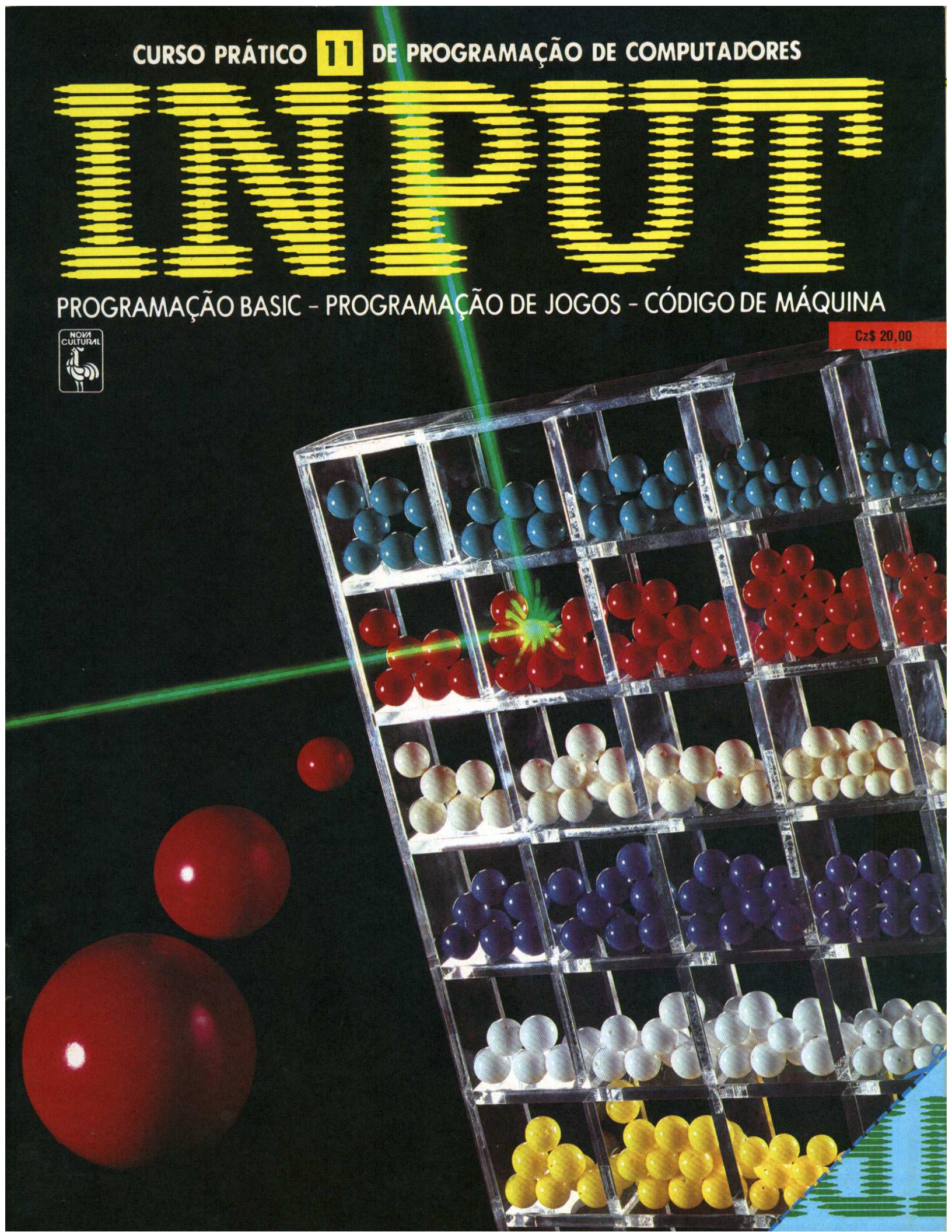
CURSO PRÁTICO 11 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

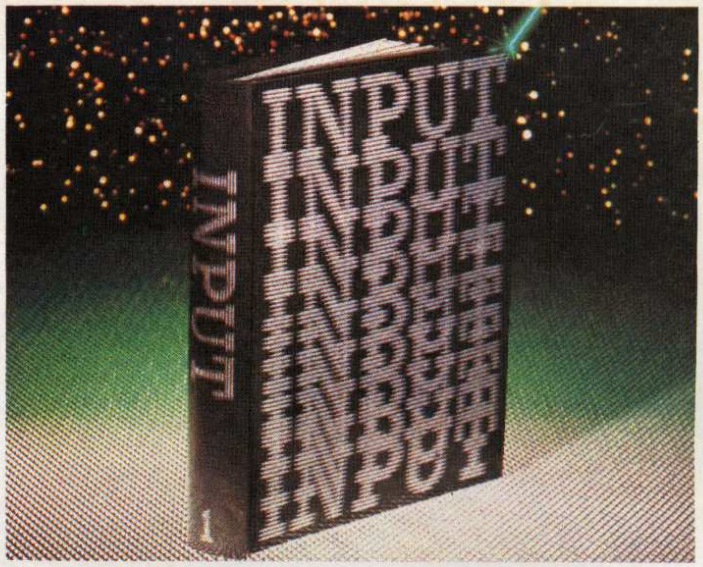
Cz$ 20,00



INPUT

Vol. 1 N.º 11

NESTE NÚMERO



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretora Editorial: Iara Rodrigues

Editor chefe: Paulo de Almeida
Editor de texto: Cláudio A.V. Cavalcanti
Editor de Arte: Eduardo Barreto
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, José Maria de Oliveira, Grace A. Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz
Secretário Gráfico: Antonio José Filho

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda. Campinas, SP.

Tradução: Maria Fernanda Sabbatini

Adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli,
Coordenação geral: Rejane Felizatti Sabbatini
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Ferrucio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atilio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Ana Maria Dilguerian, Antonio Francelino de Oliveira, Karina Ap. V. Grechi, Levon Yacubian, Maria Teresa Galluzzi, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria do Carmo Leme Monteiro, Maria Luiza Simões, Maria Teresa Martins Lopes.

Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Dívisão Gráfica da Editora Abril S.A.

CONJUNTOS DE DUAS DIMENSÕES

- DEFINIÇÃO DE MATRIZES
- ENTRE DADOS EM UMA MATRIZ
- COMO PROGRAMAR MATRIZES
- RECUPERAÇÃO DE INFORMAÇÕES
- USOS PARA AS MATRIZES

Se você tem uma grande quantidade de dados inter-relacionados, a informação pode estar perdida entre eles. Coloque-os numa matriz e o computador extrairá exatamente a informação desejada.

Agora que você já conhece como um conjunto pode ser empregado para conservar informações em um programa (lição *Conjuntos: Caixas de Informação*, página 192), podemos passar ao estudo de matrizes. Matriz (ou matrix) é todo conjunto *bidimensional* que serve de meio para armazenar grupos de dados *inter-relacionados*.

Um conjunto bidimensional é, graficamente, como uma pilha de gavetinhas, onde cada gaveta serve para armazenar um único dado. Esta é uma maneira muito conveniente de lidar com informações pois, para localizar uma delas, basta se referir a um elemento do conjunto, por meio dos seus números de linha e de coluna.

A matriz armazena itens reais, tais como os parafusos da ilustração abaixo, onde se pode encontrar imediatamente a caixa que contém parafusos de latão de 38 mm, ou para qualquer outra que se esteja procurando. Neste caso o dado da matriz é o número de parafusos em cada caixa.

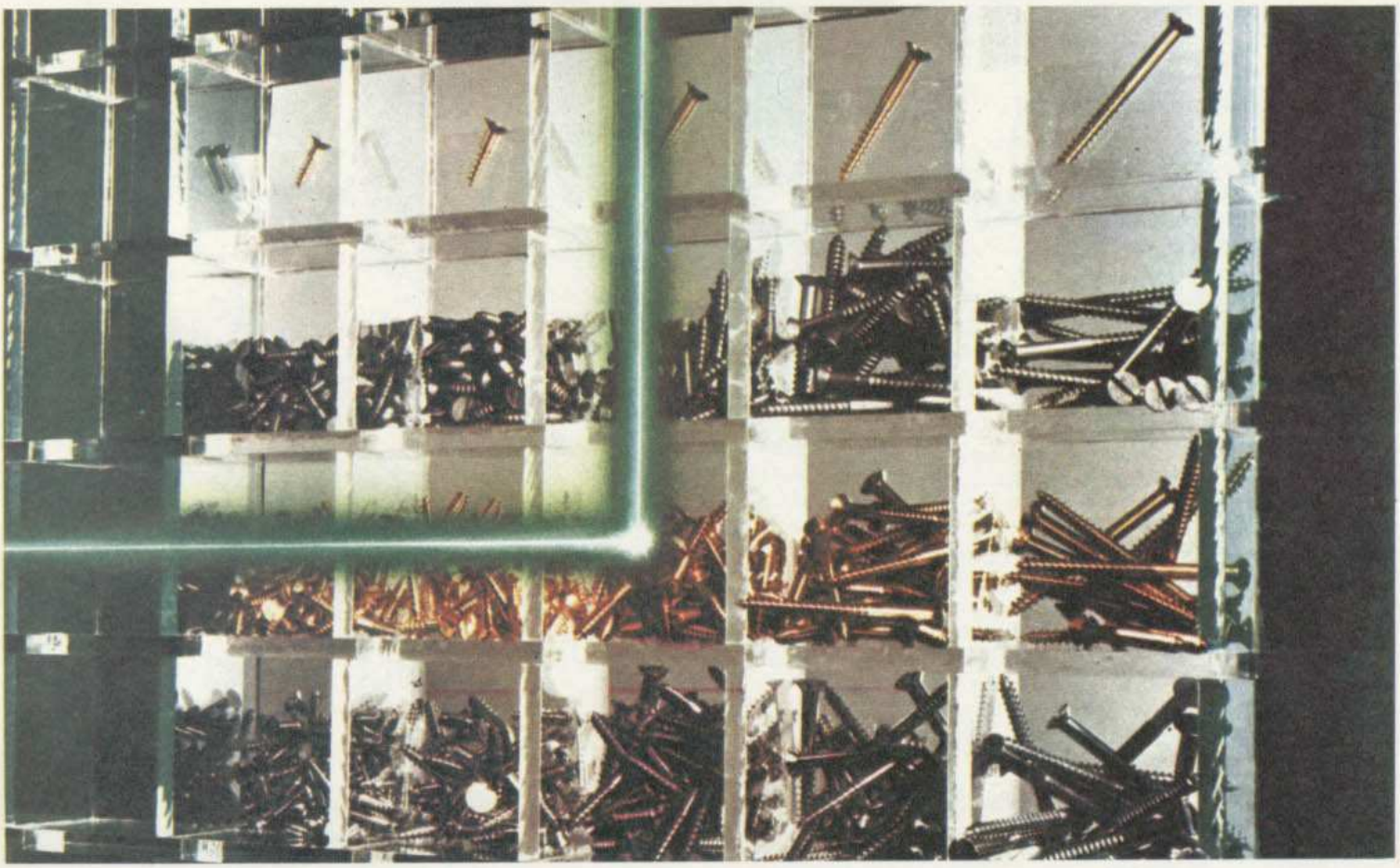
A matriz pode também guardar informações abstratas, como o número de revendedores de automóveis, discriminados segundo a sua nacionalidade e marca dos carros vendidos.

Informações como esta são freqüentemente colecionadas como resultado de levantamentos, que são a base para o programa exposto neste artigo. Entretanto, a estrutura do programa é a mesma para qualquer matriz bidimensional, pouco importando o tipo de informação que ela contém.

Como exemplo, suponha que você seja um professor interessado em classificar os animais de estimação trazidos por um conjunto de crianças. Em primeiro lugar, seria preciso anotar o nome das crianças e descrever quantos animais de cada grupo elas possuem. A lista seria mais ou menos assim:

Sílvia: dois periquitos, um coelho.
João: um cachorro, quatro peixes.
Carlos: dois hamsters, um gato.

Recurso muito útil para armazenar dados, as matrizes servem também para classificar pequenos itens como parafusos.



Jaci: um cachorro, um gato, um hamster.
Beto: um rato, um coelho, dois peixes.

Entretanto, em uma lista como essa torna-se um tanto difícil extrair informações como "quantas pessoas possuem gatos?", ou "quem tem mais de quatro animais?" — e quanto maior for a lista, mais difícil será também obter respostas.

A melhor maneira de resolver o problema seria distribuir as informações em forma de tabela ou quadro, com o nome dos animais dispostos no alto e o das crianças à esquerda. Pois bem, é exatamente assim que funciona uma matriz bidimensional.

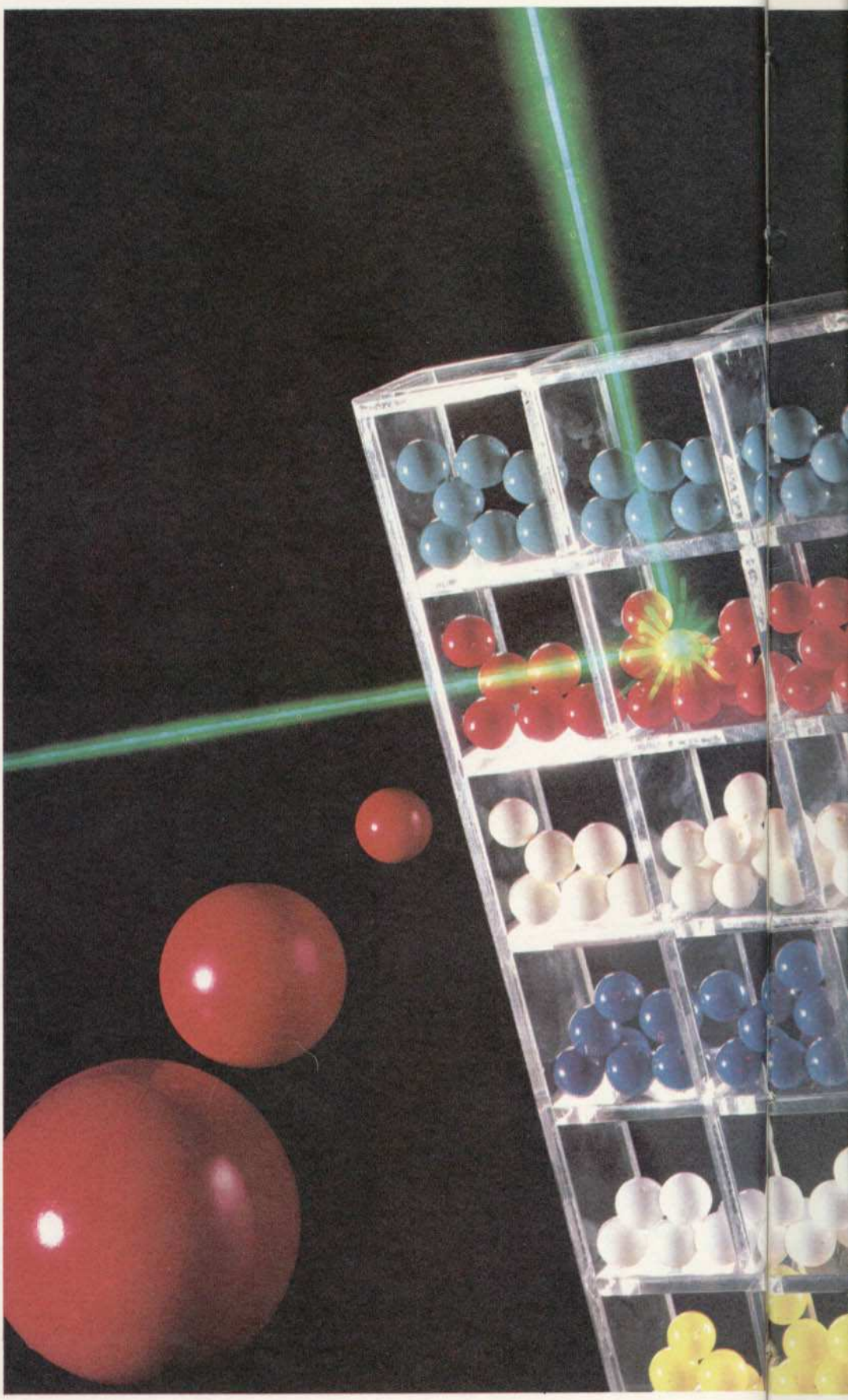
Em um exemplo pequeno como este é simples encontrar a resposta para qualquer pergunta sobre os dados. Entretanto, o computador trabalha com um infindável número de informações. Se você examinar todo o conjunto de dados para descobrir quem tem pelo menos um gato, ou se quiser saber o nome das pessoas que possuem ratos, verificará que o trabalho será tanto maior quanto mais ampla for a quantidade de dados. Por sorte, o computador pode cumprir essa tarefa em questão de segundos.

