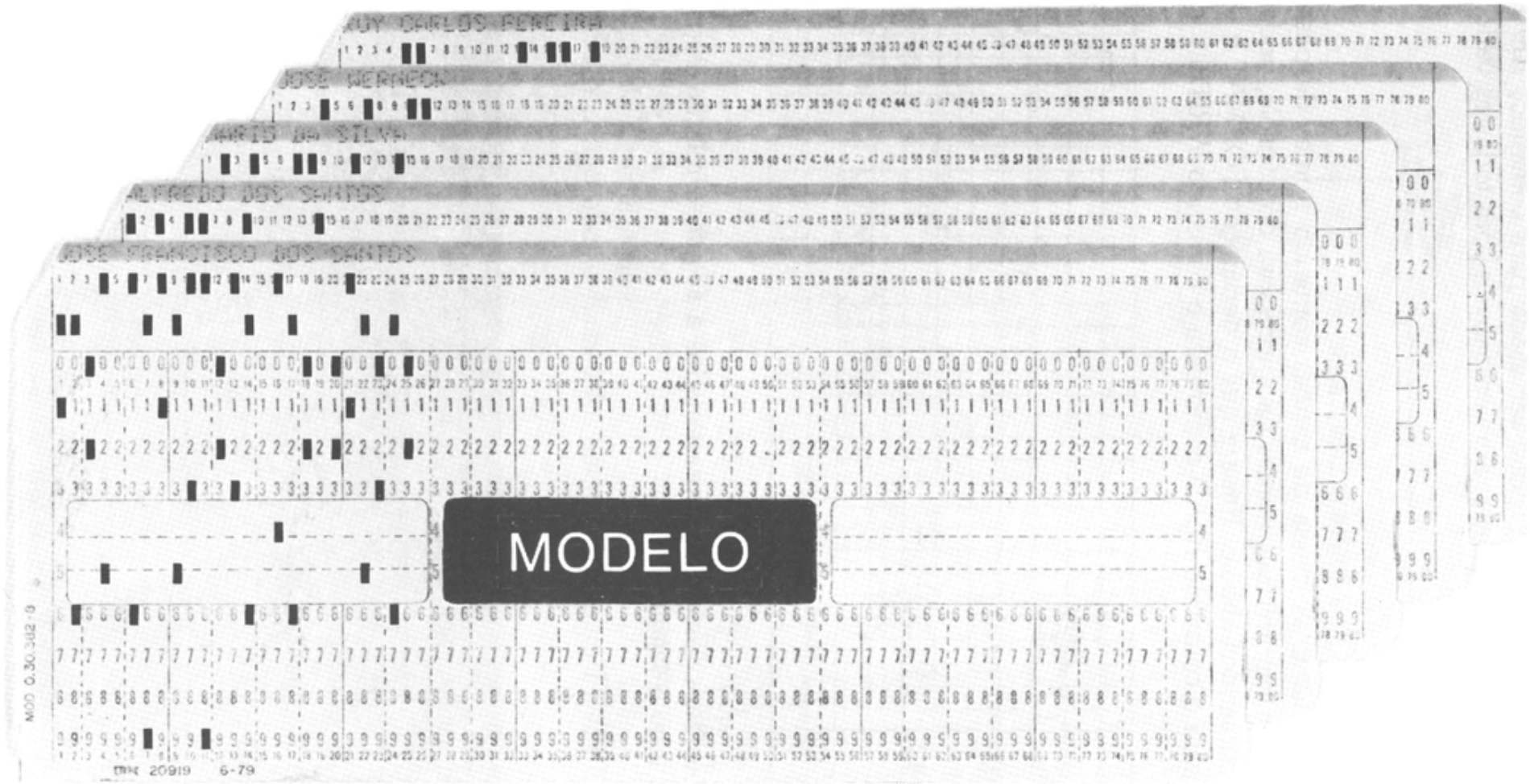
Exemplo:

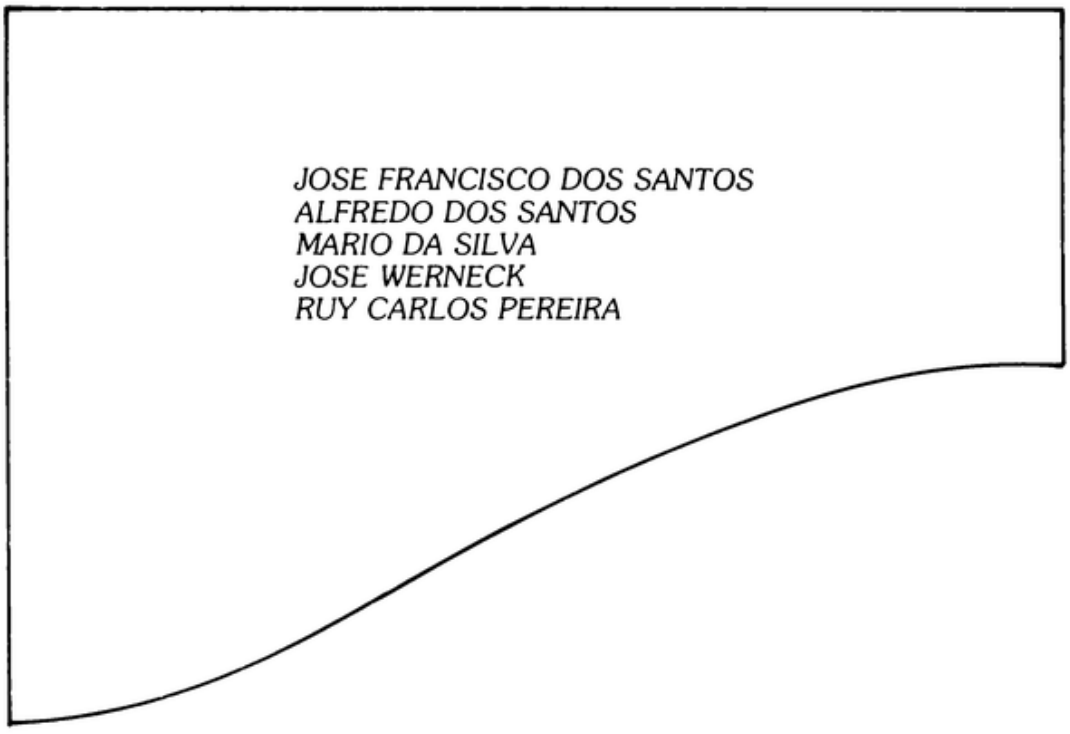
□□ Se quiséssemos ler uma massa de cartões contendo nomes de pessoas nas colunas 1 a 25 e imprimir tais nomes na impressora do console, poderíamos escrever os comandos abaixo. Repare que a declaração FORMAT é uma só e serve tanto para o comando READ como para o comando WRITE.



Resultado:

□□ Se os cartões lidos fossem os representados a seguir, a relação impressa seria a mostrada logo a seguir.


MODELO



□□ Note que a especificação do formato H não está diretamente associada a nenhuma variável escrita no comando READ ou WRITE.

□□ Alguns computadores como os IBM-360 e IBM-/370, aceitam um par de apóstrofos em lugar da especificação de formato H.

Exemplos:

a) A declaração

```
5  FORMAT  (10X, 17HFEVEREIRO DE 1981)
```

poderia ser substituída por

```
5  FORMAT  (10X, 'FEVEREIRO DE 1981')
```

b) Para imprimir o aviso "DIGITE O VALOR" na impressora do console, os comandos poderiam ser:

```
      WRITE (1, 11)
11    FORMAT ('DIGITE O VALOR')
```

Código de Formato A

□□ O código de formato A é usado, também, para trabalhar com dados alfanuméricos. A diferença básica entre o código de formato A e o código de formato H é a possibilidade de, na especificação tipo A, fazer alterações nos dados alfanuméricos.

□□ Nós vimos que na especificação tipo H não há nenhuma associação entre a especificação e a lista de variáveis do comando READ ou WRITE. Na especificação tipo A, no entanto, os dados alfabéticos estão contidos em uma variável, a qual tem de ser mencionada no comando READ ou WRITE..

□□ O formato geral do código de formato A é

Aw

onde

w — é uma constante inteira sem sinal que indica a quantidade de caracteres alfanuméricos (tamanho do campo) que serão lidos, impressos ou perfurados.

Exemplo:

```
      READ (2, 20) MES
20    FORMAT (A3)
```

□□ O mês, perfurado no cartão, será lido para a variável MES.

□□ O número *w* possui um limite, o qual varia com o tipo de computador e com o tipo de variável (inteira ou real).
□□ O IBM-/360 e o IBM-/370 aceitam até quatro caracteres alfanuméricos para serem armazenados em uma variável (real ou inteira). O IBM-1130 aceita até 2 caracteres para serem armazenados numa variável inteira e até 4 caracteres para uma variável real.
□□ O quadro abaixo facilita a visualização:

Computador	**Variável inteira**	**Variável real**
IBM /360	até 4 caracteres	até 4 caracteres
IBM-/370	até 4 caracteres	até 4 caracteres
IBM-1130	até 2 caracteres	até 4 caracteres

□□ Se, na entrada de dados, referenciados por variáveis reais, o número *w* for maior do que 4 (limite máximo permitido), somente serão lidos os quatro caracteres mais à direita.
□□ Na saída da dados, se *w* for maior do que o limite permitido, o campo excedente será preenchido com "brancos" à esquerda.
□□ Se *w* for menor do que o limite permitido, serão tomadas em consideração os caracteres mais à esquerda.
□□ As observações acima também são válidas para as variáveis inteiras, obedecendo os limites indicados no quadro acima.

Código de Formato F

□□ O código de formato F indica que a variável de entrada ou saída é do tipo real.
□□ A sua forma geral é

Fw.d

onde

w — é uma constante inteira sem sinal que especifique o tamanho total do campo utilizado.

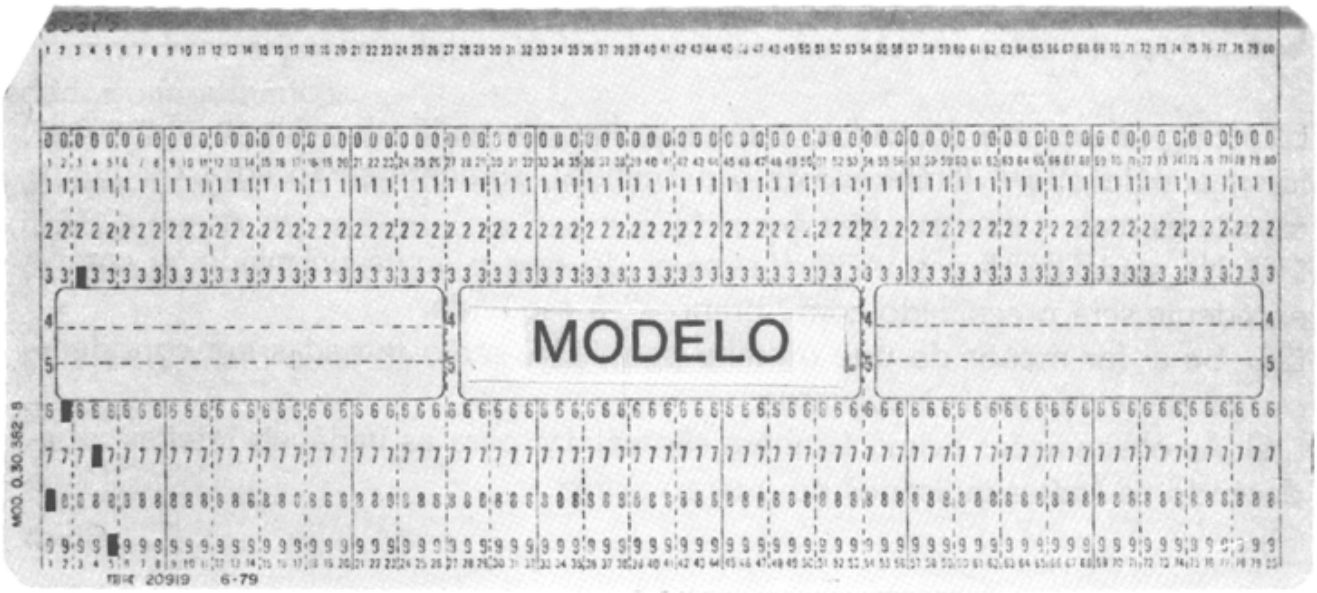
d — é uma constante inteira sem sinal que indica o número de decimais à direita do ponto.

