O seu ZX Spectrum é um poderoso computador, pelo que são inúmeras as possibilidades a explorar. Dos princípios básicos até às mais sofisticadas técnicas de programação, esta obra é um excelente guia para principiantes e também para aqueles que já possuem alguns conhecimentos neste domínio. Mais de 100 programas, rotinas e subrotinas, que o ajudarão a melhorar os seus próprios programas e que lhe possibilitarão, passo a passo, um melhor domínio da arte da programação.

EDITORIAL PRESENÇA / MARTINS FONTES

148

Como Programar o Seu ZX SPECTRUM

TIM HARTNELL e DILWYN JONES

COMO PROGRAMAR O SEU ZX SPECTRUM

TEMPOS LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Petar Trifunovitch* e *Sava Vukovitch*
4. ABC do Bridge, *Pierre Jais* e *H. Lahana*
5. Guia Prático de Fotografia, *W. D. Emanuel*
6. ABC do Judo, *E. J. Harrison*
7. Como Fazer Cinema, *Paul Petzold*
8. Bridge Moderno, *Pierre Jais* e *H. Lahana*
9. Fotografia — Técnicas e Truques I, *Edwin Smith*
10. ABC dos Estilos, da Arquitectura ao Mobiliário, *A. Aussel*
11. Fotografia — Técnicas e Truques II, *Edwin Smith*
12. A Pesca Submarina, *António Ribera*
13. Teoria dos Finais de Partida, *Yuri Averbach*
14. Aprenda Rádio, *B. Fighiera*
15. Guia do Cão, *Louise Laliberté-Robert* e *Jean-Pierre Robert*
16. ABC do Aquário, *Anthony Evans*
17. Iniciação à Electricidade e Electrónica, *Fernand Huré*
18. Os Transístores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricone*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *Curt Af Enehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez — Teoria do Meio Jogo I, *Bondarevsky*
35. Manual do Super 8 — I, *Myron A. Matzkin*
36. ABC da Criação de Periquitos, *Cyril H. Rogers*
37. O Livro dos Gatos, *Barbel Gerber* e *Horst Bielfeld*
38. Manual do Super 8 — II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. Witham Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Coker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabian Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*
51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires* e *C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelhos, *P. Surdeau* e *R. Henaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen* e *Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Böelle* e *B. Doyen*
63. Enxerto de Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *Luís Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*
67. Doces e Conservas de Fruta, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Roy Genders*
70. Calendário do Horticultor, *Luís Alsina Grau*
71. Jogos Electrónicos, *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, *Alejo Rigau*
73. Aprenda Televisão a Cores, *Gordon J, King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Poda de Árvores e Arbustos, *Roy Genders*
76. Como Treinar o Seu Cão, *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, *Heinrich Stöckle*
78. A Criação de Caracóis, *Matías Josa*
79. Rádio — Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébault*
81. Iniciação à Jardinagem, *Noel Clarasó*
82. A Congelação dos Alimentos, *Suzanne Lapointe*
83. Windfsurf — Prancha à Vela, *Ernstfried Prade*
84. Raças de Cães, *O. Hasselfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia, Embalsamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging — Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos T.V., *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie de Prémont* e *Nicole Philippi*
95. O Minitrampolim na Escola, *Sonja Helmer Jensen* e *Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, *B. Fighiera*
97. O Cultivo do Tomate, *Louis N. Flawn*
98. Pilhas Solares, *F. Juster*
99. A Criação Doméstica de Coelhos, *C. F, Snow*
100. Iniciação ao Futebol, *Wieland Männle* e *Heinz Arnold*
101. Horóscopos Chineses, *Georg Haddenbach*
102. Guia Prático de Marcenaria, *Charles H. Hayward*
103. Andebol, *Fritz* e *Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perus, Pintadas e Codornizes, *Jérome Sauze*
106. Crepes — Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott* e *M. Baker*
110. Futebol — Técnica e Táctica, *Kurt Lavall*
111. A vaca Leiteira, *Colin T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*

113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marieanne Ränk*
115. O Cubo Para Quem Já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, *M. Mrowka-W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung-Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hanns-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hil*
121. Construção de um Alimentador de Corrente, *Waldemar Baitinger*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *Christa-Maria* e *Richard Kerler*
126. A Pesca e os seus Segredos, *Armand Deschamps*
127. O Osciloscópio, *R. Rateau*
128. Guia Prático da Banda do Cidadão, *T. M. Normand*
129. Sumos e Batidos, *Manfred Donderski*
130. Introdução à Programação de Microcomputadores, *Peter C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*
133. Guia Prático de Basic, Goger Hunt
134. Introdução à Electrónica Digital, *Ian Sinclair*
135. ABC do Vídeo, *David K. Matthewson*
136. Fotografia em Movimento, *Don Morley*
137. Guia de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *Sidney F. Ray*
139. Guia Moderno da Canaricultura, *Manuel Gonçalves*
140. Minielectrónica para Amadores, *Heinz Richter*
141. ABC da Programação de Computadores, *John Shelley*
142. Tarot — O Futuro Pelas Cartas, *Edwin J. Nigg*
143. ABC da Equitação, *Dorothy Johnson*
144. Como Programar o Seu ZX 81, *Patrick Gueulle*
145. 100 Avarias TV e a Maneira Prática de as Detectar, *P. Duranton*
146. ABC da Horticultura, *Louis Giordano*
147. Basic Para Microcomputadores, *A. P. Stephenson*
148. Como Programar o Seu ZX Spectrum, *Tim Hartnell* e *Dilwyn Jones*
149. Iniciação aos Motores Diesel, *David S. Maclean*
150. 60 Jogos Para o ZX Spectrum, *Devid Harwood*
151. As Linhas da Mão, *Rose Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *Simpson* e Terrel
154. 80 Assembler para o Z Spectrum Iniciação ao Código de Máquina, *João Paulo Fragoso*
155. Aeróbica, *H. Schulz*
156. ABC do Atletismo, *Denis Watts*
157. 26 Programas Basic para Microcomputadores, *Derrick Daines*
158. Aprenda Pascal no seu Microcomputador, *Jeremy Ruston*
159. Guia Moderno da Suinicultura, *Colin Whittmore*

Tim Hartnell
Dilwyn Jones

COMO PROGRAMAR O SEU ZX SPECTRUM

2.ª edição

EDITORIAL PRESENÇA

Título original:
PROGRAMMING YOUR ZX SPECTRUM

Tradução Conceição Jardim e Lúcio Nogueira

PREFÁCIO

As suas primeiras horas com o ZX Spectrum podem ser entusiasmantes. Depois de ter executado os programas simples apresentados no manual da máquina, é provável que o leitor pense: "Sim, e agora?". Este livro foi concebido para responder a esta pergunta.

Ensiná-lo-á a programar o Spectrum partindo do princípio, até chegar às técnicas de programação sofisticadas.

E enquanto está a dar entrada a um programa, espalhando marcianos e asteróides por todo o lado, descobrirá que está de facto a aprender bastante sobre programação, sobre os computadores em geral - e sem fazer qualquer esforço.

Este livro é concebido como um simples guia, a ser usado com o computador a seu lado. O seu valor será bastante diminuído se se limitar a lê-lo. É melhor meter na máquina todos os programas que lhe aparecerem nestas páginas, não desprezando qualquer ser de outros mundos; desse modo tirará o máximo benefício deste livro e transformar-se-á num feiticeiro da programação antes de se dar conta disso.

É chegado o momento. Ligue o seu Spectrum, "acenda" a televisão, e comecemos.

Tim Hartnell, Londres
Dilwyn Jones, Bangor, País de Gales

1
COMO USAR O TECLADO

Se bem que, à primeira vista, o teclado do Spectrum possa parecer um tanto complexo, não é de facto difícil de dominar desde que se seja cuidadoso nas primeiras fases de trabalho com ele.

As duas teclas mais importantes são as de mudança de modo: a CAPS SHIFT (canto inferior esquerdo) e a SYMBOL SHIFT (segunda tecla do canto inferior direito). Procure descobri-las.

As cores destas teclas indicam a respectiva utilidade. Ligue o Spectrum, carregue na tecla CAPS SHIFT e em seguida experimente qualquer das teclas que apresentam uma letra do alfabeto impressa. Verificará que obtém a versão em maiúscula dessa letra. CAPS SHIFT também serve para obter as funções escritas a branco sobre as teclas 1 a 4. Carregue em CAPS SHIFT e na tecla 2. Em seguida carregue em cada uma das letras do alfabeto, e verificará que são impressas no visor em maiúsculas. O uso de TRUE VIDEO e INVERSE VIDEO (teclas 3 e 4) será discutido no capítulo sobre gráficos.

Deixando de parte a tecla CAPS SHIFT, vejamos agora a SYMBOL SHIFT, impressa a vermelho. Carregue nela, e simultaneamente em qualquer outra tecla (exceptuando ENTER, BREAK SPACE ou CAPS SHIFT). Verificará que obtém os símbolos que se encontram escritos nessas teclas a vermelho (por exemplo a seta para cima da tecla H).

Em seguida, carregue em qualquer tecla com uma letra do alfabeto, sem carregar simultaneamente em nenhuma das duas teclas de mudança de modo. Verificará que obtém a palavra escrita a branco na tecla em causa (IF, NEXT, DIM, etc). Chama-se-lhes *palavras-chave* e são elas que ocupam o primeiro lugar em qualquer linha de programa, questão que discutiremos mais adiante.

As palavras escritas a verde por cima das teclas são obtidas carregando simultaneamente em ambas as teclas de mudança de modo, largando-as em seguida e carregando na tecla pretendida. Por exemplo, depois de carregar em CAPS SHIFT e SYMBOL SHIFT, carregue na tecla D. Verá surgir no visor a palavra DATA: que se encontra escrita por cima dessa tecla.

As palavras a vermelho por baixo das teclas são obtidas carregando em ambas as teclas de mudança de modo, largando em seguida uma delas e carregando simultaneamente na tecla desejada. Conseguirá assim obter, por exemplo, a palavra BRIGHT se a tecla for a B, ATTR se for a L, e VERIFY se for a R.

Consegue assim obter tudo o que está escrito no teclado excepto EDIT, ENTER, GRAPHICS e DELETE.

EDIT - Pode utilizar esta "função" para modificar uma linha de um programa. Deslocando o cursor (o sinal "maior do que" >) e carregando em seguida na tecla EDIT (1) ao mesmo tempo que carrega na CAPS SHIFT trará a linha indicada por aquele cursor para a parte inferior do visor, onde poderá ser alterada. As teclas 5 e 8 permitir-lhe-ão deslocar um outro cursor ("L") ao longo da linha, no sentido das setas impressas nessas teclas.

ENTER - Carrega-se nesta tecla depois de escrever uma linha de programa na parte inferior do visor, a fim de conseguir colocá-la no corpo do programa, na ordem correcta relativamente às outras linhas. É igualmente usada depois de ter sido escrita uma ordem como RUN, a fim de indicar ao computador que a deve executar.

GRAPHICS - Se carregar em CAPS SHIFT e na tecla 9, verá o cursor transformar-se num "G". Carregando em seguida nas teclas de números poderá ver que são impressas no visor os símbolos gráficos nelas indicados. Se carregar novamente nas teclas de números mas carregar simultaneamente em CAPS SHIFT verá que

são impressos os mesmos símbolos, mas com as cores invertidas: obterá assim o "oposto" de qualquer dos símbolos gráficos. A segunda utilização do modo gráfico, para a produção de gráficos inventados por si, será descrita mais adiante.

DELETE - Esta tecla serve para "apagar" o que tiver escrito na parte inferior do visor. Carregue em CAPS SHIFT e em O, e a linha em que está a trabalhar será apagada elemento a elemento. Pode tirar o dedo de CAPS SHIFT depois de a função DELETE estar em auto-repetição, e os caracteres continuarão a ser apagados um a um.

Não se preocupe se esta descrição - que tornámos tão simples quanto nos foi possível - não lhe desvendar imediatamente todos os mistérios do teclado. Usando o seu Spectrum, e descobrindo cada função do teclado à medida que dela necessita, acabará por atingir um ponto em que o uso do teclado se transformará numa segunda natureza.

2
A DECLARAÇÃO PRINT

PRINT é provavelmente a declaração mais usada em BASIC. É ela que permite ao computador comunicar consigo através da televisão. Escreva a seguinte linha no seu Spectrum, e carregue em seguida em ENTER:

```
PRINT 5
```

Verificará que o computador imprime obedientemente o número 5. Pode utilizar a declaração PRINT para fazer o computador actuar como uma máquina de calcular. Escreva o seguinte, carregando em seguida em ENTER:

```
PRINT 5-3
```

Quando carrega em ENTER, verá que a máquina imprime o resultado correcto. Este "modo de cálculo directo" pode resolver problemas tão complexos quanto se quiser. Experimente o seguinte, não se esquecendo de carregar em ENTER depois de o ter feito a fim de obrigar o computador a cumprir aquilo que você escreveu:

PRINT SQR (8+1)

Esta ordem pede ao computador que imprima a raiz quadrada (é isto que SQR significa) da soma dos dois números entre parêntesis, isto é, a raíz quadrada de nove. Se o seu computador está a funcionar bem, deve obviamente apresentar "3" em resposta.

Vemos portando que PRINT pode ser usada no modo directo para imprimir números ou resultados de cálculos. Pode igualmente imprimir palavras no visor. Passe ao modo "maiúsculas" carregando em CAPS SHIFT e na tecla 2. Em seguida experimente o seguinte, não se esquecendo de carregar depois em ENTER:

PRINT COMO VAI?

Em vez de imprimir alegremente a frase COMO VAI?, o computador "tira da manga" aquilo a que se chama uma mensagem de erro, que imprime na parte inferior do visor. Neste caso, a mensagem de erro é "2 variable not found" (variável não encontrada) Se quiser que o computador imprima palavras, é necessário que estas sejam encerradas em aspas. Escreva agora o seguinte (carregando depois em ENTER):

PRINT "COMO VAI?"

Verá finalmente estas palavras serem impressas na primeira linha do visor.

Recapitulemos. Usada simplesmente como ordem, a frase PRINT 2+3 dirá ao computador que deve imprimir o resultado da soma em causa. Escrevendo PRINT "palavras" conseguirá levar a máquina a imprimir tudo o que estiver dentro de aspas.

Os computadores utilizám programas, e é chegado o momento de escrever um primeiro programa, bastante simples. Escreva e execute o seguinte programa. Quando inicia a execução, o que é feito carregando em RUN (tecla R) e seguidamente em ENTER, deve ver-se impresso aquilo que escrevemos por baixo das linhas do programa:

```
10 REM PROGRAMA UM
20 PRINT "ISTO É UMA DEMONSTRAÇÃO"
```



```
30 PRINT 1
40 PRINT 2
50 PRINT "CHEGAMOS AO FIM"
```

Escreva RUN, seguido de ENTER, e verá o seguinte:

```
ISTO É UMA DEMONSTRAÇÃO
1
2
CHEGAMOS AO FIM
```

Enquanto este programa se encontra no interior do computador, tentemos aprender mais alguma coisa sobre programas. Escreva LIST (o que consegue carregando na tecla K), carregue novamente em ENTER. Verificará que todas as linhas se iniciam por um número. A primeira linha, neste caso com o número 10, começa pela palavra REM. REM é uma abreviatura da palavra inglesa REMARK (em português "observação"), e é usada num programa quando se deseja explicar o que se passa no interior do programa, ou do que trata ele (como acontece neste caso), a fim de facilitar a sua compreensão quando mais tarde se quiser estudá-lo. O computador ignora as instruções REM quando as encontra no programa, não as executando.

A instrução REM é formada por um número de linha seguido pela palavra REM e finalmente por um texto. A mensagem que se segue à palavra REM pode ser formada por aquilo que quisermos - - letras, números, sinais de pontuação, gráficos, espaços - se bem que seja melhor manter as mensagens tão breves quanto possível. Se bem que tudo o que for escrito depois da palavra REM seja ignorado pelo computador quando está a executar o programa, a linha REM ocupa obviamente algum espaço na memória.

As instruções REM são muitas vezes do seguinte tipo:

```
10 REM Calcular pontuação
10 REM determinar o ângulo
```

Não há qualquer razão para incluir linhas REM num programa, ou para não incluir apenas uma, mas se o comentário que deseja acrescentar num dado ponto do programa ocupar mais do

que uma linha de texto é conveniente colocar a palavra REM no início de cada linha ocupada. Por exemplo:

```
60 REM Rotina de multiplicação que
70 REM multiplica as duas variáveis A e B
```

Desde que cada linha de comentários se inicie pela palavra REM, o computador ignorará o que ela contém (se bem que imprima uma listagem completa do programa se tal for pedido usando a ordem LIST).

Vejamos agora a questão da montagem de um programa. Escreva o número 10, e em seguida carregue em ENTER. Verá que a linha 10 do programa que meteu na máquina desaparece. É muito fácil eliminar as linhas de um programa que não desejamos manter, bastando para tal escrever o seu número, como acabámos de fazer, e carregar em seguida na tecla ENTER.

```
20 PRINT "ISTO É UMA DEMONSTRAÇÃO"
30 PRINT 1
40 PRINT 2
50 PRINT "CHEGAMOS AO FIM"
```

Acrescente agora 10 REM, carregando em ENTER.

O leitor recorda ainda que LIST é a ordem BASIC que utilizamos para levar o computador a imprimir todo o programa que contém. Todas as linhas do programa são listadas por ordem numérica e não pela ordem em que foram introduzidas na máquina. Isto é, o computador ordena automaticamente as linhas de programa. Escreva o seguinte, carregando depois em ENTER:

```
15 PRINT "Esta linha é nova"
```

O leitor verificará, no programa seguinte, que a nova linha (15) se coloca automaticamente na sua posição correcta no interior da listagem.


```
10 REM
15 PRINT "ESTA LINHA É NOVA"
20 PRINT "ISTO É UMA DEMONSTRAÇÃO"
30 PRINT 1
40 PRINT 2
50 PRINT "CHEGAMOS AO FIM"
```

Como o leitor compreendeu sem dúvida, a ordem RUN é usada para levar o computador a executar um programa previamente introduzido nele através do teclado ou carregando-o a partir de uma cassette. O computador executa todas as linhas armazenadas na sua memória, começando pela de menor número e continuando por ordem. Existem várias instruções que podem obrigar o computador a saltar para uma linha fora da ordem, mas no essencial a máquina executa um programa seguindo a ordem das linhas a menos que lhe seja ordenado que actue de outro modo.

Se deseja que o programa pare num determinado ponto, pode utilizar uma ordem chamada "STOP". Escreva 25 STOP (usando a tecla A e carregando simultaneamente em SYMBOL SHIFT); carregue depois em ENTER e execute o programa. Este imprimirá o seguinte no visor:

```
ESTA LINHA É NOVA
ISTO É UMA DEMONSTRAÇÃO
```

Em seguida, na parte inferior, aparecerá uma mensagem: 9 STOP statement, 25:1. Esta mensagem significa que foi executada a instrução STOP que encontra em primeiro lugar na linha 25.

Voltaremos a analisar PRINT com maior detalhe daqui a pouco, mas para já existe uma outra declaração que convém introduzir. A ordem NEW apagará o programa existente na memória do computador, devendo ser sempre usada quando se deseja retirar o que se encontra em memória antes de carregar ou escrever um programa novo. Se não o fizer, e se utilizar diferentes números de linha para o segundo programa, verificará que as linhas novas aparecerão misturadas com as do programa anterior. A ordem NEW é definitiva, levando o computador a "esquecer" completamente tudo o que continha.

Experimente a ordem NEW imediatamente. Escreva NEW (usando a tecla A), carregue em ENTER, e em seguida escreva

LIST e volte a carregar em ENTER. Verificará, o que não o deve espantar, que não surge qualquer listagem no visor. **Experimente** LIST 10, e verificará que obtém o mesmo resultado.

3
FORMATAÇÃO DA SAÍDA EM VISOR E TAB

Para continuar o nosso estudo da declaração PRINT, carregue em CAPS LOCK para escrever em maiúsculas e introduza e ponha em execução o programa seguinte.

```
 10 REM Formatos de saída
 20 PRINT
 30 PRINT
 40 PRINT
 50 PRINT
 60 PRINT "Viva"; 60
 70 PRINT' "Viva";70
 80 PRINT 12
 90 PRINT 1,2
100 PRINT ,1
110 PRINT ,1;2
```

Siga cuidadosamente a explicação dada em seguida, e aprenderá bastante sobre o modo como o computador formata a sua saída para o visor. Pode usar estes conhecimentos para obrigar a máquina a imprimir as saídas no ponto do visor onde você o desejar. Vamos estudar este programa linha a linha:

10 Título do programa contido numa declaração REM
20-50 Cada uma das palavras PRINT, sem nada a seguir, deslocam a posição de impressão para a linha seguinte. Isto explica as linhas em branco deixadas na parte de cima do visor quando o programa é executado.
60 Esta linha imprime a palavra "Viva" e depois, deixando um espaço em branco (contido no interior das aspas), imprime o número 60, a fim de o leitor saber a linhade que se trata.
70 A vírgula (escrita usando SYMBOL SHIFT e a tecla N), como pode ver, desloca o início da impressão para a linha central do visor.
80 Esta linha permite escrever os números 1 e 2 juntos. Note que mesmo no caso de existir um espaço entre os números, (por exemplo PRINT 1 2) o computador continuará a escrevê-los juntos.
90 Esta linha utiliza uma vírgula entre os números para espaçá-los de meio visor.
100 A vírgula no início da linha desloca o 1 para a linha central do visor, tal como aconteceu anteriormente à palavra "Viva".
110 O ponto e vírgula entre os números permite que sejam impressos juntos, tal como acontecia na linha 80.

Pode-se utilizar a vírgula e o ponto (designados por "separadores") para comandar a saída produzida no visor. Limpe o programa com NEW, e em seguida escreva no teclado e execute a série de programas que se seguem e que servem para produzir alguns efeitos novos.