DEFINA A MATRIZ

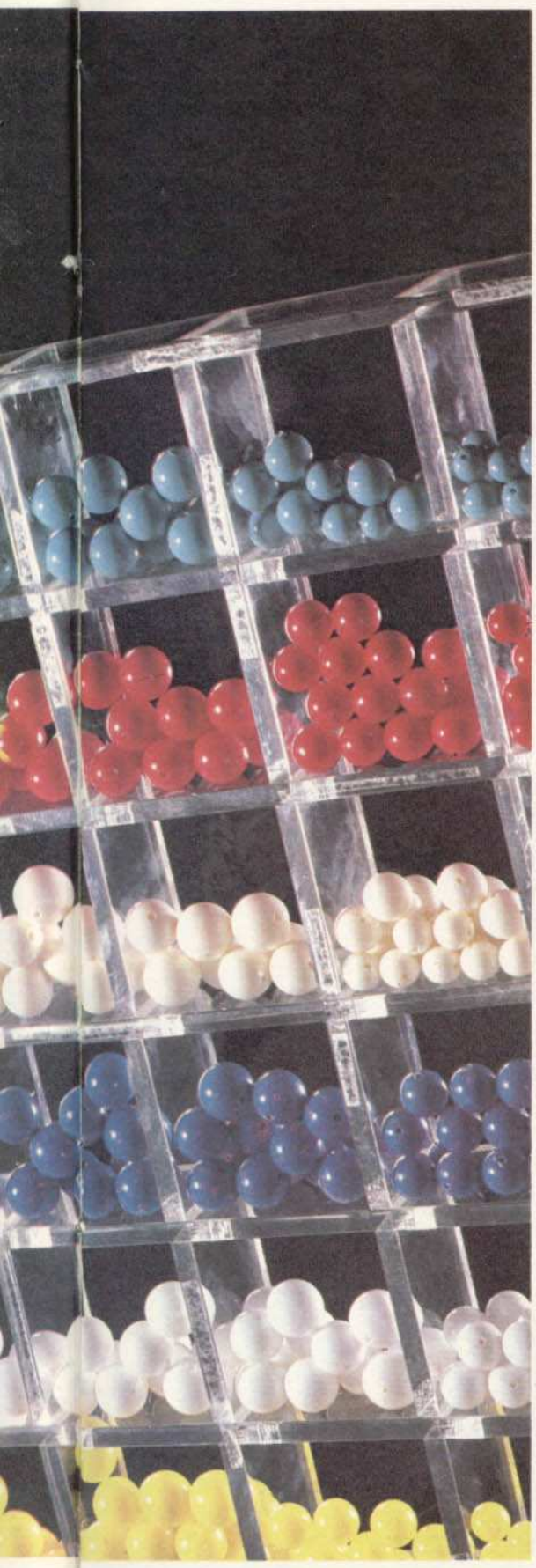
Equacionado o problema, o passo seguinte será colocar a informação dentro do computador. Se você voltar às páginas 192 a 195 verá como definir e usar uma matriz unidimensional (também chamada de conjunto). Matrizes bidimensionais são muito similares, envolvendo apenas um pouco mais de trabalho para serem definidas.

Uma matriz unidimensional pode ser escrita na forma **A(N)**, onde **N** é o número dos seus elementos. Uma matriz bidimensional, por sua vez, é definida como **A(L,C)**, onde **L** é o número de linhas e **C** o número de colunas. Na verdade, pouco importa se você a definir de forma invertida, como **A(C,L)**, com as colunas antes das linhas, desde que seja coerente com a primeira ou a segunda definições.

Contudo, mesmo trabalhando com uma matriz bidimensional, você precisará definir duas matrizes unidimensionais que sirvam de cabeçalho para as colunas e para as linhas (no nosso caso, uma matriz conterá o nome das crian-



Assim como as bolinhas da ilustração, distribuídas ordenadamente segundo seu tamanho e cor, as informações devem ser armazenadas na memória de acordo com o grupo a que pertencem.



ças e a outra o nome dos animais). A matriz bidimensional conterá os dados.

Vamos chamar as duas primeiras matrizes de **PET$ (C)** e **CH$ (R)** e a matriz de dados de **N (R,C)**. Uma outra matriz, **PT (R)**, conterá o número total de animais em cada linha. No Spectrum, as matrizes são rotuladas de **p$ (c)**, **c$ (r)**, **n (r, c)** e **p (r)**, pois é permitido apenas um caractere para a formação dos nomes. O programa não funcionará no ZX-81. Eis um programa para dimensionar a matriz e ler os dados de um comando **DATA**.

```
100 C=7:R=5
110 DIM PET$(C),CH$(R),N(R,C),P
T(R)
120 FOR J=1TOC:READPET$(J):NEXT
130 FORJ=1TOR:READCH$(J):NEXT
140 FOR J=1TOR
150 FORK=1TOC
160 READ N(J,K):NEXT:NEXT
3000 DATA PERIQUITO,GATO,CACHOR
RO,PEIXE,RATO,HAMSTER,COELHO
3010 DATA SILVIA,JOAO,CARLOS,JA
CI,BETO
3020 DATA 2,0,0,0,0,0,1
3030 DATA 0,0,1,4,0,0,0
3040 DATA 0,1,0,0,0,2,0
3050 DATA 0,1,1,0,0,1,0
3060 DATA 0,0,0,2,1,0,1
```

```
100 LET c=7: LET r=5
110 DIM p$(c,7): DIM c$(r,5):
DIM n(r,c): DIM p(r)
120 FOR j=1 TO c: READ p$(j):
NEXT j
130 FOR j=1 TO r: READ c$(j):
NEXT j
140 FOR j=1 TO r
150 FOR k=1 TO c
160 READ n(j,k): NEXT k: NEXT
j
3000 DATA "PERIQUITO","GATO","C
ACHORRO","PEIXE","RATO","HAMSTE
R","COELHO"
3010 DATA "RODRIGO","RICARDO","
FERNANDA","BEATRIZ","JUNIOR"
3020 DATA 2,0,0,0,0,0,1
3030 DATA 0,0,1,4,0,0,0
3040 DATA 0,1,0,0,0,2,0
3050 DATA 0,1,1,0,0,1,0
3060 DATA 0,0,0,2,1,0,1
```

As variáveis **R** e **C** foram usadas para facilitar a adaptação do programa: afinal, é improvável que sua matriz tenha o mesmo número de linhas e de colunas que a que apresentamos aqui. Assim, tudo o que você tem a fazer é trocar os números na linha 100 e, logicamente, as informações nas linhas dos comandos **DATA**.

Note agora que os dados entram na matriz. Existe uma linha para o cabeçalho das colunas, outra para o cabeçalho das linhas e uma para cada linha da matriz principal. Note também que você deve entrar um item de dado para cada espaço na matriz, mesmo que ele seja zero, ou o computador responderá com uma mensagem de erro.

Agora que os dados estão dentro do computador, você deve decidir o que fazer com eles. Um de seus recursos é oferecer tantas opções quanto possível: afinal, é o computador que vai trabalhar, e não você.

O MENU

Será melhor descrever as opções em forma de menu. E para a pesquisa dos animais você provavelmente precisará do seguinte:

```
300 LET fo=0: CLS
310 PRINT ''"MENU"
320 PRINT '"1- LISTAR ANIMAIS"
330 PRINT "2- LISTAR CRIANCAS"
340 PRINT "3- INTRODUZIR TIPO
DE ANIMAL"
350 PRINT "4- INTRODUZIR NOME
DA CRIANCA"
360 PRINT "5- INTRODUZIR NUMER
O DE ANIMAIS"
370 PRINT "6- MOSTRAR A MATRIZ
"
380 PRINT ''"SELECIONE OPCAO"
390 INPUT a
395 IF a<1 OR a>6 THEN  GOTO
390
400 CLS
410 GOSUB (a*100+400)
420 GOTO 300
```

No TRS-80, multiplique por 2 os valores utilizados com o **PRINT@**.

```
300 FOUND=0:CLS
310 PRINT @43,"MENU"
320 PRINT @98,"1 LISTAR ANIMAIS
330 PRINT @130,"2 LISTAR CRIANC
AS
340 PRINT @162,"3 INTRODUZIR TI
PO DE ANIMAL
350 PRINT @194,"4 INTRODUZIR NO
ME DA CRIANCA"
360 PRINT @226,"5 INTRODUZIR NU
MERO DE ANIMAIS"
370 PRINT @258,"6 MOSTRAR A MAT
RIZ"
380 PRINT:PRINT"ESCOLHA A OPCAO
";
390 INPUT A
395 IF A<1 OR A>6 THEN 390
400 CLS
410 ON A GOSUB 500,600,700,800,
900,1000
420 GOTO 300
```

```
300 FO=0:CLS
310 LOCATE 17,1:PRINT"MENU"
320 LOCATE 7,4:PRINT"1. LISTAR
ANIMAIS"
330 LOCATE 7:PRINT"2. LISTAR CR
IANÇAS"
340 LOCATE 7:PRINT"3. INTRODUZI
R TIPO DE ANIMAL"
350 LOCATE 7:PRINT"4. INTRODUZI
R NOME DE CRIANÇA"
360 LOCATE 7:PRINT"5. INTRODUZI
R NUMERO DE ANIMAIS"
370 LOCATE 7:PRINT"6. MOSTRAR A
 MATRIZ"
380 PRINT:PRINT:LOCATE 20:PRINT
"OPÇAO";
390 INPUT A
395 IF A<1 OR A>6 THEN 390
400 CLS
410 ON A GOSUB 500,600,700,800,
900,1000
420 GOTO 300
```

```
300 FO = 0: HOME
310  HTAB 18: VTAB 2: PRINT "ME
NU"
320  VTAB 5: HTAB 8: PRINT "1.
LISTAR ANIMAIS"
330  HTAB 8: PRINT "2. LISTAR C
RIANCAS"
340  HTAB 8: PRINT "3. INTRODUZ
IR TIPO DE ANIMAL"
350  HTAB 8: PRINT "4. INTRODUZ
IR NOME DE CRIANCA"
360  HTAB 8: PRINT "5. INTRODUZ
IR NUMERO DE ANIMAIS"
370  HTAB 8: PRINT "6. MOSTRAR
A MATRIZ"
380  PRINT : PRINT : HTAB 20: P
RINT "OPCAO ";
390  INPUT A
395  IF A < 1 OR A > 6 THEN 390

400  HOME
410  ON A GOSUB 500,600,700,800
,900,1000
420  GOTO 300
```

ESCREVA AS SUB-ROTINAS

As sub-rotinas a seguir servirão para o início do trabalho, embora, mais tarde, você possa adicionar outras opções. A primeira imprime apenas a lista de animais, fazendo uma procura na matriz **PET$( )** ou **p$( )**. A segunda faz a mesma coisa para imprimir a lista de crianças.

```
499 REM ** OPCAO 1 **
500 PRINT '"LISTA DE ANIMAIS"
510 PRINT ''
520 FOR j=1 TO c
530 PRINT p$(j)
540 NEXT j
550 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
560 PAUSE 0
570 RETURN
599 REM ** OPCAO 2 **
600 PRINT '"LISTA DE CRIANCAS
"
610 PRINT ''
620 FOR j=1 TO r
630 PRINT c$(j)
640 NEXT j
650 PRINT '"PRESSIONE QUALQUE
R TECLA PARA RETORNAR"
660 PAUSE 0
670 RETURN
```

```
499 REM ### OPCAO 1 ###
500 PRINTTAB(5);"LISTA DE ANIMA
IS"
510 PRINT:PRINT
520 FOR J=1TOC
530 PRINT PET$(J)
540 NEXT
550 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
560 IF INKEY$=""THEN560
570 RETURN
599 REM ### OPCAO 2 ###
600 PRINTTAB(5);"LISTA DE CRIAN
ÇAS"
610 PRINT:PRINT
620 FOR J=1TOR
630 PRINT CH$(J)
640 NEXT
650 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
660 IF INKEY$=""THEN660
670 RETURN
```

```
499  REM    *** OPCAO 1 ***
500  PRINT  TAB( 10)"LISTA DE A
NIMAIS"
510  PRINT : PRINT
520  FOR J = 1 TO C
530  PRINT PET$(J)
540  NEXT
550  PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
560  GET A$
570  RETURN
599  REM  *** OPCAO 2 ***
600  PRINT  TAB( 10);"LISTA DE
CRIANCAS"
610  PRINT : PRINT
620  FOR J = 1 TO R
630  PRINT CH$(J)
640  NEXT
650  PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
660  GET A$
670  RETURN
```

A opção 3 é a mais interessante. Se você digitar o nome de um animal, o computador imprimirá uma lista de crianças que têm no mínimo um deles, informando ainda quantos espécimes cada criança possui.

```
699 REM ** OPCAO 3 **
700 PRINT '"INTRODUZA TIPO DE
 ANIMAL"
705 DIM i$(7): INPUT  LINE i$
710 PRINT '"PESSOAS QUE TEM UM
 ";i$
720 FOR j=1 TO c
730 IF p$(j)=i$ THEN  LET fo=j
740 NEXT j
750 IF fo=0 THEN  PRINT " ANIM
AL NAO ENCONTRADO.  TENTE  OU
RA VEZ.": GOTO 700
760 FOR j=1 TO r
```

A matriz mostra alguns revendedores de carros classificados por país e marca de automóvel.

```
770 IF n(j,fo)>0 THEN  PRINT c
$(j);" ";n(j,fo)
775 NEXT j
780 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
785 PAUSE 0
790 RETURN
```

```
699 REM ### OPCAO 3 ###
700 PRINTTAB(5);"DIGITE TIPO DE
 ANIMAL"
705 PRINT"LISTA DE TODOS QUE TE
M ";
710 INPUT P$
715 PRINT:PRINT
720 FORJ=1TOC
730 IFPET$(J)=P$THENFO=J
740 NEXT
750 IFFO=0THENPRINT"ANIMAL NÃO
ENCONTRADO. TENTE DE NOVO!":GOT
0700
760 FORJ=1TOR
```

```
770 IFN(J,FO)>0THENPRINTCH$(J);
" ";N(J,FO)
775 NEXT
780 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR";
785 IFINKEY$=""THEN785
790 RETURN
```

```
699  REM  *** OPCAO 3 ***
700  PRINT  TAB( 10);"DIGITE O
TIPO DE ANIMAL"
705  PRINT  TAB( 5);"LISTA DE T
ODOS QUE TEM ";
710  INPUT P$
715  PRINT : PRINT
720  FOR J = 1 TO C
730  IF PET$(J) = P$ THEN FO =
J
740  NEXT
750  IF FO = 0 THEN  PRINT "ANI
 MAL NAO FOI ENCONTRADO. TENTE
  NOVAMENTE!": GOTO 700
 760  FOR J = 1 TO R
 770  IF N (J,FO) > 0 THEN  PRI
 NT CH$(J);" ";N(J,FO)
  775  NEXT
  780  PRINT : PRINT "PRESSION
   E QUALQUER TECLA PARA RETOR
    NAR"
    785  GET A$
     790  RETURN
```

Esta rotina é mais complicada, de modo que vale a pena explicá-la com mais detalhes. A linha 710 guarda nossa entrada na variável **P$ (i$** no Spectrum). Digamos, por exemplo, que nossa entrada seja "GATO"; assim, **P$ = "GATO"** (ou **i$ = "GATO"**). As linhas 720 a 740 verificam se há algum **"GATO"** na lista de animais. Em caso positivo, a variável **FO** receberá o número da coluna em que "GATO" foi achado. No nosso caso, **FO=2** porque "GATO" está na coluna 2.