Exemplos:

a)

```
      READ (2, 7) DADOS
7     FORMAT (F5.3)
```

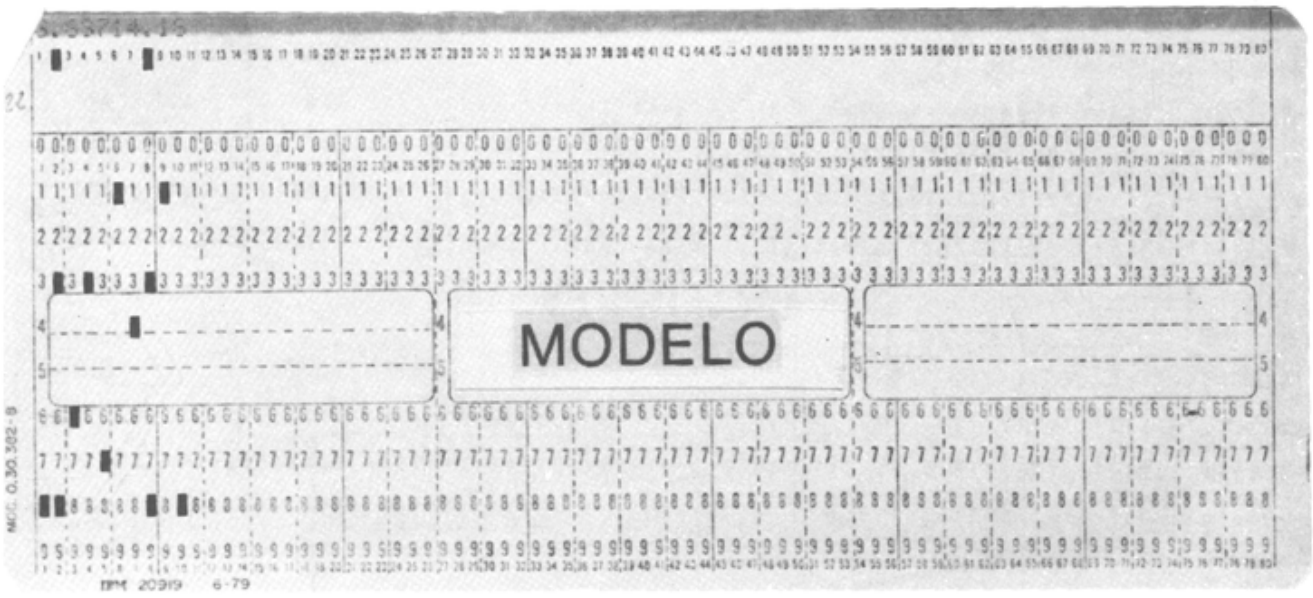
□□ Quando o cartão abaixo fosse lido, o campo DADOS conteria 86.379.



b)

```
      READ (2, 7) DADO1, DADO2
7     FORMAT (F5.3, F5.3)
```

□□ Vamos supor que o cartão abaixo será lido



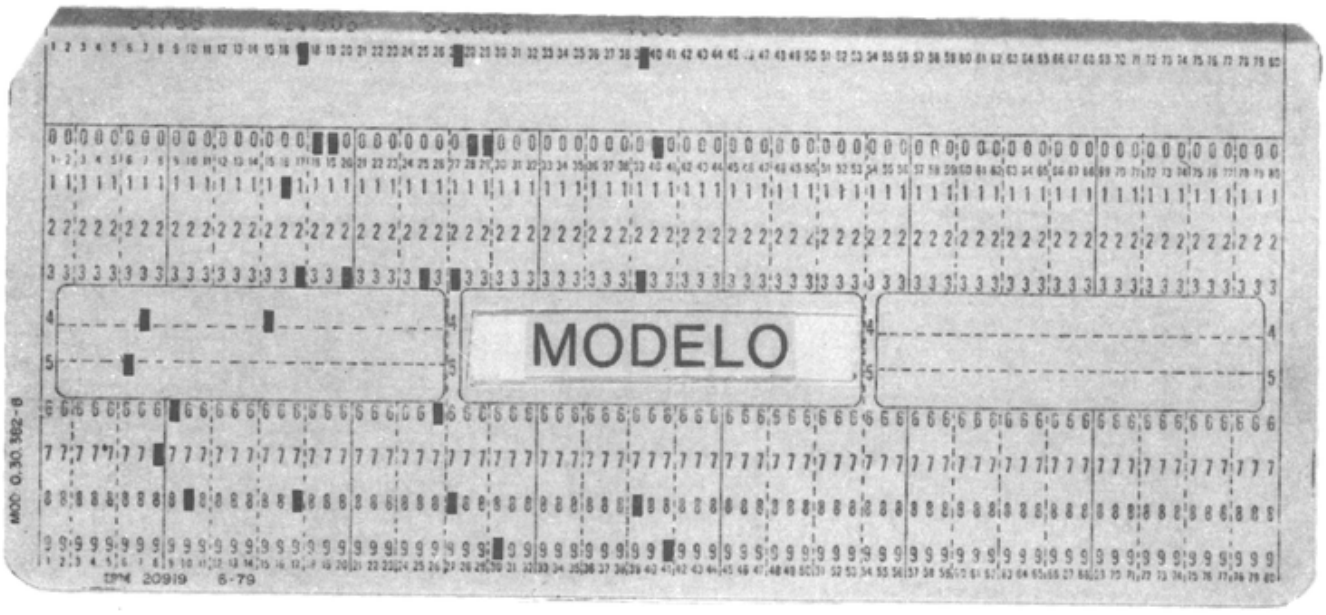
□□ Nas cinco primeiras colunas, as quais serão lidas para a variável DADO1, está perfurado o número 8.637. Neste caso, o número de decimais à direita do ponto coincidiu com a especificação F e o dado será lido sem nenhuma alteração. Após a leitura, DADO1 conteria, portanto, 8.637.

□□ Nas colunas 6 a 10, as quais serão lidas para a variável DADO2, está perfurado 14.18. O número de decimais (2) não coincide com o número de decimais especificado no código de formato F. Neste caso prevalece a posição do ponto no cartão, pois a posição do ponto PERFURADO NO CARTÃO TEM PRECEDÊNCIA sobre a posição especificada no código de formato F.

c)

```
      READ (2,7) CAMPOA,CAMPOB,CAMPOC
7     FORMAT (F10.1,F10.2,F10.3)
```

□□ Se o cartão a ser lido fosse o seguinte



após a leitura o conteúdo dos campos de leitura seria:

□□ CAMPOA — 5476.8 (A especificação do formato seria aceita, pois o número perfurado é um inteiro).

□□ CAMPOB — 41.003 (O número de decimais do cartão tem precedência sobre a posição do ponto especificada no código de formato F).

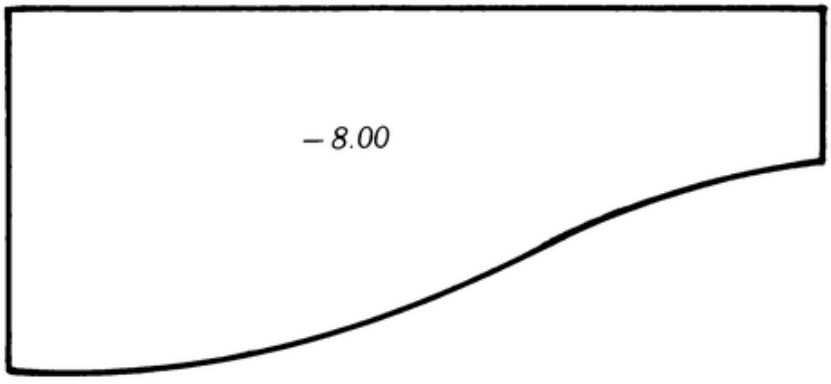
□□ CAMPOC — 36.009 — (Houve coincidência entre a posição do ponto perfurado no cartão e a posição especificada no código de formato F).

□□ O valor 4.09 seria ignorado porque não há variável para aceitá-lo no comando READ.

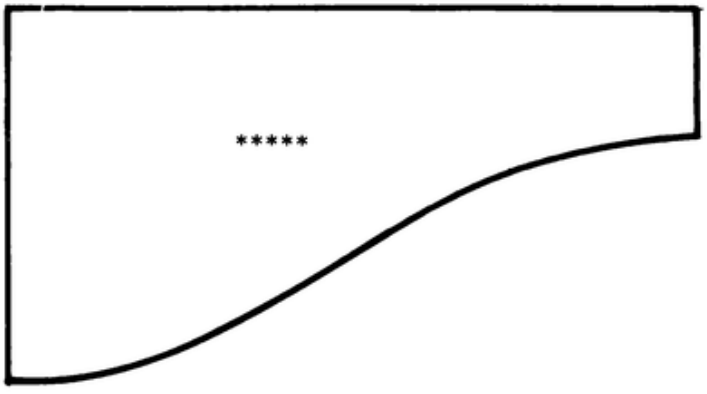
d)

```
      WRITE (1,8) A
   8  FORMAT (F5.2)
```

□□ Supondo que o valor da variável A fosse (– 8), sairia impresso no console:

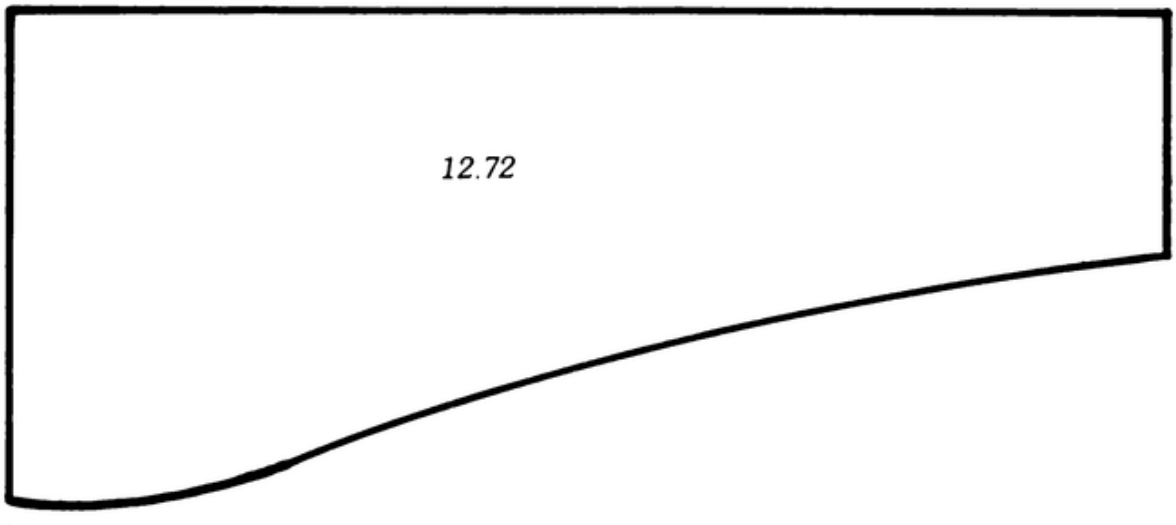


□□ Se o valor da variável A fosse 1234.5, sairia impresso no console:



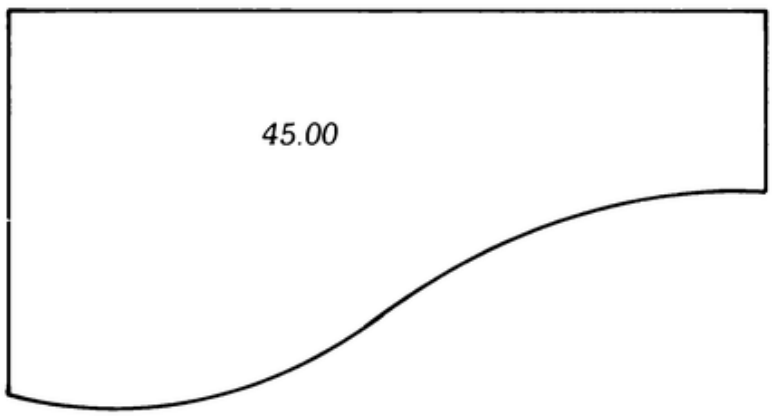
□□ O motivo da impressão dos arteriscos é o mau dimensionamento da especificação F. O programador precisa especificar um campo que caiba, além da parte inteira e das posições decimais, o próprio ponto decimal e o sinal.

□□ Se o valor da variável A fosse 12.72384 seria impresso no console:



□□ O truncamento dos decimais é feito por causa da especificação F5.2, a qual indica que o campo de impressão deve ter apenas dois decimais.

□□ Se o valor da variável A fosse 45. seria impresso no console:



□□ Quando no campo de origem houver menos casas decimais do que na especificação F, o campo impresso será preenchido com "zeros".

Código de Formato E

□□ O código de formato E indica que a variável de entrada ou saída é do tipo real e está na forma exponencial.

□□ A sua forma geral é

Ew.d

onde

w — é uma constante inteira sem sinal que define o tamanho total do campo. O campo deve ter o comprimento necessário para conter:

- a parte inteira do número (se houver);
- o zero antes do ponto (se houver);
- o sinal (se houver);
- as posições para a parte decimal;
- o ponto decimal;
- 4 posições para o expoente.