O primeiro programa, chamado PRINT DOIS, imprime muito simplesmente os números 1 a 10 de um dos lados do visor. O seguinte (PRINT DOIS-B) imprime-os a seguir uns aos outros. PRINT DOIS-C imprime os mesmos números em duas colunas bem separadas, uma delas na linha central do visor, e PRINT DOIS-D imprime-os deixando um único espaço entre cada dois.

```
10 REM print dois
20 FOR j=1 TO 10
30 PRINT j
40 NEXT j
```

```
10 REM dois - b
20 FOR j=1 TO 10
30 PRINT j;
40 NEXT j
```

```
10 REM print dois - c
20 FOR j=1 TO 10
30 PRINT j,
40 NEXT j
```

```
10 REM print dois - d
20 FOR j=1 TO 10
30 PRINT j;" ";
40 NEXT j
```

O USO DE "TAB"

TAB (de *tabular*) é uma ordem que pode ser bastante útil quando combinada com a declaração PRINT. Desloca a posição de impressão de um número de espaços que deve ser indicado imediatamente à palavra TAB. Escreva os programas PRINT DOIS-E e PRINT DOIS-F. vendo o efeito da declaração TAB em ambos.

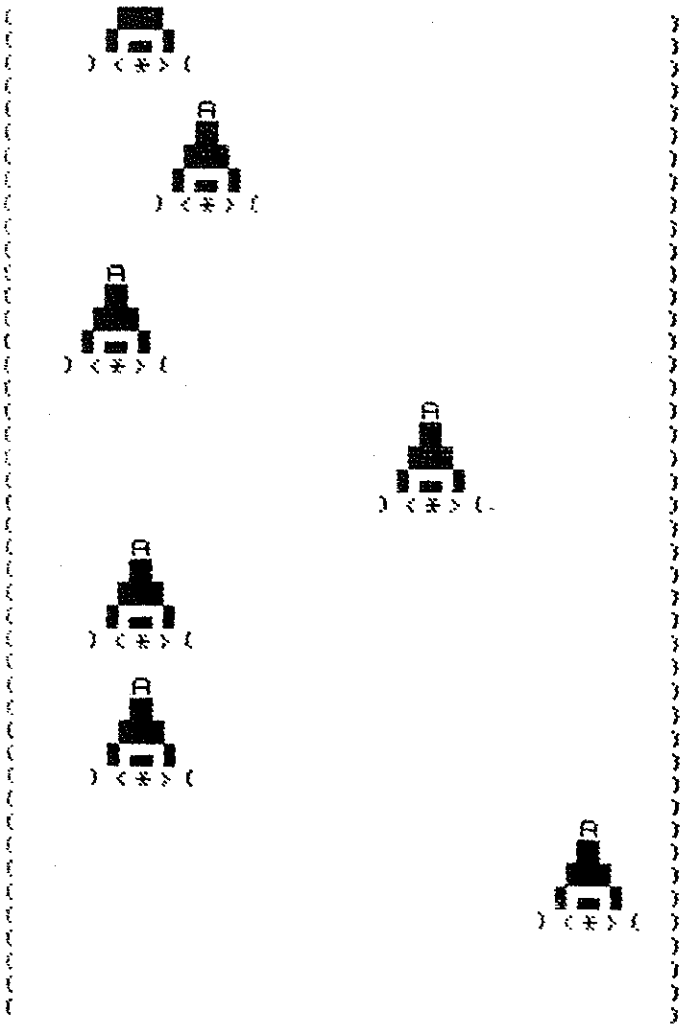
```
10 REM print dois-e
20 FOR j=1 TO 10
30 PRINT TAB j;j
40 NEXT j
```

```
10 REM print dois-f
20 FOR j=1 TO 10
30 PRINT TAB 3*j;j
40 NEXT j
```

O programa seguinte. a que chamaremos "Espaçoporto", mostra a eficácia da demarcação TAB. Escreva e faça executar o

programa. voltando depois a este livro para estudar um pouco as suas linhas mais importantes.

```
 10 REM ESPAÇOPORTO
 20 DIM a$(5,5)
 25 BORDER 0
 30 FOR j=10 TO 1 STEP -1
 40 PRINT TAB 3*j; INK 2;j
 50 FOR a=1 TO j
 52 BEEP .02,5*j
 60 NEXT a
 70 NEXT j
 71 FOR b=1 TO 5
 72 READ q$
 73 LET a$(b)=q$
 74 NEXT b
 80 REM **PROGRAMA**
 90 LET q=INT (RND*25)+1
100 FOR r=1 TO 5
110 BEEP .02,10*r
120 PRINT "(";TAB q; INK r;a$(r
); INK 0;TAB 30;")"
130 NEXT r
140 POKE 23692,-1
150 LET space=q/3
160 FOR p=1 TO space
190 PRINT "(";TAB 30;")"
200 NEXT p
210 GO TO 90
220 DATA "  A","  ■","  ■■","  ■
■■","  ■■■■"
```



As linhas mais importantes deste programa são a 120 e a 190, dado que são elas que utilizam as instruções TAB. A linha 120 comporta-se do seguinte modo.

"(" — Isto imprime um parêntesis encostado ao rebordo esquerdo do visor.

TAB 9 — 9 é um número entre 1 e 25 (escolhido na linha 90) que determina quantos espaços devem ser deixados em branco ao imprimir numa linha.

INK r — Isto determina qual a cor em que é impresso o foguete.

aS(r) — Isto determina qual a parte do foguete que será impressa. Utiliza elementos do quadro de cadeias alfanuméricas, aS, definidos pelo ciclo READ das linhas 71 a 74. Não se preocupe com tudo isto por agora, pois o assunto será estudado com maior detalhe mais adiante.

INK 0 Esta instrução volta a passar a cor de impressão para negro.

TAB 30 Depois de ter sido impressa uma parte do foguete, a posição de impressão desloca-se para a 31ª posição da linha (última coluna), onde é impresso um ")", delimitando o rebordo direito do visor.

Vejamos agora a linha 190.

Esta linha encontra-se no interior do ciclo (loop) que se inicia na linha 180 e termina na linha 200. PRINT ")": TAB 30: ")" imprime um "(" no lado esquerdo do visor, passando depois à última coluna, a trigésima-primeira (usando TAB 30, isto é, deslocando-se de 30 posições para a direita) e imprime um ")" no lado direito. A linha 190 é usada um número de vezes aleatório (determinado pelo 9 que foi escolhido na linha 90) para deixar um número variável de linhas em branco entre os vários foguetes. A linha 140 (POKE 23692,-1) assegura que o programa não pára para perguntar se queremos rolar ("scroll?") a imagem.

4
GRAVAÇÃO DE PROGRAMAS EM CASSETTE

O leitor pode querer guardar uma cópia permanente do programa anterior, por exemplo. É possível SAVE programas (passá-los

para uma cassette) ligando o gravador ao Spectrum do modo indicado no manual, e escrevendo em seguida SAVE seguido do nome do programa entre aspas. Neste caso, sugiro que utilize o nome "FOGUETES", pelo que poderá escrever SAVE "FOGUETES". Ponha a fita em movimento, depois de fazer as ligações necessárias, e carregue em seguida na tecla ENTER.

Sugerimos ao leitor que se habitue a gravar sempre duas cópias de cada programa, talvez em cassettes diferentes; se alguma delas se danificar por qualquer razão disporá ainda daquilo que se chama em gíria inglesa de um "back-up". Por outro lado, se o seu gravador não dispõe de contador, convir-lhe-á gravar apenas um programa em cada lado de uma cassette. Ponha uma etiqueta na cassette, escrevendo nela claramente o nome do programa (neste caso, "FOGUETES"). Se bem que possa parecer um desperdício usar todo um lado de uma cassette para um único programa, tal permitir-lhe-á pelo menos não se sentir frustrado ao procurar no meio de uma fita o programa que lhe interessa. Deve sempre realizar mais do que uma cópia dada a possibilidade de a fita se deteriorar, de você mesmo apagar acidentalmente o programa, ou ainda — como tantas vezes acontece — de uma cópia se recusar a "entrar" correctamente na máquina quando o desejamos.

Deve limpar frequentemente as cabeças do gravador, usando liquido e não uma das cassettes vendidas para esse fim, para que garanta a obtenção de um sinal quanto possível.

5
VERIFY, MERGE

Depois de ter gravado o programa na cassette, pode verificar se a gravação foi feita correctamente antes de eliminar o programa da memória do computador.

Suponhamos que gravou o programa escrevendo SAVE "programa". Deve rebobinar a fita, escrevendo em seguida VERIFY "programa", carregando em ENTER e pondo a fita novamente em movimento. Se tudo estiver bem, surgirá no visor o nome do programa, e no final verá na parte inferior daquele surgir a mensagem 0 OK. Saberá então que o programa foi gravado correctamente. Note porém que por vezes esta informação é enganadora: a verificação pode ser feita a um volume de saída sonora do gravador relativamente baixo e pode acontecer que apesar de aquilo que se encontra em memória corresponder exactamente ao que foi gravado, a gravação pode ter sido feita a um nível excessivamente baixo e não ser aceite depois pelo computador. Sugerimos ao leitor que faça algumas experiências com um programa sem interesse até descobrir o melhor modo de usar o seu gravador para este efeito.

É possível carregar num programa no computador quando se encontra outro dentro deste. Os dois programas juntar-se-ão um ao outro. Se ambos possuirem linhas de programa idênticas, a linha do programa novo substituirá a do programa anterior. Vejamos um exemplo. Vamos escrever o programa seguinte, passando em seguida para uma cassette e escrevendo NEW no computador a fim de eliminar o programa da sua memória.

```
10 REM Programa de teste 1
20 REM Linha 20
30 REM Linha 30
40 REM Linha 40
```

Escreveu-se depois o programa seguinte:

```
 5 REM Programa Dois
15 REM Linha 15
25 REM Linha 25
35 REM Linha 35
```

Utilizou-se em seguida a ordem MERGE "Um" (foi com este nome que se gravou previamente o programa anterior em cassette) para reintroduzir o primeiro programa na máquina.

Ao fim de alguns segundos, encontrava-se o seguinte programa dentro do computador:

```
 5 Programa Dois
10 Programa Um
15 REM Linha 15
```

```
20 REM Linha 20
25 REM Linha 25
30 REM Linha 30
35 REM Linha 35
40 REM Linha 40
```

MERGE é um modo muito prático de utilizar certas rotinas especiais, por exemplo uma rotina de alteração dos números das linhas, que podem ser assim carregadas na máquina depois de se ter escrito um outro programa. Vale a pena ter o cuidado de dar a estas rotinas altos números de linha (por exemplo acima de 9000), usando as inferiores para o programa propriamente dito. Garante-se assim que não haja dificuldades devido a sobreposição de linhas.

Carregar os programas no computador pode tornar-se por vezes difícil. Se tiver problemas ao executar um LOAD experimente algumas das seguintes sugestões:

1) Desligue o cabo de ligação ao gravador que não está a ser usado;
2) Tente alimentar o gravador de cassettes com pilhas;
3) Tente afastar o gravador do computador, assim como ambos do aparelho de televisão. Pode também desligar este quando faz SAVE, ou virá-lo ao contrário, "de costas" para o computador .. Ocorrem por vezes interferências electromagnéticas entre estes aparelhos.
4) Altere o volume do gravador de cassettes, dado que algumas das fitas possuem uma maior potência de sinal do que outras. Tente alterar igualmente a tonalidade, em particular aumentando os agudos ou diminuindo os graves.
5) Verifique se os fios não se partiram, ou se não existe qualquer junta em mau contacto.
6) Isto pode parecer óbvio, mas verifique se as fichas estão colocadas nas tomadas correspondentes... Por outro lado, as tomadas do computador não fazem por vezes bom contacto, convindo trocar de cabo.

Voltemos agora à declaração TAB.

Só nos é possível utilizar a declaração TAB escrevendo um

único número depois desta palavra. O leitor não se deve esquecer de que TAB A deslocará a posição de impressão A+1 espaços para a direita na mesma linha. A declaração PRINT pode ser seguida de AT e de dois números, como acontece por exemplo em PRINT AT 10,6: nestas condições pode-se deslocar a posição de impressão para a linha 10 (não esquecer que a primeira é a linha 0) e para a posição 6. O canto superior esquerdo do visor é 0,0, pelo que PRINT AT 0,0 indicará que a impressão se inicia nesse canto do visor. Não esqueça que o visor tem uma largura de 32 caracteres, numerados de 0 a 31.

6
PRINT AT

O programa que se segue, "Squash", utiliza instruções PRINT AT y,x para colocar uma bola (linha 620) e uma raquete (linha 150). Utiliza-se as teclas "Z" e "M" para deslocar a raquete existente na parte inferior do visor para a direita e para a esquerda respectivamente: O programa conta o tempo durante o qual o utilizador mantém a bola em jogo, e fornece no fim uma pontuação baseada nele. Carregando em qualquer tecla no final do jogo poderá jogar de novo.

A listagem pode parecer-lhe por agora bastante horrorosa. Depois de ter terminado este livro, talvez seja conveniente voltar a programas como este tentando descobrir o que faz cada linha. Ficará surpreendido com aquilo que conseguirá então "decifrar".

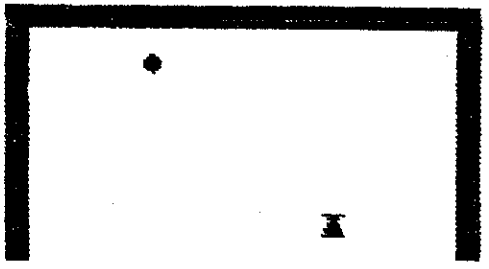
```
10 REM SQUASH
15 REM SEGUNDO PROGRAMA DE
       JEREMY RUSTON
20 GO SUB 590
25 REM DESLOCAR A RAQUETE COM Z e M
27 LET BOLA= 550
30 LET CAMPO= 300
35 LET RAQUETE= 460
40 BRIGHT 1: BORDER 4
45 PAPER 7: CLS
50 LET PONTUAÇÃO= 0
```

```
  60 GO SUB CAMPO
  70 REM ************
  80 LET PONTUACAO=PONTUACAO+INCREMENTO
 110 LET A$=INKEY$
 130 IF A$="Z" OR A$="M" THEN GO
 SUB RAQUETE
 140 GO SUB BOLA
 150 PRINT AT 19,B+11; INK 2;B$
 160 GO TO 80
 290 REM ************
 300 REM **  CAMPO  **
 310 LET X=1
 320 PRINT AT 10,10; INK 1;"
 330 FOR T=0 TO 10
 340 PRINT AT T+10,10; INK 1;"
;AT T+10,30;"
 350 NEXT T
 360 LET B$=" "+B$+" "
 380 LET Y=1
 385 LET L=1
 390 LET M=1
 400 LET B=10
 410 PRINT AT 19,11+B;B$
 420 LET INCREMENTO=207+INT (RND*
100)
 430 RETURN
 450 REM ******************
 460 REM ** DESLOCAR RAQUETE **
 480 IF A$="M" AND B=16 THEN RET
URN
 490 IF A$="Z" AND B=0 THEN RETU
RN
 510 IF A$="M" THEN LET B=B+1
 520 IF A$="Z" THEN LET B=B-1
 530 RETURN
 540 REM ******************
 550 REM *** MOVER BOLA  ***
 570 PRINT AT 11+Y,11+X;" "
 580 IF L+X>18 OR L+X<0 THEN LET
 L=-L: BEEP .01,50
 590 IF M+Y>8 OR M+Y<0 THEN LET
M=-M: BEEP .01,20
 600 LET X=X+L
 610 LET Y=Y+M
 620 PRINT AT 11+Y,11+X; INK 4;B
$
 622 IF Y<>8 THEN RETURN
 625 PRINT AT 6,7; INK 6; PAPER
```



```
2;" PONTUAÇÃO: ";PONTUAÇÃO;"  "
 630 IF Y=8 AND ABS (B-X)<=2 THE
N RETURN
 640 PRINT AT 2,2; INK 7; PAPER
3;" FINAL DO JOGO"
 650 FOR G=1 TO 50
 660 BEEP .01,G: BEEP .01,60-G
 670 NEXT G
 680 RUN
 690 REM NA LINHA 700 O SÍMBOLO GRÁFICO
         ENTRE ASPAS É A LETRA M
 700 LET D$="
 710 FOR H=0 TO 7
 715 BEEP .01,H*H
 720 READ N
 730 POKE USR "M"+H,N
 740 NEXT H
 750 REM A PRÓXIMA LETRA É
         O GRÁFICO B
 760 LET B$="
 770 FOR H=0 TO 7
 775 BEEP .01,50/(H+1)
 780 READ N
 790 POKE USR "B"+H,N
 800 NEXT H
 890 RETURN
1000 DATA BIN 00000000,BIN 00011
000,BIN 00111100,BIN 00111110,B
IN 00111110,BIN 00111110,BIN 0
0111100,BIN 00001000
1010 DATA BIN 11111111,BIN 00111
100,BIN 00011000,BIN 00111100,BI
N 00111100,BIN 01111110,BIN 01111
110,BIN 11111111
```

PONTUAÇÃO 5152



As letras entre aspas nas linhas 700 e 760 devem ser escritas em modo gráfico (carregando primeiro nas teclas CAPS SHIFT e 9).

7
CORES E GRÁFICOS

O Spectrum está equipado com poderosos comandos gráficos que o leitor poderá usar para melhorar os seus programas. Estas instruções são simples de usar, e capazes de produzirem uma vasta gama de efeitos.

É possível controlar três coisas com o comando de cursor: a margem, acedida pela ordem BORDER, o fundo, acedido por PAPER e a cor de "escrita", acedida por INK.

Existem na máquina oito cores (contando o preto e o branco), numeradas entre zero e sete. São as seguintes:

0 - negro
1 - azul
2 - vermelho
3 - magenta (púrpura)
4 - verde
5 - cião (azul-esverdeado)
6 - amarelo
7 - branco

Quanto menor for o número, mais escura é a cor. Num aparelho de televisão a preto e branco, os números mais pequenos aproximam-se do negro, e os outros do branco.

Quando se liga o Spectrum, o fundo e a margem surgem automaticamente a branco e a tinta a negro. Isto significa que o visor se apresenta completamente em branco, e que qualquer programa escrito no teclado será impresso a negro.

As cores de PAPER e INK podem ser usadas de dois modos.

O primeiro é globalmente: isto significa que no caso de uma linha de programa dizer PAPER 6, seguido de CLS (limpar o visor), todo o fundo ficará de cor amarela, exceptuando evidentemente a margem. Do mesmo modo, a linha de programa que inclua a instrução INK 2 levará a máquina a imprimir daí em diante tudo vermelho.

Mas as cores podem também ser usadas "localmente". Se escrevermos PRINT INK 1; PAPER 7; "Viva!", o Spectrum imprimirá a palavra "Viva!" a azul sobre uma faixa branca. É possível fazer o mesmo controlo local em instruções INPUT. Se deseja dar a entrada a uma sequência alfanumérica, pode escrever INPUT INK 2; PAPER 6; "Qual é o seu nome?"; aS, e a pergunta será impressa a vermelho sobre uma faixa amarela.

Exprimentemos agora estes comandos de cor, usando para tal o seguinte programa:

```
  20 REM DEMONSTRAÇÃO DE CORES
  30 FOR B=0 TO 7
  40 FOR P=0 TO 7
  50 FOR I=0 TO 7
  60 BORDER B
  70 PAPER P: CLS
  80 INK I
  90 PRINT AT 10,10;" MARGEM ";B;
TAB 10;"FUNDO" ";P;TAB 10;"TINTA ";
I
 100 FOR U=1 TO 60: NEXT U: BEEP
 .15,P*I+2*B
 110 NEXT I
 120 NEXT P
 130 NEXT B
```

Este programa passa todas as combinações possíveis de BORDER, PAPER e INK. Como o leitor pode verificar, demora bastante tempo a ser executado (existem 8*8*8 combinações possíveis), se bem que várias destas não sejam muito eficazes (tinta branca sobre fundo branco e com margem branca, por exemplo...) Certos grupos de cores não são agradáveis. Outros, no entanto, são bastante interessantes, valendo a pena anotá-los numa folha de papel.

A linha que limpa o visor (70 CLS) é necessária para tornar "global" a cor de fundo. A instrução PAPER, sem CLS, apenas altera o fundo das palavras impressas. Experimente novamente o programa sem essa linha. Os comando de tinta (INK) usados no programa e que são automaticamente "globais" quando seguidos por CLS, tornam-se locais se estiverem associados a uma instrução PRINT ou INPUT. Uma ordem INK global (como por exemplo INK 2) não é alterada por uma ordem local que a contrarie, como PRINT INK 1) dado que a cor de tinta volta automaticamente à

anterior assim que encontra no programa nova instrução PRINT sem parâmetros de cor.

Execute em seguida o programa que vamos apresentar, que mostra até que ponto é possível misturar cores escolhidas aleatoriamente.

```
  5 REM Pirâmide
  7 REM © Hughes, Hartnell
 10 BORDER 7
 15 CLS
 20 LET b=16
 50 LET t=0
 60 LET s=0
 70 LET l=20
 80 LET t=t+b
 90 FOR n=s TO s+b*2-2
100 PRINT AT l,n; INK-INT (RND*
6)+1;"■"
105 BORDER INT (RND*6)+1
110 NEXT n
120 LET l=l-1
130 LET b=b-1
140 LET s=s+1
150 IF b>0 THEN GO TO 80
155 BORDER 1
160 GO TO 160
```

O programa desenha uma pirâmide com pequenos blocos coloridos. A margem cintila de forma alarmante durante toda a execução do programa, e finalmente (linha 155) torna-se azul. A linha 160, que se limita a reenviar para si mesma, serve para suprimir a mensagem "OK" que de outro modo estragaria a imagem. Sai-se do programa fazendo BREAK.

O pequeno quadrado negro no final da linha 100 pode ser obtido directamente no teclado passando ao modo gráfico (CAPS SHIFT e tecla 9) e carregando na tecla 8, mantendo ainda a pressão na tecla CAPS SHIFT. O inverso dos outros caracteres pode ser obtido carregando simplesmente em INV. VIDEO (CAPS SHIFT e 4). Volta-se aquilo que é chamado TRUE VIDEO carregando de novo em CAPS SHIFT e em 3. O fundo negro atrás das letras em inverso passam a ter a cor INK, e as próprias letras passam à cor PAPER, o que pode ter um efeito bastante eficaz como se pode ver no programa que se segue:


```
 15 PAPER 5
 17 CLS
 20 FOR g=1 TO 100
 30 INK RND*7
 40 PAPER RND*7
 50 BORDER RND*7
 70 PRINT AT RND*21,RND*9; "ISTO É
UMA DESMONSTRAÇÃO! "
 80 NEXT g
```

Na linha 70, as palavras sublinhadas devem ser escritas em modo "INV. VIDEO", obtido do modo já indicado.