Se o computador atingir a linha 750 e **FO** for igual a 0 (ou seja, se seu valor for igual a 0 na linha 300), isso significa que o animal pedido não está na lista. Em conseqüência o programa recomeçará.

As linhas 760 a 775 percorrem cada elemento na coluna 2. Se algum elemento for maior que 0, isso significa que a pessoa nessa linha tem um gato e seu nome será impresso junto com a informação de quantos animais dessa espécie ela tem.

Verifique o programa e veja se você pode seguir o que o computador está fazendo a cada passo. A opção 4 permite

entrar o nome de uma pessoa para ver a lista de seus animais.

O programa funcionará de modo quase igual ao da última rotina. Desta vez, entretanto, o computador procurará por uma linha particular da matriz, e não por uma coluna, como acontecia no exemplo anterior.

```
799 REM ** OPCAO 4 **
800 PRINT ''"INTRODUZA O NOME
DA CRIANCA"
805 DIM f$(5): INPUT  LINE f$
810 PRINT "ANIMAIS PERTENCENTE
S A ";f$
815 PRINT ''
820 FOR j=1 TO r
830 IF c$(j)=f$ THEN  LET fo=j
840 NEXT j
```

Que problemas podem ocorrer com laços múltiplos?

Definir uma matriz permite exercitar seus conhecimentos sobre os comandos **FOR...NEXT** e **IF...THEN** (tente contar quantos desses comandos existem nos programas a| resentados aqui; você logo vai perceber que eles são muitos). Porém, a parte mais difícil da programação de matrizes diz respeito ao uso de laços **FOR...NEXT** múltiplos, ou ''aninhados'' (isto é, um dentro do outro). Com efeito, é nessa parte que os iniciantes, e mesmo os programadores experientes, mais se confundem.

Examine as linhas dos programas do artigo. Elas são responsáveis pela leitura dos dados armazenados nas linhas **DATA**.

Cada linha de **DATA** corresponde a uma linha da matriz de dados, de modo que o **FOR...NEXT** que controla a leitura das linhas deve vir antes (laço mais externo). Se a linha de **DATA** contivesse os elementos de uma coluna da matriz, teríamos que usar como **FOR...NEXT** externo aquele que lê as colunas.

É por isso que o conjunto é dimensionado como **N (L,C)**, em vez de **N (C, L)**. A ordem dos índices de linha e coluna no comando **DIM** não é muito importante, desde que você os faça corresponder aos laços **FOR...NEXT**.

Os cabeçalhos para as linhas e colunas devem ser definidos separadamente, para facilitar a compreensão.

```
850 IF fo=0 THEN  PRINT ''"NOM
E NAO ENCONTRADO.TENTE OUTRA V
EZ": GOTO 800
860 FOR j=1 TO c
870 IF n(fo,j)>0 THEN  PRINT n
(fo,j);" ";p$(j)
875 NEXT j
880 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
885 PAUSE 0
890 RETURN
```

```
799 REM ### OPCAO 4 ###
800 PRINTTAB(2);"DIGITE O NOME
DE UMA CRIANÇA"
805 PRINT"LISTA DE TODOS OS ANI
MAIS QUE PERTENCEM A ";
810 INPUT F$
815 PRINT:PRINT
820 FORJ=1TOR
830 IFCH$(J)=F$THENFO=J
840 NEXT
850 IF FO=0THENPRINT:PRINT"NOME
 NÃO ACHADO. TENTE NOVAMENTE!":
GOTO800
860 FORJ=1TOC
870 IFN(FO,J)>0THENPRINTN(FO,J)
;" ";PET$(J)
875 NEXT
880 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR";
885 IFINKEY$=""THEN885
890 RETURN
```

```
799   REM  *** OPCAO 4 ***
800   PRINT  TAB( 5);"DIGITE O N
OME DE UMA CRIANCA"
805   PRINT "LISTA DE TODOS OS A
NIMAIS QUE PERTENCEM A ";
810   INPUT F$
815   PRINT : PRINT
820   FOR J = 1 TO R
830   IF CH$(J) = F$ THEN FO = J

840   NEXT
850   IF FO = 0 THEN  PRINT : PR
INT "NOME NAO ENCONTRADO. TENTE
 NOVAMENTE!": GOTO 800
860   FOR J = 1 TO C
870   IF N(FO,J) > 0 THEN  PRINT
 N(FO,J);" ";PET$(J)
875   NEXT
880   PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
885   GET A$
890   RETURN
```

A opção 5 pede que você entre um número e imprime a lista de pessoas que têm no mínimo esse número de animais. Ela também informa quantos animais cada pessoa tem.

Digite então a próxima sub-rotina da seguinte forma:

```
899 REM ** OPCAO 5 **
900 PRINT '" DIGITE UM NUMERO
PARA LISTAR    TODOS OS QUE TE
M AO MENOS ESTE  NUMERO DE ANI
MAIS"
910 INPUT a
915 PRINT ''
920 FOR j=1 TO r
930 LET p(j)=p(j)+n(j,k)
935 NEXT k
940 LET p(j)=0
945 LET p(j)=0
950 NEXT j
955 IF fo=0 THEN  PRINT ''"NIN
GUEM TEM TANTOS ANIMAIS !"
960 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
970 PAUSE 0
980 RETURN
```

```
899 REM ### OPCAO 5 ###
900 PRINT"DIGITE UM NÚMERO PARA
 LISTAR TODOS QUE TEM AO MENOS
ESTE NÚMERO DE ANIMAIS";
910 INPUTA
915 PRINT:PRINT
920 FORJ=1TOR
925 FORK=1TOC
930 PT(J)=PT(J)+N(J,K)
935 NEXTK
940 IFPT(J)>=ATHENPRINTCH$(J);"
 ";PT(J):FO=1
945 PT(J)=0
950 NEXTJ
955 IFFO=0THENPRINT:PRINT"NINGU
ÉM TEM TANTOS ANIMAIS!"
960 PRINT:PRINT"PRESSIONE QUALQ
UER TECLA PARA RETORNAR"
970 IF INKEY$=""THEN970
980 RETURN
```

```
899   REM  *** OPCAO 5 ***
900   PRINT "DIGITE UM NUMERO PA
RA LISTAR TODOS QUE TEM AO MENO
S ESTE NUMERO DE ANIMAIS";
910   INPUT A
915   PRINT : PRINT
920   FOR J = 1 TO R
925   FOR K = 1 TO C
930 PT(J) = PT(J) + N(J,K)
935   NEXT K
940   IF A <  = PT(J) THEN  PRIN
T CH$(J);" ";PT(J):FO = 1
945 PT(J) = 0
950   NEXT
955   IF FO = 0 THEN  PRINT : PR
INT "NINGUEM TEM TANTOS ANIMAIS
!"
960   PRINT : PRINT "PRESSIONE Q
UALQUER TECLA PARA RETORNAR";
970   GET A$
980   RETURN
```

Antes de comparar o número que você imprimiu com os totais para cada criança, o computador precisa calcular esses totais. As linhas 920 e 925 definem os laços que percorrem as linhas e colunas da matriz, enquanto a linha 930 cal-

cula o total de animais em cada linha. Os totais são armazenados em um conjunto chamado **PT( )**, ou **p( )**, no Spectrum. Se o total de cada linha for maior ou igual ao número que você entrou, o nome da criança será exibido na tela, juntamente com o número de animais que ela possui.

Assim que o nome de alguém for impresso, o indicador **FOUND** (ou seja, achado, em inglês) será igualado a 1. Se **FOUND** ainda for igual a 0 quando o computador atingir a linha 955, isso significa que nenhuma criança tem aquele número de animais que você solicitou, e o programa lhe dará essa informação por meio de mensagem.

A última opção — de número 6 — mostra a matriz na tela.

Como é difícil encaixar todos os nomes dos animais de estimação no topo da tabela, o programa imprime números de referência, em vez de palavras. Em compensação, ele imprime os nomes dos proprietários.

```
 999 REM ** OPCAO 6 **
1000 PRINT '"CRIANCAS","ANIMAIS
"
1010 PRINT 'TAB 9;
1015 FOR j=1 TO c
1020 PRINT j;"  ";
1025 NEXT j: PRINT
1030 FOR j=1 TO r
1035 PRINT 'c$(j);TAB 9;
1040 FOR k=1 TO c
1050 PRINT n(j,k);"  ";
1060 NEXT k
1065 NEXT j
1070 PRINT ''" PRESSIONE QUALQU
ER TECLA PARA   RETORNAR"
1080 PAUSE 0
1090 RETURN
```

```
999 REM### OPCAO 6 ###
1000 PRINT @32,"CRIANCAS","ANIM
AIS"
1010 PRINT @72,;
1015 FOR J=1 TO C
1020 PRINT J
1025 NEXT:PRINT
1030 FOR J=1 TO R
1035 PRINT:PRINT CH$(J) TAB(8);
1040 FOR K=1 TO C
1050 PRINT N(J,K);
1060 NEXT K,J
1070 PRINT:PRINT:PRINT " PRESSI
ONE QUALQUER TECLA PARA   RETOR
NAR"
1080 IF INKEY$="" THEN 1080
1090 RETURN
```

```
999  REM  *** OPCAO 6 ***
1000  PRINT "CRIANCAS","ANIMAIS
"
1010  HTAB 15
1015  FOR J = 1 TO C
1020  PRINT J;" ";
1025  NEXT : PRINT
1030  FOR J = 1 TO R
1035  PRINT : PRINT CH$(J); TAB
( 15);
1040  FOR K = 1 TO C
1050  PRINT N(J,K);" ";
1060  NEXT : NEXT
1070  PRINT : PRINT : PRINT "PR
ESSIONE QUALQUER TECLA PARA RET
ORNAR";
1080  GET A$
1090  RETURN
```

```
999 REM ### OPCAO 6 ###
1000 PRINT"CRIANÇAS","ANIMAIS"
1010 LOCATE 15
1015 FORJ=1TOC
1020 PRINTJ;
1025 NEXT:PRINT
1030 FORJ=1TOR
1035 PRINT:PRINTCH$(J);TAB(15);
1040 FORK=1TOC
1050 PRINTN(J,K);
1060 NEXT:NEXT
1070 PRINT:PRINT:PRINT"PRESSION
E QUALQUER TECLA PARA RETORNAR"
;
1080 IFINKEY$=""THEN1080
1090 RETURN
```

USOS PARA AS MATRIZES

Uma matriz usada dessa forma é chamada de base de dados. É possível criar bases de dados para os mais diversos fins: para um levantamento da vida animal, por exemplo. Numa pesquisa desse gênero, a base de dados mostraria o número de certos animais, assim como o tipo de habitat onde eles são encontrados. As linhas da matriz indicariam os tipos de animais, e as colunas, os habitats. Em seguida, os dados poderiam ser examinados para verificar a distribuição dos animais em determinadas regiões ou épocas do ano.

A base de dados pode ser aplicada também em levantamentos de tráfego em estradas: para verificar, por exemplo, que tipos de carros passam por cada seção de uma rodovia. Ela se presta ainda para aplicação no campo da nutrição. Assim, a matriz conteria os dados de cada tipo de alimento, com relação ao seu conteúdo de proteínas, gorduras, carboidratos e calorias. Com uma matriz desse tipo, seria fácil imprimir todos os alimentos que apresentassem uma certa quantidade de proteínas, ou tantas calorias, ou que oferecessem certas combinações de elementos.

MICRO DICAS

COMO USAR A DECLARAÇÃO *REM*

Não basta que o programa em BASIC esteja correto ou passe por todos os testes: é preciso ainda que ele esteja *bem documentado* e estruturado de forma lógica e clara.

Para evitar confusões por parte do usuário e mesmo do programador na compreensão do que foi digitado, costuma-se utilizar um mecanismo conhecido como *documentação interna* do programa. Este identifica, por meio de títulos e comentários colocados na listagem, as diversas partes do programa. Isso é feito pelo comando **REM**, que é uma abreviatura da palavra *REMark* (*comentário*, em inglês). O **REM** não é uma declaração executável (ou seja, ao encontrá-la em um programa, o computador simplesmente a ignora e passa adiante).

Uma das funções do **REM** consiste em identificar o programa, prestando informações (chamadas *cabeçalho*) como: para que serve, para que máquina ele está destinado, quem o escreveu, quando etc. Eis um exemplo:

```
10 REM ----------------------
15 REM PROGRAMA CREPE
25 REM Simular o jogo de crepe
30 REM VERSAO:1 DATA:20/02/1986
35 REM NOME P/ GRAVACAO: CREP01
40 REM
45 REM COMPUTADOR: TK-90X
50 REM MEMORIA NECESSARIA:16 K
55 REM PROGRAMADO POR: RENATO
       SABBATINI
60 REM ----------------------
```

A colocação de cabeçalhos como esse evita muitos dissabores.