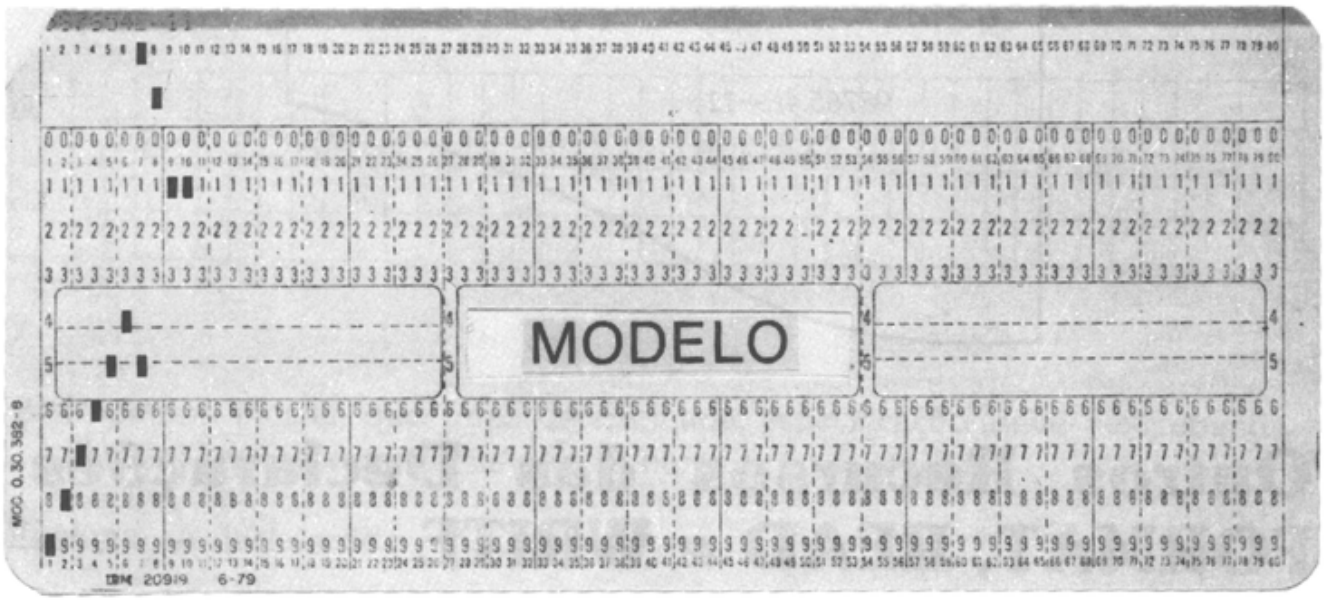
d — é uma constante inteira sem sinal que indica quantas posições decimais existem depois do ponto.

Exemplos:

a)

```
      READ (2, 15) VALOR
15    FORMAT (E10.5)
```

□□ Se o cartão lido fosse o abaixo, a variável VALOR ficaria com o valor 9.87654×10^{-11}



b)

```
      READ (2, 15) VALOR
15    FORMAT (E10.1)
```

□□ Se o cartão lido fosse o apresentado no exemplo (a), a variável VALOR ficaria com o valor 98765.4×10^{-11}

c) Se após a leitura do exemplo (b) o programa tivesse o comando

```
WRITE (1, 15)
```

□□ A especificação seria a mesma, pois o FORMAT com o número 15 tem de ser único no programa. Neste caso, já vimos no exemplo (b) que a variável VALOR conteria 98765.4 x 10^{-11}. A linha que seria impressa no console quando o WRITE fosse executado seria a seguinte:

```
98765.4E-11
```

Outros Recursos das Declarações FORMAT, READ e WRITE

1. Quando os grupos de especificações são repetitivos, isto é, quando há repetição de especificações iguais, podemos simplificar a escrita da declaração FORMAT. Para indicar a repetição de um grupo de especificações, coloca-se o grupo a ser repetido entre parênteses, precedido do número de repetições.

Exemplos:

a) a declaração

```
15  FORMAT  (1X, 2(F5.2, F5.1))
```

equivale a

```
15   FORMAT (1X, F5.2, F5.1, F5.2, F5.1)
```

b) a declaração

```
8   FORMAT (2 (I4, 3X), I2)
```

equivale a

```
8   FORMAT (I4, 3X, I4, 3X, I2)
```

2. Podemos indicar que *uma* determinada especificação deve ser repetida. Para tal basta colocar antes dela, o número de repetições.

Exemplo: A declaração

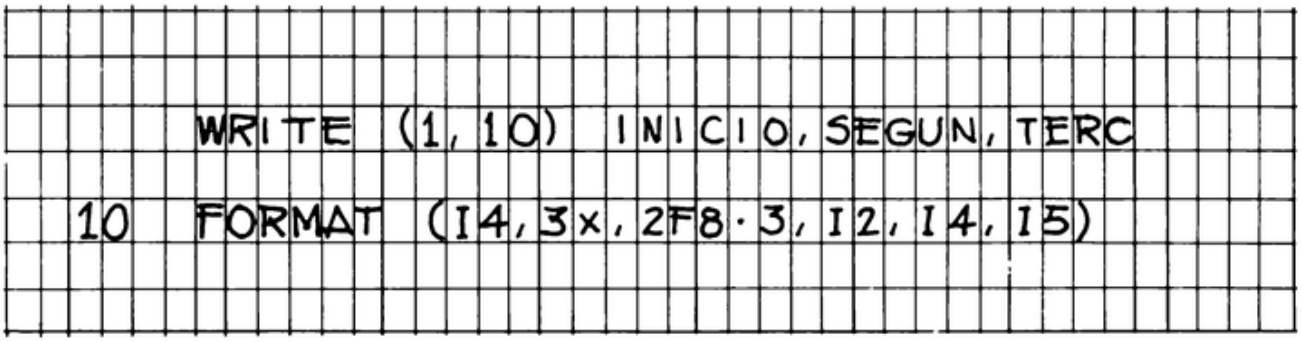
```
5   FORMAT (2F3.2, 3I4)
```

equivale a

```
5   FORMAT (F3.2, F3.2, I4, I4, I4)
```

3. Quando há mais especificações no FORMAT do que variáveis no comando READ ou WRITE, serão utilizados apenas as especificações para as quais existem variáveis.

Exemplos:



□□ Neste exemplo, INICIO é impresso com a especificação I4. A especificação 3X não está associada, como já vimos, a nenhuma variável. Por isso SEGUN é impresso com a especificação F8.3. TERC também é impresso com a especificação F8.3 (porque a especificação F8.3 deve ser tomada duas vezes).

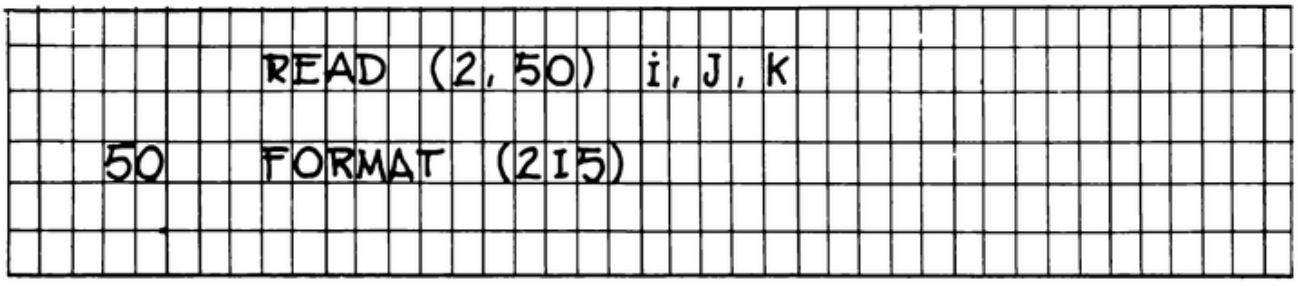
□□ As especificações I2, I4 e I5 são ignoradas, pois não há variáveis associadas a elas.

4. Quando há mais variáveis no comando READ ou WRITE do que especificações na declaração FORMAT, após serem utilizadas todas as especificações fornecidas ocorrerá o seguinte:

   - na saída haverá uma mudança de linha (se for impressão), de registro (se for gravação) ou de cartão (se for perfuração). Serão repetidas as especificações a partir da mais próxima do parêntese mais à esquerda.

Exemplos:

a) Os comandos



equivalem a

```
      READ (2, 50) I, J
      READ (2, 50) K
50    FORMAT (2I5)
```

ou

```
      READ (2, 50) I, J
50    FORMAT (2I5)
      READ (2, 51) K
51    FORMAT (I5)
```

b)

```
      READ (2, 22) A, B, C, D, E, F, G, H
22    FORMAT (2(F2.2, F3.3, F4.4), 2F5.5)
```

A declaração acima equivale a

```
      READ (2, 22) A, B, C
      READ (2, 22) D, E, F
22    FORMAT (F2.2, F3.3, F4.4)
      READ (2, 23) G, H
23    FORMAT (2F5.5)
```

5. Já vimos que o caráter / indica que as especificações que o antecedem estão em um registro e as que o sucedem estão em outro registro.

Exemplo:

□□ Os comandos abaixo servem para ler 2 cartões. O primeiro cartão será lido com as especificações I2, I3 e I3. O segundo cartão será lido com as especificações F4.2 e F4.2.

```
      READ (2, 3) I, J, K, Y, Z
3     FORMAT (I2, 2I3 / 2F4.2)
```

□□ Quando houver um grupo de *n* caracteres / adjacentes, o computador considerará que é para proceder com n-1 registros, como se segue:

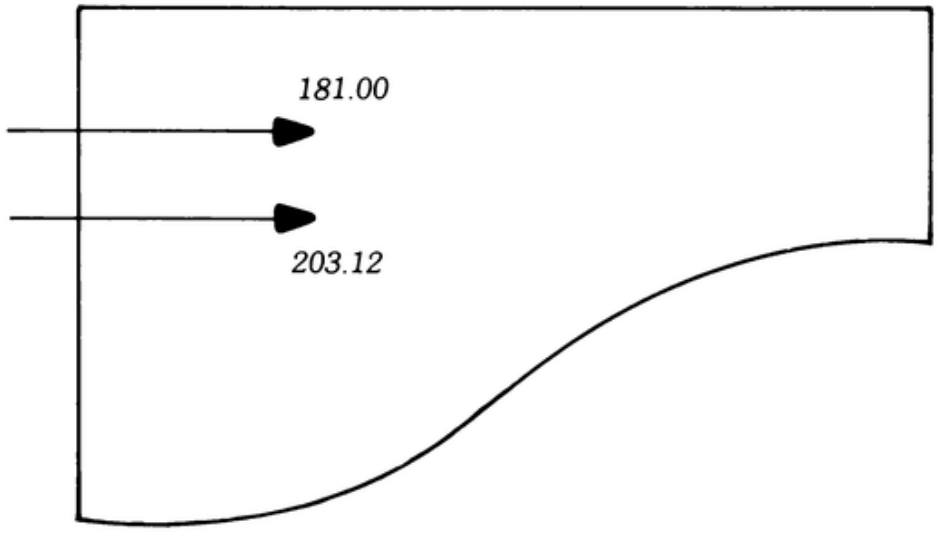
□□ — ignorá-los se o comando for READ;

□□ — se o comando for WRITE, deverá imprimir n-1 linhas em branco ou gerar n-1 registros em branco.

Exemplo:

```
      WRITE (1, 18) A, B
18    FORMAT (1X, F5.2, ///, F5.2)
```

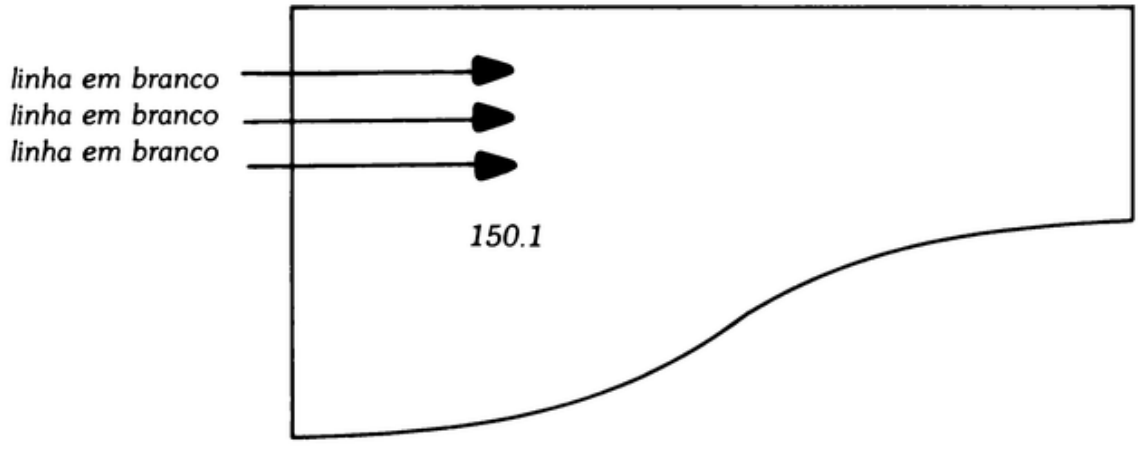
□□ Se as variáveis A e B contivessem, respectivamente, 181.00 e 203.12, as linhas impressas no console seriam:



□□ Se o grupo de caracteres / estiver no início ou no fim da declaração FORMAT, o computador ignorará tantos registros quantos forem os caracteres / do grupo.

```
      WRITE (1,16) A
16    FORMAT (///,F3.1)
```

□□ Se a variável A contivesse 150.1 o resultado da impressão no console seria



6. Controle de espacejamento nas impressoras

□□ Até agora, nos nossos exemplos a respeito de linhas a serem impressas, temos usado o comando para imprimir na impressora do console. A escolha de tal impressora foi proposital, pois nos demais tipos de impressora (IBM-1132, IBM-1403 e IBM-3211, por exemplo), a primeira posição do registro de saída não é impressa, tendo uma função especial. A primeira posição do registro de saída, quando for comandada uma impressão, serve para indicar qual o espacejamento vertical desejado. Entende-se por espacejamento vertical o movimento vertical do formulário contínuo na impressora.