O uso das cores num programa pode produzir óptimos resultados, como se mostra no programa seguinte. "Código de cores". Trata-se de uma variante do Mastermind mas, como poderá ver ao executar o programa, este exige de si que adivinhe um código de quatro cores, e não de quatro números ou letras ao contrário do que acontece na maior parte das versões deste jogo. Escreva e execute o programa, voltando em seguida ao livro para estudar a explicação dos comandos de cor e de gráficos nele usados.

```
 10 REM Código de cores
 20 POKE 23609,100
 30 DIM C(4)
 40 DIM G(4)
 50 DIM H(4)
 60 INK 0: BORDER 0: PAPER 7: C
LS
 70 PRINT ''TAB 3; "Estou a pensar num
código de quatro cores"
 80 PRINT "Tem 10 tentativas para advinhar o código"
 90 PRINT "Vou escolher algumas das seguintes cores"
'"ESE COLOURS"''
100 FOR C=1 TO 6
110 PRINT TAB 4+C; INK 0;C;;" >
>"; INK C;"■"
120 NEXT C
130 PRINT '' "As cores são diferentes entre si"
140 PRINT '' "Carregue em qualquer tecla"
150 PAUSE 4E4
160 CLS
170 PRINT AT 1,5;
```

```
160 FOR C=1 TO 6
190 PRINT INK 0;C;">"; INK C;"■
";
200 NEXT C
210 PRINT
220 LET C(1)=INT (RND*6)+1
230 LET Z=1
240 LET Z=Z+1
250 LET C(Z)=INT (RND*6)+1
260 LET J=0
270 LET J=J+1
280 IF C(J)=C(Z) THEN GO TO 230
290 IF J<Z-1 THEN GO TO 270
300 IF Z<4 THEN GO TO 240
310 FOR G=1 TO 10: POKE 23692,-
1
320 PRINT INK 0;"Indique o número a adivinhar
";G
330 INPUT A
335 FOR Q=1 TO 32: PRINT CHR$ 8
;: NEXT Q
337 REM Na linha seguinte deixe 32 espaços entre aspas
340 PRINT"                                "
350 FOR Z=1 TO 4
360 LET G(Z)=A-10*INT (A/10)
370 LET H(Z)=G(Z)
380 LET A=INT (A/10)
390 NEXT Z
400 LET B=0: LET W=0
410 FOR Z=1 TO 4
420 IF C(Z)<>G(Z) THEN GO TO 45
0
430 LET B=B+1: BEEP .2,B*15
440 LET G(Z)=0
450 NEXT Z
460 FOR Z=1 TO 4
470 IF G(Z)=0 THEN GO TO 520
480 FOR J=1 TO 4
490 IF C(Z)<>G(J) THEN GO TO 51
0
500 LET W=W+1: BEEP .2,60-B*15
510 NEXT J
520 NEXT Z
530 FOR T=4 TO 1 STEP -1
540 PRINT INK H(T);"■ ";
550 NEXT T
560 PRINT INK 0;"";B;" NEGRO";
570 IF B<>1 THEN PRINT "S";
580 PRINT "E ";W;" BRANCO";
```



```
590 IF W<>1 THEN PRINT "S"
600 IF W=1 THEN PRINT
610 IF B=4 THEN PRINT "Conseguiu, em";G;
" Tentativas";
620 IF G>1 AND B=4 THEN PRINT "
ES"
630 IF B<>4 THEN NEXT G
650 PRINT '''"O código era";
660 FOR H=1 TO 100: NEXT H
670 FOR T=4 TO 1 STEP -1
680 FOR H=1 TO 50: NEXT H
690 BEEP .2,T*10: PRINT INK C(T
);"■ ";
700 NEXT T
710 FOR H=1 TO 60: BEEP .01,H:
NEXT H
720 POKE 23692,-1
730 PRINT '''"Deseja outro jogo?"
735 PRINT TAB 8;"Escreva s ou n"
740 LET A$=INKEY$: IF INKEY$=""
THEN GO TO 740
750 IF CODE A$<>CODE "N" THEN R
UN
760 CLS
770 PRINT '' INK RND*6;TAB RND*
15;"OK, Adeus!"
780 POKE 23692,-1
790 FOR H=1 TO 25
800 NEXT H
810 GO TO 770
```

No início, o programa imprimirá o seguinte:

Estou a pensar num código de quatro cores

Tem 10 tentativas para adivinhar o código

Vou escolher algumas das seguintes cores:

As cores são diferentes entre si

Carregue em qualquer têcla

A linha 20 (POKE 23609,100) altera o "click" produzido pela tecla num "beep", um som mais longo, o que serve para confirmar o contacto da tecla premida. É muito útil recorrer a isto quando se está a programar. A linha 80 define as cores da margem, do fundo e da tinta. A rotina entre as linhas 100 e 120 imprime as seis cores (um quadrado de cada) numa linha em diagonal, com os números junto às cores a que correspondem. A linha 150 espera até que se carregue em qualquer tecla antes de continuar a execução.

A rotina entre as linhas 220 e 300 escolhe as cores, garantindo que sejam todas diferentes. A linha 210, entretanto, deslocou o ponto de impressão de uma linha (usado o apóstrofo existente na tecla 7, a vermelho), e as linhas 180 a 200 imprimem as seis cores atravessando o visor na parte superior, juntamente com os números correspondentes.

A linha 310 inicia o ciclo que permite a entrada de dez adivinhas. A segunda metade da linha 310 (POKE 23692,-1) assegura que no caso de o visor encher rolará automaticamente, sem requerer uma resposta à pergunta "scroll?" que de outro modo surgiria na parte inferior do visor. Este POKE é também usado com grande frequência.

A linha 320 pede que seja dada entrada a uma adivinha, e depois de isto ser feito (linha 330) utiliza-se o movimento para trás (CHR$ 8) 32 vezes para voltar a cobrir a linha que pede a adivinha. A linha 340 enche esta linha de espaços em branco. Isto significa que a linha "Indique adivinhe número 2" é apagada, mas que as adivinhas anteriores (e o código de cores em cima) não o são, tornando possível observar as adivinhas anteriores para facilitar a resposta. Indica-se a nova adivinha escrevendo um número de quatro algarismos, usando o código indicado na parte superior do visor: isto significa que para indicar azul basta carregar em 1.

A rotina entre as linhas 350 e 390 divide o número por nós indicado em quatro algarismos diferentes. As variáveis para negros (B) e brancos (W) são passadas a zero na linha 400 e em seguida compara-se a cor com o código de quatro algarismos que o computador escolheu, fazendo soar pequenos "beeps" quando encontra "brancos" ou "negros". Se acertou, o programa dá-lhe a conhecer o facto. Se não, e não tiver utilizado ainda as 10 adivinhas, são-lhe indicados quais os algarismos correctos na posição correcta (negros) e quais os correctos em posição errada (brancos), sendo-lhe pedida uma nova adivinha.

O leitor já sabe que pode escrever PRINT AT 4,7 "teste" para imprimir a palavra teste na linha 4, coluna 7. O caracter de comando CHR$22 actua de modo semelhante, mas com uma diferença. Para obter o mesmo resultado é necessário escrever

```
PRINT CHR$22 + CHR$4 + CHR$7; "teste"
```

No entanto, como o Spectrum permite concatenar (**somar** cadeias alfanuméricas, ou seja, juntá-las umas às outras), pode-se juntar todos estes CHR$'s obtendo uma única cadeia. Isto pode ser bastante útil, se quisermos especificar uma dada posição de impressão várias vezes ao longo de um programa. Execute o programa seguinte, e verá como tudo isto funciona.

```
10 LET a$=CHR$ 22+CHR$ 4+CHR$
7
20 PRINT a$; "TEST"
```

A declaração TAB pode ser substituída por instruções CHR$ n, onde n é o número de espaços que se deseja deixar livres, acrescentadas a CHR$ 23. Execute o programa seguinte para observar um exemplo. No entanto como CHR$ 23 espera ser seguida de facto por dois números (n e m que têm o mesmo efeito de PRINT TAB n + 256*m), pode-se preceder a informação contida entre aspas por um espaço, ou uma letra a mais (x no nosso caso) que não será impressa. Corra o programa seguinte, e verá que em vez de imprimir *TESTE no visor, a máquina escreverá simplesmente TESTE.

```
10 LET a$=CHR$ 23+CHR$ 4
20 PRINT a$;"XTEST"
30 GO TO 20
```

No início deste capítulo, discutimos as oito cores e vimos como estas podiam ser usadas na informação impressa (INK), na cor de fundo (PAPER) e na da margem (BORDER). A informação impressa pode ser modificada usando dois comandos adicionais - BRIGHT e FLASH. A rotina que se segue mostra estes dois comandos em acção. Escreva e execute o programa -, voltando depois ao livro para uma breve discussão destas duas novas declarações.

```
 40 PRINT INK 4;"NORMAL ";
 45 PRINT BRIGHT 1; INK 4;" Brilhan
te ";
 50 PRINT INK 4; PAPER 2;"NORMA
L ";
 55 PRINT BRIGHT 1; INK 4; PAPE
R 2;"Brilhante ";
 60 PRINT FLASH 1; INK 4;"Cintilante
";
 65 PRINT BRIGHT 1; FLASH 1; IN
K 4;"Brilhante ";
 70 PRINT FLASH 1; PAPER 2; INK
 4;"Cintilante ";
 75 PRINT BRIGHT 1; FLASH 1; PA
PER 2; INK 4;"Brilhante "
```

Se bem que o efeito de cintilação (FLASH) seja impossível de passar despercebido, pode ser necessário prestar um pouco mais de atenção para ver o efeito de BRIGHT, depois de ter executado este programa, observe a palavra "Brilhante", imediatamente abaixo de "Normal" no topo do visor. Verá que o verde tem um tom diferente. O branco sobre o verde (a sexta linha) mostra o efeito de BRIGHT com maior clareza. Compare a "leveza" da palavra "Brilhante" aqui quando comparada com a palavra "Cintilante", acima. No caso das palavras não cintilantes impressas a verde sobre vermelho (uma combinação bastante horrível), verifica-se que a palavra "brilhante" é um pouco mais fácil de ler do que a "normal".

Se bem que os números zero a sete sejam os normalmente usados com as declarações INK, PAPER e BORDER, podem ser

usados outros números. Usando 8 (por exemplo em PAPER 8) indica-se ao computador que seja o que for que estiver impresso nesse ponto a cor se manterá igual. Isto não é particularmente útil em programação normal, mas o número 9 já pode ser bastante eficaz.

"9" refere-se ao contraste, e assegura que no caso de estar a usar um fundo claro as palavras serão impressas a negro, ou a claro sobre um fundo escuro. É deste mesmo modo que varia a cor da declaração INPUT em função da cor BORDER. O programa seguinte mostra o resultado destas instruções, imprimindo letras escolhidas aleatoriamente, em posições igualmente aleatórias, sobre uma cor de fundo variável. Deixe o programa correr durante algum tempo, voltando depois ao livro.

```
 60 PAPER RND*8
 70 CLS
 80 INK 9
100 FOR G=1 TO 32
110 PRINT AT RND*20,RND*30;CHR$
(65+INT (RND*26));
120 NEXT G
130 GO TO 60
140 PRINT AT RND*20,RND*30; OVE
R 1;CHR$ (65+INT (RND*26));
```

A palavra OVER é bastante útil, e pode produzir alguns efeitos interessantes. O leitor notou certamente uma linha aparentemente inútil no final do programa anterior. Usando o comando EDIT, coloque a linha 140 em vez da 110, e altere 32 no final da linha 100 para 300. Notará que, por vezes, as letras são impressas por cima de outras. A ordem OVER significa que a nova letra não apaga a anterior, limitando-se a complementá-la, "subtraindo" uma da outra de modo a produzir uma forma nova. Isto permite-nos construir alguns caracteres próprios. Escreva e execute o programa seguinte para o fazer. É muito difícil prever o efeito de "soma" de várias letras. Por exemplo, um "o" minúsculo e um "w" minúsculo podem combinar-se para produzir o que parece ser T maiúsculo!

```
10 OVER 1
20 FOR G=1 TO 16
30 INPUT "Indique uma letra"; A$;
40 INPUT "Indique uma letra"; B$;
50 PRINT AT G,G; A$; CHR$ 8; B$
60 NEXT G
```

O leitor recorda-se certamente do que dissémos sobre o facto de CHRS22 e CHRS23 poderem ser usadas para substituir PRINT AT e TAB. e sobre o modo como estas instruções podem ser somadas (concatenação) a fim de incluir toda a ordem numa única cadeia. O mesmo pode ser feito com outras ordens. Os caracteres de comando. e as ordens que substituem. são CHRS 16 - INK: CHRS 17 - PAPER: CHRS 18 - FLASH: CHRS 19 - BRIGHT: CHARS 20 - INVERSE: CHRS 21 - OVER. Estas ordens são seguidas do caracter que correspondem à cor pretendida. Podem por outro lado ser somados. como já dissémos e podemos verificar no programa seguinte.

```
 20 INPUT PAPER 6; INK 1; "INDIQUE
UMA COR DE INK";INK
 30 INPUT INK 2; "INDIQUE UMA COR DE
PAPER"; PAPER
 40 INPUT FLASH 1; BRIGHT 1; IN
K 4; PAPER 2; "INDIQUE UMA PALAVRA"; AS
 50 LET A$=CHR$ 16+CHR$ INK+CHR
$ 17+CHR$ PAPER+A$
 60 PRINT AT 10,10; A$
```

A linha 60 deste programa pode igualmente ser acrescentada à cadeia AS. Talvez queira experimentar fazê-lo como exercício.

No manual do computador são explicados alguns outros caracteres de comando. também se podendo encontrar nele uma discrição completa dos vários efeitos disponíveis na fiada superior de teclas.

Se quiser verificar até que ponto pode ser eficaz o uso da cor. mesmo quando usada num programa simples. escreva e execute a rotina seguinte. Se os "beeps" o aborrecerem. elimine as linhas

90 e 100. Se deseja que a figura seja construída mais rápidamente. altere o 7 no final da linha 40. substituindo-o por 6.

```
 10 PAPER 7: BORDER 0: CLS
 20 LET A=RND*10
 30 LET B=RND*16
 40 LET Z=RND*7
 50 PRINT AT A,B; INK Z;"■"
 60 PRINT AT 21-A,B; INK Z;"■"
 70 PRINT AT 21-A,31-B; INK Z;"
■"
 80 PRINT AT A,31-B; INK Z;"■"
 90 IF RND>RND THEN GO TO 20
100 BEEP RND/30,RND*60-RND*60
110 GO TO 20
```

Quando tiver executado este programa durante algum tempo. modifique-o do seguinte modo:

```
 10 PAPER 7: BORDER 0: CLS
 20 LET A=RND*10
 25 LET F=RND
 30 LET B=RND*16
 40 LET Z=RND*6
 50 PRINT AT A,B; FLASH F; BRIG
HT 1; INK Z;"■"
 60 PRINT AT 21-A,B; FLASH F; B
RIGHT 1; INK Z;"■"
 70 PRINT AT 21-A,31-B; FLASH F
; BRIGHT 1; INK Z;"■"
 80 PRINT AT A,31-B; FLASH F; B
RIGHT 1; INK Z;"■"
 90 IF RND>RND THEN GO TO 20
100 BEEP RND/30,RND*60-RND*60
110 GO TO 20
```

O leitor verificará que o programa faz cintilar todos os pontos impressos. acrescentando um FLASH aleatório em cada execução. Nas instruções FLAHS e BRIGHT. "1" significa que é usada a cintilação ou brilho. e "0" nega-os. Tal como acontece em várias outras instruções. FLASH e BRIGH não executam INT sobre um número aleatório. arredondando-o porém para o inteiro mais próximo (enquanto que INT arredonda sempre para o inteiro mais próximo *inferior* ao valor fraccionário). Nestas condições. o efeito da

linha 25 no programa consiste em "Ligar" a função FLASH em alguns ciclos da execução, "desligando-a" nos outros. Pode-se ver isto alterando a instrução RND na linha 25 para 1, executando depois o programa durante algum tempo, e em seguida para 0 executando-o também durante algum tempo. Finalmente, o leitor poderá estar interessado em modificar a listagem de modo a obter o programa seguinte, "Sopa de letras", um nome que compreenderá facilmente depois de o ver em execução. Esta última versão inclui muitos dos aspectos discutidos até agora.

```
  7 REM SOPA DE LETRAS
 10 PAPER 7: BORDER 0: CLS
 20 LET A=RND*10
 25 OVER 1
 30 LET B=RND*16
 40 LET Z=RND*7
 45 LET A$=CHR$ (65+RND*26)
 50 PRINT AT A,B; BRIGHT 1; INK
Z;A$
 60 PRINT AT 21-A,B;; BRIGHT 1;
INK Z;A$
 70 PRINT AT 21-A,31-B; BRIGHT
1; INK Z;A$
 80 PRINT AT A,31-B; BRIGHT 1;
INK Z;A$
 90 GO TO 20
```

INSTRUÇÃO PLOT

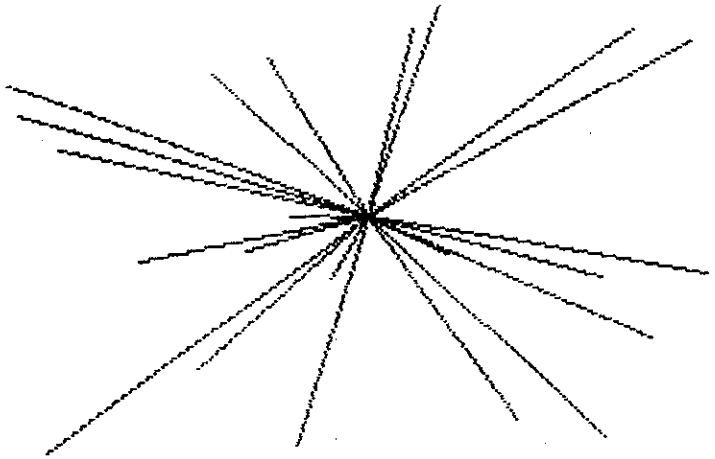
As instruções PLOT permitem construir gráficos de grande resolução, como se pode verificar executando o programa "Galáxia" e "Seno". Depois de ter executado este último, verificará que apesar de a resolução obtida nos pontos ser de 256X192, a resolução obtida na cor é apenas 32 x 22. Com efeito, a cor é aplicada separadamente ao visor preenchido pelo desenho. Apesar desta falta de resolução na cor, é ainda possível criar imagens com uma elevada resolução.


```
 10 REM Galáxia
 20 PAPER 0: BORDER 0: CLS
 30 LET c=255: LET d=175
 40 INK RND*7
 50 LET a=c*RND
 60 LET b=d*RND
 70 PLOT a,b: PLOT a,d-b
 80 PLOT c-a,b: PLOT c-a,d-b
 90 IF RND>.5 THEN GO TO 60
 95 INK RND*7
100 GO TO 50
```

```
 10 REM Seno     Sine
 20 REM © Colin Hughes,
        Hartnell    1982
 30 BORDER 2: CLS
 40 FOR x=0 TO 63 STEP .5
 50 LET y=20*SIN (x/32*PI)
 60 IF y=0 THEN GO TO 100
 70 FOR n=0 TO y STEP SGN y/4
 80 PLOT INK RND*6;x*3+30,3*(n+
30)
 90 NEXT n
100 BEEP .1,x: NEXT x
```

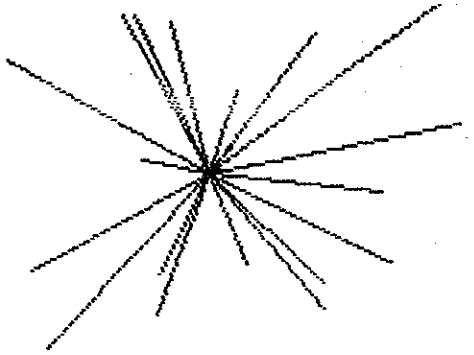
INSTRUÇÃO DRAW

Pode-se verificar a qualidade dos gráficos escrevendo e executando o programa seguinte — Vidros partidos — que utiliza a ordem DRAW. O fundo recebe a cor branca (linha 30), sendo depois escolhidas aleatoriamente as cores de margem e de tinta. A linha 70 verifica se estas são diferentes. Se não o são, é escolhida uma nova cor de tinta. O visor é limpo (linha 100), e escolhe-se em seguida um par de coordenadas. Marca-se um ponto no centro do visor (linha 130), e desenha-se uma linha entre este ponto e as coordenadas previamente escolhidas. O desenho da linha é realizado pela instrução DRAW, que determina o comprimento da linha, o seu ângulo, e apenas necessita de conhecer o ponto de partida. Este ponto é indicado neste programa usando PLOT.

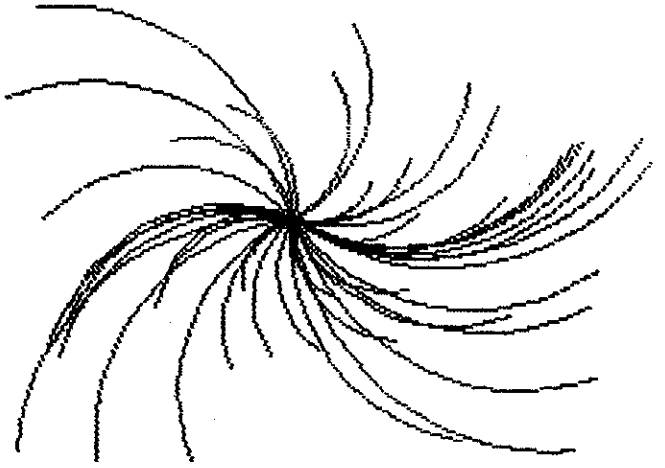


Vidros partidos

```
 10 REM Vidros partidos
 20 REM © Hartnell, Ruston 1982
 30 PAPER 7
 50 LET a=INT (RND*8)
 60 LET b=INT (RND*7)
 70 IF a=b THEN GO TO 60
 80 BORDER a
 90 INK b
100 CLS
110 LET c=INT (RND*256)-128
120 LET d=INT (RND*172)-85
130 PLOT 128,86
140 DRAW c,d
145 BEEP .01,RND*100-50
150 IF RND>0.02 THEN GO TO 110
160 RUN
```

A instrução DRAW desenha linhas quando contém apenas dois números. Estes números correspondem às coordenadas de PLOT do último ponto da linha. Se se acrescentar um terceiro número, a ordem DRAW desenhará parte de um círculo, servindo o terceiro número para especificar o arco que deve ser descrito. O programa que se segue - "Curvas Interrompidas" - é igual ao anterior excepto no final da linha 140, onde desenha uma espécie de versão encurvada do desenho anterior, rodando a linha de PI /2 radianos quando é desenhada.