Outra função do **REM** é informar para que servem certas partes do programa. Por exemplo:

```
350 REM ---------------------
355 REM TESTA SE NUMERO DE NA
       VES=ZERO
360 REM SE E', TERMINA O PRO-
       GRAMA
365 REM ---------------------
370 IF NAVES=0 THEN GOTO 999
```

O ideal seria colocar um comentário para cada linha do programa. Isso pode ser feito facilmente nos micros que aceitem comandos múltiplos por linha:

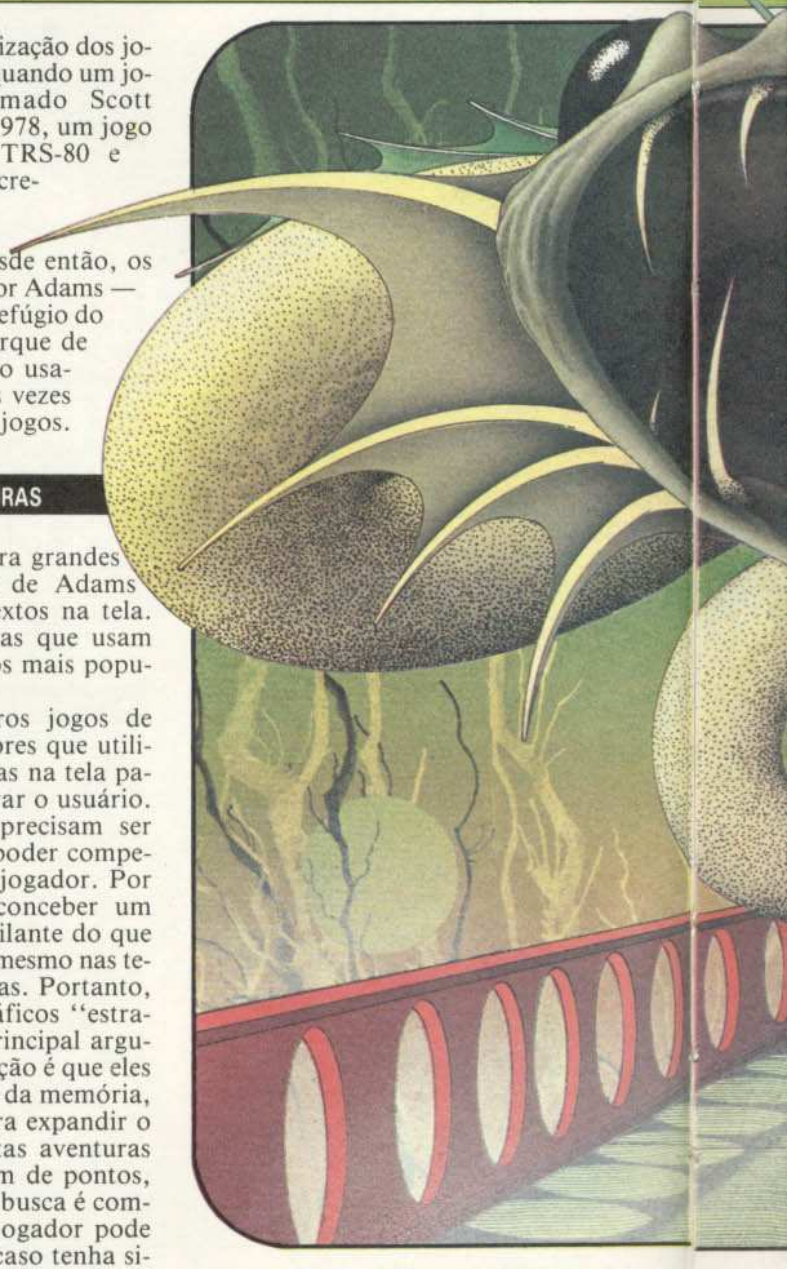
```
500 GOSUB 890 :REM SUBR.
CALCULO
```

O emprego de comentários é simplificado, em vários micros, pelo apóstrofo ('), usado como abreviatura do comando **REM**:

```
10 ' Isto é um comentário
```

COMO PLANEJAR UMA AVENTURA

Sair da rotina e fazer da vida uma aventura errante: quantos de nós já não sonhamos com isso? Veja como fazer essa mudança, vivendo aventuras incríveis... no computador.



Os jogos de aventura são uma boa alternativa para quem está cansado dos arcaicos videogames e fliperamas. Neles, o jogador é totalmente envolvido por um mundo fantástico, criado unicamente pela imaginação do programador.

A ORIGEM DOS JOGOS DE AVENTURAS

A idéia de criar jogos de aventura surgiu da popularidade, nos EUA e na Grã-Bretanha, dos jogos de ação não-computadorizados, tais como *Masmorras e Dragões* e, ao mesmo tempo, do desejo de se utilizar os computadores para fazer coisas mais interessantes do que processar dados.

Por outro lado, os jogos não-computadorizados não atendiam plenamente à demanda do público, que exigia emoções cada vez mais fortes. Essa demanda só foi satisfeita pelos jogos de aventuras programados para o computador.

Em *Masmorras e Dragões* os jogadores assumem o papel de certos personagens, que lutam encarniçadamente (na imaginação) num lugar conhecido como *Masmorra*. Esse lugar, por sua vez, é criado por outro jogador, que faz o papel do *Senhor da Masmorra*. Nos jogos de aventura, o programador vive um personagem parecido com este, podendo inventar, assim, seu próprio mundo.

Ao contrário de outros jogos, em *Masmorras e Dragões* os jogadores não podem escolher as características de seus personagens; isso depende inteiramente do jogo. Em algumas versões mais sofisticadas, os jogadores podem selecionar o equipamento e os mantimentos que os personagens vão usar, desde que isso seja feito no início do jogo, antes de começar a aventura. O primeiro jogo de aventura destinado a computadores foi escrito para máquinas de grande porte, em linguagem FORTRAN, e não em BASIC. O programa ocupou aproximadamente 300k de memória, ou seja, bem mais do que a memória RAM total da maioria dos micros pessoais. Apesar do seu tamanho e complexidade, esse jogo conquistou milhares de usuários de computadores nos Estados Unidos e no Canadá; eles passaram a dedicar suas noites a tentar decifrar os mistérios do jogo.

Na realidade, a popularização dos jogos de aventura começou quando um jovem programador chamado Scott Adams desenvolveu, em 1978, um jogo para microcomputador TRS-80 e provou que era possível escrever uma aventura razoável em um espaço de memória bem menor. Desde então, os temas de jogos adotados por Adams — A Terra da Aventura, O Refúgio do Pirata, O Mistério do Parque de Diversões etc. — têm sido usados e recusados inúmeras vezes pelos escritores de outros jogos.

TIPOS DE AVENTURAS

Como as aventuras para grandes computadores, os jogos de Adams apresentavam somente textos na tela. Na verdade, os programas que usam apenas textos são ainda os mais populares.

Existem ainda inúmeros jogos de aventura para computadores que utilizam ilustrações desenhadas na tela para situar, orientar e motivar o usuário. Entretanto, os gráficos precisam ser muito sofisticados, para poder competir com a imaginação do jogador. Por exemplo, você poderia conceber um monstro bem mais horripilante do que seria capaz de reproduzir, mesmo nas telas gráficas mais avançadas. Portanto, é bem possível que os gráficos "estraguem" sua diversão: o principal argumento contra a sua utilização é que eles desperdiçam grande parte da memória, que poderia ser usada para expandir o campo da aventura. Certas aventuras apresentam uma contagem de pontos, sempre que uma etapa da busca é completada. Desse modo, o jogador pode avaliar seu desempenho, caso tenha sido derrotado em algum estágio. Alguns programas chegam ao ponto de classificar esse desempenho com conceitos, que podem ir de "palhaço" a "perito".

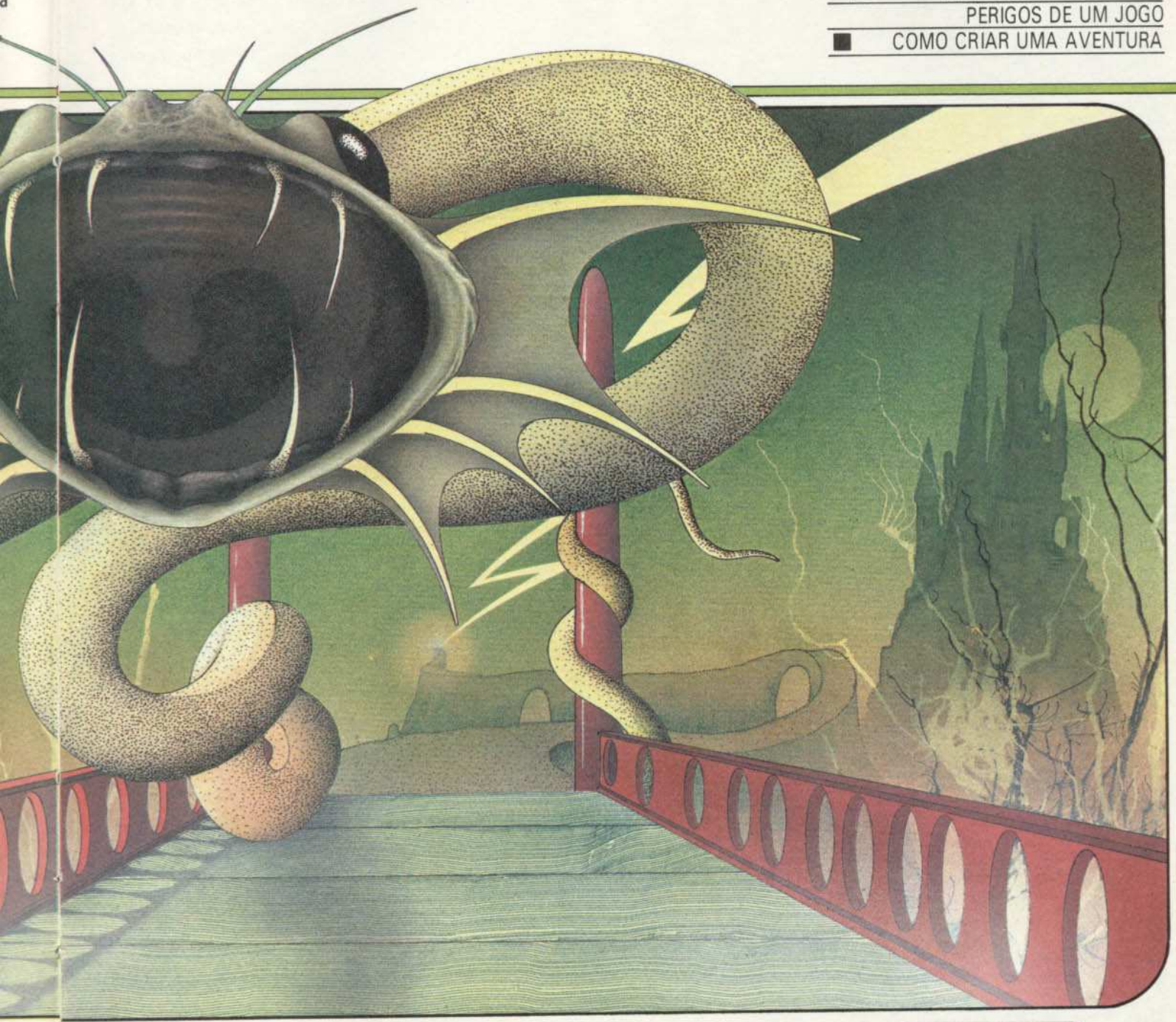
Outros jogos não fornecem qualquer dica de como o jogador está se saindo ou da proximidade do desfecho.

Seja qual for o caso, porém, a diversão maior está na solução de uma série aparentemente infinita de charadas, depois da qual a aventura chega ao fim. Alguns costumam durar muitas horas, ou dias, podendo ser interrompidos e retomados a qualquer momento.

UMA HISTÓRIA SEM FIM

Normalmente, quando se inicia uma aventura através do comando **RUN**, são

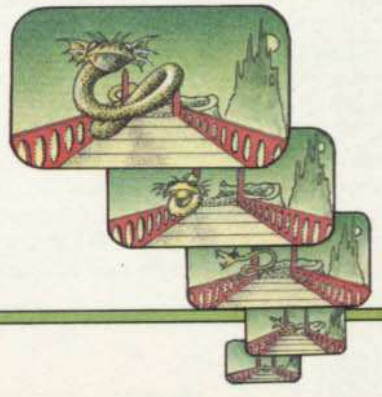
- JOGOS DE AVENTURA
- COMO JOGAR
- DICAS PARA VENCER OS PERIGOS DE UM JOGO
- COMO CRIAR UMA AVENTURA



dadas informações sobre o mundo em que se acaba de entrar: este pode ser um lugar exótico na Terra, um planeta distante, ou um local imaginário. O jogo pode acontecer em épocas diversas. Geralmente, são fornecidas informações como: quem governa o lugar; quem é você e como fazer para atingir o objetivo e ganhar o jogo. Depois disso, aparecerá uma primeira descrição do local, provavelmente dizendo algo como:

```
 VOCÊ SE ENCONTRA PRÓXIMO A UM
ENORME CALDEIRÃO CHEIO DE UM
LÍQUIDO VERDE BORBULHANTE. HÁ
UM CHEIRO DEMONÍACO NO AR. UMA
COLHER ESTÁ NO CHÃO.
VOCÊ PODE IR PARA O LESTE
                    OESTE
                    NORTE

E AGORA?
```



Agora, você tem que decidir o que fazer. Será conveniente usar a colher para misturar o líquido ou mesmo experimentá-lo? Não seria melhor deixar isso de lado? Ou talvez fosse preferível procurar um recipiente para levar um pouco do líquido verde com você?

Se decidir usar a colher, digite algo assim: **PEGUE COLHER**; a resposta do computador será: OK ou VOCÊ AINDA NÃO PODE PEGAR A COLHER!, ou alguma outra mensagem.

Em cada etapa do jogo você deve informar ao computador exatamente o que pretende fazer. A forma de fazer isso dependerá apenas do tipo de jogo. A maioria deles espera você digitar seus comandos no computador através de um verbo seguido de substantivo; por exemplo: **PEGUE COLHER**.

Jogos mais sofisticados aceitam sentenças completas, mas isso é uma exceção à regra. Esses tipos de jogos permitem-lhe dizer algo como: **MATE O INSETO PISANDO NELE ENQUANTO CANTA "O TICO-TICO NO FUBÁ"**.

A maioria dos jogos de aventura consegue entender abreviaturas das palavras válidas. Por exemplo, é muito comum, numa aventura, digitar **N** em vez de **NORTE**. As direções podem ser pontos cardeais — **N**, **S**, **L** e **O** — ou colaterais — **SE**, **SO**, **NE** e **NO** —, ou instruções como **PARA CIMA**, **PARA BAIXO**.

Um jogo típico de aventura é baseado numa grade de locais possíveis, que assume geralmente a forma de um quadrado. Dependendo da imaginação do programador, esses lugares podem representar ambientes tão variados como os quartos de um castelo, os subterrâneos de uma mina etc.

DICAS PARA VENCER

Habitualmente, existe apenas uma solução para a aventura: por exemplo, encontrar um pote de ouro e levá-lo para o fim do arco-íris, ou matar um guerreiro medieval e escapar ileso etc. Até chegar a esse objetivo, o jogador precisa resolver toda uma série de problemas. O mais provável é que ele tenha que fazer muitas tentativas diferentes até concluir a aventura.

Existem algumas regras básicas e dicas que o ajudarão a resolver mais rapidamente a maioria dos jogos. Quase todos os objetos que você encontrar nas aventuras serão de alguma utilidade. A existência de muitas pistas completamente falsas representa um desperdício de memória, embora seja necessário es-

tar sempre atento para alguns objetos que podem ser "facas de dois gumes". Por exemplo, o jogador poderia estar carregando uma sacola com moedas de ouro para passar pelo pedágio de uma ponte; caso ele se decidisse a nadar no rio, porém, o peso delas o faria afundar. Como regra geral, é aconselhável tentar carregar o maior número possível de objetos, mas algumas vezes apenas uma parte deles lhe será realmente útil.