O programador pode escolher:

- Imprimir o registro na linha imediatamente seguinte à última linha já impressa. Este tipo de espacejamento é conhecido como espacejamento simples.
- Imprimir o registro na 2.ª linha após a última linha impressa, ficando, pois, uma linha em branco entre o registro a imprimir e a última linha já impressa. Este tipo de espacejamento é conhecido como espacejamento duplo.
- Imprimir o registro na 1.ª linha da folha seguinte. Este tipo de espacejamento é chamado de "salto para canal 1".
- Imprimir o registro sobre a última linha impressa. Este tipo de impressão é muito usado quando desejamos reforçar a linha impressa, fazendo com que a linha, por ficar mais escura, sobressaia das demais. Neste caso, diz-se que o registro é impresso sem espacejamento.

□□ Para comandar o espacejamento desejado basta colocar o caráter apropriado, escolhido na tabela abaixo, na 1.ª posição do registro que se deseja imprimir.

Caráter	Tipo de Espacejamento
b	espacejamento simples
0 (zero)	espacejamento duplo
1	salto para canal 1
+	sem espacejamento

Exemplos:

a)

```
      WRITE (5, 20) I
20    FORMAT (1H1, I4)
```

□□ Neste exemplo, o valor de I será impresso na 1.ª linha da folha seguinte.

b)

```
      WRITE (5, 20) I
20    FORMAT (1X, I4)
```

□□ Neste exemplo, o valor de I será impresso na linha seguinte à última linha já impressa.

c)

```
      WRITE (5, 20) I
20    FORMAT ('0', I4 / 1H+, I4)
```

□□ Neste exemplo, a impressora deixaria uma linha em branco após a última linha já impressa; em seguida imprimiria o valor de I, por causa do formato '0' que aparece no início da especificação FORMAT. O formato 1H+ faria com que o valor de I fosse impresso em cima do valor de I já impresso anteriormente.

10

Desvios

□□ Os comandos de um programa são executados, normalmente, na ordem em que eles são escritos. O programador, pode, no entanto, modificar esta seqüência. Tal quebra de seqüência é conhecida como "desvio".
□□ Se o programa deve modificar a sua seqüência normal independentemente de quaisquer condições, o desvio é chamado de "desvio incondicional". Se a quebra de seqüência ficar subordinada a uma ou a várias condições, o desvio é chamado de "desvio condicional".

Desvio Incondicional

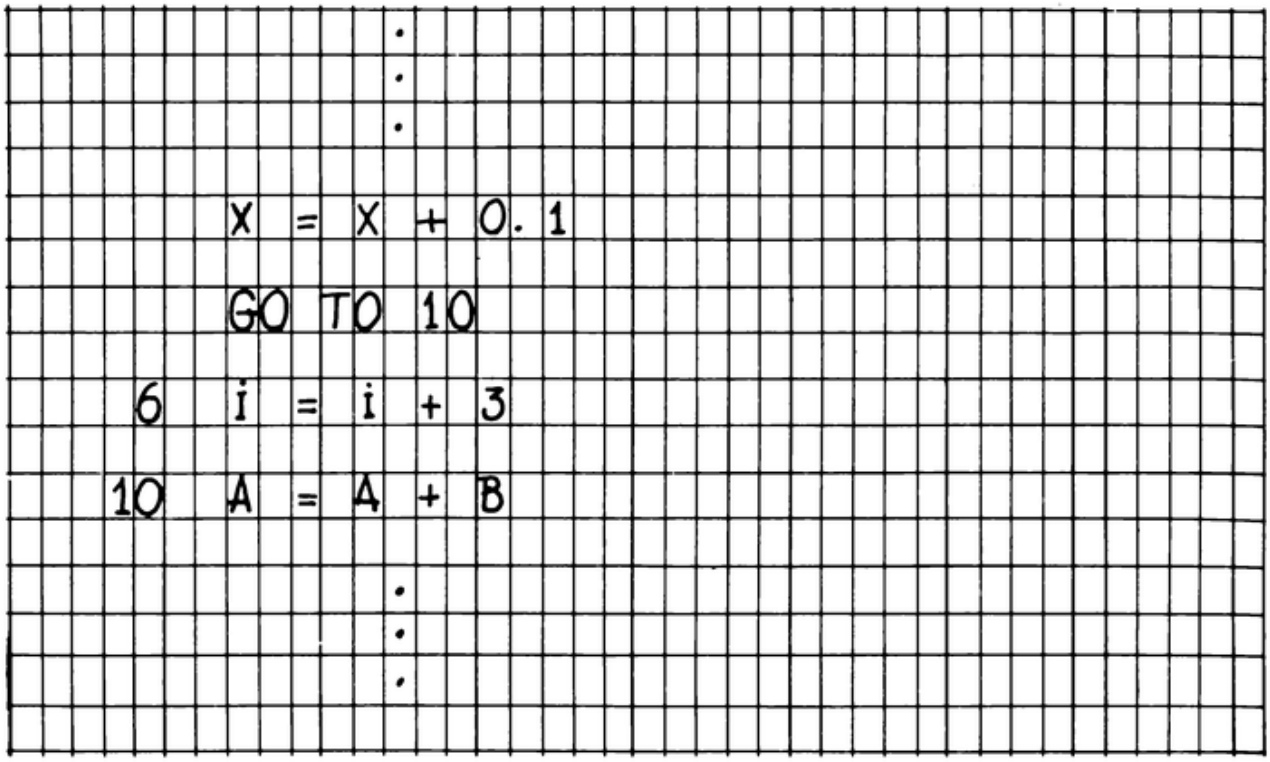
□□ O desvio incondicional é conseguido como o comando GO TO, cuja forma é

GO TO n

onde

n — é o número de um comando FORTRAN do programa.

Exemplo:



□□ Neste exemplo, após a execução do comando X = X + 0.1, o comando seguinte será o comando GO TO 10. Este passará o controle para o comando 10 e não para o comando 6, isto é, depois da execução do comando GO TO o comando seguinte será A = A + B.

Desvio Condicional

□□ O desvio condicional pode ser obtido com dois tipos de comandos.

- o comando IF;
- o comando GO TO indexado.

Comando IF

Há dois tipos de comandos IF:

- o comando IF aritmético;
- comando IF lógico.

Comando IF Aritmético

□□ No comando IF aritmético o programa desvia para um determinado comando, dependendo do resultado do cálculo de uma expressão aritmética. A sua forma geral é

IF (E) n_1, n_2, n_3

onde

e — é uma expressão aritmética.

n_1, n_2, n_3 — são números de comandos FORTRAN do programa. Se o resultado da expressão for negativo, o programa desvia para n_1. Se for zero, desvia para n_2. Se for positivo, desvia para n_3.
□□ Para fins didáticos, apresentamos o esquema abaixo:

IF (E) n_1, n_2, n_3

↓ ↓ ↓

− 0 +

Exemplo: IF (A + B) 10, 20, 30

□□ Se o resultado de A + B for negativo, o controle do programa passará para o comando 10. Se for zero, passará para o comando 20. Se for positivo, o controle do programa passará para o comando 30.
□□ O Fortran aceita expressão de uma só variável. Neste caso, o desvio dependerá exclusivamente do valor da variável.

Exemplo:

IF (J) 5, 10, 15

□□ Se o valor de J for negativo, o controle do programa passará para o comando 5. Se for zero, passará para o comando 10. Se for positivo, passará para o comando 15.

Comando IF Lógico

□□ Nem todos os computadores aceitam o comando IF lógico. Os IBM-/360 e IBM-/370, por exemplo, aceitam. O IBM-1130, não.
□□ A forma geral do comando IF lógico é

IF (L) C

onde

L — é uma expressão lógica

C — é um *comando,* o qual não pode ser outro IF lógico nem um comando DO (estudado no Capítulo 11).

□□ Entende-se por expressão lógica as expressões dos tipos

x.GT.y
x.GE.y
x.LT.y
x.LE.y
x.EQ.y
x.NE.y

cujos significados são:

x.GT.y — Se a expressão *x* for maior do que (*G*reater *t*han) a expressão *y*.

x.GE.y — Se a expressão *x* for maior que ou igual a (*G*reater than or *e*qual to) a expressão *y*.

x.LT.y — Se a expressão *x* for menor (*l*ess *t*han) do que a expressão *y*.

x.LE.y — Se a expressão *x* for menor que ou igual a (*l*ess than or *e*qual to) expressão *y*.

x.EQ.y — Se a expressão *x* for igual (*e*qual to) a expressão *y*.

x.NE.y — Se a expressão *x* for diferente (*n*ot *e*qual to) da expressão *y*.

□□ Se ocorrer a condição imposta na expressão lógica, o controle do programa passará para o comando indicado no parâmetro C.

Exemplos:

a) IF (A.NE.5.0) A = B + 1.0

b) IF (K.EQ.L) I = I + 1

c) IF (3.5 * A.GT.35.0) GO TO 18

Comando GO TO Indexado

□□ A sua forma geral é

GO TO (n_1, n_2, n_3, ... n_m), i

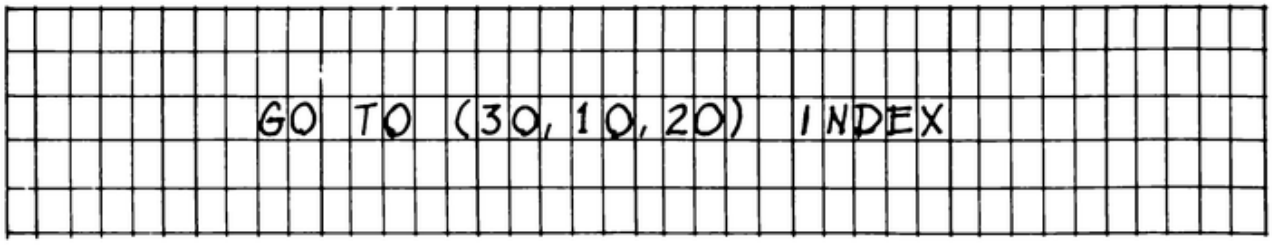
onde

n_1, n_2, n_3, ... n_m — são números de comandos.

i — é uma variável inteira cujos valores tem de ser, obrigatoriamente, um dos valores 1, 2, 3 ... m.

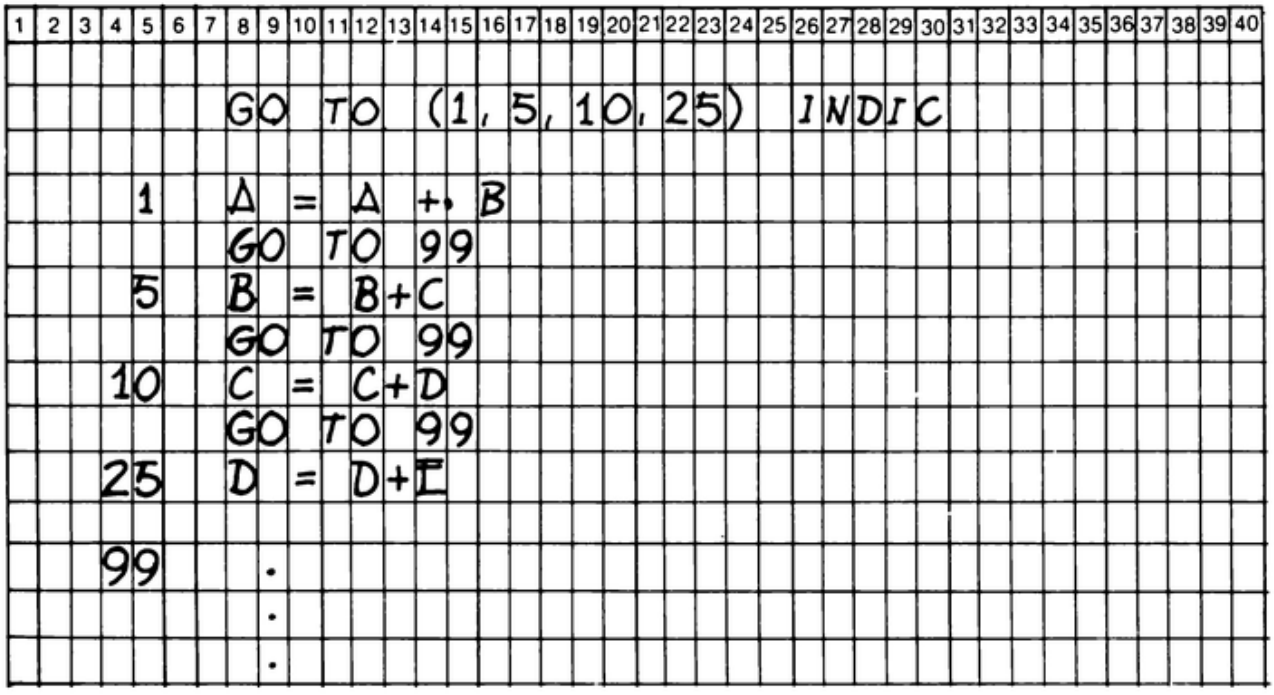
□□ Se o valor de *i* for 1, ocorrerá o desvio para o comando n_1; se for 2, o desvio será para n_2; se o valor de *i* for 3, o controle do programa passará para n_3; se for *m*, o desvio será para n_m.