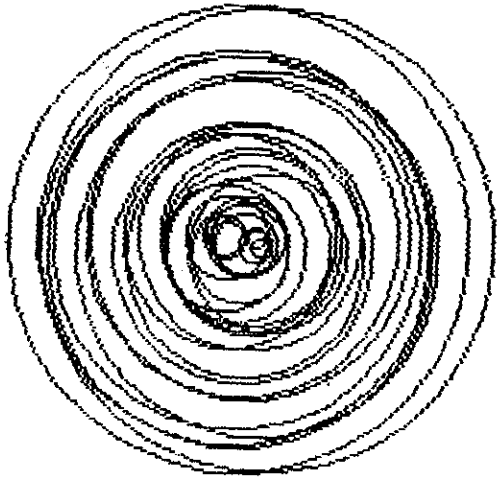
```
 10 REM Broken curves
 20 REM © Hartnell, Ruston 1982
 30 PAPER 7
 50 LET a=INT (RND*8)
 60 LET b=INT (RND*7)
 70 IF a=b THEN GO TO 60
 80 BORDER a
 90 INK b
100 CLS
110 LET c=INT (RND*256)-128
120 LET d=INT (RND*172)-85
130 PLOT 128,86
140 DRAW c,d,PI/2
145 BEEP .01,RND*100-50
150 IF RND>0.015 THEN GO TO 110
160 RUN
```

Em seguida apresentamos outra versão, com mudança de cores:

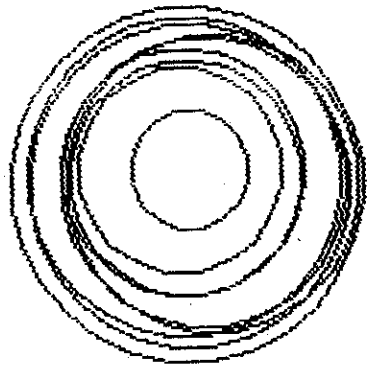
```
 10 REM  Curvas interrompidas
 15 REM  Mudança de cores
 20 REM © Hartnell, Ruston 1982
 30 PAPER 7
 50 LET a=INT (RND*8)
 60 LET b=INT (RND*7)
 70 IF a=b THEN GO TO 60
 80 BORDER a
 90 INK b
100 CLS
110 LET c=INT (RND*256)-128
120 LET d=INT (RND*172)-85
130 PLOT 128,86
135 IF RND>0.5 THEN INK RND*6
140 DRAW c,d,PI/2
145 BEEP .01,RND*100-50
150 IF RND>0.015 THEN GO TO 110
160 RUN
```

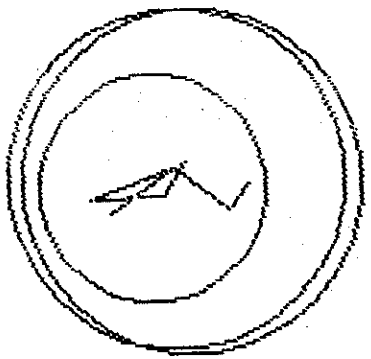

INSTRUÇÃO CIRCLE

Existe ainda um outro comando gráfico, CIRCLE, que como seria de esperar desenha círculos. O programa "Visão em túnel" define uma margem azul clara, com PAPER branco, e desenha uma série de círculos em cores aleatórias, em torno de um ponto central que pouco varia de um círculo para outro. Os raios são também variáveis. O primeiro número depois da palavra CIRCLE é a coordenada X do centro, e o segundo Y; o terceiro é o raio.



```
 10 REM visão em túnel
 15 BORDER 5: PAPER 7: CLS
 20 CIRCLE INK RND*6;128+RND*10
-RND*10,86+RND*7-RND-7,RND*65
 30 IF RND>.92 THEN CLS
 40 BEEP RND/3,RND*100-30
 50 GO TO 20
```





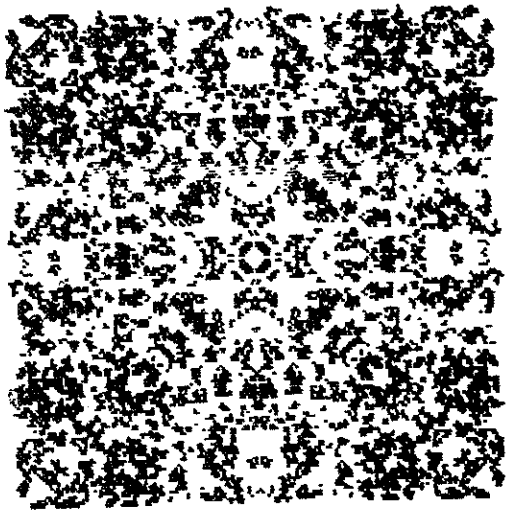
```
  10 REM Visão em túnel
  15 REM com o movimento aleatório
  20 CIRCLE INK RND*6;128+RND*10
-RND*10,86+RND*7-RND-7,RND*65
  25 PLOT 128,86
  30 DRAW INK RND*6; OVER 1,32-R
ND*64,21-RND*42
  40 BEEP RND/4,RND*50-30
  50 IF RND>.33 THEN GO TO 30
  60 IF RND<.93 THEN GO TO 20
  70 RUN
```

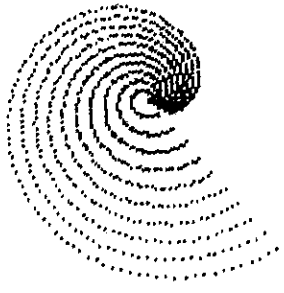

Finalmente. apresentamos alguns programas para demonstrar a eficácia que os gráficos do Spectrum podem atingir.

```
   5 PAPER RND*7
   7 REM Pontos de cores aleatórias
  10 REM screen demo
  20 BORDER 2
  25 PAPER 7
  27 CLS
  30 PRINT INK RND*5+1;AT RND*21
,RND*31;"■"
  40 GO TO 30
```

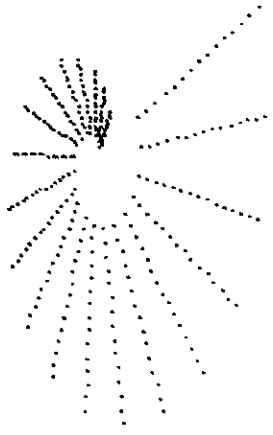
```
  10 REM Som e visão
  20 BORDER RND*7: PAPER RND*7:
CLS
  25 PRINT AT 10,8; INK RND*7;"S
OUND and VISION"
  30 BEEP RND*50/200,RND*50
  40 GO TO 20
```

```
  10 REM Tapete
  12 REM © Gourlay, Hartnell
  15 BORDER 1+RND: PAPER 7: CLS
  20 RANDOMIZE
  25 LET p=85: LET s=45
  30 LET x=RND*p: LET y=RND*p
  35 INK RND*6
  37 FOR g=1 TO RND*30
  40 PLOT p+x+s,p+y
  50 PLOT p+y+s,p+x
  60 PLOT p-x+s,p+y
  70 PLOT p+y+s,p-x
  80 PLOT p-x+s,p-y
  90 PLOT p-y+s,p-x
 100 PLOT p+x+s,p-y
 110 PLOT p-y+s,p+x
 120 LET x=x+RND+RND-1
 130 LET y=y+RND+RND-1
 135 NEXT g
 140 IF RND<.2 OR ABS x>p OR y>p
THEN GO TO 40
 150 IF RND<.01 THEN GO TO 25
 160 GO TO 30
 190 RUN
```





```
  10 REM REMOINHO
  20 PAPER 7: CLS
  30 INK 0
  40 BORDER 7
  50 FOR a=1 TO 10
  60 FOR j=1 TO 7 STEP .1
  70 PLOT a*j*COS (j)+110,a*j*SI
N (j)+100
  80 NEXT j
  90 INK a/2
 100 NEXT a
```

```
  10 REM Spokesman
  20 PAPER 7: CLS
  30 INK 0
  40 BORDER 7
  50 FOR a=5 TO 19
  60 FOR j=1 TO 7 STEP .3
  70 PLOT a*j*COS (j)/2+110,a*j#
SIN (j)+100
  80 NEXT j
 100 NEXT a
```

```
  10 REM CAMPO DE FORÇAS
  15 REM (muito lento)
  20 PAPER 7: CLS
  30 INK 0
  40 BORDER 7
  50 FOR a=2 TO 19
  60 FOR j=1 TO 7 STEP .01
  70 PLOT a*(j+4*SIN (j))*COS (j
)+110,a*(j+2*SIN (j))*SIN (j)+10
0
  80 NEXT j
  90 INK a/3.5
 100 NEXT a
```

ARTE ABSTRACTA

O programa que se segue, escrito por Jeremy Ruston, move duas bolas no visor (X,Y e L,M), desenhando linhas entre elas. Um outro aspecto é que de vez em quando são alteradas as velocidades, fazendo as bolas ressaltarem em pontos que não correspondem à margem do visor. Para eliminar isto, retirar a linha 116.

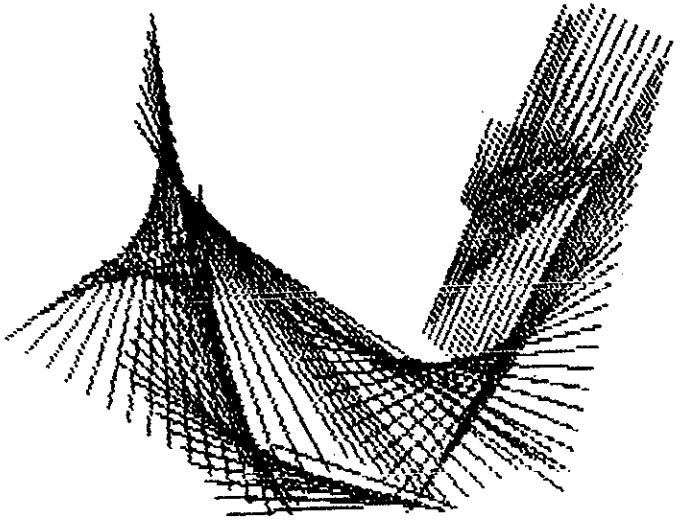
Descrição do programa

5 - Define as cores, impressão a negro sobre um fundo branco. Esta linha é necessária porque o leitor pode executar o programa com diferentes efeitos de cores.
10 - Escolhe aleatoriamente uma coordenada x para a primeira bola.
20 - Escolhe aleatoriamente uma coordenada y para a primeira bola.
30 - Escolhe aleatoriamente uma coordenada x para a segunda bola.
40 - Escolhe aleatoriamente uma coordenada y para a segunda bola.
50 - Inicializa as variáveis requeridas para a definição das velocidades das bolas.
60 - Define um gerador de números aleatórios aceitável.
100 - Chama a subrotina da linha 1000. Esta subrotina escolhe velocidades aleatórias para as bolas.
115 - A variável "num" guarda o número de passos que serão executados antes de serem escolhidas novas velocidades.
116 - Se "num" atinge zero, é chegado o momento de escolher outras velocidades. A rotina na linha 1000 define igualmente um novo valor de "num".
120 - Desloca para x,y.
130 - Junta x,y a L,M
140 - Se ao somar a coordenada x da primeira bola à sua velocidade obtiver um resultado equivalente a lançá-la para fora do visor, inverte o sentido do movimento, negando a velocidade.

150 - Ver 140.
160 - Ver 140.
170 - Ver 140.
180 - Soma as velocidades da primeira bola às suas coordenadas.
190 - Faz o mesmo para a segunda bola.
210 - Produz um número aleatório entre 0 e 199, inclusivé
220 - Se o número é 1 faz RUN,limpando o visor e criando um novo padrão.
230 - De outro modo desenha a linha seguinte.
1000 - Define a velocidade x da primeira bola, dando-lhe um valor aleatório na gama - V a + V
1010 - Ver 1000
1020 - Ver 1000.
1030 - Ver 1000.
1040 - Escolhe um valor aleatório para "num". Note que esta variável não pode ter o valor zero.
1050 - Volta ao programa principal.

O leitor pode produzir linhas mais afastadas ou próximas entre si alterando os valores de U eV na linha 50.

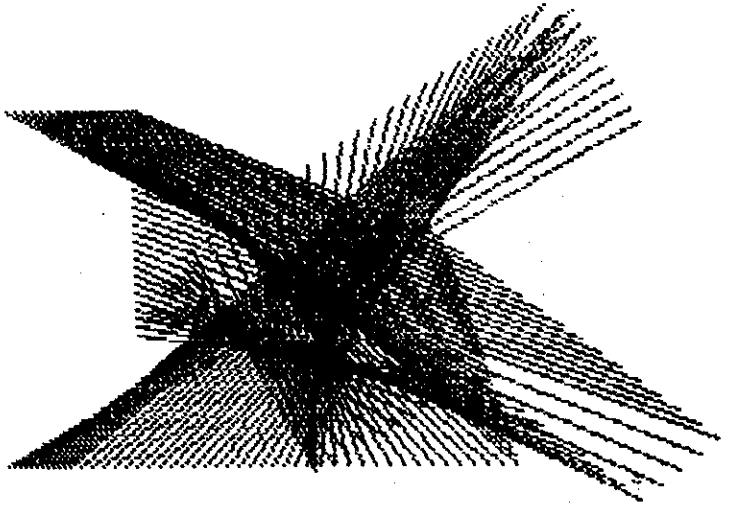
É interessante alterar a instrução DRAW da linha 130 de modo a desenhar uma curva - mas tenha cuidado com a possibilidade de a linha sair fora do visor, provocando a paragem da execução com uma mensagem de erro...

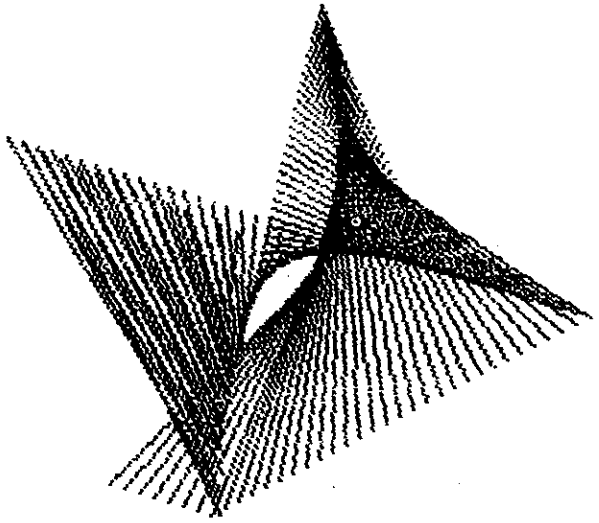


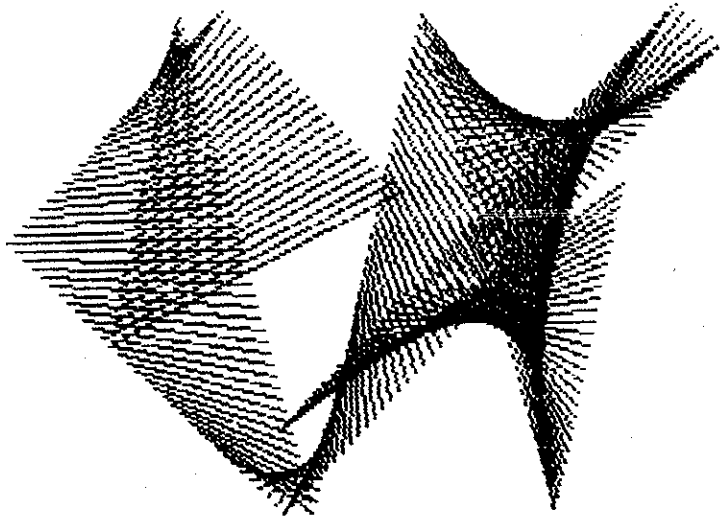
```
    5 BORDER 7: PAPER 7: CLS : IN
K 0
   10 LET x=INT (RND*256)
   20 LET y=INT (RND*176)
   30 LET l=INT (RND*256)
   40 LET m=INT (RND*176)
   50 LET u=15: LET v=7
   60 DEF FN r(x)=INT (RND*x)
  100 GO SUB 1000
  110 REM repetir
  115 LET num=num-1
  116 IF num=0 THEN GO SUB 1000
  120 PLOT x,y
  130 DRAW l-x,m-y
  140 IF x+a>255 OR x+a<0 THEN LE
T a=-a
  150 IF y+b>175 OR y+b<0 THEN LE
T b=-b
  160 IF l+c>255 OR l+c<0 THEN LE
T c=-c
  170 IF m+d>175 OR m+d<0 THEN LE
T d=-d
  180 LET x=x+a: LET y=y+b
  190 LET l=l+c: LET m=m+d
  200 REM UNTIL FALSE
```

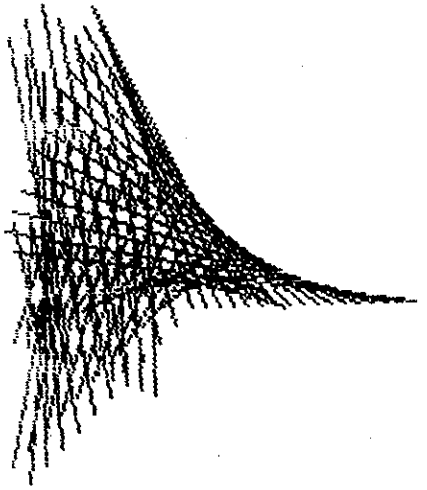


```
  210 LET cont=FN r(200)
  220 IF cont=1 THEN RUN
  230 GO TO 110
 1000 LET a=FN r(u)-v
 1010 LET b=FN r(u)-v
 1020 LET c=FN r(u)-v
 1030 LET d=FN r(u)-v
 1040 LET num=FN r(20)+10
 1050 RETURN
```



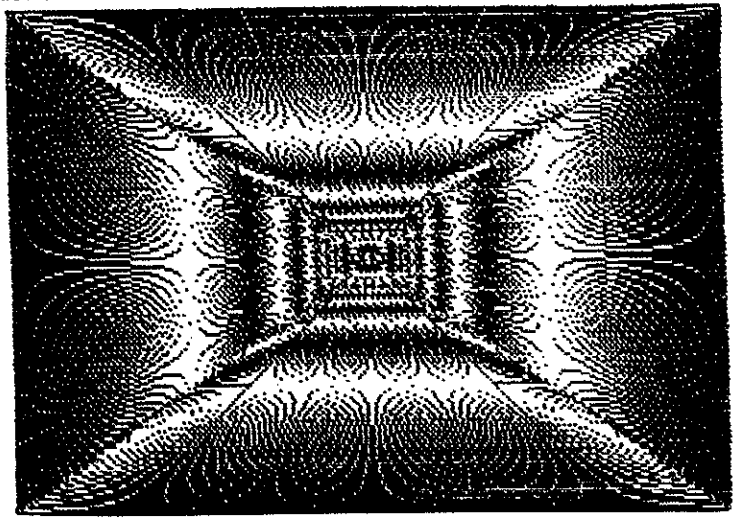






MOIRÉ

Este programa desenha padrões de Moiré. Representa uma série de linhas desenhadas a partir do centro do visor para cada



ponto da periferia deste. Como estas linhas são desenhadas em OVER 1, obtém-se o efeito apresentado mais adiante.

Talvez o leitor deseje modificar o programa de modo a utilizar um ponto central diferente. Experimente eliminar as referências OVER, e aumentar o STEP nas duas instruções FOR das linhas 10 e 50 para cerca de 4. Obterá então padrões mais realistas, mas devido à resolução relativamente baixa do visor do Spectrum o efeito não é tão bom como o obtido por exemplo no computador BBC.

A cor não dá bom resultado com este programa devido à grande proximidade dos pontos desenhados.