Quase sempre, a maioria dos objetos é usada somente uma vez durante uma aventura. Mas há exceções: uma espada, por exemplo, poderia ser usada muitas vezes para derrubar gigantes, dragões ou bandidos. Dessa forma, se você for obrigado a limitar o número de objetos que leva consigo, deve ter em mente que é mais seguro descartar os que já foram usados uma vez.

Outra regra geral: desenhe *sempre* um mapa, marcando nele os nomes dos aposentos, a posição dos sentinelas e dos objetos distribuídos pelos quartos, *todas* as entradas e saídas com suas direções e outros pontos de referência que lhe possam ser úteis.

O mapa o fará economizar tempo e esforço quando você estiver retornando para algum ponto anterior — manobra que você repetirá várias vezes durante o jogo. Se tiver que abandonar algum objeto, por não poder carregar tudo, não se esqueça de marcar a posição dele no mapa. A importância do mapa está em que ele lhe permitirá ter certeza de que explorou todas as possibilidades da aventura — o que, em muitos casos, salvará sua "vida".

Certos jogos permitem que você peça um inventário do que está carregando. Assim, quando estiver diante de uma charada, procure saber exatamente o que tem em mãos, digitando **INVENTÁRIO**, **LISTA**, ou algo semelhante, dependendo da aventura.

Há jogos, ainda, que dão ao jogador a possibilidade de pedir ajuda. A forma de fazê-lo varia, mais uma vez, conforme o conteúdo da aventura. Freqüentemente, porém, você obterá algo assim, em resposta ao seu pedido:
NENHUMA AJUDA POSSÍVEL.

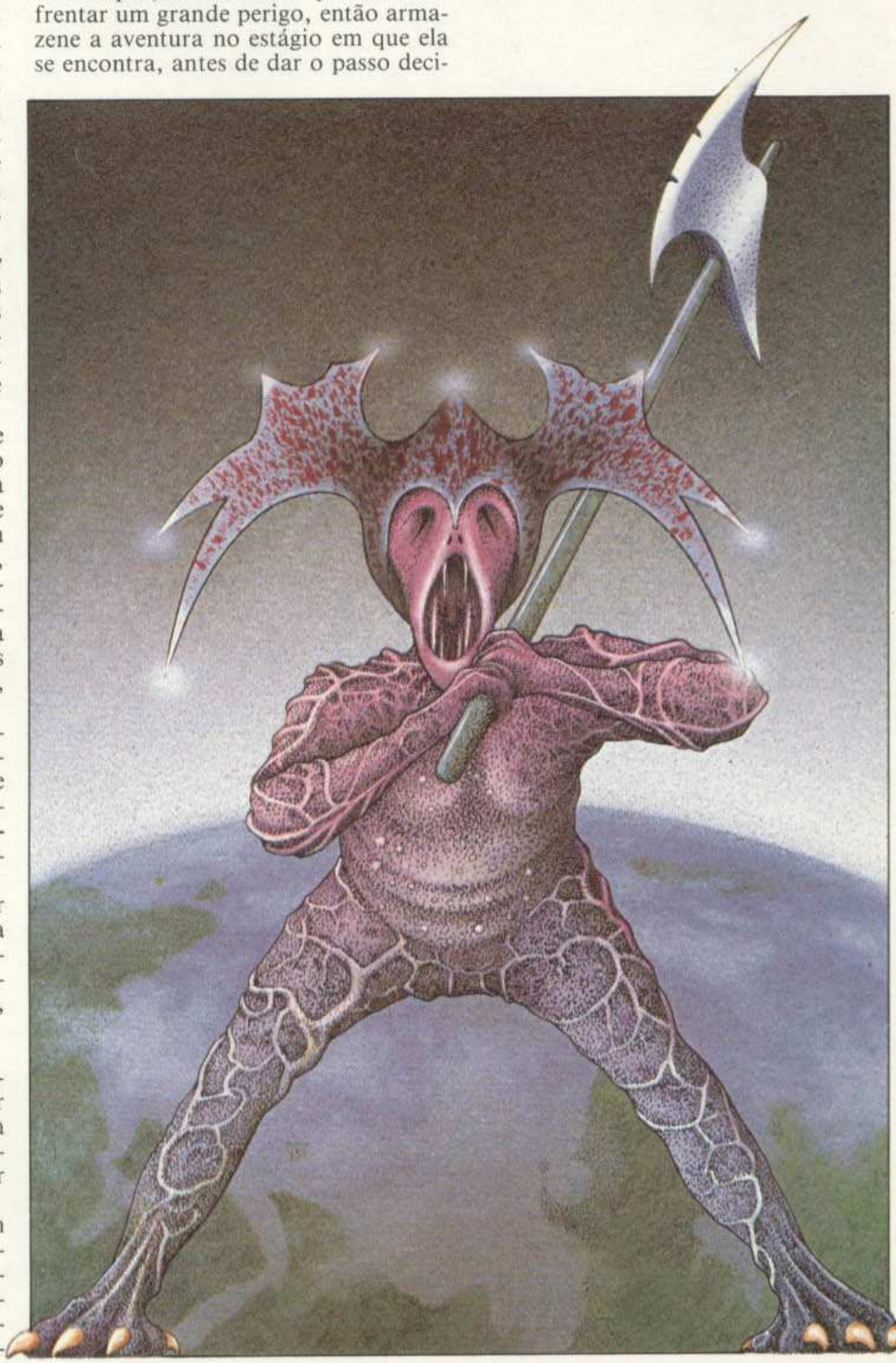
Alguns jogos seguem cuidadosamente o enredo de um livro específico (por exemplo, *A Ilha do Tesouro*, escrito em 1833 por Robert Louis Stevenson). Nestes casos, é altamente recomendável ler o livro em questão.

Outros jogos inspiram-se apenas em alguns capítulos ou idéias de um romance famoso. Assim, se você encontrar numa aventura algum elemento que o faça recordar um texto literário e tiver dificuldades para resolver um determinado problema, tente encontrar a resposta consultando o livro. Um dicionário de sinônimos pode ser de grande utilidade, permitindo que o jogador explore todas as variações possíveis de uma determinada frase. Por exemplo: o programador pode ter usado no jogo a palavra POLIR, em vez de LUSTRAR.

Uma última dica: se o jogo permitir o uso do comando **GRAVAR** em algumas etapas, e você estiver prestes a enfrentar um grande perigo, então armazene a aventura no estágio em que ela se encontra, antes de dar o passo decisivo, pois, se você for "morto", poderá voltar atrás e continuar a partir do ponto em que parou.

CRIE AS SUAS PRÓPRIAS AVENTURAS

A criação de aventuras é uma boa maneira de dominar a linguagem BASIC. Quase todos os aspectos importan-



tes dessa linguagem são utilizados nesse tipo de jogo: variáveis, matrizes, formatos de telas, cadeias alfanuméricas, comandos **IF**, e assim por diante.

Muitos jogos de aventuras à venda ainda são escritos em BASIC, pois não há muita necessidade de velocidade de execução do programa.

Antes de programar um jogo desse gênero, você precisa ter uma idéia bem clara do que pretende fazer e do enredo da aventura. A seguir, planeje cuidadosamente a trama com suas charadas, perigos, objetos etc.

Para começar, pegue uma folha de papel e faça um rascunho de algumas idéias. Não se preocupe se ainda não tem uma visão completa da ordem do jogo: o que você precisa inicialmente é de uma noção do roteiro, do local da aventura e de algumas charadas para o jogador resolver.

Quanto espaço de memória necessito para escrever um jogo de aventura?

Nas próximas lições de INPUT desenvolveremos uma aventura simples, ocupando cerca de 5K de BASIC. Como todo primeiro passo, esse jogo não trará episódios de grande emoção. Seu objetivo é apenas introduzi-lo ao mundo fascinante da aventura.

Os melhores jogos têm um grande número de aposentos e charadas e são muito difíceis, devido ao efeito crescente de problemas cada vez mais complicados. Com uma quantidade limitada de memória, você será forçado a determinar um número menor de problemas, que por sua vez serão mais difíceis do que o normal. Isso, contudo, não é muito satisfatório.

Um jogo como esse exige um computador com, no mínimo, 16K de memória; mas sugerimos uma máquina com 48 ou 64K para jogos realmente desafiantes.

O que devo fazer se cair numa armadilha durante a aventura?

Não desista; tente novamente quando tiver uma nova idéia ou estratégia.

Se você realmente empacar, os jogos de aventuras comercializados oferecem um folheto com dicas e respostas. Muitas vezes, é preciso solicitar pelo correio um mapa da solução.

Outro recurso para ajudá-lo são as revistas do gênero, que costumam apontar a solução completa de aventuras complexas.

Além de livros, você pode inspirar-se em filmes, programas de TV e peças teatrais, ou mesmo inventar novas histórias. Pode também buscar idéias em outras aventuras. Mas o melhor recurso para sua inspiração é fazer trabalhar a imaginação.

Ao mesmo tempo, procure alcançar um certo equilíbrio entre o desafio e as condições para vencê-lo. Não é conveniente criar uma aventura que qualquer um poderia resolver no espaço de meia hora. Em contrapartida, seu jogo não deve apresentar problemas insolúveis. A regra básica é: "dê aos jogadores uma chance"... mas não exagere.

Tente não deixar muitos aposentos ou locais vazios. Além de ocupar muito espaço de memória, estes não contribuem para a resolução dos problemas e tornam o enredo cansativo.

Por outro lado, procure evitar que as suas primeiras aventuras sejam excessivamente complexas, para que os problemas que aparecem ao longo da história possam ser resolvidos por quem ainda não tem prática; familiarize-se com o tema antes de aventurar-se em qualquer enredo extravagante; mantenha-se alerta quanto ao espaço restante de memória, pois programas como esses tendem a se tornar gigantescos.

No jogo de aventura que será elaborado nas próximas partes do curso de *Programação de jogos*, encontraremos muitas declarações **REM** (comentários, para entender o fluxo do programa). Entretanto, para economizar memória em jogos com longas aventuras, é preferível trabalhar sem tais declarações, mesmo que no começo elas facilitem a criação da trama.

Você poderá também poupar espaço de memória na descrição dos aposentos. Descrições muito curtas, porém, tendem a estragar todo o prazer suscitado pela aventura.

O ESPAÇO RESTANTE DE MEMÓRIA

Ao escrever um longo jogo de aventura, é fácil descobrir quando se está chegando ao limite de memória do computador. Obviamente, os problemas de desperdício de memória tornam-se mais graves no caso de máquinas com memórias menores (na verdade, seria perda de tempo tentar escrever um programa de aventura em um computador que tenha menos do que 16K de RAM).

Digite a seguinte linha, em modo direto:

```
PRINT (PEEK 23730 + 256* PEEK
23731) - (PEEK 23653 + 256*
PEEK 23654)
```

O que foi digitado leva em consideração tanto o espaço ocupado pelo programa como o usado para armazenar as variáveis. A melhor contagem para descobrir a quantidade de memória restante só será possível, portanto, depois que o programa tiver sido rodado.

Para saber quanto espaço está sobrando, subtraia esse valor da quantidade total de memória RAM existente no computador.

Nos microcomputadores das linhas TRS-80 e TRS-Color, o que resta da memória pode ser descoberto digitando-se:

```
PRINT MEM
```

Essa instrução levará em conta o espaço ocupado pelo programa e pelas variáveis. Assim, é melhor rodar antes o programa, durante seu desenvolvimento (se isso for possível), a fim de conseguir uma indicação real do total de memória usado por ele.

Para saber quanto espaço está sobrando, subtraia esse valor da quantidade total de memória RAM existente no computador. Há uma maneira bem fácil de se aumentar o espaço livre de memória no TRS-Color para seus programas em BASIC. Antes de começar a programar, digite esses dois comandos **POKE**:

```
POKE 25,6
POKE 26,1
NEW
```

Isso economizará uns 6K extras para seu programa, pois enganará o computador, colocando o programa em BASIC dentro de um espaço normalmente reservado para gráficos de alta resolução.

Neste caso os comandos **POKE** excluem os comandos para gráficos na sua aventura. Se eles existirem, o seu programa será alterado.

Quando você armazenar o programa em fita ou disco e quiser carregá-lo mais tarde, não se esqueça de usar antes os comandos **POKE** e **NEW**.

Nos microcomputadores das linhas Apple e MSX, o que resta da memória pode ser descoberto digitando-se:

```
PRINT FRE(0)
```

A função **FRE** (abreviatura de **FREe**, ou livre, em inglês) retorna o espaço livre na memória RAM, ou seja, aquele não ocupado pelo programa e pelas variáveis.

PROGRAMAS EM CÓDIGO DE MÁQUINA

- EXERCÍCIO EM ASSEMBLY
- UM PROGRAMA QUE DESLOCA A TELA PARA OS LADOS
- COMO FUNCIONA O PROGRAMA EM DIVERSOS COMPUTADORES

Agora que você já sabe traduzir do Assembly para o sistema hexa, passe a operar com código de máquina.

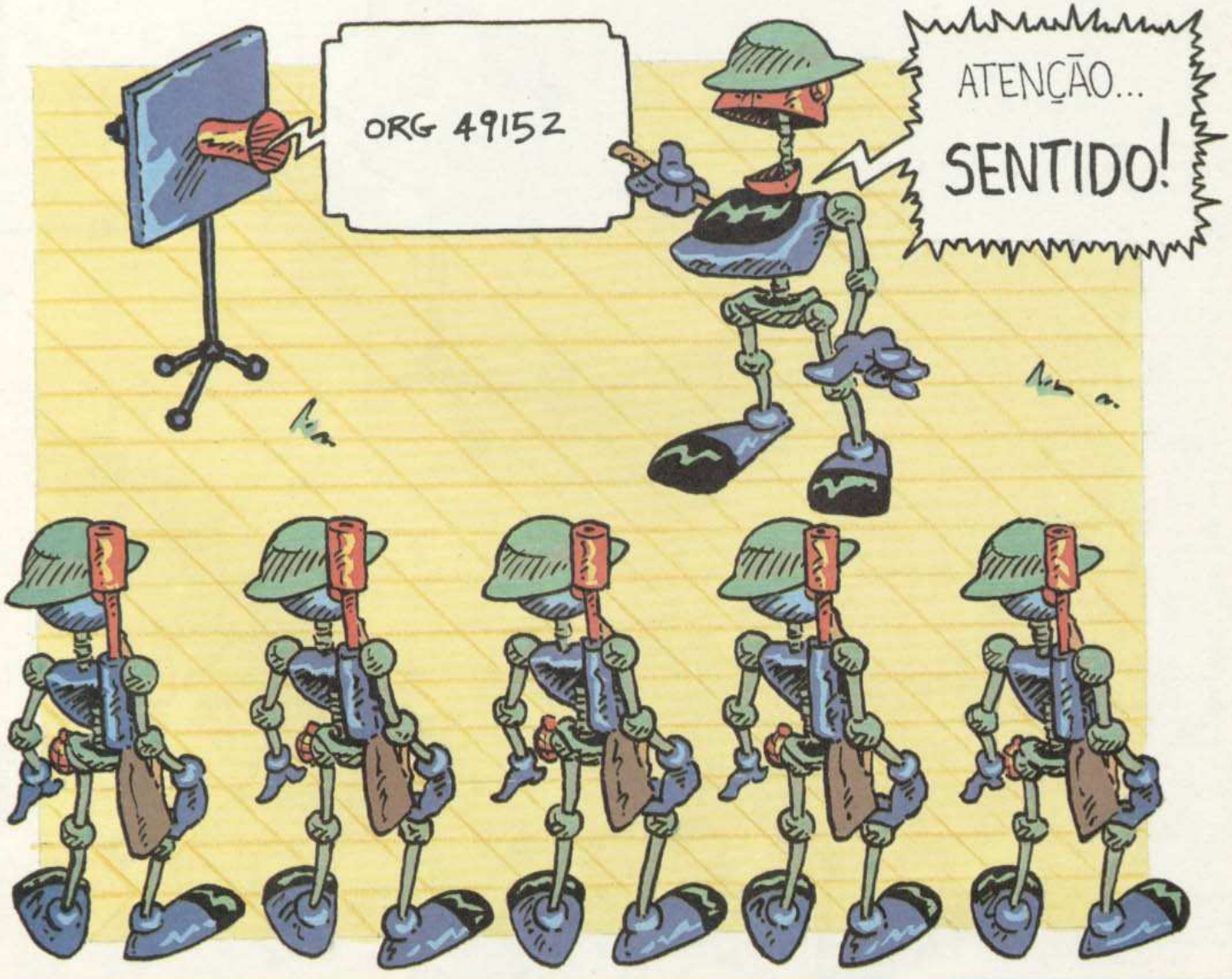
Os programas em Assembly apresentados a seguir deslocam a tela um caractere para a esquerda ou para a direita. Monte-os na memória do micro usando seu programa monitor. Se você quiser gravar programas para uso futuro, coloque-os em locais diferentes da memória.