Exemplos:



a) Se o valor de INDEX for 1, o desvio será para o comando de número 30; se for 2, o desvio será para o comando de número 10; se for 3, o desvio será para o comando de número 20.

b) Vamos supor que num programa há quatro cálculos distintos a serem efetuados. No comando 1, o programa calcula A = A + B; no comando 5 o programa calcula B = B + C; no comando 10 o programa calcula C = C + D e no comando 25 o programa calcula D = D + E. O cálculo a ser efetuado dependerá do valor contido na variável INDIC. O trecho do programa para fazer a opção entre os quatro cálculos a serem efetuados poderia ser o seguinte:



□□ O desvio ocorrerá como explicado a seguir.

□□ Se INDIC for igual a 1 o controle do programa passará para o comando 1.

□□ Se INDIC for igual a 2 o controle do programa passará para o comando 5.
□□ Se INDIC for igual a 3 o controle do programa passará para o comando 10.
□□ Se INDIC for igual a 4 o controle do programa passará para 25.

Comando STOP

□□ O comando STOP termina a execução do programa. As suas formas gerais são

STOP ou STOP n

onde

n — é uma constante inteira sem sinal, cujo valor máximo é 9999.

□□ Quando o programa executa um comando STOP n, a execução termina e o operador da máquina pode ver qual o número *n* que lhe é mostrado.
□□ Como o programa aceita vários comandos STOP em diversos pontos (embora só um seja executado), o número *n* serve para indicar ao operador em qual comando STOP o programa terminou.

Exemplo:

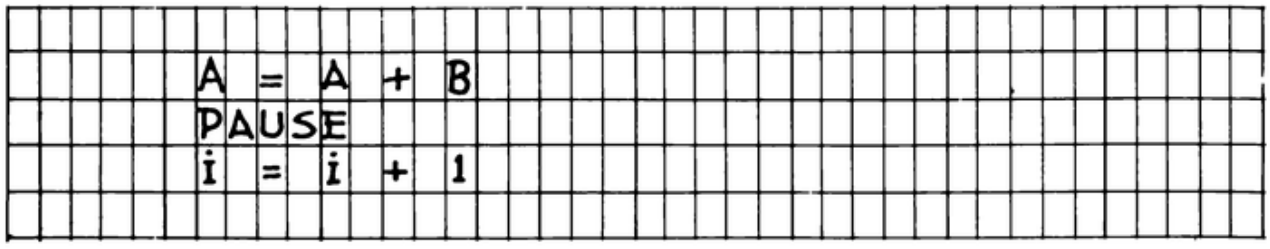
□□ No trecho do programa abaixo, vamos supor que o comando STOP 9999 é o fim normal do programa; e que o comando STOP 1000 indica um fim anormal.
□□ Assim, dependendo da lógica do programa, se o computador parar por causa do comando STOP 1000, o operador tomará conhecimento de que algo anormal ocorreu durante a execução do programa, podendo tomar as providências necessárias.

```
    IF (A) 20, 20, 30
20  STOP 9999
30  STOP 1000
```

Comando PAUSE

□□ O comando PAUSE é semelhante ao comando STOP. A diferença é que o programa não encerra a execução, mas apenas pára temporariamente. Quando o operador da máquina apertar a tecla 'START' do computador, o processamento do programa reinicia no comando seguinte ao PAUSE.

Exemplo:

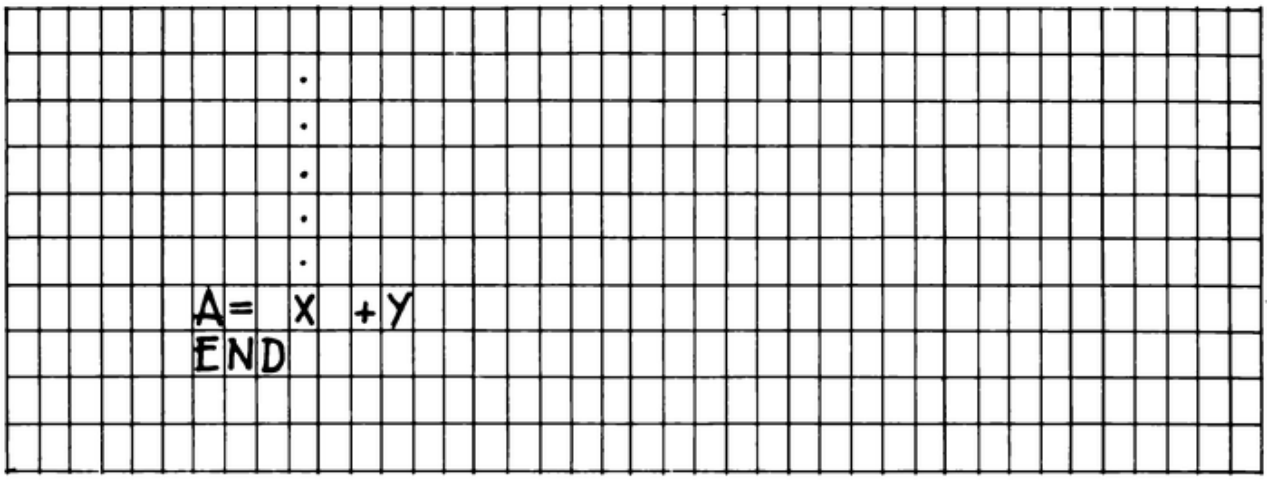


□□ Neste exemplo, o programa após executar A = A + B, o processamento se interromperia. Quando o operador apertasse a tecla START do computador o programa executaria a instrução I = I + 1 e as seguintes.

Comando END

□□ O comando END não gera nenhuma instrução no programa depois de compilado. Ele serve apenas para indicar ao compilador que o programa fonte acabou e, por isso, ele tem de ser o último comando de todo o programa fonte.

Exemplo:



11

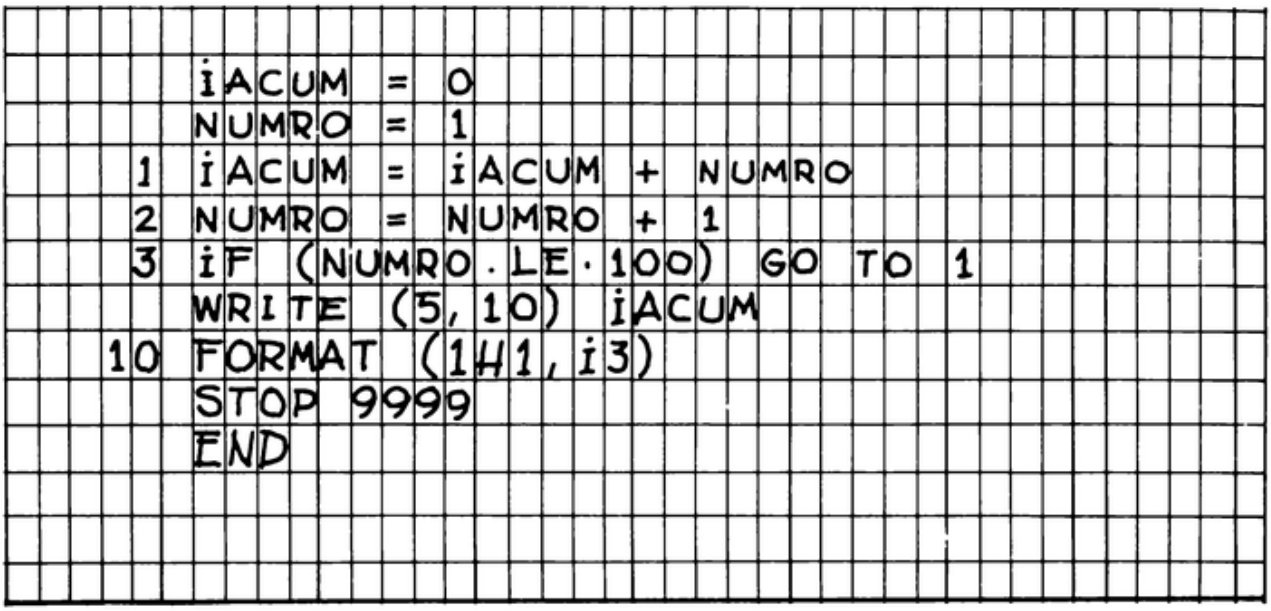
Comandos DO e CONTINUE

* * *

Comando DO

□□ O comando DO é usado para provocar a execução várias vezes de um trecho do programa.

□□ Vamos supor que quiséssemos calcular a soma dos números de 1 a 100. O programa que faria este cálculo poderia ser



□□ Neste exemplo, os comando 1, 2 e 3 são executados 100 vezes.

□□ É freqüente a necessidade de termos de escrever grupos de instruções como estas que têm de ser repetidas muitas vezes. O comando DO é uma forma bastante simples e poderosa de escrevê-las. Após as explicações a seguir, veremos como ficaria o programa do exemplo acima com o emprego do comando DO.

□□ A forma geral do comando DO é

$$DO\ n\ i = m_1, m_2, m_3$$

onde

n — é o número do comando do programa até onde (inclusive) atua o comando DO.

i — é uma variável inteira chamada de "índice do comando DO".
m_1 — é o valor inicial do índice;
m_2 — é o valor limite do índice;
m_3 — é o valor de incremento do índice.

□□ m_1, m_2, e m_3 podem ser constantes inteiras sem sinal e maiores do que zero ou variáveis inteiras que contenham valores sem sinal e maiores do que zero.

□□ Quando m_3 é omitido, o valor de incremento é considerado 1.

□□ O comando imediatamente após o comando DO e os seguintes até o comando *n* (inclusive) são chamados de "intervalo" ou "domínio" do comando DO.

Funcionamento do Comando DO

□□ O comando DO executa os comandos do seu domínio, fazendo primeiramente i = m_1. Após a primeira execução ele soma m_3 ao valor de *i* e executa novamente os comandos do seu domínio enquanto o valor de *i* não exceder o valor de m_2. Quando o valor de *i* exceder o valor de m_2, o controle do programa passará para o comando seguinte ao comando *n*.

Exemplo:

□□ Vamos escrever agora o programa para calcular a soma dos números de 1 a 100, usando o comando DO

```
      IACUM = 0
      DO 3 NUMRO=1,100,1
    3 IACUM = IACUM + NUMRO

    4 WRITE (5,10) IACUM
   10 FORMAT (1H1, I3)
      STOP 9999
      END
```

□□ O comando DO funcionaria da seguinte maneira:

a) colocaria 1 em NUMRO;
b) executaria IACUM = IACUM + NUMRO;
c) somaria 1 em NUMRO e executaria novamente o item *b* enquanto NUMRO não fosse maior do que 100;
d) quando NUMRO fosse maior do que 100 o controle do programa passaria para o comando 4.

Observações Sobre o Comando DO

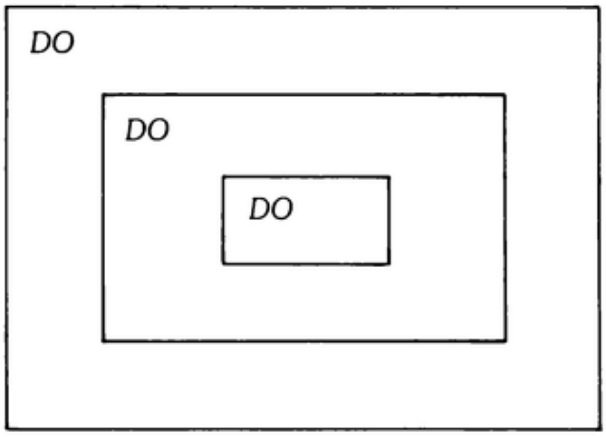
a) os valores de i, m_1, m_2 e m_3 não devem ser alterados por nenhum comando dentro do domínio do comando DO.
b) Não é permitido o desvio de fora do domínio de um comando DO para dentro dele. É permitido, no entanto, o desvio de dentro para fora do domínio de um comando DO, desde que não seja no último comando do domínio.
c) O último comando do domínio não pode ser: GO TO, IF ou STOP. Veja o item "comando CONTINUE" neste capítulo.
d) A primeira declaração do domínio do comando DO não pode ser uma especificação. Tem de ser um comando executável.
e) Quando o programa sai do domínio do comando DO por causa de um desvio, os valores que i, m_1, m_2 e m_3 continham no momento da transferência ficam disponíveis.
f) São permitidos um ou mais comandos DO dentro do domínio de outro comando DO. A este agrupamento de comandos DO chamamos de "ninho de DOs".