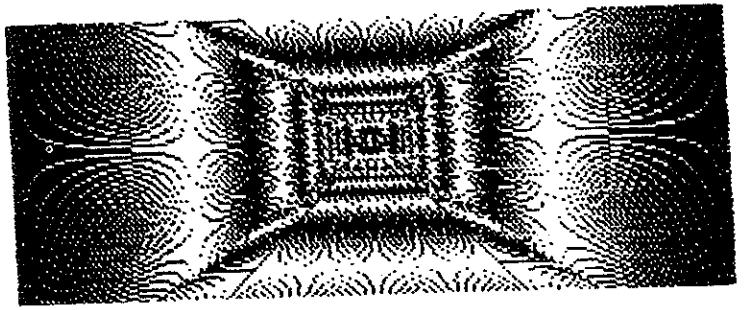
```
  1 PAPER 0: INK 7: BORDER 0: C
LS
  5 REM PADRÕES MOIRÉ
  6 REM By Jeremy Ruston
  7 REM ********************
 10 FOR x=0 TO 255
 20 PLOT x,0
 30 DRAW OVER 1;255-x*2,175
 40 NEXT x
 50 FOR y=0 TO 175
 60 PLOT 0,y
 70 DRAW OVER 1;255,175-y*2
 80 NEXT y
```

```
  5 PAPER 0: CLS : BORDER 0
 10 FOR X=0 TO 255
 20 PLOT X,0
 30 DRAW OVER 1;255-X*2,175
 40 NEXT X
 50 FOR Y=0 TO 175
 60 PLOT 0,Y
 70 DRAW OVER 1;255,175-Y*2
 80 NEXT Y: REM © J. Ruston
 85 LET S=RND+.5
 90 FOR X=255 TO 0 STEP -S
100 PLOT X,0
110 DRAW OVER 1,255-X*2,175
120 NEXT X
130 FOR Y=0 TO 175 STEP S
140 PLOT 0,Y
150 DRAW OVER 1,255,175-Y*2
```



```
160 NEXT Y
165 PAUSE 200
170 INK RND*7: PAPER 0: CLS
180 GO TO 85
```

```
  5 REM Spinning triangles
  6 REM © Jeremy Ruston
  8 REM ************************
  9 INK 6: PAPER 0: BORDER 0: C
LS
 10 LET L=RND*256
 20 LET M=RND*176
 30 LET N=RND*256
 40 LET O=RND*176
 50 LET P=RND*256
 60 LET Q=RND*176
 70 LET A=RND*5
 80 LET B=RND*5
 90 LET C=RND*5
100 LET D=RND*5
110 LET E=RND*5
120 LET F=RND*5
130 REM REPEAT
140 PLOT L,M
150 DRAW N-L,O-M
160 DRAW P-N,Q-O
170 DRAW L-P,M-Q
171 FOR X=1 TO 50: NEXT X
180 IF L+A>255 OR L+A<0 THEN LE
T A=-A
190 IF M+B>175 OR M+B<0 THEN LE
T B=-B
200 IF N+C>255 OR N+C<0 THEN LE
T C=-C
210 IF O+D>175 OR O+D<0 THEN LE
T D=-D
220 IF P+E>255 OR P+E<0 THEN LE
T E=-E
230 IF Q+F>175 OR Q+F<0 THEN LE
T F=-F
240 CLS
250 LET L=L+A: LET M=M+B
260 LET N=N+C: LET O=O+D
270 LET P=P+E: LET Q=Q+F
280 REM até ser falsa
290 GO TO 130
```



DECLARAÇÃO POINT

A declaração POINT corresponde a PLOT do mesmo modo que SCREENS corresponde a PRINT AT. Ou seja, pode-se usar POINT para determinar a presença ou não de um ponto de tinta numa determinada posição.

```
10 REM PLOT...POINT
20 FOR A=1 TO 100
30 PLOT RND*100,RND*100
40 NEXT A
50 LET COUNT=0
60 FOR G=1 TO 1000
70 IF POINT (RND * 100, RND * 100)=1 THEN
BEEP 1,1: LET COUNT= COUNT + 1; PRINT AT 1,1;
"PERCENTAGEM DE ÊXITOS É"; COUNT /G *100;" % "
; AT 3,1; "NÚMERO DE TENTATIVAS: ";G
```

Vemos aqui um programa muito simples para demonstrar a utilidade desta declaração. O programa espalha alguns pontos pelo visor, verificando depois a presença destes de modo aleatório. De cada vez que POINT (ver a linha 70) encontra um PLOT na posição que está a verificar, soa um BEEP. O programa deve produzir uma percentagem de êxitos de cerca de 1%.

Se POINT (X,Y) é igual a 1, isto significa que existe um ponto de tinta nessa posição. Se não, POINT (X,Y) é igual a zero. Experimente executar o programa anterior com o ciclo A prolongado até

1000, e o G até 10000. A percentagem de êxitos deve ser superior cerca de 10 vezes (ou seja, equivaler a aproximadamente 10%). O leitor consegue dizer porquê?

8
A IMPRESSORA

Existem três comandos associados à impressora — LLIST, LPRINT e COPY.

LLIST — Esta instrução passa toda a listagem do programa para a impressora. Não é possível "convencer" o computador a imprimir apenas uma parte do programa. A instrução actua do mesmo modo que LIST, excepto no que se refere a imprimir em papel e não no visor da televisão.

LPRINT — Esta instrução equivale igualmente a PRINT, imprimindo em papel e não no visor. Pode ser usada em modo directo, por exemplo LPRINT "Viva!", ou como instrução de programa, por exemplo 10 LPRINT "A resposta é":a.

COPY — Esta instrução copia todo o conteúdo do visor, depois de ter sido executado um programa, para a impressora. Como esta não representa as cores, nem permite dispor por exemplo de um fundo escuro e tinta clara, a cópia na impressora pode ser considerada menos interessante do que a imagem do visor. Experimente usar a ordem INVERSE em pontos definidos para melhorar a cópia em papel.

9
NÚMEROS ALEATÓRIOS

Os números aleatórios são muito úteis para jogos. Examinemos o modo como estes números são produzidos, e utilizemo-los em alguns programas simples.

O computador permite-nos produzir dois números aleatórios de vírgula flutuante entre zero e um.

Escreva e execute o programa seguinte. para ver uma possível gama de números entre zero e um:

```
10 PRINT RND
20 GO TO 10
```

Obterá uma lista de números semelhante à que se segue:

```
.00112291504      0.69433594
.08581543         .075531006
0.43719482        0.6658763
0.79025269        0.941253566
0.2691803         0.594086569
0.18934631        0.556688477
0.201866904       0.766686096
0.14257813        0.51483154
                  0.61291504
```

O leitor verificará que os números aleatórios inteiros são em geral muito mais úteis do que estes números entre zero e um. Para obter números inteiros. utilize uma instrução do tipo INT (RND * 30)+ 1w. Execute o programa seguinte: é provável que obtenha uma lista de números como a representada abaixo:

```
10 REM Inteiros aleatórios
20 LET A=INT (RNDX 100)+1
30 PRINT TAB 8:A
40 GO TO 20
```

```
14
93
19
15
51
9
41
13
72
70
86
28
44
66
15
```



```
60
96
38
36
26
85
1
```

O computador parte do número indicado entre parêntesis (a que se chama *argumento* da função) e escolhe números aleatórios entre um e esse número. Para obter números aleatórios negativos. basta colocar um sinal menos à frente da palavra INT. Experimente fazê-lo e execute novamente o programa: obterá um resultado do seguinte tipo:

```
-97               -88
-68               -78
-40               -50
-88               -80
-46               -6
-91               -61
-68               -9
-42               -15
-86               -46
-52               -87
-9                -25
```

O leitor pode usar o gerador de números aleatórios em qualquer aplicação em que se torne necessário simular uma actividade aleatória. como por exemplo a distribuição de sementes num jardim. a distribuição de núvens no céu. ou o resultado do lançamento de dois dados. O programa seguinte indica os resultados do lançamento de um dado com seis faces. Escreva-o e execute-o algumas vezes.

```
10 REM *Lançamento de dados*
20 PRINT "Quantas vezes deseja rolar"
30 PRINT "o dado?"
40 INPUT A
50 CLS
60 PRINT "Resultados obtidos dos"; A;" Rolamentos"
```

```
 70 FOR B=1 TO A
 80 LET C=INT (RND*6) +1
 90 PRINT ,C
100 NEXT B
```

No visor serão impressos valores do seguinte tipo:

Resultados obtidos:

6
5
6
6
3

TOURADA

Vamos ver em seguida um jogo muito simples que ilustra o funcionamento do gerador de números aleatórios. O jogo não é de facto particularmente interessante, mas vale a pena escrevê-lo e executá-lo. Depois de ter jogado algumas vezes volte ao livro para estudar um pouco o programa. Ficará bastante satisfeito com o muito que já aprendeu...

O leitor é um toureiro. O touro atacá-lo-á 10 vezes. O toureiro escolhe um número aleatório entre um e três e o touro também. Enquanto os números forem diferentes você mantém-se em jogo. Se o touro escolhe o mesmo número, o jogo termina. O utilizador recebe uma pontuação no fim.

```
 10 REM tourada
 20 LET pontuação=0
 30 FOR G=1 TO 10
 40 PRINT AT 4,4; INK 2; "O touro cor
re sobre si!"
 50 PRINT ' TAB 4; INK 1; "Qual o seu
movimento (1 a 3)?"
 70 INPUT A
 80 IF A<1 OR A>3 THEN GO TO 70
 90 LET B=INT (RND*3) +1
```



```
100 IF A=B THEN GO TO 220
120 PRINT ' TAB 4; INK 4; "Olé!";INK 2;G
130 BORDER RND*7
140 PRINT ' ' INK 2; "O touro escolheu";B
150 PRINT ' INK 4; "Você escolheu";A
160 FOR H=1 TO 10
170 BEEP .1,RND*50-RND*50
180 NEXT H
190 CLS
200 NEXT G
210 GO TO 230
220 PRINT ' INK 2; "Foi atingido pelo touro!"
230 FOR T=1 TO 50: BEEP .05,RND
*50: NEXT T
240 PRINT ' INK 2; "PONTUAÇÃO"
;100*(G-1)
```

Um exemplo de execução:

O touro corre sobre si!
Qual o seu movimento (1 a 3)?
Foi atingido pelo touro!

Note que o apóstrofo (') das linhas 120, 140, 150, 220 e 240, que se obtém carregando na tecla 7, é usado para deslocar o ponto de impressão de uma linha para baixo.

Estudemos este programa linha a linha:

10 REM com título do jogo

20 Define a variável "pontuação", inicializando-a para o valor 0. Dentro em pouco estudaremos as variáveis.

30 Inicia o ciclo FOR /NEXT, a fim de contar até 10. Estes ciclos serão discutidos mais adiante.

40 Imprime no visor a informação de que o touro está a atacar.

50 Pede ao jogador um número entre 1 e 3.

70 Aceita o número escrito pelo jogador.

80 Verifica se o número se encontra entre 1 e 3; se não, reenvia à linha 70, obrigando o utilizador a escrever outro número.

90 Define B. um número "escolhido" aleatoriamente pelo touro também entre 1 e 3.
100 Compara o número do jogador (A) com o touro (B). e se forem iguais envia para a linha 220 que o avisa de que se pode despedir do mundo dos vivos.
120 Cumprimenta o jogador pelo modo como recebeu o touro.
130 Altera a cor da margem.
140 Diz ao jogador o número do touro.
150 Recorda ao jogador o seu próprio número.
160-180 Introduz um pequeno intervalo. com música. antes do ataque seguinte...
190 Limpa o visor.
200 Volta a executar o ciclo.
210 Se o jogador sobreviveu 10 vezes. imprime a pontuação.
220 Diz ao toureiro que foi atingido...
230 Pequeno intervalo com música.
240 Imprime pontuação.

Se ler bem esta explicação e estudar a listagem. deve aprender já bastante sobre programação. Existe no programa um certo número de ordens específicas que devemos estudar em seguida com algum pormenor. mas o leitor já começará provavelmente a aperceber-se de como se constrói um programa.

10
VARIÁVEIS

O leitor notou certamente no programa anterior que se usavam letras para representar números. A letra A foi atribuída a um número (na linha 70) entre um e três. e B foi atribuída do mesmo modo na linha 90. As letras A e B deste programa são chamadas variáveis.

Existem dois tipos de variáveis: numéricas e alfanuméricas. ou de cadeia.

É possível usar como variável numérica praticamente qualquer combinação de letras. desde que comece por uma letra e não contenha qualquer símbolo ou sinal de pontuação. "Pontuação" ou "D 17" são nomes válidos para variáveis. mas "1D7" não. As variáveis numéricas são muito simples de usar. Pode-se atribuir uma

variável deste tipo a qualquer número que se encontre dentro da gama tratada pelo computador. O Spectrum ignora os espaços existentes no interior do nome das variáveis. e não distingue entre letras maiúsculas e minúsculas (pelo que aS é o mesmo que AS).

A propósito. como o leitor já sabe provavelmente. o computador utiliza a notação científica para apresentar os números muito grandes no visor. indicando um único algarismo inteiro e até oito casas decimais. seguido da letra E (símbolo de exponenciação) e a potência de dez pela qual se deve multiplicar o número. Escreva e execute o programa seguinte que ilustra o uso da variável A. atribuída a um número que está a ser multiplicado repetidamente por 10 e impresso em seguida.

```
10 REM Notação científica
20 LET A= 1234
30 PRINT A
40 LET A= 10*A
50 GO TO 30
```

```
1234
12340
123400
1234000
12340000
1.234E+8
1.234E+9
1.234E+10
1.234E+11
1.234E+12
1.234E+13
1.234E+14
1.234E+15
1.234E+16
1.234E+17
1.234E+18
1.234E+19
1.234E+20
1.234E+21
1.234E+22
1.234E+23
1.234E+24
```

Note que quando o número passa a ter mais de oito algarismos (neste caso, 12340000) começa a ser impresso em notação científica, sob a forma de um número, uma vírgula, mais números depois da vírgula, a letra E e uma potência de 10. Experimente prever durante quanto tempo será necessário executar o programa até exceder o maior número que a máquina pode tratar, e depois obrigue a máquina a executá-lo até ao fim para ver se acertou...

Se olharmos para a listagem poderemos aprender alguma coisa sobre variáveis. A variável é atribuída escrevendo muito simplesmente a palavra LET, o nome da variável, o sinal de igual e o valor que lhe queremos atribuir. Se dissermos LET A= 99, a instrução PRINT A escrita em seguida provocará o aparecimento desse número no visor. A linha 40 parece um pouco estranha. O asterisco (*) indica "multiplicação" em linguagem BASIC. A linha 40 parece dizer que A é igual a 10 vezes A, o que em termos matemáticos não é verdadeiro. É no entanto assim que se utiliza a instrução LET em BASIC.

VARIÁVEIS DE CADEIA

As variáveis de cadeia (sequências alfanuméricas) são designadas por uma letra seguida do sinal S. Escreva LET AS= "Olá", carregue em ENTER e escreva PRINT AS. ENTER: a máquina responderá imprimindo "Olá" no visor. O utilizador pode incluir tudo o que quiser, números, letras, símbolos ou sinais de pontuação no interior das aspas, designado o conjunto por uma variável de cadeia. Uma série de letras ou qualquer outro caracter dentro de aspas é conhecida pelo nome de "cadeia".

"GRILOS"

Existe, curiosamente, uma relação entre a temperatura ambiente e o número de vezes que o grilo canta por minuto. O programa que se segue, que ilustra o uso de variáveis com nomes bastante compridos, relaciona o número de vezes que o grilo canta por minuto com a temperatura em graus Fahrenheit. Escreva-o e execute-o algumas vezes. Note que a variável "canto" é inicializada para 80 na linha 20. Este valor é relacionado com o da temperatura na linha 30, sendo esta última variável usada na instrução PRINT da linha 40. A variável "canto" é incrementada de um número aleatório entre um e sete na linha 60; depois de um intervalo ou atraso (linhas 70 e 80) o programa pode correr durante bastante tempo (até exceder o maior número que a máquina pode tratar) se não for interrompido usando o comando BREAK.

```
A temperatura é 62 graus
quando o grilo canta 87 vezes

A temperatura é 63 graus
quando o grilo canta 90 vezes

A temperatura é 63 graus
quando o grilo canta 92 vezes

A temperatura é 64 graus
quando o grilo canta 96 vezes

A temperatura é 65 graus
quando o grilo canta 98 vezes
```

```
10 REM Canto do grilo
20 LET canto= 80
30 LET temperatura= INT ((canto /4) + 40.5)
40 PRINT "A temperatura é ";INK 2; temperatura; " graus"
50 PRINT "quando o grilo canta"; INK 2; canto; INK0; "vezes"
60 LET canto= canto+INT (RND * 7)
70 FOR J= 1 TO 100
80 NEXT J
90 POKE 23692,-1
100 GO TO 30
```

Se bem que seja mais demorado escrever nomes de variáveis bastante compridos, estes têm uma vantagem relativamente a nomes como A, B ou C2. De cada vez que os observar numa listagem saberá imediatamente aquilo que representam, sem necessidade de ir consultar a linha onde a variável é definida. Escreva e execute o programa seguinte.

```
10 REM * * Variáveis * *
20 LET WS= "O número é"
30 LET número= 3
50 PRINT W$; número
60 PRINT "O quadrado de": número
70 PRINT TAB 5: "é": número * número
80 PRINT "e a raíz quadrada"
90 PRINT "é": SQR (número)
```

Resumindo:

— Variável numérica — pode ter qualquer nome, desde que comece por uma letra e não contenha quaisquer símbolos ou sinais.

— Variável de cadeia ou alfanumérica — Trata-se de uma letra seguida pelo símbolo S, que é usada como nome de qualquer entidade contida dentro de aspas. Todas as variáveis são inicializadas ou alteradas usando instruções LET, seguidas do nome da variável, do sinal de igual e finalmente o valor que lhe é atribuído.

11
INPUT

A instrução INPUT é usada para obter informações do utilizador enquanto o programa está em execução. O computador pára quando atinge uma instrução INPUT, e espera que seja dada entrada a uma informação através do teclado antes de continuar a executar o programa.

Escreva e ponha em execução o programa seguinte, que mostra INPUTs numéricos em acção. O programa esperará que você indique um número e carregue em ENTER, esperando em seguida por um novo número. Depois de você ter carregado novamente em ENTER, imprimirá a soma dos dois números.

```
10 REM ** input **
20 INPUT x
30 PRINT ,x
40 INPUT y
50 PRINT ,y
60 LET z=x+y
70 PRINT ,"_____"
80 PRINT ,z
90 PRINT ,"_____"
```

```
469
247
____
?715
____
```

Note que o utilizador não faz a mais pequena ideia daquilo que este programa faz antes de o executar. Existe uma forma simples de rectificar isto, programando os INPUTs de modo a fazerem perguntas ao utilizador. O programa anterior pode ser facilmente alterado, de modo a que o utilizador não tenha qualquer dúvida sobre o que está a fazer.

```
10 REM ** input **
20 INPUT "Dê-me um número";X
30 PRINT ,x
40 INPUT "Outro número";Y
50 PRINT ,y
60 LET z=x+y
70 PRINT ,"_____"
80 PRINT ,z
90 PRINT ,"_____"
```

Se executar este programa verificará que o computador escreve as frases na parte inferior do visor. esperando depois pela sua resposta.

Note que muitos dos comandos usados para controlar a saída PRINT podem também ser usadas para controlar a entrada INPUT. como se ilustra no programa seguinte.

COMBATE

```
  5 REM ** Combate      **
  6 POKE 23609,100
 10 LET Pontuação= 0
 15 FOR j=1 TO 20
 20 PRINT AT 1,8; INK j/3; "Tentati
va";j
 30 INPUT INK j/3; "Indique um número en
tre 1 e 10"; a
 40 IF a<1 OR a>10 THEN GO TO 3
0
 50 PRINT AT 10,0; INK 3; "O seu
número é";a;AT 12,6; INK 1; "Pontuação:";pontuação

 60 FOR g=1 TO 4
 70 LET b=INT (RND*10)+1
 80 PRINT AT 3,3; INK 2; b," "
 85 FOR m=1 TO 10: BEEP .01,3.5
*m: NEXT m
 90 IF b=a THEN GO TO 110
100 NEXT g
110 IF a=b THEN LET pontuação= pontuação
+1: PRINT AT 14,6; INK 4; "Bem joga
do!"
140 IF a<>b THEN PRINT AT 14,6;
INK 0; FLASH 1; "Pouca sorte"
150 PRINT AT 12,6; INK 2; "A pon
tuação é "; pontuação
160 IF pontuação=5 THEN GO TO 250
170 FOR t=1 TO 20
180 BEEP .01,2*t: BEEP .01,20-t
 NEXT t
190 CLS
200 NEXT j
210 PRINT BRIGHT 1; INK 2; "O jogo
terminou"
```



```
220 PRINT FLASH 1; BRIGHT 1; IN
K 2; "E você só pontuou ";pontuação
230 PRINT ' INK 1; "O seu valor é";pontuação
/.05;"por cento"
240 STOP
250 PRINT INK RND*6; FLASH 1;TA
B 4; "Conseguiu!"
260 FOR m=1 TO 4: BEEP .01,RND*
50: PAUSE 3: NEXT m
270 PRINT INK RND*6; FLASH 1;TA
B 8; "Ganhou!!"
280 POKE 23692,-1
290 GO TO 250
```

Exemplo de execução:

Tentativa 1

5

O seu número 3
A pontuação é 0
Pouca sorte

Em "Combate" você escolhe um número entre um e dez. O computador escolhe até quatro números entre um e dez. Se qualquer deles for igual ao seu, a sua pontuação é aumentada de um. Se conseguir uma pontuação de cinco, em vinte tentativas, ganha. Se não obtém uma pontuação final em termos de percentagem. Depois de ter executado o programa, volte ao livro para o estudar linha a linha. Se bem que este programa seja bastante trivial, a sua execução e estudo permitir-lhe-ão aumentar o seu conhecimento de vários aspectos da linguagem BASIC, além de ilustrarem o funcionamento da instrução INPUT.

5 Título
6 Produz um som ao carregar nas teclas
10 Coloca a zero a variável "pontuação"
15 Inicializa o ciclo FOR /NEXT principal para contar as tentativas
20 Imprime o número da tentativa

30 Aceita a entrada para a variável A. usando J para definir a cor.
40 Verifica se a entrada é válida
50 Imprime o número escolhido e a pontuação.
Note que as instruções PRINT podem ser "encadeadas" deste modo. com pontos e vírgulas actuando como separadores e usando TAB ou AT.
60-100 Produz até quatro números. Depois de cada um destes ser produzido (linha 70). é impresso (linha 80). ouve-se um som (linha 85). e compara-se o número com o do jogador (linha 90)
110 A pontuação é aumentada de um se a tentativa teve êxito. e o programa imprime "Bem jogado!"
140 Imprime "Pouca sorte!" se a tentativa não teve êxito.
150 Imprime novamente a pontuação.
170-180 Pequeno atraso com BEEP's antes da nova execução.
190 Limpa o visor
200 Final do ciclo FOR /NEXT principal.
210-240 Final do jogo. se perder
250-290 Final do jogo. se ganhar.

JUROS COMPOSTOS

O programa que se segue mostra novamente o comportamento da instrução INPUT. ilustrando também o uso de nomes explícitos para as variáveis. o que torna mais fácil a compreensão da listagem. Talvez interesse ao leitor gravar este programa em cassete. dado que é passível de alguma aplicação prática.

```
  10 REM Juro
  20 REM Simples e composto
  30 INPUT INK 1;TAB 6; "Capital?";capital
  40 INPUT INK 2;TAB 6; "JURO?"; Juro
  60 INPUT INK 4;TAB 4; "Durante quantos
anos? ";anos
  80 PRINT INK 2; "ANO"  TAB 7; "S
IMPLES"; TAB 17; "COMPOSTO";  TAB 27;
"DIF" ."
  90 POKE 23692,-1
 100 PRINT INK 1;"
 110 FOR M=1 TO anos
 120 LET simples= capital + M*capital* (juro /100)
 130 LET composto= INT (100* capital* (1 + juro /100)
+M) /100
 140 LET DIFF=INT (100* (Composto
-simples +  .005)) /100
 160 PRINT M;TAB 7; "simples"; TAB 17;
"composto"; TAB 27; "DIF"
 170 NEXT M
```

Exemplo de execução (em libras):

ANO	SIMPLES	COMPOSTO	DIF.
1	108.25	108.25	0
2	116.5	117.16	0.66
3	124.75	126.84	2.09
4	133	137.31	4.31
5	141.25	148.64	7.39
6	149.5	160.9	11.4
7	157.75	174.17	16.42
8	166	188.54	22.54
9	174.25	204.1	29.85
10	182.5	220.94	38.44
11	190.75	239.17	48.42
12	199	258.9	59.9

Este programa determina os juros simples e compostos. para um capital e uma taxa de juro por nós indicada. ao longo de um certo número de anos. O exemplo utiliza um capital de $100. a 8.25% durante 12 anos.