Os programas funcionam independentemente de suas posições na memória, de forma que você mesmo pode resolver onde é melhor colocá-los. Para deslocar a tela mais de uma posição no sentido horizontal, chame a rotina em código de máquina de dentro de um laço **FOR...NEXT**. Outra opção é deslocar a tela somente quando certa tecla é pressionada, usando **INKEY$** ou **GET$**.

Este programa desloca a tela para a esquerda (já está traduzido).

```
        ld de,16384      11 00 40
        ld hl,16385      21 01 40
        ld b,192            06 C0
loop    push bc                C5
        ld a,(de)              1A
        ld bc,31         01 1F 00
        ldir                ED B0
        ld(de),a               12
        inc de                 13
        inc hl                 23
        pop bc                 C1
        djnz loop           10 F3
        ret                    C9
```

Os comandos **ld de, 16384** e **ld hl, l, 16385** carregam os endereços das duas



primeiras posições da tela do Spectrum nos registros DE e HL. Já **ld b,192** transporta o registro B com o número 192 (há 192 linhas na tela desse micro, e B é usado para contá-las).

Como o registro B será usado para contar outras coisas também, seu conteúdo ficará temporariamente armazenado na pilha (*stack*). Todavia, nenhum comando transfere apenas o conteúdo de B para a pilha: assim, temos que fazer essa transferência com o conteúdo de B e C ao mesmo tempo usando **push bc**.

O acumulador é então carregado com o conteúdo da posição de memória cujo endereço está no registro DE, pelo comando **ld a (DE)**. Este é um exemplo de endereçamento indireto. Lembre-se de que o conteúdo de DE é 16384.

O comando **ld bc, 31** carrega os registros BC com o número 31. Cada linha tem 32 caracteres, mas a transferência do primeiro caractere para a última posição da linha é feita independentemente. Assim, basta contar até 31.

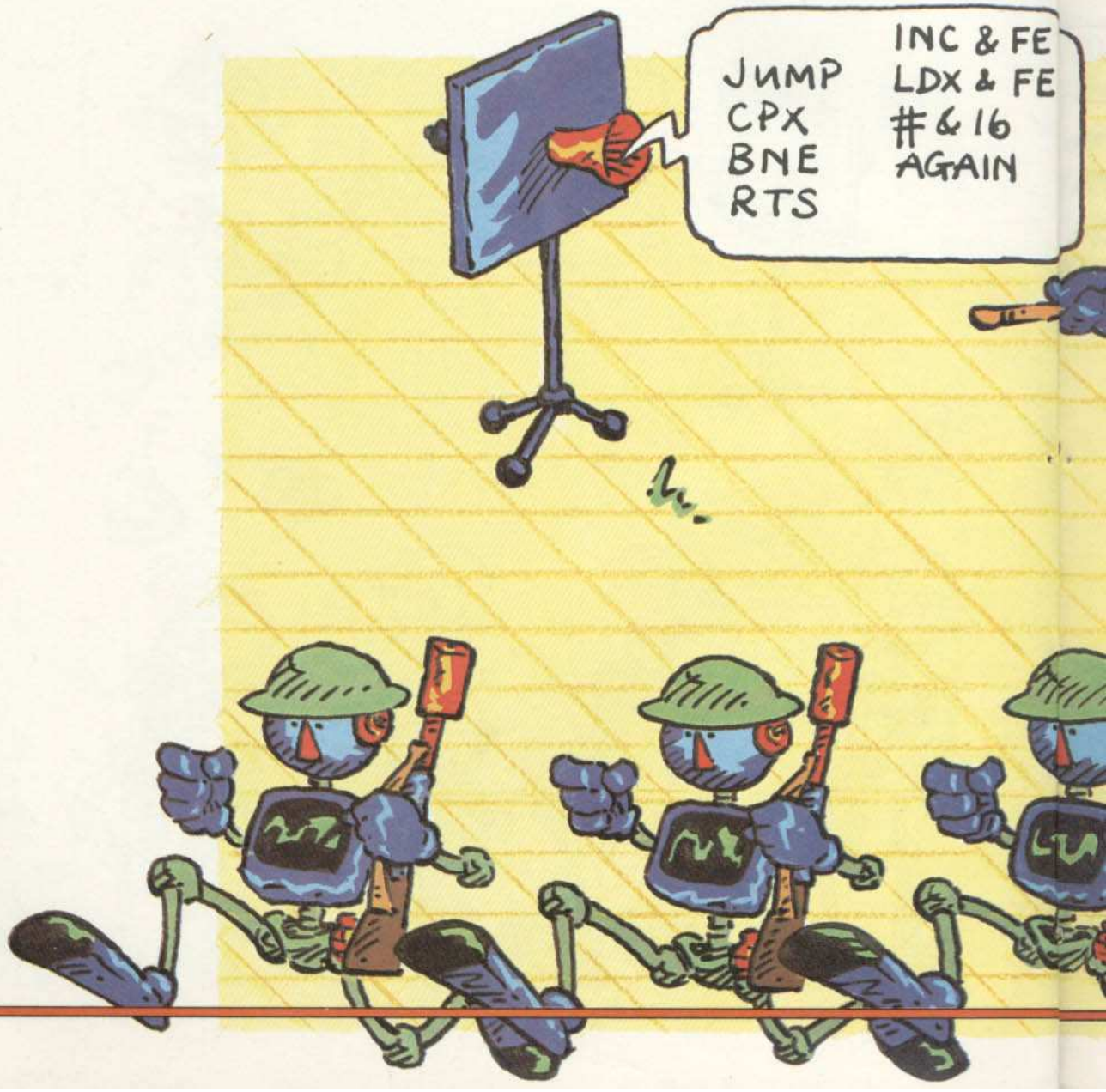
A instrução mais poderosa aqui utilizada é **ldir**, que significa "carregar, somar um e repetir" (=*load, increment and repeat*). Compreender o que ela faz é crucial para o entendimento do programa: o conteúdo da posição de memória cujo endereço está em HL — inicialmente, 16385 — é transferido para a posição cujo endereço está em DE — inicialmente, 16384. A seguir, deve-se somar 1 aos conteúdos de HL e DE, subtrair 1 do conteúdo de BC e verificar se o conteúdo de BC foi reduzido a 0. Enquanto isso não acontecer a instrução será repetida.

O que estava na segunda posição do vídeo passa então para a primeira; o que estava na terceira posição vai para a segunda, e assim até que termine a primeira linha do vídeo, o que acontece quando o registro de BC é reduzido a 0. A última transferência é a da 32ª posição, que passa para a 31ª. Nessa altura, o conteúdo de BC é subtraído de 1 e se torna 0, e o programa passa para a instrução seguinte.

O conteúdo do acumulador é transferido por **ld (de), a** para a posição cujo endereço está em DE. O acumulador contém o código do caractere que ocupava a primeira posição da memória de vídeo: 16384, endereço que estava em DE antes de **ldir** começar a incrementar esse registro.

Esse processo de somar 1 ao conteúdo de DE foi executado sucessivamente desde então, e agora DE contém o endereço da última posição da primeira linha. Este é justamente o local para onde desejamos transferir o caractere que estava no início da linha.

Os comandos **inc hl** e **inc de** somam



1 aos registros HL e DE, de forma que estes passam a conter os endereços das duas primeiras posições da linha seguinte. Como o conteúdo da primeira posição é transferido para o final da linha, independentemente de **ldir**, devemos incrementar o valor desses registros da mesma forma.

O antigo conteúdo de BC é recuperado da pilha pelo comando **pop bc**. O comando **djnz** subtrai 1 ao valor de B e "pula" sucessivamente para outros locais do programa, até que esse valor se iguale a 0 (= *decrement and jump if not zero*). No início do programa, carregamos o registro B com 192. Esse valor fica armazenado temporariamente na pilha para liberar o uso de BC. O comando **djnz** subtrai 1 a esse valor, que passa a ser 191 (valor diferente de 0). Assim, a instrução volta para a primeira ocorrência do rótulo **loop**.

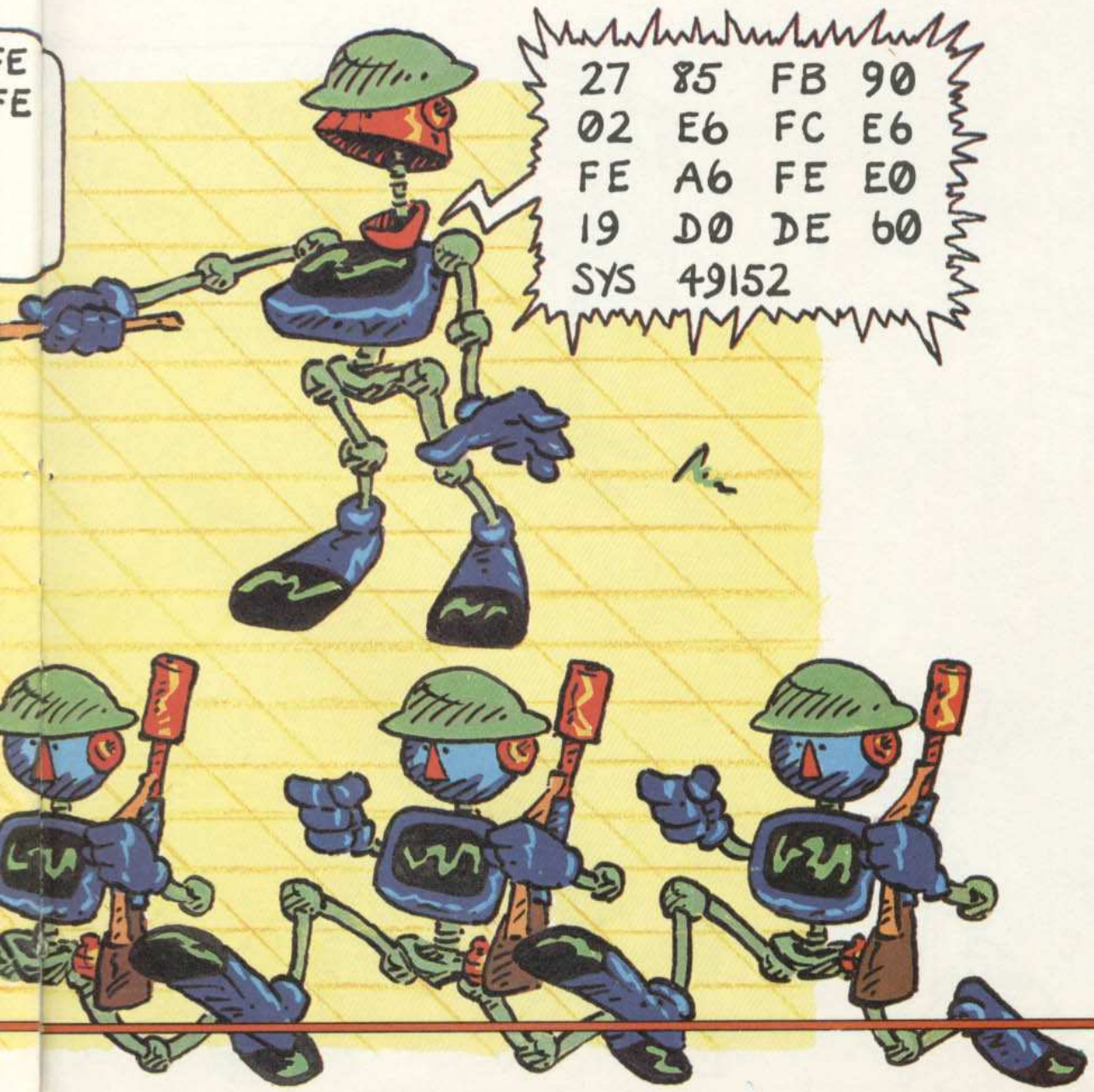
Esse salto continua a ocorrer enquanto o conteúdo de B vai sendo reduzido de 192 até chegar a 0. O fato de esse valor ser transferido temporariamente para a pilha não altera o funcionamento das coisas. Desta forma, uma linha é deslocada de cada vez. Quando B finalmente contém o valor 0, após a última subtração feita por **djnz**, não ocorre salto, e o programa passa a instrução seguinte.

O comando **ret** significa retorne. Nenhuma rotina em código de máquina funcionará direito sem uma instrução desse tipo no final. O microprocessador tentará executar qualquer coisa que esteja após os códigos de seu programa. O conteúdo dessa parte da memória geralmente não faz sentido, e o mais comum é a perda de controle do computador e do conteúdo da memória.

Mesmo quando isso não ocorre, o microprocessador continua lendo e tentando executar tudo o que encontra até o final da memória. Chegando lá, ele retorna ao início da memória onde encontra as rotinas de inicialização do BASIC. Estas são executadas e a memória se apaga.

Tente traduzir para código de máquina e introduzir em seu micro a seguinte rotina Assembly.

```
      ld de,22527
      ld hl,22526
      ld b,192
loop  push bc
      ld a,(de)
      ld bc,31
      lddr
      ld(de),a
      dec hl
      dec de
      pop bc
      djnz loop
      ret
```



Esse programa começa pelo final da memória de vídeo e vai transferindo os conteúdos das linhas "de trás para a frente". A diferença entre ele e o anterior é o uso da instrução **lddr**, que transfere o caractere da posição cujo endereço está em HL para a posição de endereço em DE, subtraindo 1 aos conteúdos de HL, DE e BC e repetindo o processo até que o valor em BC seja reduzido a 0. O comando **lddr** transfere blocos de memória em sentido contrário a **ldir**.

O salto é do mesmo tamanho que no programa anterior. Verifique se calculou seu valor corretamente, olhando a listagem (conte os bytes a partir do final da instrução de salto, ou seja, incluindo o byte cujo valor está sendo calculado).