Exemplo:

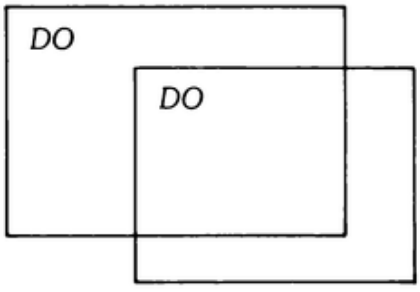
```
      .
      .
      DO 30 N=1,10,1
      DO 20 i=1,5

   20 iACUM = i+1

   30 J = 0
      .
      .
      .
```

□□ Neste exemplo, enquanto N for igual a 1, o valor de I variará de 1 até 5 e o comando 20 será executado 5 vezes. Quando o valor de I for maior do que 5, o valor de N será alterado para 2 e o comando 20 será executado mais 5 vezes. A rotina continuará desta maneira até que o valor de N seja 10, quando então o comando 20 será executado as últimas 5 vezes.

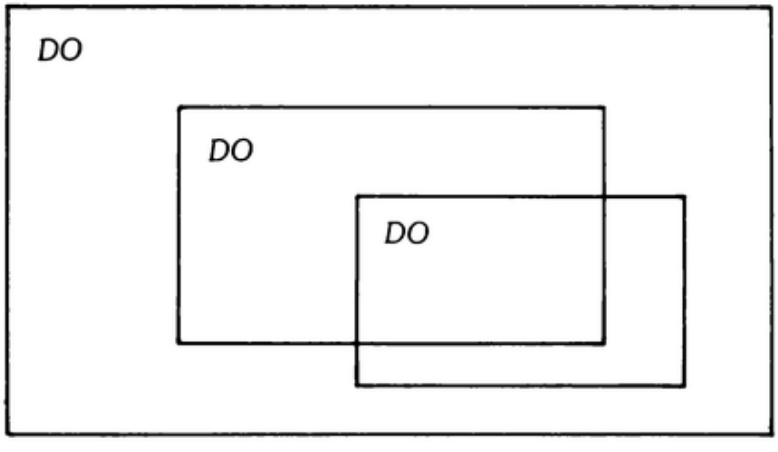
g) Num ninho de DOs (comando DO dentro do domínio de outro comando DO) o domínio do comando DO mais interno tem de estar totalmente dentro do domínio do comando DO mais externo.
Graficamente o esquema seria o seguinte:



Ninho de DOs válido.



Ninho de DOs inválido.



Ninho de DOs inválido.

Comando CONTINUE

□□ O comando CONTINUE não gera nenhuma instrução no programa compilado. A sua função é indicar que o programa deve seguir a seqüência normal onde ele é inserido. Ele pode ser inserido em qualquer lugar do programa.
□□ O seu maior uso é para encerrar o domínio de um comando DO que teria de terminar com um dos comandos proibidos.

Exemplo:

```
      .
      .
      DO 3 i=2, 8, 2
      A = B + C
      iF (A) 3, 3, 20

    3 CONTiNUE
      .
      .
      .
   20 STOP 7777
```

12

Variáveis Subscritas e Indexadas

□□ Até agora nós já estudamos como armazenar dados em variáveis inteiras ou reais. Se desejássemos contar, por exemplo, quantas palavras perfuradas numa massa de cartões começam com a letra A, poderíamos criar um acumulador chamado LETRAA. Se desejássemos, no entanto, saber qual o total geral de palavras que estão na memória já distribuídas por grupos cuja letra inicial sejam iguais, seria bastante trabalhoso ter de citar cada um dos 26 grupos correspondentes às 26 letras do alfabeto e escrever os comandos para fazer a acumulação.

□□ Para facilitar a resolução deste tipo de problema em que um grupo de variáveis pertence a um determinado conjunto, devemos utilizar as chamadas variáveis subscritas ou variáveis indexadas.

□□ A sua forma geral é

NOME (índice)

onde

NOME — é o nome da variável, o qual segue rigorosamente as regras já estabelecidas para variáveis inteiras e reais.

índice — é uma variável inteira e sem sinal que indica qual a variável do grupo que desejamos.

Exemplo:

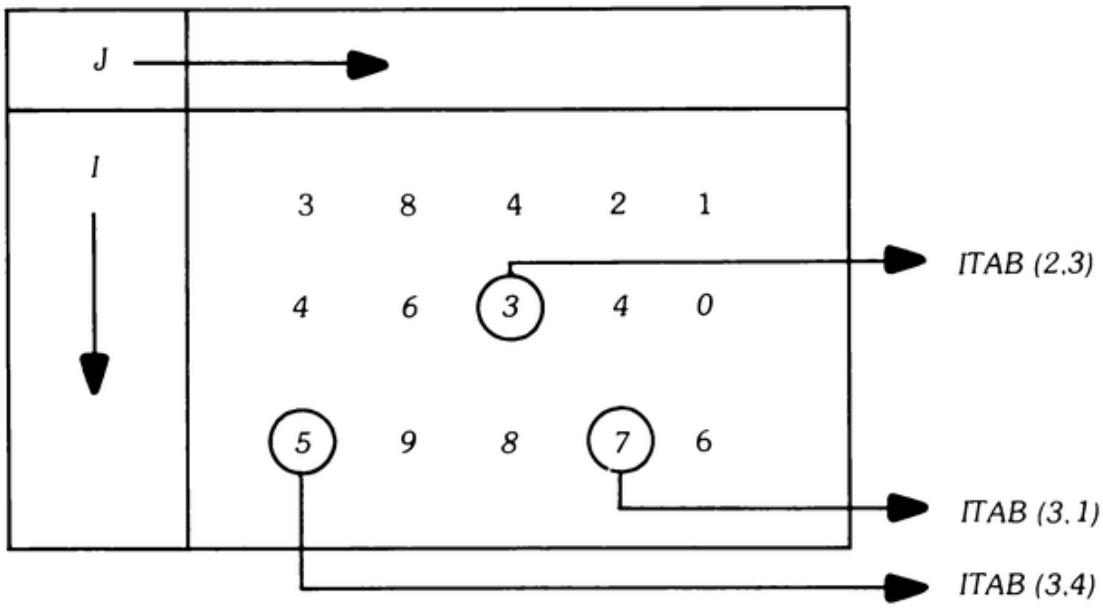
```
J = 0
DO 10 I=1,26
J = J + LETRA (I)
```

□□ A variável subscrita é LETRA [I]

□□ É possível, também, fazer variáveis subscritas com duas ou três dimensões. As suas formas são as seguintes:

NOME (i)	uma dimensão
NOME (i, j)	duas dimensões
NOME (i, j, k)	três dimensões

□□ A variável com dois subscritos pode ser representada pela tabela:



□□ Consideremos os elementos desta tabela como sendo ITAB (I,J). O valor de I pode variar de 1 a 3 e o valor de J pode variar 1 a 5.

Exemplos:

ITAB (2,3)	elemento 3
ITAB (3,4)	elemento 7
ITAB (3,1)	elemento 5

Formas Permitidas de Subscritos

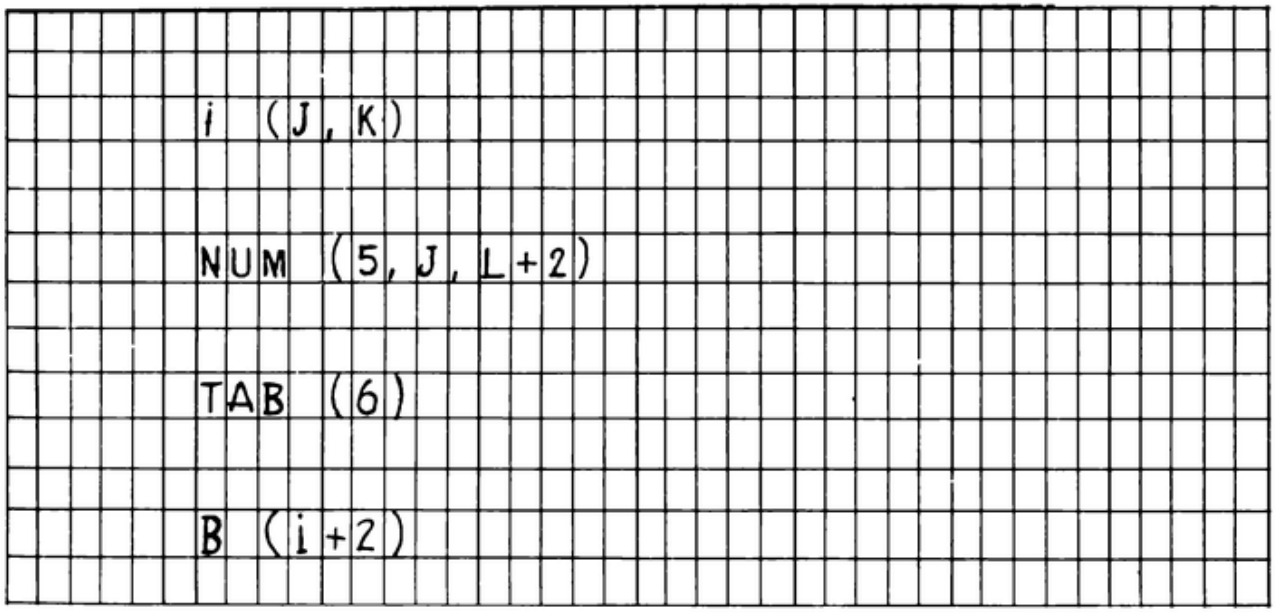
□□ O Fortran aceita que os subscritos *i, j e k* sejam:

- variáveis inteiras sem sinal (v);
- constantes inteiras sem sinal (c) e (c_1);
- expressões dos tipos:

v + c
v − c
c * v
c1 * v + c2

c1 * v − c2

□□ *Exemplos* de variáveis subscritas válidas:



Especificação DIMENSION

□□ Vimos que o FORTRAN aceita variáveis subscritas com um, dois ou três subscritos. Vimos que as variáveis subscritas podem ser compostas de vários elementos. Agora veremos como podemos definir quantos elementos e quantos subscritos têm as variáveis. Para passar estas informações para o compilador existe a declaração DIMENSION cuja forma geral é

DIMENSION v_1 (i_1), v_2 (i_2), ... v_n (i_n)

onde

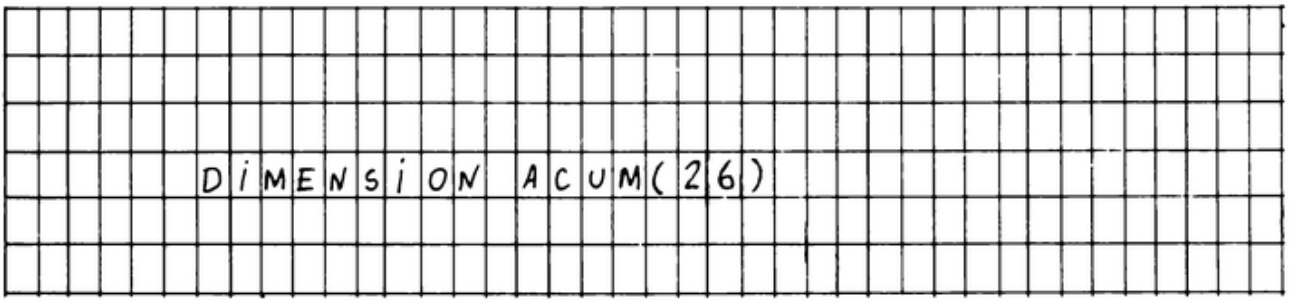
v_1, v_2, ... v_n — são nomes de variáveis reais ou inteiras que representam os conjuntos.

i_1, i_2, ... i_n — são constantes inteiras sem sinal, separadas por vírgula, que indicam o número máximo de cada subscrito.

□□ As declarações DIMENSION devem aparecer no início do programa.