Para parar um programa durante um INPUT de uma cadeia alfanumérica (notar que BREAK não actua durante estas entradas) utiliza-se a tecla de cursor 5 para sair do interior das aspas. escrevendo em seguida STOP (A+CAPS SHIFT) seguido de ENTER. Se se trata de um INPUT numérico. sem aspas. basta escrever STOP e ENTER. Em ambos os casos pára a execução do programa com a mensagem 9.

É útil poder rejeitar INPUTs não válidos. *antes* de poderem provocar o "estoiro" do programa.

Se convidar outra pessoa para jogar, analise as suas respostas do seguinte modo:

```
555 INPUT "Deseja outro jogo? ";RS
556 IF RS (1)= "S" THEN RUN
```

Existe uma lei qualquer que diz que o utilizador tem o mau hábito de responder carregando apenas em ENTER — deixando a máquina com uma cadeia nula. Não há portanto melhor solução do que RS(1) - não existe, o que lhe será indicado muito rapidamente pelo computador sob a forma de uma mensagem de erro!

Vejamos outro método possível:

```
550 DIM R$(1)
555 INPUT     "Deseja outro jogo?";R$
556 IF R$="S" THEN RUN
```

Como RS foi DIMensionada previamente, terá de consistir num único caracter, seja o que for que se escreva. Se se carrega apenas em ENTER.RS será um espaço dado que é isso que é atribuido a RS depois de uma instrução DIM e uma instrução INPUT nula não alterará a variável. Se a INPUT tiver vários caracteres de comprimento, só caberá em RS um deles. Se este caracter for S, o programa será executado mais uma vez. Este método tem a vantagem de, no caso de o utilizador escrever uma resposta muito longa como "Sim, por favor, meu querido computador, gostaria muito de tornar a executar este belíssimo programa", a máquina não necessita de guardar tudo isto na memória. O segundo método é mais convencional e utiliza menos uma linha de programa do que a rotina anterior, se bem que guarde toda a resposta em memórias sem necessidade de o fazer:

```
550 DIM R$(1)
555 INPUT "Deseja outro jogo?";RS
556 IF CODE R$=CODE "S" THEN RU
N
```


Este programa explica-se a si mesmo - se o primeiro caracter da resposta tem um código igual ao do caracter S (ou seja, é S) então o programa será executado. As entradas nulas são consideradas como significando que o jogador não quer jogar outra vez, dado que carregar simplesmente em ENTER produz uma cadeia nula e o código de uma cadeia nula é 0, como o de um espaço.

Verificar a primeira letra da INPUT do utilizador é bastante fácil como o leitor já comprendeu. Torna-se um pouco mais difícil quando se deseja verificar uma INPUT inteira, por exemplo para ter a certeza de que o utilizador deu entrada a quaisquer sinais de pontuação ou incluiu letras numa INPUT numérica. Vejamos primeiramente as entradas alfabéticas. Os operadores relacionais <, >, >=, <=, <> são aqui muito úteis. Considere o caso de uma INPUT em que é necessária uma palavra e não se deve dar entrada a nenhuma outra coisa.

```
 10 INPUT A$
 15 IF A$="" THEN GO TO 10
 20 FOR A=1 TO LEN A$
 30 IF A$(A) <"A" OR A$(A) >"Z" T
HEN GO TO 10
 40 NEXT A
```

A linha 15 assegura que as entradas nulas (ou seja, as INPUT's respondidas apenas com ENTER) são rejeitadas. O ciclo que se inicia na linha 20 varre toda a cadeia da entrada caracter a caracter, e se descobre um caracter que não é uma letra envia uma mensagem para o exterior pedindo uma nova entrada, porque o programa salta imediatamente para a linha 10. Tal como está, no entanto, este programa não permite o uso de espaços entre as palavras.

Altere a linha 30 do seguinte modo para poder introduzir espaços:

```
 30 IF (A$(A) <"A" OR A$(A) >"Z")
AND A$(A) <>" " THEN GO TO 10
```

Este esquema pode ser facilmente estendido de modo a incluir sinais de pontuação, letras e espaços (ou seja, excluindo números, palavras-chave, símbolos, etc.). É pouco mais difícil detectar a

presença de uma dada palavra na entrada, por exemplo quando existe uma linha no final de um programa convidando o utilizador a executá-lo novamente, e se este responde "sim" o programa automaticamente volta ao seu início. É muito fácil colocar a INPUT num ciclo e indicar a palavra "sim" do seguinte modo:

```
7010 INPUT "Deseja jogar novamente?";A$
7020 FOR A=1 TO LEN A$-2
7030 IF A$(A TO A+2)="SIM" THEN
RUN
7040 NEXT A
7050 STOP
```

Se se escrever "SIM" ou "SIM, POR FAVOR", o programa volta ao início. Se for dada a entrada a uma palavra cujo comprimento seja inferior ao da palavra procurada (excepto a cadeia nula), provocará um erro porque a linha 7030 espera uma entrada de comprimento pelo menos igual à palavra procurada. A cadeia nula é aceite porque então LEN AS é zero, transformando a linha 7020 em FOR A = 1 TO 0, pelo que a cadeia é completamente ultrapassada, não se colocando o problema. Experimente ainda escrever ASSIMPTOTA - o programa volta igualmente ao início porque detectou a presença das três letras SIM. Torna-se portanto necessária uma rotina que detecte se o caracter de ambos os ladós das três letras é diferente de uma letra. É necessário ter cuidado ao fazê-lo porque não podemos examinar o caracter antes e depois das três letras no caso de estas ocorrerem no início ou no final da resposta do utilizador — se as tentarmos examinar neste caso produzirão uma mensagem de erro. Apresentamos em seguida uma rotina que tem este facto em conta, acrescentando caracteres desnecessários no início e no final da resposta AS.

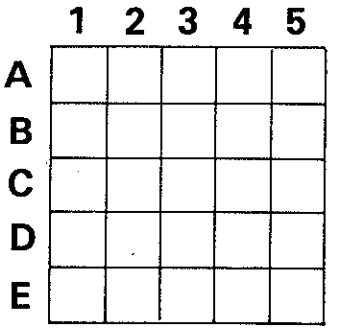
```
7010 INPUT "Deseja jogar novamente? ";AS
7015 LET A$=" "+A$+" "
7020 FOR A=2 TO LEN A$-3
7030 IF A$(A TO A+2)="SIM" AND (
A$(A-1)<"A" OR A$(A-1)>"Z") AND
(A$(A+3)<"A" OR A$(A+3)="Z") THE
N RUN
7040 NEXT A
7050 STOP
```


A rotina permite usar qualquer comprimento de resposta até ao comprimento máximo de uma cadeia. Se se deseja alterar a palavra procurada pela rotina pode valer a pena indicá-la por uma variável cujo valor alfabético é alterado pelo program ou recorrendo a um INPUT colocadonum ponto qualquer do programa. Será necessário fazer as seguintes modificações na rotina para utilizar diferentes palavras:

```
7010 INPUT "Indique palavra a procurar";S$
7020 INPUT "Indique frase";A$
7040 LET A$=" "+A$+" "
7050 LET LS=LEN S$
7060 LET LA=LEN A$
7070 FOR A=2 TO LA-LS
7080 IF A$(A TO A+LS-1)=S$ AND (
A$(A-1)<"A" OR A$(A-1)>"Z") AND
(A$(A+LS)<"A" OR A$(A+LS)>"Z") T
HEN RUN
7090 NEXT A
```

Se a rotina é um pouco extensa, desde que se utilize sempre a mesma palavra pode-se evitar o recurso a SS e LS e substituí-las pela palavra e comprimento exactos. Veja o exemplo dado anteriormente a propósito da palavra "SIM".

Observemos agora um outro tipo de INPUT muito usado em jogos — tanto jogos com redes de posições, como acontece em algumas aventuras, como em jogos de tabuleiro — isto é, uma INPUT envolvendo a indicação das coordenadas de um local. Pode por exemplo ser usado um tabuleiro do seguinte tipo:



As coordenadas são normalmente indicadas sob a forma de uma letra seguida de um número, por exemplo C3 ou C3B4, neste último caso se é necessário indicar a posição de partida e a de chegada, por exemplo em xadrez. Se se pretende dar entrada às coordenadas sob a forma de uma letra seguida de um número, é provável que mais tarde ou mais cedo alguém — deliberada ou acidentalmente — apresente as coordenadas pela ordem inversa e bloqueie o programa. A rotina que se segue detectará automaticamente o erro da ordem dos caracteres. Aplica-se ao tabuleiro da figura anterior, e para a modificar para outras redes de posições basta alterar os caracteres entre aspas nas linhas 30 e 40. A linha 50 foi incluída apenas para se poder observar o efeito da rotina.

```
  10 INPUT A$
  20 IF LEN A$<2 THEN GO TO 10
  25 LET A$=A$( TO 2)
  30 IF A$(1)>="1" AND A$(1)<="5
" AND A$(2)>="A" AND A$(2)<="E"
THEN LET A$=A$(2)+A$(1)
  40 IF A$(1)<"A" OR A$(1)>"E" O
R A$(2)<"1" OR A$(2)>"5" THEN GO
 TO 10
  50 PRINT A$
```

A rotina é muito rápida de executar. É bastante difícil bloquear o programa com esta rotina, mas estou certo de que um leitor inteligente descobrirá um modo de o fazer. Se o descobrir, modifique a rotina de modo a evitar que o erro ocorra de novo.

A rotina para coordenadas com quatro caracteres é mais complexa. A ideia deste INPUT consiste em dar a conhecer ao computador o local de onde se parte e aquele para onde se vai: por exemplo, E3D4 significa que se está a mover uma peça de E3 para D4. Comecemos por dispor as letras e os números ordenadamente.

```
  10 INPUT A$
  20 IF LEN A$<4 THEN GO TO 10
  30 LET A$=A$( TO 4)
  40 IF A$(1)>="1" AND A$(1)<="5
" AND A$(2)>="A" AND A$(2)<="E"
THEN LET A$( TO 2)=A$(2)+A$(1)
  50 IF A$(3)="1" AND A$(3)<="5"
```



```
 AND A$(4)>="A" AND A$(4)<="E" T
HEN LET A$(3 TO )=A$(4)+A$(3)
  60 IF A$(1)<"A" OR A$(1)>"E" O
R A$(2)<"1" OR A$(2)>"5" OR A$(3
)<"A" OR A$(3)>"E" OR A$(4)<"1"
OR A$(4)>"5" THEN GO TO 10
  70 PRINT A$
```

Note que pode encurtar estas duas rotinas usando a instrução DIM. No primeiro programa pode acrescentar

5 DIM AS(2)

e retire as linhas 20 e 25. No caso do segundo programa, acrescente

5 DIM AS(2)

e retire as linhas 20 e 30. Ambas as versões asseguram que a cadeia AS tem exactamente o comprimento desejado, não sendo mais curta nem mais comprida. Se der entrada a uma cadeia com mais de quatro caracteres na segunda rotina, o resto será ignorado. Se for mais curta do que quatro caracteres são acrescentados espaços no caso de usar a linha 5 (depois rejeitados na linha 60) ou rejeitados na linha 20 se estiver a usar a versão não modificada. Depois de vermos o modo de validar letras e números vejamos como podemos validar movimentos correctos. Será necessário observar o exemplo do tabuleiro apresentado anteriormente. Suponhamos que temos uma peça de damas no quadrado E3. Será necessário definir os movimentos válidos dessa peça. Como se sabe esta peça só pode mover-se um quadrado de cada vez, em diagonal para a frente. O quadrado final pode ser D2 ou D4. Antes de continuar, conseguirá o leitor descobrir a relação entre estas coordenadas?

Como a peça só se pode mover para a frente um quadrado de cada vez, deve terminar o movimento num quadrado cuja letra é alfabeticamente vizinha de E. Ora, no seu computador, os códigos das letras que se seguem alfabeticamente são seguidos, pelo que o código de D é igual ao de E menos 1. Assim, se o código da letra corresponde ao quadro de onde a peça parte não é superior em uma unidade ao do quadro final, o movimento não é legal. Por outro lado, o número do quadro de onde se parte deve ser igual ao do quadrado final menos 1. Nestas condições:

```
  65 IF (CODE A$(1)<>CODE A$(3)+
1) OR (CODE A$(2)<>CODE A$(4)+1)
```

```
OR (CODE A$(2)=CODE A$(4)-1) TH
EN GO TO 10
```

O leitor necessitará obviamente de adaptar estas rotinas aos casos concretos que ocorrem nos seus programas: apenas apresentamos aqui exemplos das rotinas que poderá estar interessado em incluir nos seus programas. Estes exemplos servem ainda para mostrar o tipo de raciocínio que é necessário fazer para resolver problemas deste tipo. Sugiro que siga sempre o método seguinte:

(1) Defina exactamente o que deseja conseguir.
(2) Defina exactamente o que é permitido, e aquilo que não será aceite como válido (por exemplo a cadeia nula).
(3) Como pode impedir as respostas ou situações não válidas, ou rejeitá-las no caso de ocorrerem?
(4) Tente descobrir mentalmente se a sua rotina faz aquilo que pretende ou não, usando alguns exemplos.
(5) Se pensa que a rotina está correcta, introduza-a no computador e experimente-a usando alguns valores permitidos a fim de verificar se os trata correctamente.
Depois de isto feito, experimente-a com todos os tipos possíveis de entradas (por exemplo indicando uma coordenada impossível, como F9 no nosso exemplo). Pode finalmente passar ao teste mais importante.
(6) Deixe um amigo experimentar a rotina, com "ordens" para tentar fazê-la falhar. As rotinas anteriores têm um erro, mas não lhe vou dizer qual...
Finalmente, estudemos as entradas numéricas. Limpe o computador com NEW e escreva o seguinte:

```
10 INPUT A
20 GO TO 10
```

Faça executar este pequeno programa e veja se consegue bloqueá-lo de qualquer modo: não deve ser difícil. Tente dar entrada de uma letra: experimente escrever STOP: tente indicar um número muito grande ou muito pequeno: ou experimente indicar uma palavra-chave ou um sinal aritmético como "+".

Os sinais aritméticos levam o computador a apresentar um erro de sintaxe, se bem que não parem o programa. As palavras-chave e os símbolos também provocam isto, mas letras levam o programa a parar com a mensagem de erro 2, o que significa que foi usada uma variável não definida. Variável? Sim - quando se responde com uma letra a um INPUT numérico o computador pensa que se está a dar entrada a uma variável, facto que por vezes se pode tornar bastante útil. Usando ainda este programa, indique 1 da primeira vez que lhe é pedido um valor, e na segunda vez escreva A. Verificará que o computador aceita esta entrada. De facto, da primeira vez a máquina atribuíu o valor 1 a A, e quando você pede que aceite o valor A, que a máquina já conhece, esta aceita-o. Escreva em seguida STOP. O programa pára com uma mensagem de erro. Experimente em seguida escrever PRINT A, e obterá o valor 1; conclui-se portanto que o programa parou antes de actualizar o valor de A. De facto, se o utilizador bloquear a máquina numa entrada numérica, o computador mantém geralmente o valor anterior dessa variável.

Isto não é particularmente importante, mas em certas circunstâncias, se conseguir fazer arrancar novamente o programa, a variável não será pelo menos desconhecida para o computador.

O modo mais simples de resolver estes problemas consiste em utilizar INPUT's de cadeias e fazer a sua avaliação usando VAL. Experimente:

```
10 INPUT A$
20 LET A=VAL A$
30 PRINT A
40 GO TO 10
```

O leitor verificará que é fácil bloquear a máquina mesmo neste caso, e que muitos dos problemas que ocorrem com as entradas numéricas parecem continuar a ocorrer neste caso. No entanto, este método apresenta a vantagem de não bloquear a execução até se aplicar VAL. Pode-se portanto tratar a cadeia antes de aplicar VAL, removendo todos os erros que possa conter, sem que a máquina páre - isto é, pode ***processar*** a cadeia alfanumérica. Note por outro lado que VAL pode actuar sobre tudo o que for numérico, não apenas com números. Experimente o seguinte:

```
PRINT VAL "RND"
PRINT VAL "SGN -7"
PRINT VAL "A*2"
```
(como é óbvio, esta instrução só funciona se tiver definido previamente A)
```
PRINT VAL "COS1"
```

Ocorrem problemas ao aplicar VAL no caso de se usarem variáveis não definidas, ou de se dar entrada a declarações não numéricas, palavras-chave ou símbolos. É necessário eliminar tudo isto antes de se usar a instrução VAL. O caso mais simples é aquele em que apenas são permitidas entradas numéricas, podendo-se proceder do seguinte modo:

```
10 INPUT A$
20 FOR F=1 TO LEN A$
30 IF A$(F)<"0" OR A$(F)>"9" T
HEN GO TO 10
40 NEXT F
50 LET A=VAL A$
60 PRINT A
```

O leitor consegue descobrir o que falha nesta rotina?

Evidentemente essa nossa velha amiga, a cadeia nula, que transforma o ciclo em FOR= 1 TO 0, tornando-o inútil, o que obriga a incluir a linha

15 IF AS= ""THEN GO TO 10

A rotina obriga-nos a dar entrada a um novo número se da primeira vez foi introduzido algo que não fosse um número. Este esquema pode ser ampliado de modo a permitir a entrada de símbolos aritméticos e nomes de variáveis se se quiser, mas não parece haver grande utilidade em fazê-lo.

```
30 IF (A$(F)<"0" OR A$(F)>"9")
AND (A$(F)<>"+" AND A$(F)<>"-")
THEN GO TO 10
```


Esta linha permite-lhe dar entrada ao símbolo de adição e exponenciação. Para permitir outras funções basta acrescentá-las no segundo grupo de aspas, ligando cada condição por um "AND" ("E" lógico). Pode acontecer que lhe seja útil fazê-lo algum dia.

Veiamos agora algumas ordens estreitamente relacionadas com a capacidade do computador para "pensar".

12
GO TO

É necessário que as linguagens de programação permitam enviar para diferentes linhas do programa enquanto este é executado. Se isto não fosse possível, o programa seria sempre executado em sequência, começando pelas linhas de número inferior e parando no fim. Uma das instruções que nos permite saltar para qualquer ponto do programa é GO TO. A instrução GO TO consiste num número de linha seguido pela palavra chave GO TO e outro número de linha, ou por uma operação (como por exemplo em GO TO 2*X, ou GO TO 200+340).

Se o computador encontra a instrução

140 GO TO 190.

salta imediatamente da linha 140 para a linha 190. Chama-se a isto um salto "incondicional". Ou seja, trata-se de um salto que é executado pelo programa independentemente de qualquer condição. Ao chegar à linha 190, o programa continua a executar as linhas por ordem, até chegar ao fim ou a qualquer outra instrução de salto que o obrigue a saltar para uma nova linha.

Podemos usar a declaração GO TO para produzir programas que continuem indefinidamente em execução. Isto pode ser bastante interessante, em particular no final de um jogo. Escreva e faça executar o seguinte:

```
10 PRINT INK RND*6;" ... Você ganhou!...";
20 POKE 23692,-1
30 BEEP .01,RND*50
40 GO TO 10
```

13
IF...THEN GO TO

A instrução IF aqui estudada tem uma função semelhante a GO TO. mas apenas enviará para a nova linha de programa se estiverem preenchidas certas condições. Realiza portanto um salto "condicionado". Esta instrução é constituída por um número de linha, seguindo-se as palavras IF THEN GO TO separadas por uma relação que deve ser definida antes de abandonar a linha. Existem por outro lado seis operadores que podem ser utilizados para comparar duas variáveis. São os seguintes:

= igual a
> maior do que
< menor do que
>< diferente de
> = maior ou igual a
< = menor ou igual a

Estes operadores são usados para ligar as palavras IF e THEN, formando a condição que deve ser respeitada.

Vejamos um exemplo:

70 IF Z >= 10 THEN GO TO 100

O computador compreenderá esta linha do seguinte modo: se (IF) o valor da variável Z é superior ou igual a 10, então (THEN) passar à linha 100. Se Z for inferior a 10, o programa continuará a execução normal, passando à linha que se segue.

Consegue-se dar assim ao computador uma capacidade para tomar decisões: é com base nela que a máquina parece pensar.

Como o leitor já terá talvez descoberto, o computador não é "indeciso" (a menos que lhe diga para sê-lo), e toma uma decisão clara sempre que enfrenta uma opção. Aquilo que faz depende no entanto do que lhe dizemos para fazer, normalmente depois da palavra THEN. Ilustremos isto com um exemplo simples, um programa que imprime números em palavras.

```
10 INPUT 'INK RND*6; "Indique um número entre 1 e 3";a
30 IF a=1 THEN PRINT "um "
40 IF a=2 THEN PRINT "dois "
50 IF a=3 THEN PRINT "três "
60 GO TO 10
```

O leitor não é forçado a incluir apenas uma condição entre IF e THEN. Voltando ao exemplo acima, suponhamos que só nos seria permitido sair do emprego às 18 horas se tivéssemos terminado um dado trabalho:

IF (se) forem seis horas AND (e) tiver acabado o seu trabalho,
THEN (então) vá para casa.

Quando se pretende juntar duas ou mais condições podem-se usar três palavras diferentes para as ligar entre si. Trata-se de AND (E lógico), OR (OU lógico) e NOT (negação lógica). Se encontrar uma expressão condicional com um AND juntando ambas as partes, o computador só fará o que lhe é indicado depois de THEN no caso de *ambas* essas condições se verificarem. Por exemplo, se forem seis da tarde mas ainda não tiver terminado o seu trabalho, não pode ir para casa.