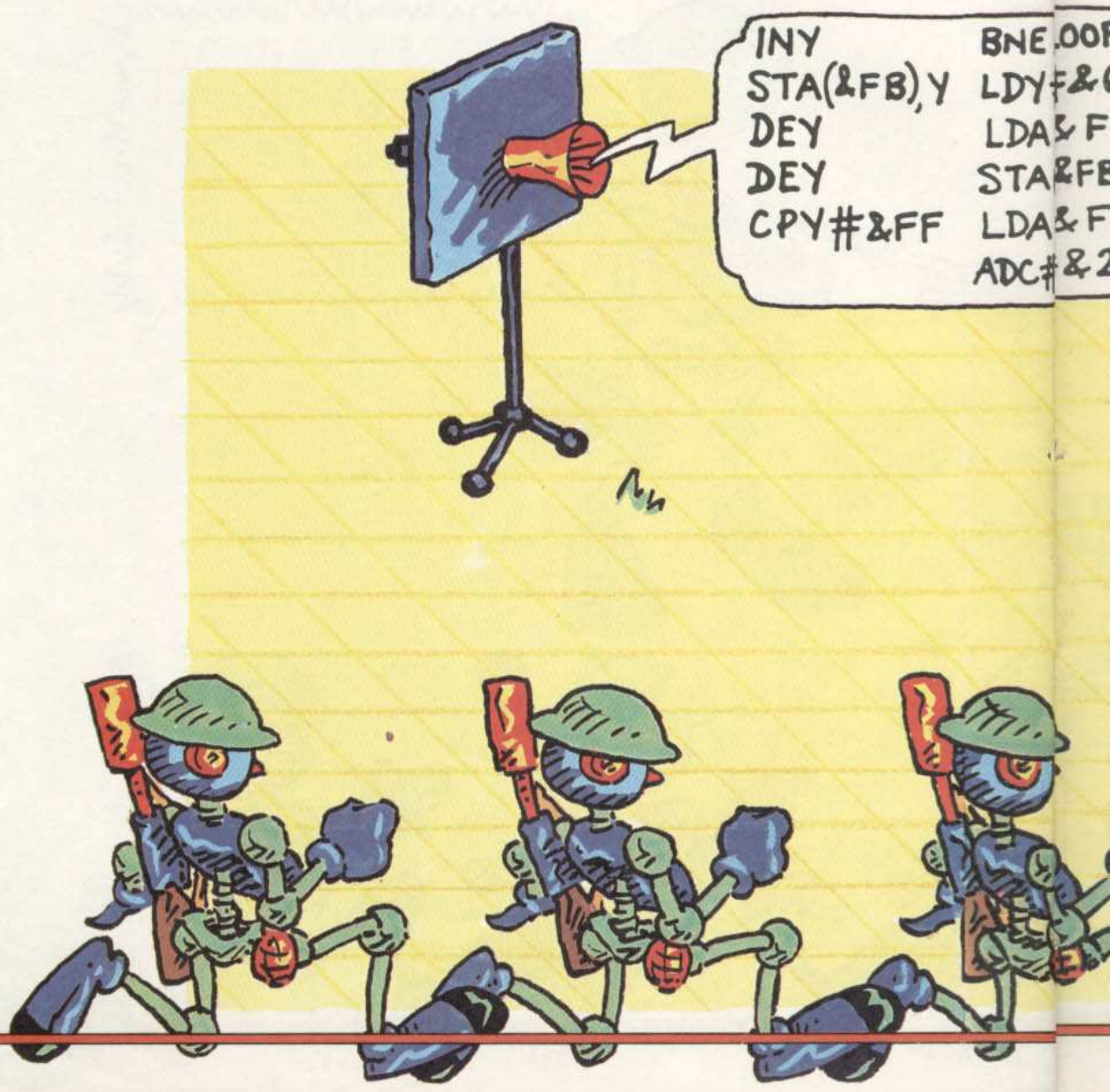
O programa a seguir desloca a tela para a esquerda no ZX-81 com expansão de memória. Ele já está traduzido do Assembly para código de máquina.

```
      LD DE,(16396)   ED 5B 0C 40
      INC DE                    13
      LD H,D                    62
      LD L,E                    6B
      INC H,L                   23
      LD B,24                06 18
LOOP  PUSH BC                   C5
      LD A,(DE)                 1A
      LD BC,31              01 IF 00
      LDIR                    ED B0
      LD(DE),A                  12
      INC HL                    23
      INC HL                    23
      INC DE                    13
      INC DE                    13
      POP BC                    C1
      DJNZ LOOP              10 F1
      RET                       C9
```

O PROGRAMA NO ZX-81

A memória de vídeo do ZX-81 não tem posição fixa; as posições 16396 e 16397 contêm o endereço da primeira posição do vídeo. A instrução **LDDE, (=16396)** carrega os registros D e E com os valores contidos em 16396 e 16397 — o conteúdo de 16396 vai para E, e o outro vai para D, seguindo a convenção "baixo-alto" do Z-80. Agora, o programa sabe onde está a memória de vídeo.

Esse micro conta com um caractere de retorno de carro (NEW LINE) no início de cada linha que não aparece na tela. Sua função é economizar o espaço ocupado pelo vídeo. Se a tela estiver vazia, a memória usada pelo vídeo corresponderá aos 24 caracteres NEW LINE, que marcam onde cada linha deve começar.



Quando é usada uma expansão, esses caracteres ficam no final de cada linha. Não devemos interferir nessa organização.

Tal organização é responsável pelas diferenças entre os programas do ZX-81 e os do Spectrum (leia com atenção a descrição do programa do Spectrum).

O primeiro caractere do vídeo é um NEW LINE. Como não devemos alterá-lo, a instrução **INC DE** soma 1 ao conteúdo de DE, que se torna então o endereço da segunda posição do vídeo.

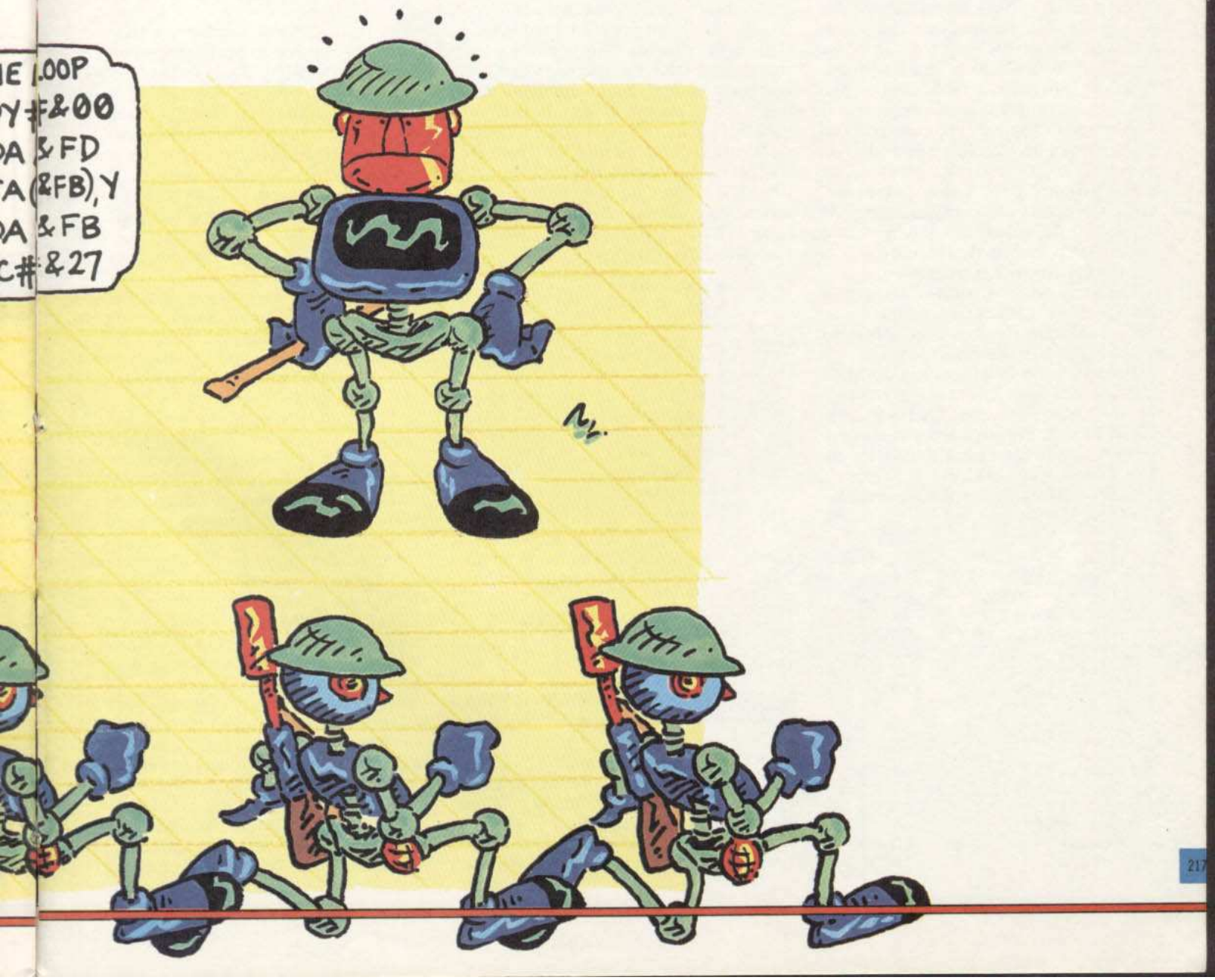
As instruções **LD H,D** e **LD L,E** copiam o conteúdo do par DE no par de registros HL. **INC HL** soma 1 a HL, que passa a conter o endereço da terceira posição do vídeo. **LD B,24** carrega o registro B com o número de linhas da tela do ZX-81. O restante do programa funciona de maneira idêntica à do Spectrum. A única diferença é o número de comandos **INC HL** e **INC DE** que, no ZX-81, somam 1 a esses registros. Isso é feito uma vez para mover o programa para a linha seguinte e uma segunda vez para evitar interferências da linha.

Agora, tente obter os códigos hexadecimais correspondentes ao seguinte programa em Assembly, que desloca a tela para a direita.

```
      LD HL,(16396)
      LD DE,790
      ADD HL,DE
      LD D,H
      LD E,L
      INC DE
      LD B,24
LOOP  PUSH BC
      LD A,(DE)
      LD BC,31
      LDDR
      LD(DE),A
      DEC HL
      DEC HL
      DEC DE
      DEC DE
      POP BC
      DJNZ LOOP
      RET
```

Esse exemplo é parecido com o segundo programa apresentado para o Spectrum. As diferenças entre eles são as mesmas que encontramos antes.

A organização da memória de vídeo do MSX é bem diferente da que existe em outras máquinas. Esse computador utiliza um microprocessador diferente da CPU para controlar a tela, dispondo de 16K que não estão na RAM para utilizar como memória de vídeo. Desta forma, essa porção da memória é tratada como dispositivo periférico, e o acesso a ela não é tão fácil quanto o acesso à RAM.



Isso significa que não podemos usar o comando **ld** para colocar ou ler um caractere na memória de vídeo. O comando Assembly utilizado para enviar dados a um dispositivo periférico é o **out**.

Quando a memória de vídeo de um computador está na RAM, podemos facilmente situar um caractere em qualquer posição vertical ou horizontal. Se a tela for tratada como um dispositivo periférico, contudo, a instrução **out** simplesmente colocará os caracteres na mesma ordem em que forem enviados, a partir da posição do cursor.

Temos, então, dois problemas: para controlar a posição do cursor, necessitamos de sub-rotinas da ROM; além disso, os 16K da memória de vídeo têm uma organização complicada, que varia conforme o tipo de tela selecionada, permitindo a exibição de caracteres, cores e sprites.

Enquanto não aprendemos como funcionam as sub-rotinas da ROM e a organização da memória de vídeo, vamos usar um artifício que dê ao MSX uma "memória de vídeo" na RAM, cujo acesso é bem mais fácil. Isso é feito pelas duas rotinas que se seguem.

A primeira copia os 960 caracteres da tela de textos de 40 colunas, no topo da memória, onde esta é manipulada de maneira semelhante à dos outros micros. A segunda copia os caracteres da "memória de vídeo" da RAM para a verdadeira memória de vídeo do MSX, que é um dispositivo periférico.

Embora todas as rotinas apresentadas funcionem independentemente de sua localização na memória, daremos exemplos utilizando endereços.

Proteja o topo da memória do MSX para colocar os programas e a "memória de vídeo" (=use **CLEAR 200, &HDFFF**). A seguir, carregue os programas usando o monitor. Utilize os endereços iniciais −8192 e −8155 para a primeira e a segunda rotinas, respectivamente.

```
      ld c,152          0E 98
      ld hl,-7936    21 00 E1
      ld a,4            3E 04
      ld b,240          06 F0
loop  inir              ED B2
      dec a               3D
      jrnz,loop         20 F9
      ret                 C9
```

```
      ld c,152          0E 98
      ld hl,-7935    21 01 E1
      ld a,4            3E 04
      ld b,240          06 F0
loop  outir             ED B3
      dec a               3D
      jrnz,loop         20 F9
      ret                 C9
```

Para fazê-las funcionar, digite ainda o seguinte programa em BASIC:

```
1000 SCREEN 0
1005 DEF USR1=-8192
1007 DEF USR3=-8155
1010 FOR I=0 TO 960
1020 VPOKE BASE(0)+I,RND(1)*256
1030 NEXT I
1040 VPOKE BASE(0),32:X=USR1(0)
1050 CLS:FOR I=1 TO 1000:NEXT I
1060 VPOKE BASE(0),32:X=USR3(0)
1080 GOTO 1080
```

Os possíveis erros podem ser corrigidos por meio do monitor. Para possibilitar isso, o programa em BASIC começa na linha 1000, acima do monitor. Se não apagarmos o programa monitor, poderemos rodar o novo programa com **RUN 1000**.

As duas rotinas fornecidas devem ser chamadas de um programa em BASIC. São especialmente importantes os comandos **VPOKE** das linhas 1020 e 1040, que posicionam o cursor no início da tela para que ela possa ser lida pela rotina **USR1** e escrita pela rotina **USR2**.

Agora, você pode montar a rotina de deslocamento horizontal propriamente dita. Ela é parecida com as rotinas dos outros micros que usam o Z-80. Não apague as duas rotinas anteriores. Utilize o endereço inicial −8112.

```
      ld de,-7935     11 01 E1
      ld hl,-7934     21 02 E1
      ld b,24            06 18
loop  push bc               C5
      ld a,(de)             1A
      ld bc,39        01 27 00
      ldir               ED B0
      ld (de),a             12
      inc de                13
      inc hl                23
      pop bc                C1
      djnz loop           10 F3
      ret                   C9
```

Para vê-la funcionando, acrescente ainda as seguintes linhas ao programa em BASIC apresentado anteriormente:

```
1006 DEF USR2=-8112
1035 FOR I=1 TO 1000
1050 A=USR2(0)
1070 NEXT I
```

COMO FUNCIONAM OS PROGRAMAS

A primeira rotina transfere os 960 caracteres da tela de 40 colunas para o topo da memória. O comando **ld c,152** coloca o valor 152 no registro C. Este é o número da "porta" correspondente à memória de vídeo. Todo dispositivo periférico tem um número desse tipo.

Escolhemos os endereços de −7936 a −6976 (57600 a 58560; o computador não reconhece inteiros acima de 32768 em código de máquina) para colocarmos a nossa "memória de vídeo" na RAM. Assim, o primeiro endereço dessa memória é colocado no registro HL por **ldhl, −7936**.