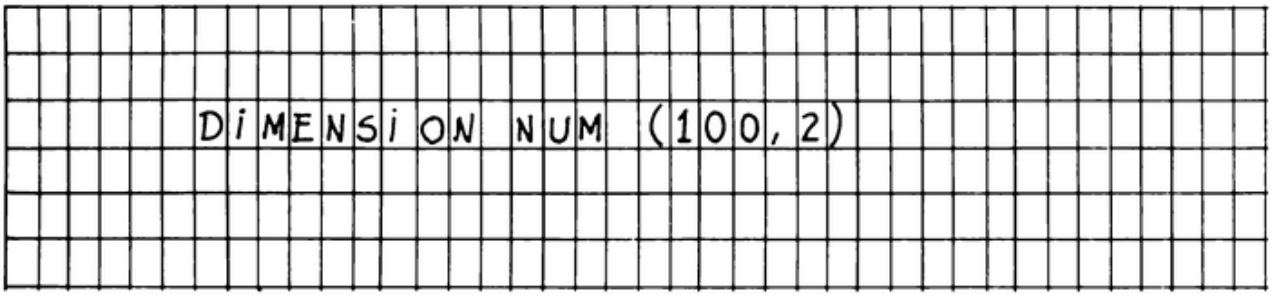
Exemplos:

a)



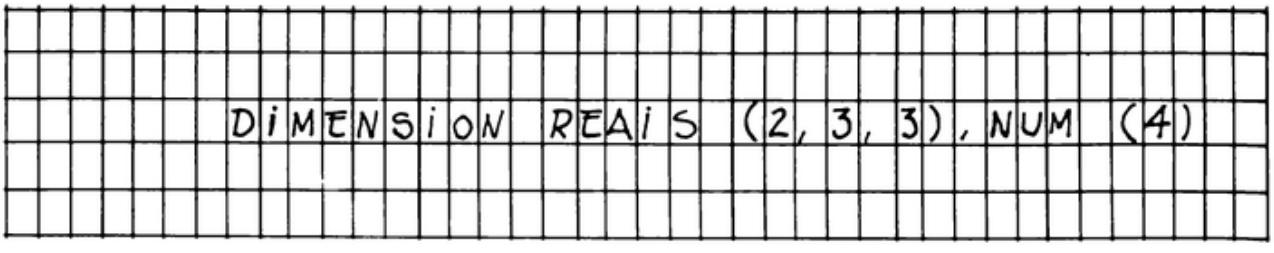
Reserva 26 variáveis reais para o conjunto ACUM.

b)



Reserva 200 variáveis inteiras (100 x 2) para o conjunto NUM.

c)



Reserva 18 variáveis reais (2 x 3 x 3) para o conjunto REAIS (2 x 3 x 3) e quatro variáveis inteiras para o conjunto NUM.

Comando READ com DO Implícito

□□ Podemos escrever um comando READ em que o comando DO esteja implícito. Usando este recurso podemos orientar o computador para ler somente alguns cartões desejados e ignorar os demais.

Exemplos:

a)

```
      DO 1 i=1,30
    1 READ (2,5) J(i)
    5 FORMAT (i10)
```

□□ Este trecho do programa provocará a leitura de 30 cartões. O mesmo resultado poderia ser conseguido com os comandos.

```
      READ (2,5) (J(i), i=1,30)
    5 FORMAT (i10)
```

□□ O DO implícito fica colocado entre parênteses no comando READ.

b)

```
      READ (2,20 (CAMPO(i), i = 1,30,5)
```

□□ Neste exemplo, a leitura só será feita para os campos CAMPO(1), CAMPO(6), CAMPO(11), CAMPO(16), CAMPO(21) e CAMPO(26).

* * *

13

Exemplos de Programas

□□ Apresentamos, apenas como exemplo, alguns programas bem simples que servirão para orientá-lo na escrita dos seus *próprios*.

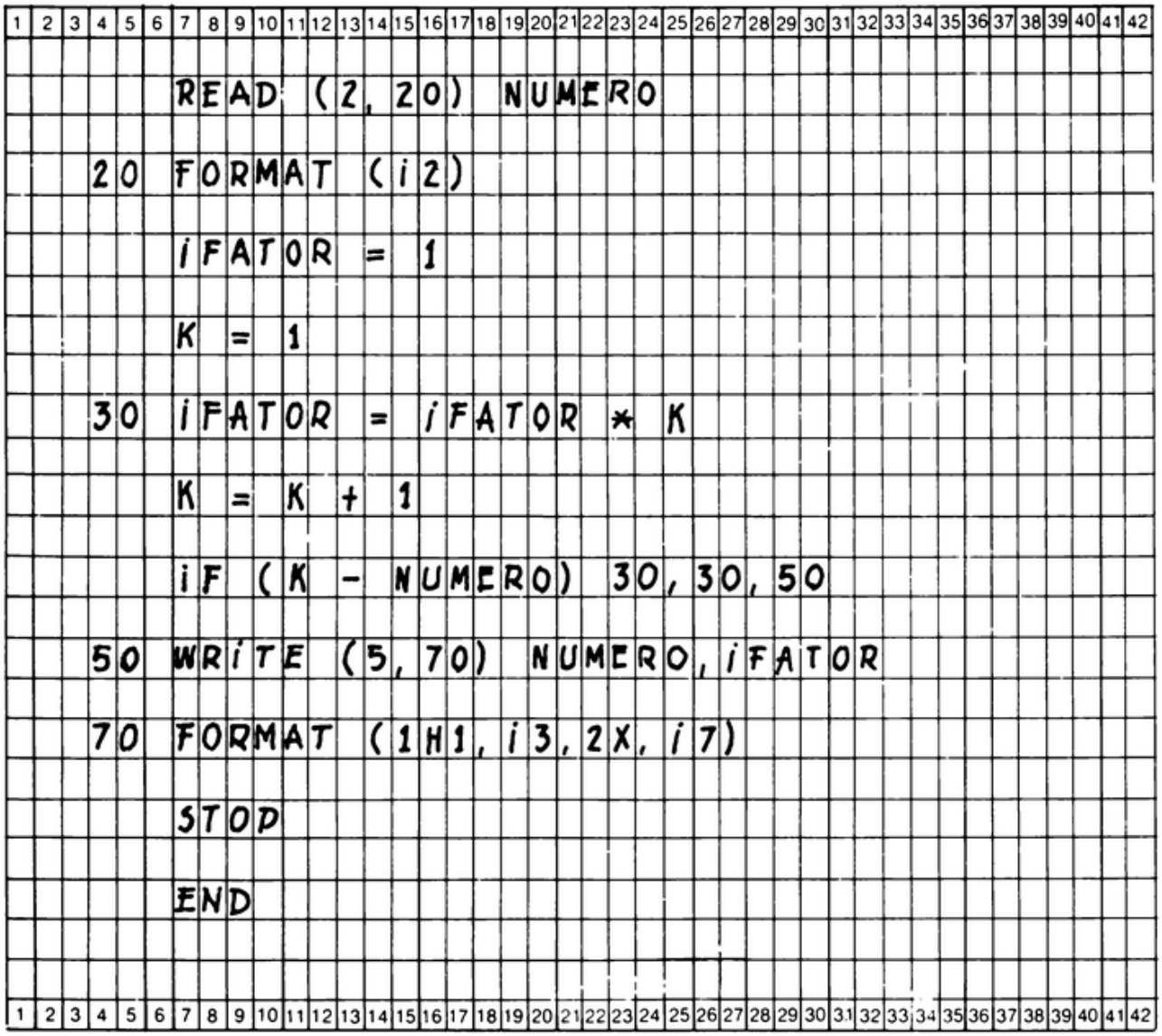
Cálculo da Média Aritmética

Exemplo 1 — Escrever um programa para calcular a média aritmética dos números de 1 a 50.

```
      IACUM = 0
      NUM = 1
    5 iACUM = iACUM + NUM
      NUM = NUM + 1
      iF (NUM.LE.50) GO TO 5
      MEDARi = iACUM / NUM
      WRiTE (5, 10) MEDARi
   10 FORMAT (i10)
      STOP
      END
```

Cálculo de Fatorial

Exemplo 2 — Fazer um programa para calcular o fatorial de um número inteiro, o qual será fornecido nas colunas 1 e 2 de um cartão. Sabe-se que o maior número que será perfurado no cartão é 10.



□□ Usando o comando DO, teríamos esta outra solução para o exemplo anterior:

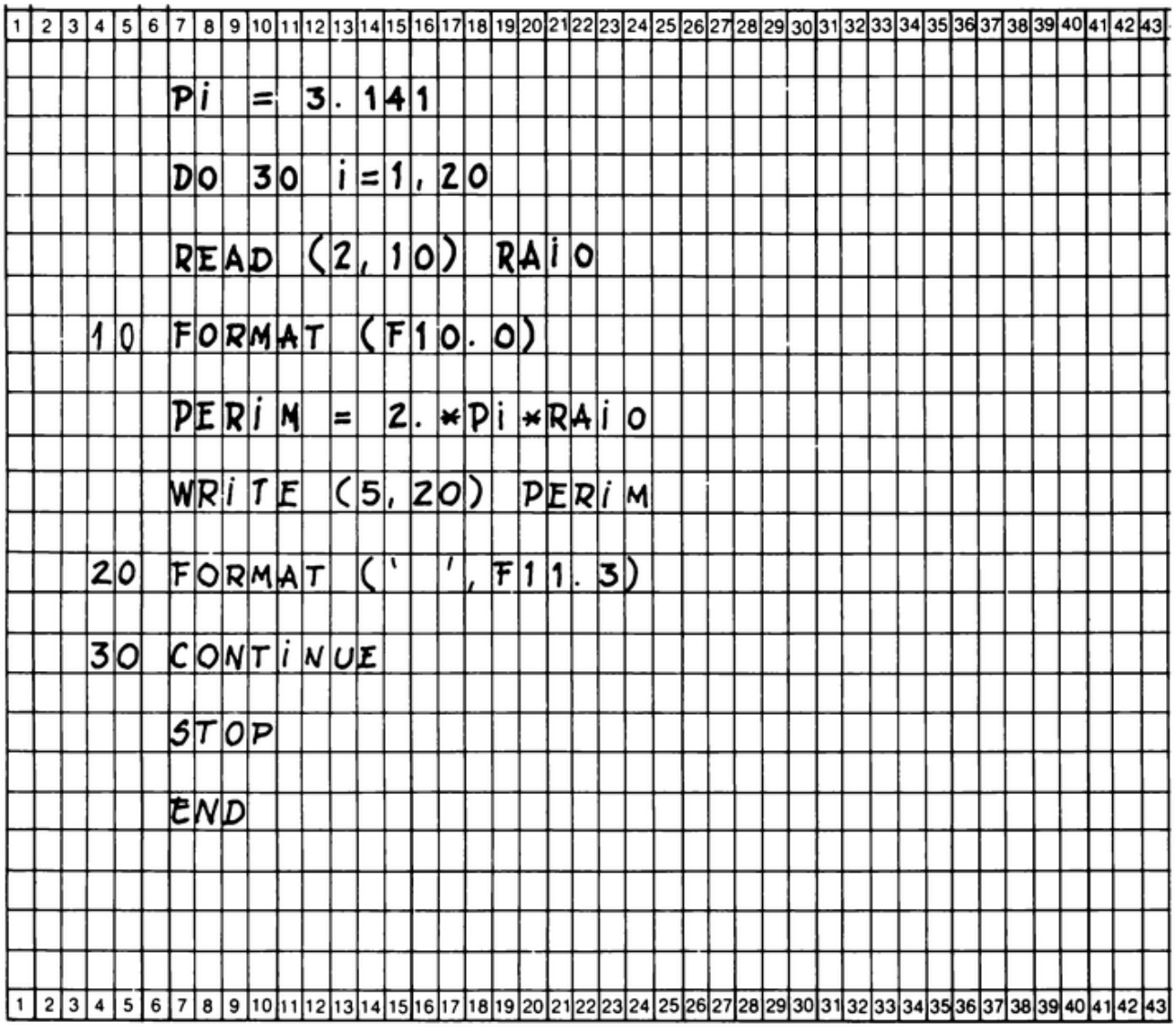
```
      READ (2, 20) NUMERO
      FORMAT (I2)
      IFATOR = 1
      DO 30 K=1, NUMERO
      IFATOR = IFATOR * K
   30 CONTINUE
      WRITE (5, 70) NUMERO, IFATOR
   70 FORMAT (1H1, I3, 2X, I7)
      STOP
      END
```

Perímetro da Circunferência

Exemplo 3 — Escrever um programa em FORTRAN para calcular o perímetro de uma circunferência (Fórmula 2 π R). O raio da circunferência será fornecido através de um cartão perfurado, nas 10 primeiras colunas. O raio fornecido nas colunas 1/10 será sempre um número inteiro sem posição decimal. Fornecer o resultado com 3 decimais e considerar o valor de π como 3,141.

```
      PI = 3.141
      READ (2, 10) RAIO
   10 FORMAT (F10.0)
      PERIM = 2.*PI*RAIO
      WRITE (5, 20) PERIM
   20 FORMAT (1H1, F11.3)
      STOP
      END
```

Exemplo 4 — Escrever o programa do exemplo 3, porém para calcular o perímetro das circunferências cujos raios serão fornecidos nos 20 primeiros cartões lidos.