Para ilustrar a noção de "verdadeiro" e "falso" relativamente a estas condições lógicas, experimente este programa:

```
10 INPUT a
20 INPUT b
30 IF a=1 AND b=1 THEN PRINT "verdadeiro"
40 GO TO 10
```

Experimente indicar diferentes valores à máquina e veja os resultados. Experimente alterar os valores na linha 30 para ver o efeito obtido. Tome nota das suas respostas até compreender o que se está a passar.

Vejamos agora a palavra OR. Pense nela do seguinte modo: IF forem seis da tarde OR o patrão diz que pode sair THEN vá para casa — isto é, faça uma coisa destas quando *uma* das alternativas for verdadeira. Mais correctamente, faça qualquer coisa quando pelo menos uma das alternativas for verdadeira, porque não interessa quantas o são (podem ser ambas) desde que pelo menos uma se verifique. Você pode portanto ir para casa em quaisquer circunstâncias às seis da tarde, mas também pode ir se o patrão lhe der autorização — ambas as condições lhe permitem fazê-lo indepen-

dentemente uma da outra. Experimente este novo programa do mesmo modo que experimentou o anterior.

```
10 INPUT a
20 INPUT b
30 IF a=1 OR b=1 THEN PRINT
"verdadeiro"
40 GO TO 10
```

A última palavra a estudar é NOT. Não junta expressões como as duas anteriores, mas altera o significado de tais expressões. Estude o seguinte:

IF NOT o patrão disser que pode ir para casa THEN continua a trabalhar

Isto significa que a menos que lhe seja dito que se pode ir embora deve continuar a trabalhar. Acontece portanto que o computador estuda a expressão e decide que se não for verdadeira faz qualquer coisa (para este fim ignora NOT ao verificar se a condição é ou não verdadeira). Ou seja, IF NOT... THEN é verdadeira sempre que a condição que se segue a NOT for falsa. Só é feito o que se indica a seguir a THEN no caso de a condição *não* estar cumprida. Experimente o seguinte:

```
10 INPUT a
20 INPUT b
30 IF a<>b THEN PRINT " verdadeiro "
40 GO TO 10
```

Talvez este programa o confunda um pouco inicialmente, mas se experimentar diferentes valores de A e B acabará por descobrir os valores de A e B que melhor ilustram o funcionamento de NOT.

Talvez o leitor tenha notado que usámos o sinal= em todos os exemplos apresentados até agora. Não se esqueça porém de que este é apenas um dos operadores disponíveis. Voltamos a apresentar uma lista dos seis que pode utilizar.

< menor que
> maior que
<= menor ou igual a
>= maior ou igual a
<> diferente de
= igual a

Altere os programas anteriores de modo a utilizar todos estes operadores. Experimente-os o número de vezes necessário para conseguir prever o resultado de cada vez que os executa. Experimente combinações de AND, OR e NOT e veja por que ordem são executadas. Veja se consegue alterar os resultados colocando parêntesis nas expressões. Note que isto não dará resultado em todos os casos, pelo que se uma certa expressão der problemas deixe-a e experimente outra. A avaliação é feita por *prioridades*, questão que será tratada adiante mais pormenorizadamente.

Podemos aplicar expressões condicionais a cadeias ou a números.

```
10 INPUT a$
20 IF a$<>"FRED BLOGGS" THEN N
EW
```

Faça executar este programa para ver o que acontece. Da primeira vez que o fizer, escreva o nome FRED BLOGGS em letras maiúsculas. O programa pára normalmente. Não pode dizer que seja muito interessante. Execute-o de novo e desta vez escreva o seu próprio nome (se não se chamar FRED BLOGGS). Desta vez o programa auto-destruir-se-á devido à instrução NEW da linha 20. Se substituir FRED BLOGGS pelo seu nome ou por um número de código, terá um programa que só funcionará para si ou para aqueles que conhecerem esse código, destruindo-se a si mesmo se alguma outra pessoa o quiser usar.

Vejamos agora os valores em expressões condicionais. Antes do mais utilizaremos os operadores de relacionamento já referidos. O leitor verificará que "verdadeiro" é representado por 1 e "falso" por um 0.

```
10 INPUT a
20 INPUT b
30 LET x=(a=b)
40 PRINT x
50 GO TO 10
```

Os parêntesis na linha 30 não são de facto necessários, mas ajudam a tornar o significado da expressão mais claro. A expressão em parêntesis é reduzida a um 0 ou a um 1 — o que depende dos valores comunicados à máquina. Experimente este programa usando todos os operadores de relacionamento, e tome nota dos resultados. Deve obter um 0 ou um 1 de cada vez — talvez lhe pareça que isto é um pouco restritivo, mas como de facto este valor 0 ou 1 é um número, pode depois ser utilizado como qualquer outro número. O melhor modo de o fazer consiste em multiplicá-lo, operação que não alterará os valores verdadeiros mas modificará os falsos (dado que qualquer número multiplicado por 0 dá 0). Experimente alterar o programa tendo isto em conta:

```
10 INPUT a
20 INPUT b
30 LET x=(a=b)*2
40 PRINT x
50 GO TO 10
```

Desta vez deve obter um valor de 0 no caso de uma expressão falsa e de 2 no de uma verdadeira. A importância de tudo isto reside em que estes valores das operações condicionais são números, podendo ser tratados como tais. Segue-se um programa de um jogo muito simples, "Apanha-bolas", que ilustra a utilidade do que discutimos até agora.

```
 10 REM * Apanha-bolas *
 20 LET S=0
 30 FOR J=1 TO 9
 40 PRINT AT 10,2*J; INK J/2;J
 50 NEXT J
 60 FOR G=30 TO 1 STEP -1
 70 PRINT AT 5,3; INK 6; PAPER
2; FLASH 1;" TEMPO >";G;" PONTUAÇÃO >
";S;" "
 80 LET A=INT (RND*9)+1
 90 PRINT AT 9,2*A; FLASH 1; IN
K 2;"■"
100 FOR H=1 TO 24
110 BEEP .01,50/H
120 LET A$=INKEY$
130 IF A$<>"" THEN GO TO 170
140 NEXT H
```



```
150 LET G=G-1
170 LET S=S+(A$=STR$ A)*A
180 PRINT AT 9,2*A;" "
190 NEXT G
200 PRINT AT 5,3; INK 6; PAPER
2; FLASH 1;" TEMPO >";0;" PONTUAÇÃO >
";S; PAPER 6; INK 2;" "
210 PRINT AT 13,3; FLASH 1; INK
0; PAPER 6; BRIGHT 1;"É TUDO, AMIG
OS!!"
```

O objectivo deste jogo consiste em carregar na tecla correspondente ao número sob o ponto móvel: por exemplo, se o ponto aparece sobre 3 deve-se carregar na tecla 3, e pontua-se este número. O número de tentativas restantes é continuamente apresentado no visor, tal como a pontuação. A linha 170 é aquela que mais nos interessa por agora. Nela, se STRS A (o valor de A convertido para a cadeia, de tal modo que possa ser comparado com a tecla premida) é igual à tecla em que se carrega então o valor lógico é 1 porque a expressão é verdadeira. Qualquer que seja o valor, é multiplicado pelo valor de A. Se for 0 a pontuação não se altera. Se for 1 a pontuação altera-se de 1 * A, ou seja, de A. A pontuação é contada pela variável S. O número de tentativas é indicado pela variável F.

Observemos agora os valores de expressões condicionais envolvendo as operações lógicas AND, OR e NOT. X AND Y têm o valor

X se Y é verdadeiro (não-zero)

0 se Y é falso (zero)

X e Y podem ser expressões como X= 2 ou Y= 2 * B. Uma aplicação muito vulgar disto é o comando de movimentos no visor. São muitos os jogos que utilizam as teclas de cursor (as teclas 5, 6, 7 e 8, que possuem setas por cima). Vejamos um modo de deslocar no visor um objecto para a esquerda ou para a direita:

```
 10 LET X=15
 20 IF INKEY$="5" AND X>1 THEN
LET X=X-2
 30 IF INKEY$="8" AND X<30 THEN
 LET X=X+2
 40 PRINT AT 21,X; INK 2;"■";AT
21,X;" "
 50 GO TO 20
```

Este programa move um ponto vermelho duas colunas de cada vez ao longo da fiada inferior do visor. Pode conseguir o mesmo com:

```
  10 LET X=15
  20 LET X=X-(INKEY$="5" AND X>1
)*2+(INKEY$="8" AND X<30)*2
  40 PRINT AT 21,X; INK 2;"■";AT
21,X;" "
  50 GO TO 20
```

ou ainda:

```
  10 LET X=15
  20 LET X=X-(2 AND INKEY$="5" A
ND X>1)+(2 AND INKEY$="8" AND X<
30)
  40 PRINT AT 21,X; INK 2;"■";AT
21,X;" "
  50 GO TO 20
```

Um facto a notar nestes dois últimos programas é que as expressões entre parêntesis adquirem o valor do número que se encontra antes do primeiro AND se todas as expressões após este forem verdadeiras. Compare-se com a expressão X AND Y que acabámos de estudar. Neste caso X é um número (2) e não uma expressão. Pode-se pensar na linha 20 anterior como equivalendo a:

> 20 LET X= X-(2 se for premida a tecla "5" e se o valor de X for maior do que 1; de outro modo. 0) + (2 se a tecla "8" for premida e o valor de X for inferior a 30, senão 0).

Pode-se perguntar qual a razão de complicar tanto uma coisa que poderia ser resolvida igualmente bem por uma série de instruções IF... THEN. A resposta é que quando usada convenientemente e nas circunstâncias apropriadas é possível substituir várias instruções ou linhas de programa por uma única expressão condicional se bem que longa, poupando portanto espaço de memória e tornando talvez mais rápida a execução. Por outro lado, quando estamos familiarizados com estas expressões condicionais, verificamos que por vezes tornam as listagens muito mais claras do que quando utilizamos uma longa lista de instruções IF... THEN.

Vejamos agora o funcionamento da declaração OR.

X OR Y tem o valor
  1 se Y é verdadeiro (não zero)
  X se Y é falso (zero)

Suponhamos um condutor de um camião que deseja um programa para calcular o bilhete que deve ser pago por uma criança, e que o limite de idade para estes bilhetes a preço reduzido é 14 anos.

```
10 INPUT "Indique preço"; preço
20 INPUT "Indique idade"; idade
30 LET preço= preço * (0.5 OR idade > 14)
40 PRINT "O preço reduzido é"; preço
```

As linhas 10 e 20 perguntam qual o preço normal para adulto e a idade do jovem passageiro. Para compreendermos melhor, traduzamos o que se segue em português (excepto a palavra-chave...):

LET preço= preço * (0.5 a menos que a idade seja superior a 14).

Se a expressão que se segue a OR for verdadeira, a expressão entre parêntesis tem o valor 1. No entanto, se essa expressão é falsa (o jovem tem menos de 14 anos) a expressão toma o valor indicado antes de OR. Este número pode igualmente ser uma variável, se assim quisermos. Por si só, esta rotina não parece grandemente vantajosa em relação a:

```
30 IF idade >= 14 THEN LET preço= preço * 0,5
```

No entanto, quando existem várias categorias de bilhetes o método que utiliza OR pode ser aplicado à avaliação de todas elas com uma só linha de instruções.

"NOT X" toma o valor de 0 no caso de a relação X ser verdadeira, e o valor 1 se aquela relação for falsa. O melhor modo de ilustrar o que acabamos de dizer será um exemplo como o seguinte:

```
10 INPUT a
20 INPUT b
```

```
30 PRINT a;TAB 4;b;TAB 8;NOT a
=b
40 GO TO 10
```

O que o utilizador vê no visor são os dois números indicados na linha 10 e 20, seguidos de um 0 ou um 1. A partir dos resultados obtidos, o leitor deve tentar descobrir as relações entre A e B que produzem o valor na terceira coluna. Experimente outros operadores relacionais em vez do = da linha 30.

Finalmente, observemos dois casos interessantes. Em primeiro lugar consideremos a linha seguinte:

```
10 IF a=1 THEN IF b=2 THEN PRI
NT "verdadeiro"
```

Esta linha é equivalente a:

```
10 IF a=1 AND b=2 THEN PRINT ".
verdadeiro"
```

Exige no entanto um maior espaço de memória. Uma outra diferença é que se não tivermos definido previamente a variável B a versão que utiliza AND bloqueará o programa com a mensagem de erro 2. No entanto, se a primeira parte da outra versão for falsa o programa passa por cima do resto da linha. Talvez o leitor encontre nos seus programas uma aplicação para isto.

O segundo caso com interesse não é propriamente algo de estranho, é mais qualquer coisa que normalmente as pessoas esquecem. Compare estes dois programas:

```
10 INPUT a
20 IF a THEN PRINT a
```

```
10 INPUT a
20 IF NOT a THEN PRINT a
```


Seria de esperar que estes programas funcionassem porque não existem quaisquer operadores relacionais para comparar A com seja o que for. Aqui, no entanto, o valor de A é considerado verdadeiro se não for zero, ou zero se se usar NOT. Tal como todo o resto neste capítulo, experimente com os exemplos até compreender exactamente aquilo que a rotina faz. Verificará que estas instruções podem ser muito potentes, e a sua programação pode melhorar bastante em resultado disto.

Pode-se usar IF /THEN GO TO para terminar uma mensagem de "vitória" ao fim de um certo número de ciclos. Escreva e faça executar o seguinte:

```
10 LET x=0
20 PRINT "VOCÊ GANHOU";
30 LET x=x+1
40 IF x<25 THEN GO TO 20
```

Isto assegurará que a frase "Você ganhou" será impressa um certo número de vezes.

Como vimos, IF /THEN não é apenas usado para enviar para outras linhas. Escreva NEW, e introduza em seguida na máquina o programa seguinte. Verificará que tem um efeito semelhante ao anterior, apesar de IF... não enviar para qualquer número de linha.

```
10 LET x=0
20 LET x=x+1
30 IF x< 25 THEN PRINT "VOCÊ GANHOU";
40 GO TO 20
```

Este programa não é tão útil como o anterior, porque continuará a ser executado indefinidamente mesmo depois de deixar de imprimir a frase pretendida. É fácil descobrir isto executando-o, carregando depois em BREAK, e escrevendo em seguida PRINT X ENTER.

Talvez valha a pena mencionar o facto de o computador ser uma criatura bastante dogmática. Se especificarmos num programa que determinada rotina deverá ser efectuada apenas se Z for igual a 6, o programa continuará em execução num ciclo sem fim se Z não tiver exactamente esse valor, por muito próximo que esteja dele

(por exemplo 5,999999). Se pensar que o valor pode ter uma diferença muito reduzida, fraccionária, relativamente ao valor pretendido, não se esqueça de especificar um operador relacional que defina a condição "maior do que 5.5 ou "maior do que 5,9", e não apenas igual a 6.

14
IF / THEN / ELSE

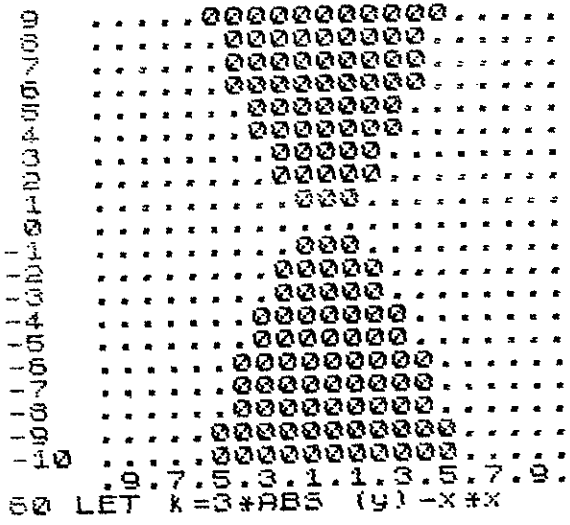
Muitos dialectos de BASIC incluem uma opção ELSE, usada na declaração IF... THEN... ELSE. Não existe esta função na linguagem BASIC do nosso computador, mas a lógica que este utiliza pode ser usada para produzi-la.

IF... THEN... ELSE é uma variante muito útil da instrução IF. O computador pode ser programado para fazer qualquer coisa se a condição verificada for verdadeira, e qualquer outra coisa, diferente de passar simplesmente à linha seguinte, se for falsa.

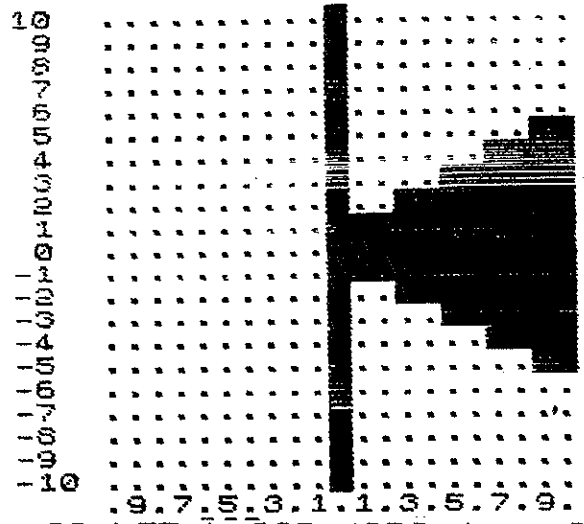
Pode-se usar a variante que se apresenta a seguir em substituição de IF... THEN... ELSE a fim de produzir alguns gráficos bastante interessantes. Basta indicar a função que se pretende representar em gráfico na linha 55. Este não é o método mais eficaz de programar computadores, mas é útil como meio de demonstrar o modo de substituir a instrução IF... THEN... ELSE. Ao ser executado, o programa avalia K de cada vez que chega à linha 55. A linha 70 observa o valor de K e imprime um zero se K for maior ou igual a 0.5, ou um ponto se K for inferior a 0.5. Isto equivale a uma linha que contenha a instrução: IF K maior ou igual a 0,5 imprimir "0" ELSE imprimir " ". Cada um dos outros gráficos utiliza um valor diferente de K, produzido pela linha 55. A condição verificada na linha 70 também varia. Faça executar os exemplos dados, usando símbolos gráficos escolhidos por si na linha 70, e crie depois exemplos seus. É provável que se veja forçado a alterar a escala para certas funções.


```
  10 REM GRÁFICO
  20 FOR y=10 TO -10 STEP -1
  30 IF y<>10 AND y<>-10 AND y>-
1 THEN PRINT " ";
  40 PRINT y;TAB 4;
  50 FOR x=-10 TO 10
  60 LET k=y-x*x/2+7
  70 PRINT ("0" AND k>=.5)+(","
AND k<.5);
  80 NEXT x
  90 PRINT
 100 NEXT y
 110 PRINT TAB 4;".9.7.5.3.1.1.3
.5.7.9."
```

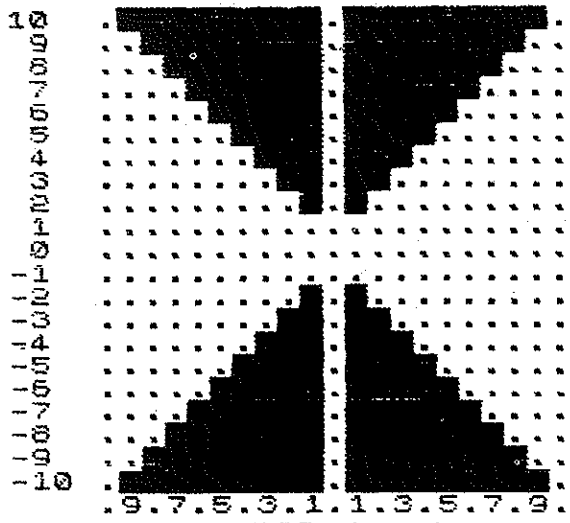
```
10      .....0000000000.....
 9      .....0000000000.....
 8      .....0000000000.....
 7      .....0000000000.....
 6      .....0000000000.....
 5      ......00000000......
 4      ......00000000......
 3      ......00000000......
 2      ......00000000......
 1      .......0000000.......
 0      .......0000000.......
-1      .......0000000.......
-2      .......0000000.......
-3      ........00000........
-4      ........00000........
-5      .........000.........
-6      .........000.........
-7      ....................
-8      ....................
-9      ....................
-10     ....................
        .9.7.5.3.1.1.3.5.7.9.
```



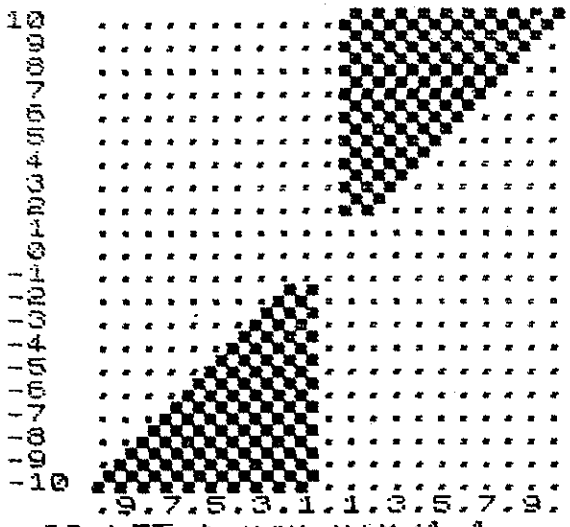
```
60 LET k=3*ABS (y)-x*x
```



```
60 LET k=SQR (ABS (y*x*2))-x
70 PRINT ("■" AND k<=.5)+("."
AND k>.5);
```

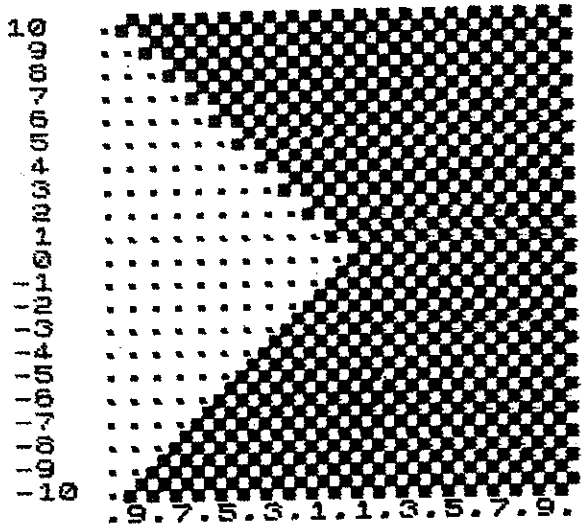
```
60 LET k=ABS (y*x)-x*x
70 PRINT ("■" AND k>=.5)+("."
AND k<.5);
```



```
60 LET k=y*x-x*x/1.1
```

```
 70 PRINT ("■" AND k>=.25)+(" " AND k<.25);
```

Experimente descobrir o que deveria encontrar-se na linha 55 para produzir o gráfico seguinte:



15
CICLOS FOR / NEXT

Os ciclos FOR / NEXT são uma outra ferramenta muito útil do BASIC usado no Spectrum. É conveniente estudá-los, agora, dado que a última série de programas utilizava bastante dois ciclos deste tipo, o ciclo Y que se iniciava na linha 30 e terminava na 100, e o ciclo X que se iniciava na linha 50 e terminava na 80. Como estes são um pouco mais complexos do que os ciclos FOR / NEXT simples, deixaremos a sua discussão para mais tarde.