O valor 4 é colocado no acumulador pelo comando **ld a,4**. O mesmo é feito com 240 no registro B, por **ld b,240**. A e B funcionam como contadores.

A instrução **inir** é a mais importante do programa. Ela coloca o valor lido na porta cujo número está em C na posição de memória cujo endereço está em HL. A seguir, a mesma instrução soma 1 ao conteúdo de HL e subtrai 1 a B. Se o conteúdo de B não for 0, todo o processo será repetido (*input, increment and repeat*). Quando o conteúdo de B se torna 0, o programa passa à instrução seguinte.

A instrução **dec a** subtrai 1 ao valor em A. Se, após essa operação, o conteúdo de A for diferente de 0, **jrnz** desviará o curso do programa para a primeira ocorrência do rótulo **loop** (*jump relative if not zero*). Isso fará com que o processo de ler 240 caracteres na porta 152 seja repetido. Caso o valor em A seja reduzido a zero, o salto não ocorrerá, e o programa passará então à instrução seguinte.

O comando **ret** faz com que a rotina retorne ao BASIC.

Em resumo, esse programa transfere os primeiros 960 caracteres da tela, a partir da posição do cursor.

Ele faz isto repetindo quatro vezes a leitura e transferência de 240 caracteres (4×240=960).

A segunda rotina é essencialmente igual à primeira; apenas, trocamos o comando **inir** por **outir** (=*output, increment and repeat*) que, em vez de ler na porta 152, envia para esse dispositivo os 960 caracteres que estavam no topo da memória.

A terceira rotina é, com algumas modificações, a mesma utilizada para o Spectrum. Ela age sobre a "memória de vídeo" na RAM e, para funcionar, deve ser usada juntamente com as rotinas anteriores. Leia atentamente a explicação do programa do Spectrum.

As únicas diferenças são os endereços iniciais da memória de vídeo (−7935 e −7934), o número de linhas da tela (24) e o número de caracteres por linha, menos 1 (39). O restante é exatamente igual.

Tente então traduzir o programa que faz deslocamento horizontal da tela para a direita, do Assembly para código de máquina. Para rodá-lo, serão necessárias

as rotinas iniciais, bem como um programa em BASIC.

```
      ld de,-6976
      ld hl,-6977
      ld b,24
loop  push bc
      ld a,(de)
      ld bc,39
      lddr
      ld (de),a
      dec hl
      dec de
      pop bc
      djnz loop
      ret
```

O TRS-80 já vem com um monitor. Se seu sistema não tem unidade de disco, responda **N** à pergunta inicial do computador: BASIC (S/N)? Entramos, assim, no monitor. Para aprender os comandos específicos, consulte seu manual.

Caso seu sistema tenha unidade de disco, pressione simultaneamente **BREAK** e **RESET**, liberando a última tecla primeiro. Desse modo, o sistema se comportará como se não houvesse unidade de disco.

O programa montado com o monitor permanecerá na memória, não sendo apagado nem pelo DOS nem pela tecla **RESET**, nem pelo BASIC, se for dada uma resposta adequada à pergunta "Memória Usada?" (resposta adequada é um endereço inferior ao do programa montado).

A rotina que se segue desloca a tela um caractere para a esquerda.

```
      ld de,15360     11 00 3C
      ld hl,15361     21 01 3C
      ld b,16            06 10
loop  push bc               C5
      ld a,(de)             1A
      ld bc,63        01 3F 00
      ldir               ED B0
      ld (de),a             12
      inc de                13
      inc hl                23
      pop bc                C1
      djnz loop          10 F3
      ret                   C9
```

O PROGRAMA NO TRS-80

Este é essencialmente o mesmo programa fornecido para o Spectrum. Mas existem algumas diferenças.

São elas: o endereço inicial da memória de vídeo, que é 15360 no TRS; o número de linhas da tela (16) e o número de caracteres por linha (64). Os comandos são, portanto, idênticos.

A rotina a seguir desloca a tela para a direita.

```
      ld de,16383
      ld hl,16382
      ld b,16
loop  push bc
      ld a,(de)
      ld bc,63
      lddr
      ld (de),a
      dec hl
      dec de
      pop bc
      djnz loop
      ret
```

Dirija-se à seção do Spectrum para comentários.

Este programa desloca a tela para a esquerda. Já foi traduzido do Assembly para linguagem de máquina. Ele não funciona em sistemas com disquete.

```
      LDX#1024        8E 04 00
LOOP  LDB,X+             E6 80
      PSHS B             34 04
      LDB#31             C6 1F
JUMP  LDA,X+             A6 80
      STA-2,X            A7 1E
      DECB                  5A
      BNE JUMP           26 F9
      PULS B             35 04
      STB-1,X            E7 1F
      CMPX#1536       8C 06 00
      BLO LOOP           25 EA
      RTS                   39
```

COMO FUNCIONA O PROGRAMA

A memória de vídeo do computador começa no endereço 1024; assim, **LDX #1024** carrega este valor no registro X. **LDB,X+** carrega o acumulador B com o conteúdo da posição de memória cujo endereço está em X. A seguir, soma 1 ao conteúdo desse registro. **PSHB** armazena temporariamente na pilha o conteúdo do acumulador B — que é o código do primeiro caractere do vídeo.

O comando **LDB#31** carrega B com o número 31. Esse registro é usado como contador. Embora existam 32 caracteres por linha, a operação de deslocar cada caractere uma posição para a esquerda será feita 31 vezes. A transferência do primeiro caractere da linha para a última posição é feita separadamente, depois que o resto da linha é deslocado para a esquerda.

O comando **LDA,X+** carrega o acumulador A com o conteúdo do endereço que está em X; esse endereço está uma posição à frente do endereço carregado em B, pois foi somado 1 a X. Depois disso, a instrução soma 1 a X novamente. **STA – 2,X** armazena o conteúdo de A no endereço situado duas posições antes do conteúdo de X. Isso fica uma posição à esquerda de onde esse caractere foi retirado, já que X foi incrementado depois.

A instrução **DECB** subtrai 1 ao conteúdo de B, diminuindo esse valor de 31 até 0. Essa operação afeta o sinalizador 0 (ou *flag* zero), de forma que um resultado 0 em qualquer operação estabelece esse sinalizador. **BNE** verifica se o sinalizador foi estabelecido. Caso isto não tenha ocorrido, o curso do programa sofrerá um desvio. **JUMP** é o rótulo (*label*), de forma que o programa salta até a instrução **LDA,X+** e desloca para a esquerda o caractere seguinte, até chegar ao final da linha. Quando isso acontece, **DECB** reduz o conteúdo do registro B a 0 e o sinalizador é estabelecido. Agora, o salto não ocorre mais, e o programa passa à instrução seguinte a **BNE**.

O comando **PULS B** recupera o valor no topo da pilha, colocando-o em B. **STB – 1,X** faz com que o conteúdo de B seja armazenado uma posição atrás do endereço em X. Esta é a última posição da linha; o primeiro caractere é então recuperado da pilha. Assim, a linha é toda deslocada para a esquerda, e seu final passa a ser ocupado pelo primeiro caractere.

A instrução **CMPX#1536** compara o endereço em X com 1536, que é o último endereço da memória de vídeo. **BLO** desviará o programa se o conteúdo de X for menor que 1536 (*Branch if LOwer than*). Loop é o rótulo, de forma que, se o programa não deslocar toda a tela, chegando ao seu endereço final, o programa saltará para a instrução **LDB,X+** e começará a deslocar a linha seguinte.

Se o registro X contiver 1536, o programa terá deslocado toda a tela, e a instrução seguinte será executada.

O comando **RTS** retorna ao BASIC. Toda rotina em linguagem de máquina deve terminar com essa instrução. Se isso não ocorrer, o microprocessador tentará executar qualquer comando que esteja após o seu programa.

O programa seguinte desloca a tela para a direita. Traduza os mnemônicos Assembly para código de máquina (os endereços estão em decimal e devem ser convertidos para hexa).

```
      LDX#1536
LOOP  LDB,-X
      PSHS B
      LDB#31
JUMP  LDA,-X
      STA 1,X
      DECB
      BNE JUMP
      PULS B
      STB,X
```

```
      CMPX#1024
      BGT LOOP
      RTS
```

Este programa trabalha no sentido inverso ao anterior, começando pelo fim da memória de vídeo (1536). Ele desloca as linhas da tela para a direita, até chegar ao endereço 1024. Na realidade, 1536 é uma posição posterior ao final da tela, mas o pós-byte, −X, subtrai 1 ao conteúdo do registro X após a execução da instrução **LDX**. Assim, se **LDB, −X** carregar B com o conteúdo da última posição da tela, o programa deverá começar com um endereço que está uma posição na frente, dentro do registro X.

Com exceção desse detalhe, o programa funciona de maneira idêntica ao anterior (confira o tamanho dos saltos que calcular, consultando a listagem).

O Apple II já vem com um monitor. Para entrar, digite **CALL − 151**. Os comandos específicos estão no manual do computador. Para usar as rotinas em seus programas BASIC, basta acionar o comando **CALL N**, onde **N** é o endereço inicial da rotina em código. A organização da memória de vídeo do Apple II abriga duas páginas de vídeo com diferentes modos de utilização, além de uma página de texto. A maior complicação, porém, é a diferença na ordem em que as linhas de texto ou de pontos são representadas na tela e na memória. A segunda linha de texto na tela, por exemplo, não é o segundo conjunto de 40 bytes da memória de vídeo.

Assim, para não tornar muito complicado este primeiro programa em Assembly para o Apple II, fizemos com que ele deslocasse para a esquerda somente a primeira linha de vídeo.

Este programa não funcionará no TK-2000. Todavia, os usuários desse micro não precisam aprender a montar programas a mão, uma vez que dispõem do mini-Assembler. Use o endereço inicial hexa 320 (800 em decimal).

```
      LDA #$00          A9 00
      STA $FB           85 FB
      LDA #$04          A9 04
      STA $FÇ           85 FC
      LDA #$00          A9 00
      STA $FE           85 FE
      LDY #$00          A0 00
      LDA ($FB),Y       B1 FB
      STA $FD           85 FD
      LDY #$01          A0 01
LOOP  LDA ($FB),Y       B1 FB
      DEY                  88
      STA ($FB),Y       91 FB
      INY                  C8
      INY                  C8
      CPY #$28          C0 28
      BNE LOOP          D0 F5
      LDY #$27          A0 27
      LDA $FD           A5 FD
      STA ($FB),Y       91 FB
      RET                  60
```

Use ainda este programa em BASIC:

```
5  HOME
10  FOR I = 1 TO 40
20  VTAB 1: PRINT  CHR$ (64 + I
);
30  NEXT
40  FOR I = 1 TO 1000
50  CALL 800
60  NEXT
70  GOTO 70
```

O PROGRAMA NO APPLE II

O comando **LDA #$00** carrega 0 no acumulador, e **STA $FB** coloca esse valor no endereço 0020. Analogamente, **LDA #$04** e **STA $FC** colocam 04 em 00FC. Não há um comando único que armazene valores diretamente na memória.

O endereço da primeira posição da memória de vídeo do Apple é 0400 (em hexa). 00FB e 00FC são posições da página 0 da memória. As posições dessa página requerem somente um byte no seu endereçamento. **LDA #$00** e **STA $FE** colocam 0 em 00FE, posição que será usada como contador.

A instrução **LDY #$00** carrega o registro-índice Y com o deslocamento (ou seja, 0, pois o programa tem que começar no início da tela), **LDA (FB),Y** carrega o acumulador com o conteúdo da posição de memória cujo endereço está em 00FB e 00FC, mais um deslocamento provocado pelo número que está em Y. 00FB e 00FC apontam para a posição 0400, início da tela de textos. Assim, essa instrução posiciona o primeiro caractere no acumulador. **STA $FD** o coloca na posição 00FD.

A seguir, **LDY #$01** carrega o registro Y com 1. **LDA ($FB),Y** carrega o acumulador com o conteúdo da posição de endereço dado por 00FB e 00FC, mais um deslocamento provocado por Y. Desta vez, contudo, Y contém 1 em vez de 0. Assim, essa instrução põe no acumulador o caractere da posição 0401.

A instrução **DEY** subtrai 1 ao registro Y, e **STA ($FB),Y** coloca o caractere cujo código está em A na posição dada por 00FB e 00FC, mais o deslocamento em Y. Como Y foi diminuído de 1 nesse processo, ele tem como resultado a transferência de cada caractere para uma posição à esquerda.

O registro Y tem 1 duas vezes somado ao seu conteúdo por intermédio da repetição do comando **INY**, apontando assim para a próxima posição da tela **CPY #$28** compara o conteúdo de Y com 28 em hexa, ou 40 em decimal, que é o total de caracteres por linha. **BNE LOOP** verifica o sinalizador 0. Se ele não for estabelecido, o microprocessador voltará para o endereço rotulado por **LOOP** e continuará de lá.

O comando **BNE** envia o programa de volta a **LOOP** até que Y acabe de contar de 0 a 40, movendo os caracteres da linha para a esquerda. Quando o conteúdo de Y chega a 40, a condição de **BNE** não é satisfeita e o microprocessador continua na instrução seguinte.

A instrução **LDY #$27** carrega Y com 27 hexa, ou 39 decimal. **LDA $FD** carrega o acumulador com o conteúdo da posição de memória 00FD.

Já o comando **STA ($FB),Y** coloca o conteúdo do acumulador no endereço dado por 00FB e 00FC, mais o deslocamento em Y. Ele posiciona o primeiro caractere da linha na posição 0427, que é a última da primeira linha de texto.

A instrução **RTS** diz ao microprocessador para voltar ao BASIC. Qualquer rotina em linguagem de máquina tem que terminar com **RTS**, caso contrário o microprocessador seguirá memória afora, tentando executar qualquer coisa que houver no caminho. O programa seguinte desloca a primeira linha de vídeo para a direita. Tente calcular os códigos correspondentes.

```
      LDA #$00
      STA $FB
      LDA #$04
      STA $FC
      LDA #$00
      STA $FE
      LDY #$27
      LDA ($FB),Y
      STA $FD
      LDY #$26
LOOP  LDA ($FB),Y
      INY
      STA ($FB),Y
      DEY
      DEY
      CPY #$FF
      BNE LOOP
      LDY #$00
      LDA $FD
      STA ($FB),Y
      RET
```

Este programa opera da mesma forma que o anterior, exceto que somamos onde subtraímos e vice-versa, para que o deslocamento seja feito no sentido inverso.

Os saltos são do mesmo tamanho, de forma que os valores calculados podem ser verificados na outra listagem (o salto inclui o final da instrução **BNE**, ou seja, o byte que contém seu tamanho).

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO DE JOGOS

Acompanhe a criação de uma aventura e crie o seu próprio jogo. Roteiros, personagens e mapas.

PROGRAMAÇÃO BASIC

Formas de ampliar a utilização dos comandos gráficos e criar novas ilustrações na tela. Cores, círculos e arcos.

PROGRAMAÇÃO BASIC

O que é programação estruturada. Estruturas embutidas. Escolhas múltiplas.

CÓDIGO DE MÁQUINA

Traduzir Assembly para código de máquina: um trabalho para o Apple II.

CURSO PRÁTICO 12 DE PROGRAMAÇÃO DE COMPUTADORES

DIGITAL

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

Cz$ 20,00

NOVA CULTURAL