Raízes Reais de Uma Equação do Segundo Grau

Exemplo 5 — Escrever um programa para calcular as raízes reais de uma equação do segundo grau. A fórmula para cálculo das raízes é:

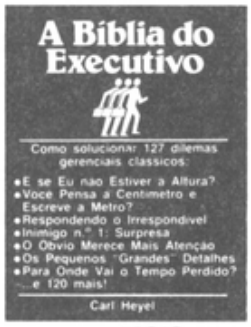
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

□□ Os valores de *a*, *b* e *c* são inteiros sem sinal que serão fornecidos num cartão, perfurados nas colunas 1/10, 11/20 e 21/30, respectivamente. Se $(b^2 - 4ac)$ for negativo, imprimir uma mensagem informando que as raízes não são reais.

```
      READ (2,20) A,B,C
   20 FORMAT (3F10.0)
      IF (B**2-4.*A*C) 30,60,60
   30 WRITE (5,40)
   40 FORMAT (1H1,'AS RAIZES NAO SAO REAIS')
   50 STOP
C
C     AS RAIZES SAO X1 E X2
C
   60 X1 = (-B + SQRT(B**2-4.*A*C)) / (2.*A)
      X2 = (-B - SQRT(B**2-4.*A*C)) / (2.*A)
      WRITE (5,80) X1,X2
   80 FORMAT (1H1,F10.2,3X,F10.2)
      STOP
      END
```

Empresa

Estas obras,
em linguagem clara
e objetiva,
vão ajudá-lo na árdua
tarefa de organizar e
dirigir uma empresa.



10618

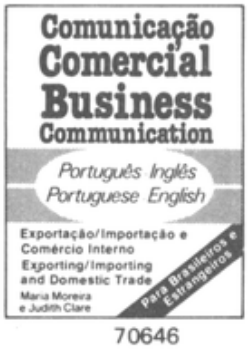
70419

40091

18548

40057

38040



70646

30052

58621

30312

20139

58554

Computação
Processamentos de Dados


10627

90032

98087

10067

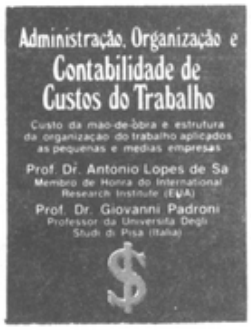
28614

90071

30228

10160

Aprender a
Programação Assembler
servirá para complementar
profissões em geral
e alcançar novas
posições no
mercado de trabalho

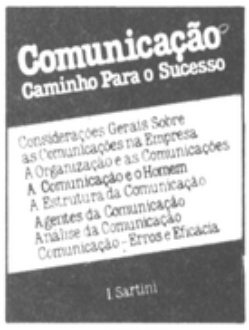


70792

28001

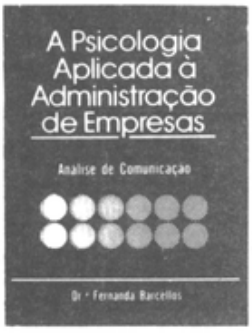
98566

48191



98588

10434



10594

98535

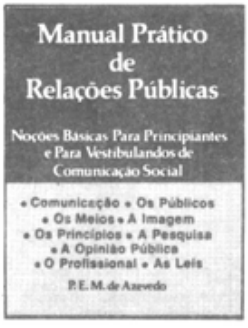
40141

40219

68075

60459

40592



78195

58005

68528

38510

10778

As principais carreiras, os requisitos básicos e as perspectivas de cada uma

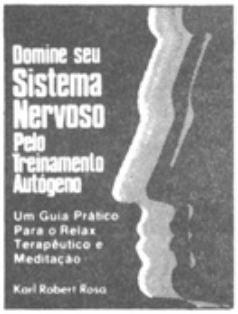
90354

60719

A linguagem BASIC para os *micros* e importantes instruções para quem possui ou pretende possuir um

PSICOLOGIA

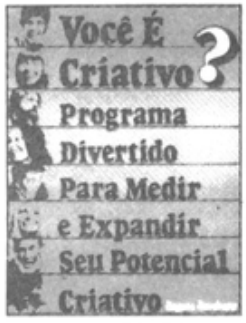
APERFEIÇOAMENTO PESSOAL



80069

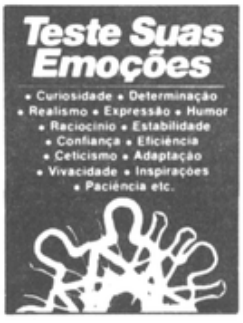
10201

28564

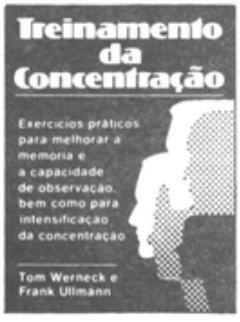


50454

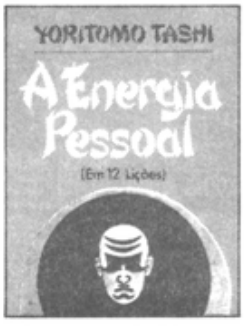
68058



20061



40446



48059

88060

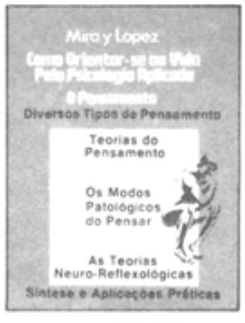
68061

48062

70016

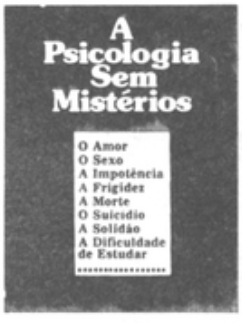
90287

18184

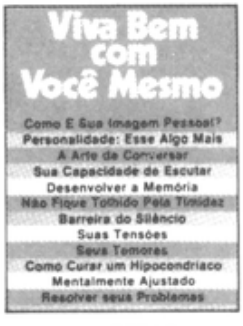


58506

98549

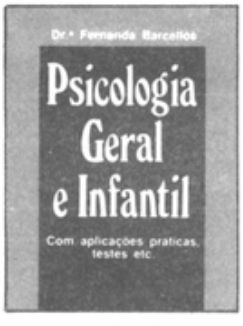


58196



88513

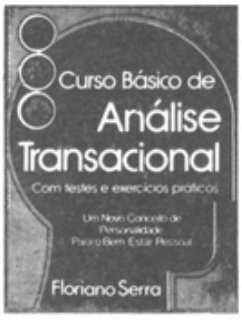
28144



20531

20643

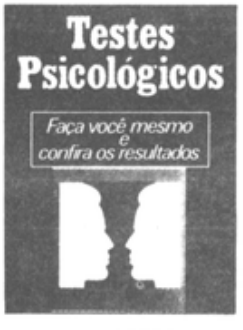
30004



98185

18601

40172



18590

68108

10188

20268

70503

50406

78603

90135

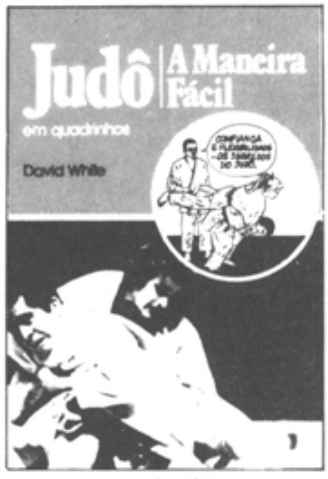
28192

50065

Esportes de Luta

Defenda-se! Aprenda com os mestres, através de livros com fotos e ilustrações explicativas.

90709



60087

28094

80153

48143


40351

18145

90359

90628

98137

18511

68142

98168

68139

10358

60154

80363

18167

20397

38138

88172

10750

18136

98017

18016

70209

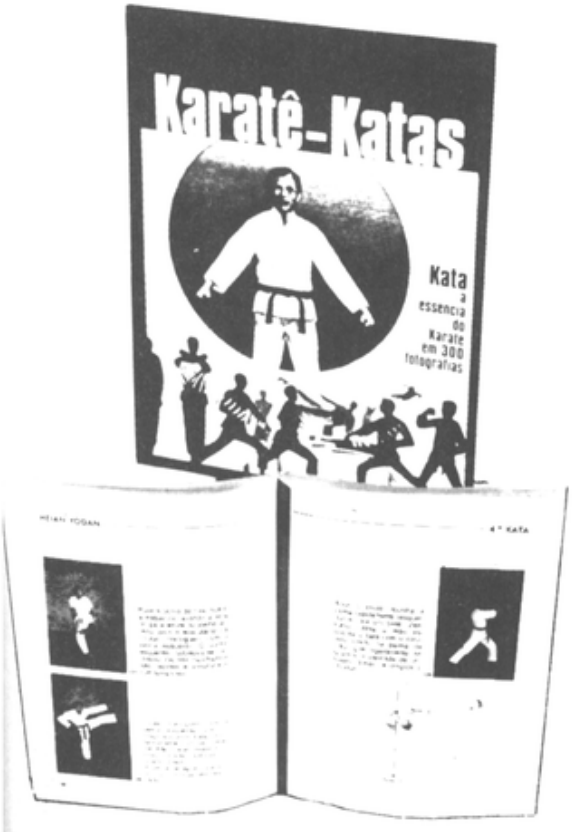
38023

58148

.Automóvel
.Motocicleta

18539

O método de treinamento de Bruce Lee, o gênio das artes marciais.



88155

Qualquer pessoa pode exercitar sozinho o Kata, em qualquer lugar, pois ele não requer qualquer equipamento.
Para ambos os sexos.

20433

38183

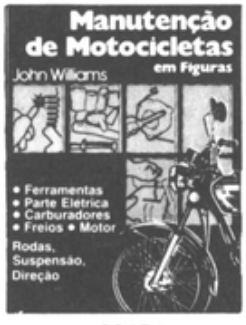
60185

80394

98194

18419

80444



88558

10174

78438

98437

90409

78536

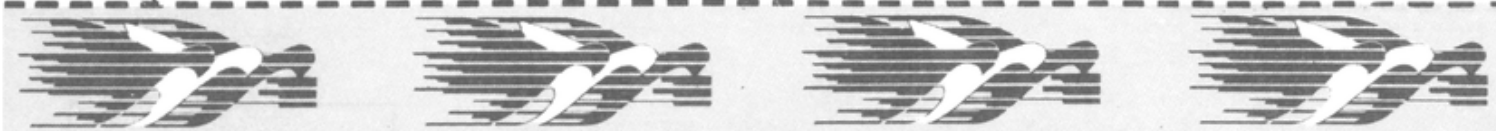
80525

70596

Por Reembolso Postal

CX. POSTAL 1880
CEP 20001 — RIO DE JANEIRO — RJ

(Não mande nada adiantado)
Você paga no Correio ao retirar o volume. Leve Identidade (indispensável).



Desejo receber por **Reembolso Postal** os livros:

Quant.	N.º	Quant.	N.º	Quant.	N.º	Quant.	N.º	Quant.	N.º

Escreva sempre o número e a letra, quando houver.

Q.T.

Nome:

Endereço (rua e número):

Bairro:

CEP n.º Cidade: Estado:

Preencha com letras MAIÚSCULAS ou à máquina.

FL16 304

* Os livros serão remetidos ao preço atual conforme as Séries, Coleções ou símbolos. (Pelo preço do livro que tem em mãos é possível avaliar os outros pela coluna ao lado).

Para pedidos pequenos será cobrado uma taxa de despesas.

PREÇOS:

Séries, Coleções e símbolos em ordem crescente aproximada para avaliação dos preços dos livros.

Fantasminha (Infantil)
Enrola e Desenrola Infantil
▮
*x1
Enrola e Desenrola Juvenil
Séries de Ficção
Edijovem
1001 Receitas Ediouro
Picolé-Livro
Bordado
Elefante (17 e 12 anos)
Grandes Filósofos
Grandes Personagens
O que É, o que É?
Sabedoria e Pensamento
▮▮
*x2
Espiritualismo
Horas de Ouro
Mitologia Grega (Infantil)
▼
Boa Cozinha
*x3
Realismo Fantástico
Guias Pintura e Desenho
*x4
▼ ▼

Dentro de cada coleção ou série os livros têm preço igual

ISR - 52-312/81
UP. CT PRINCIPAL
DR/RIO

CARTÃO-RESPOSTA-COMERCIAL

Não é necessário selar

O selo será pago por:

EDITORA TECNOPRINT S.A.

20299

RIO DE JANEIRO – RJ

Programação Fortran

Foram criadas várias formas de instruir o computador por meio de códigos e regras mais simples do que os códigos de máquina e as regras que regem estes códigos de máquina.

O conjunto destes códigos e regras simplificadas é conhecido como linguagem de programação. O Fortran, abreviação de Formula Translator, é muito empregado em processamento científico.