Um ciclo FOR / NEXT é constituído por duas declarações usadas para comandar uma série de novas execuções de uma mesma parte de um programa. FOR inicia o ciclo especificando o número de vezes que o ciclo deve ser executado, e ainstrução NEXT ocorre no final da sequência reenviando a execução para a instrução que se segue àquela onde se encontra a ordem FOR.

As declarações FOR são constituídas por um número de linha, a palavra FOR, uma variável numérica (uma única letra), um sinal de igual, 'uma expressão numérica (um número ou uma variável numérica anteriormente definida), a palavra TO e, finalmente, uma outra expressão numérica (um número ou uma variável previamente definida) diferente do primeiro. Isto pode parecer complicado, mas de facto não o é.

Vejamos então a aparência da linha FOR:

```
100 FOR J= 1 TO 100
```

A linha NEXT, que termina o ciclo, tem a seguinte forma:

```
200 NEXT J
```

A declaração NEXT é portanto, constituída por um número de linha, a palavra NEXT e a variável utilização na declaração FOR anterior. Esta declaração serve apenas para dar a conhecer ao computador o ponto onde termina a sequência de programação que deve repetir. Quando o valor da variável de comando (J) atinge o indicado na declaração FOR (o segundo valor numérico nela especificado), o programa passa uma última vez pelo ciclo e em seguida continua para a linha que se segue à declaração NEXT.

Escreva e execute o seguinte:

```
10 FOR a=1 TO 10
20 PRINT TAB 4;a;TAB 8;a*a
30 NEXT a

 1    1
 2    4
```

```
3     9
4     16
5     25
6     36
7     49
8     64
9     81
10    100
```

A variável de comando é A, e a linha 20 imprime o seu valor e o quadro deste. Note que os limites do ciclo são indicados explicitamente na linha 10 (1 TO 10).

Observe o seguinte exemplo:

```
10 LET a=5
20 LET b=16
30 FOR c=a TO b
40 PRINT TAB 4;c;TAB 8;c/10;TAB 14;c/a
50 NEXT c
```

```
5     0.5     1
6     0.6     1.2
7     0.7     1.4
8     0.8     1.6
9     0.9     1.8
10    1       2
11    1.1     2.2
12    1.2     2.4
13    1.3     2.6
14    1.4     2.8
15    1.5     3
16    1.6     3.2
```

Note que neste programa o ciclo FOR /NEXT tem como limite duas variáveis, A e B, que foram definidas previamente. Você verificará que existem muitos programas em que se deseja utilizar um ciclo FOR / NEXT limitado, sendo os seus limites resultantes de resultados previamente obtidos pelo programa.

CICLOS SECUNDÁRIOS

Como já vimos, um ciclo FOR / NEXT permite-nos alterar o valor de uma variável (incrementando-a de 1 nos casos que já vimos), e repetir uma série de acontecimentos programados um certo número de vezes. Suponhamos agora que era necessário actuar sobre duas ou mais variáveis. Neste caso necessitaríamos de fazer variar ambas. Isto pode ser conseguido muito facilmente encerrando ciclos uns dentro dos outros, cada um deles controlado por um conjunto de declarações FOR / NEXT.

Escreva e execute o programa seguinte, que utiliza um ciclo B no interior de outro ciclo, A.

```
10 REM Ciclos secundários
30 FOR a=1 TO 12
40 FOR b=1 TO 12
50 PRINT TAB 8;b;" vezes";a;" são";a*b
60 NEXT b
70 PRINT
80 POKE 23692,-1
90 NEXT a
```

Quando se faz executar este programa, verifica-se que imprime uma tabela de multiplicação, entre 1 X 1 e 12 X 12. Vejamos parte do que é impresso:

```
 8 vezes 4 são 32
 9 vezes 4 são 36
10 vezes 4 são 40
11 vezes 4 são 44
12 vezes 4 são 48

 1 vezes 5 são 5
 2 vezes 5 são 10
 3 vezes 5 são 15
 4 vezes 5 são 20
```

5 vezes 5 são 25
6 vezes 5 são 30
7 vezes 5 são 35
8 vezes 5 são 40
9 vezes 5 são 45

Neste programa, a variável de comando A mantém-se no valor 1, enquanto o ciclo comandado por B varia de 1 até 12. Depois do PRINT da linha 70, a variável de comando A aumenta para 2 e o ciclo B é novamente executado entre 1 e 12, desta vez com A sempre igual a 2, assim continuando até o ciclo B ser integralmente cumprido com A igual a 12. O leitor pode utilizar qualquer número de ciclos contidos uns nos outros.

É vital que as variáveis de comando dos ciclos se apresentem pela ordem correcta, isto é, que o primeiro ciclo seja o último a terminar, etc. Experimente trocar as linhas 60 e 90 deste programa, e veja o que acontece.

Apresentamos em seguida parte dos resultados então fornecidos, que manifestamente não corresponde ao que se pretende:

2 vezes 13 são 26
3 vezes 14 são 42
4 vezes 15 são 60
5 vezes 16 são 80
6 vezes 17 são 102
7 vezez 18 são 126
8 vezes 19 são 152
9 vezes 20 são 180
10 vezes 21 são 210
11 vezes 22 são 242
12 vezes 23 são 276

Utilize a mesma variável para tantos fins quantos se lembrar, em particular quando recorre a ciclos FOR / NEXT. Não utilize outra letra como nome da variável de comando de um segundo ciclo FOR /NEXT se já tiver terminado a execução do ciclo anterior, pois o uso de várias variáveis de comando (e o mesmo se aplica a qualquer outro tipo de variável desnecessária) apenas serve para gastar memória.

STEP

Para o nosso estudo seguinte, necessitamos do programa "Foguetes" já apresentado.

As linhas importantes para o nosso estudo são a 30, a 40 e a 70. Quando executar o programa verificará que este imprime os números 10 a 1. É a palavra STEP (na linha 30) a seguir ao 1 que controla este resultado. Altere o -1 que se segue a STEP para -2, e veja o que acontece. Se não for especificado qualquer STEP ("passo") o computador considera que você deseja um passo positivo de 1, que foi precisamente aquilo de que necessitámos nos exemplos anteriores.

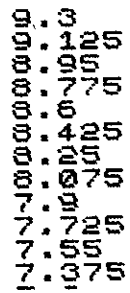
A ordem STEP é portanto usada no interior de um ciclo FOR / /NEXT para permitir ao utilizador especificar o valor do incremento (positivo ou negativo) da variável de comando. STEP não deve ser obrigatoriamente um número inteiro, se bem que seja necessário garantir - no caso de o número que se segue a To ser inferior ao que lhe é anterior - que STEP seja negativo. Experimente os exemplos seguintes:

```
10 FOR a=100 TO 1 STEP -12.5
20 PRINT TAB 8;a
30 NEXT a
```

```
 100
 87.5
 75
 62.5
 50
 37.5
 25
 12.5
```

```
10 FOR a=10 TO 1 STEP -0.175
20 PRINT TAB 8;a
30 NEXT a
```

```
 10
 9.825
 9.65
 9.475
```



Num ciclo FOR / NEXT. STEP não é obrigatoriamente um número inteiro: pode ser fraccionário. decimal. o resultado de um cálculo. e nem sequer deve dar exactamente o valor limite da variável de ciclo. Este continua a ser executado enquanto a sua variável de comando tiver um valor menor ou igual ao limite. Não é fácil alterar o valor de STEP durante o ciclo. Se o valor limite tiver sido excedido. o ciclo será desprezado:

```
10      FOR F = 1 TO 0
20      PRINT "X"
30      NEXT F
```

O leitor pode utilizar esta ideia para impedir a execução de ciclos em certas condições: por exemplo. se não se deseja que seja desenhada uma linha negra no caso de X ser igual a 6:

```
1000    FOR F = (X = 6) * 33 TO 31
1010    PRINT CHR$ 143;
1020    NEXT F
```

Para verificar se o valor limite foi ou não excedido utiliza-se a própria linha que contém a instrução FOR. Uma experiência interessante consiste em usar um valor de 0 para STEP. A variável de comando nunca é incrementada. e portanto o ciclo não termina! Pode-se sair de ciclos FOR / NEXT sem quaisquer problemas. mas convém que o leitor não se esqueça de que só pode saltar para o interior de um ciclo no caso de a variável de comando já ter sido

definida (isto é. se de facto esse mesmo ciclo já tiver sido usado previamente).

Não deve esquecer que num ciclo FOR / NEXT a execução passa da linha onde se encontra a declaração NEXT para a que se segue à declaração FOR. Por outro lado. se é certo que em certas versões de BASIC. usadas noutras máquinas. é possível omitir o nome da variável de comando depois da declaração NEXT (aceitando-a o computador como referindo-se à mais recente variável de comando e incrementando esta). tal não acontece no caso do seu Spectrum. onde deve sempre indicar a variável a seguir a NEXT.

16
GOSUB E RETURN

Chama-se subrotina a um bloco de instruções no interior de um programa que servem para executar uma tarefa bem definida. O programa principal é executado linha após linha até ser chamada a subrotina. utilizando a instrução GOSUB. O computador passa então para a linha de programa especificada a seguir a essa palavra. e continua por ordem a partir desse ponto até encontrar a instrução RETURN. Esta constitui um sinal que indica à máquina que deve voltar ao programa principal. para a linha *que se segue* à que chamou a subrotina.

As subrotinas são úteis sempre que é necessário executar um certo número de cálculos repetidas vezes e em diferentes pontos do programa. Por exemplo. num programa de contabilidade. pode ser necessário calcular várias vezes um imposto e fazê-lo relativamente a diferentes artigos. tratados em diferentes pontos do programa.

Sempre que é necessário fazê-lo a execução é enviada à subrotina que calcula o imposto. mantendo-se nela até encontrar a palavra RETURN e voltando depois à linha que se segue a GOSUB.

As subrotinas são escritas exactamente do mesmo modo que o programa principal. tratando-se de facto de um programa no interior de outro e estando limitadas por duas instruções. uma contendo GOSUB e outra RETURN. A instrução GOSUB é constituída por um número de linha. a palavra GOSUB e outro número de linha. A linha

```
40 GO SUB 100
```

indica ao computador que deve passar para a linha 100 e continuar a

executar as instruções por ordem, tal como no caso de a mesma linha conter a instrução GO TO 100. No entanto, quando o programa atinge a palavra RETURN, a execução volta ao programa principal, neste caso para a linha que se segue à 40.

Segue-se um exemplo muito simples mostrando o funcionamento destas instruções. Escreva-o e execute-o algumas vezes, voltando depois ao livro para o discutirmos.

```
 10 REM Demonstração GOSUB / RETURN
 20 POKE 23609,100: REM
    produz um BEEP quando se carrega numa tecla
 30 INPUT "indique um número ";A
 40 GO SUB 100
 50 GO TO 30
 90 REM subrotina
100 PRINT "O seu número é":A
110 PRINT A; "ao quadrado é ":A ↑ 2
120 RETURN
```

A linha 30 pede-lhe que indique um número, e a linha 40 transfere a execução para uma subrotina que se inicia na linha 100. São realizados os cálculos pretendidos, e em seguida impressos os seus resultados, ainda no interior da subrotina; finalmente, a linha 120 reenvia para a linha que se segue à que chamou a subrotina, isto é, para a linha 50. Como esta contém um GO TO, a execução passa novamente para a linha 30, que pede um novo número e repete todo o processo.

Escreva e execute o programa seguinte, que põe em competição dois submarinos numa corrida. Sempre é uma aplicação mais interessante das subrotinas...

```
 10 REM Corrida de submarinos
 15 PAPER 5: BORDER 5: CLS
 20 LET A$="▟▙  "
 30 LET computador= 28
 40 LET Jogador=28
 50 LET X=5
 55 BEEP .01,RND*50
 60 GO SUB 100
 70 LET X=10
 75 BEEP .01,RND*50
 80 GO SUB 100
 90 GO TO 50
100 IF X= 5 THEN LET computador= computador-RND:
```



```
PRINT AT X, computador: INK 6;A$: IF computador < 2 T
HEN PRINT AT 0,0: PAPER 6: FLASH 1: BRIGHT 1: "Ganha
o computador": STOP
110 IF X=10 THEN LET Jogador=Jogador-RND: PRINT AT X,
    Jogador; INK 2; A$: IF Jogador < 2 THEN PRINT AT 0, 0;
PAPER 6; FLASH 1; "Ganha o Jogador"; STOP
120 RETURN
```

Como pode verificar, existem dois "submarinos" no visor. O de cima é o do computador, e o de baixo pertence-lhe a si. Carregue em RUN, depois em ENTER, e verá os submarinos moverem-se ao longo do visor da direita para a esquerda. Quando um deles atinge a margem o programa pára, imprimindo "Ganha o computador" ou "Ganha o jogador" conforme o caso.

Note que A$, o submarino, se estende por mais de uma linha de programa. Depois de escrever o "periscópio" (o sinal gráfico 5), o leitor deve escrever de seguida 29 espaços em branco usando a tecla SPACE. A seguir escreva os símbolos gráficos finais, seguidos de um último espaço. Este último espaço é bastante importante: se quiser descobrir porquê não o escreva, e execute o programa.

É chegado o momento de estudarmos um outro recurso do Spectrum, muito útil para a criação de jogos ou para melhoramento de outros programas — o som.

17
O SOM

A instrução BEEP pode ser usada para melhorar os seus programas, acrescentando ruídos apropriados quando eliminar algum invasor extraterrestre, quando envia uma bola contra a parede, ou consegue derrotar o computador em qualquer outro jogo de perícia.

Se bem que possa parecer à primeira vista que um único canal sonoro, aliás com um volume bastante reduzido, que por outro lado interrompe qualquer outra actividade da máquina enquanto é executado, é de facto bastante limitado, é no entanto possível usá-lo para aumentar imensamente o interesse dos nossos programas.

O leitor deve ligar um pequeno altifalante ao seu Spectrum, através da tomada EAR. Se bem que isto não aumente muito o

volume, a disponibilidade de uma segunda fonte sonora aumenta bastante a sua eficácia.

A instrução BEEP (carregar em ambas as teclas SHIFT, e em seguida na SYMBOL SHIFT juntamente com Z) possui dois parâmetros (ou seja, devem existir dois números à frente da palavra BEEP). O primeiro indica a duração da nota "tocada", e o segundo é a altura do som.

O leitor pode ter uma ideia dos ruídos produzidos fazendo executar a rotina que se segue:

```
 5 REM música aleatória
10 BEEP RND/RND/3,RND*60-35
12 BORDER RND*7
15 BEEP RND/RND/2,RND*80-45
20 BORDER RND*7
25 BEEP RND/RND/3,RND*130-65
30 PAPER RND*7
40 CLS
45 BEEP RND/RND/2,RND*40-5
50 GO TO 10
```

O resultado do uso de BEEP num ciclo, ou num conjunto de ciclos, pode ser bastante interessante, como se observa na seguinte demonstração:

```
  10 REM música cíclica
  20 FOR A=-60 TO 60
  30 FOR B=.01 TO .03 STEP .01
  40 BEEP B,A: BEEP B,A/10*B: BE
EP B,ABS (A)
  50 NEXT B
  60 NEXT A
```

Os BEEP's produzidos aleatoriamente, dentro de certos limites, podem também ser interessantes:

```
10 LET ALTURA= INT (RND *24) - 12
20 LET DURAÇÃO= (INT(RND * 8) + 1) /30
30 BEEP DURAÇÃO, ALTURA
40 IF RND > = .7 THEN GO TO 30
50 GO TO 10
```


Como pode descobrir no seu manual, certos números bem definidos produzem notas musicais — por exemplo 0 produz o Dó central da escala de piano, 12 produz o Dó da oitava acima, e -3 produz o Lá abaixo do Dó central. O programa que se segue transforma o seu Spectrum num "vibrafone" no qual se usam as teclas da fiada inferior para produzir notas (da oitava normal): Dó (tecla Z), Ré (tecla X), Mi (tecla C), Fá (tecla V); Sol (tecla B); Lá (tecla N); Si (tecla M); Dó (tecla K). As notas soarão enquanto estivermos a carregar nas teclas.

```
  10 REM PIANO
  20 LET A=CODE INKEY$
  30 IF A<66 OR A>90 THEN GO TO
20
  40 LET B=12*(A=75)+2*(A=83)+4*
(A=67)+5*(A=86)+7*(A=66)+9*(A=78
)+11*(A=77)
  50 BEEP .04,B
  60 IF INKEY$<>"" THEN GO TO 50
  80 GO TO 20
```

Depois de dominar este assunto, pode acrescentar um pouco de cor acompanhando a altura das notas, acrescentando a seguinte variação na linha 70 do programa anterior:

```
  10 REM PIANO
  20 LET A=CODE INKEY$
  30 IF A<66 OR A>90 THEN GO TO
20
  40 LET B=12*(A=75)+2*(A=83)+4*
(A=67)+5*(A=86)+7*(A=66)+9*(A=78
)+11*(A=77)
  50 BEEP .04,B
  60 IF INKEY$<>"" THEN GO TO 50
  70 BORDER B/2: PAPER B/2: CLS
  80 GO TO 20
```

Se lhe parece que tocar o seu vibrafone dá muito trabalho, pode obrigar o Spectrum a fazê-lo em seu lugar. Como pode verificar estudando a linha de DATA do programa seguinte, o "Spectrum bem temperado" utiliza a escala natural de Dó.

```
 10 REM O Spectrum bem temperado
 20 DIM A(8)
 30 FOR B=1 TO 8
 40 READ A(B)
 50 NEXT B
 70 LET B=INT (RND*8)+1
 75 LET M=(INT (RND*4)+1)/10
 80 BEEP M,A(B)
 85 IF RND>.9 THEN GO SUB 110
 90 GO TO 70
100 DATA 0,2.039,3.86,4.98,7.02
,8.84,10.88,12
110 LET Z=RND
112 LET M=A(1)*(Z>=.5)+A(8)*(Z<
.5)
115 BEEP 1,M
120 PAUSE 25
130 RETURN
```

O leitor encontrará muitos exemplos do uso de BEEP nos programas deste livro, que lhe darão bastantes ideias sobre os sons que pode acrescentar aos seus próprios programas.

O último programa deste capítulo utiliza o som como um ingrediente muito importante. Nele o computador escolhe um número entre 1 e 4 e coloca-o no visor, acompanhando-o com um BEEP e uma mudança de cor. O leitor deve então carregar na tecla correspondente a esse mesmo número. A máquina voltará a repetir o primeiro número, e acrescentar-lhe-á outro. O leitor deve repetir depois ambos os números. Se a máquina escolhe o mesmo número duas vezes seguidas o leitor deve premir duas vezes a tecla correspondente. Ganha o jogo se conseguir recordar uma sequência de sete algarismos correctamente.

```
  1 REM Jogo sonoro
  5 LET a$=""
 10 FLASH 1
 20 PAPER 7
 30 FOR a=1 TO 7
 32 PRINT AT 10,10; "Prepare-se"
 35 BORDER RND*7
 40 LET a$=a$+STR$ (INT (RND*4)
+1)
 45 PAUSE 5
 50 NEXT a
 55 FLASH 0: CLS
 60 LET x=1
 70 FOR q=1 TO x
 72 LET l=4*(CODE a$(q)-48)
 73 LET t=VAL a$(q)
 75 BEEP .05,10*t
 80 PRINT AT l,7; INK t;"[■■■■]";"
";a$(q);AT l+1,7;"[■■■■]";AT l-1,7
;"[■■■■]"
 85 BORDER RND*7
 90 PAUSE 20-x
100 PRINT AT l,7; INK 6;"[▚▞▚]";
AT l+1,7;"[▚▞▚]";AT l-1,7;"[▚▞▚]"
102 PAUSE 4
105 CLS
110 NEXT q
120 FOR b=1 TO x
122 IF INKEY$<>"" THEN GO TO 12
2
123 LET t$=INKEY$
124 IF CODE t$=0 THEN GO TO 123
125 CLS
126 LET y=4*(CODE t$-48)
130 PRINT AT y,7; INK y/4;"[■■■■]"
;t$;AT y-1,7;"[■■■■]";AT y+1,7;"[■■■■]"
145 BEEP .04,2.5*y
146 IF CODE t$<>CODE (a$(b)) TH
EN GO TO 300
147 PAUSE 7
148 CLS
150 NEXT b
155 IF x=7 THEN PRINT "Você ganhou!"
: BORDER RND*7: PAPER RND*7: GO
TO 155
160 LET x=x+1
165 PAUSE 50
170 GO TO 70
300 PRINT "Você pontuou ";X-1
310 BORDER RND*7
320 PAPER RND*7
330 CLS
335 BEEP .02,RND*30
340 GO TO 300
```

18
DEFINIÇÃO DE FUNÇÕES

O BASIC usado no Spectrum permite-nos definir funções no

interior de um programa, funções essas que podem depois ser utilizadas sempre que se quiser durante a execução desse programa. A declaração DEF FN pode poupar bastante espaço e tempo, dado que se torna possível definir cálculos bastante complexos de uma só vez e atribuindo-lhes um nome bastante curto, indicado sempre que se deseja usá-los.

Existem quatro componentes na instrução de definição de uma função:

— A palavra DEF
— O nome da função, que consiste nas letras FN seguidas do nome propriamente dito (uma letra se se trata de uma função numérica, uma letra e um S se se trata de uma função alfanumérica).
— O argumento da função que se segue ao nome, entre parêntesis.
— A fórmula, usando o argumento em causa, que se pretende usar.

Tudo isto parece um pouco complicado, bastante mais do que o é na prática. Observemos o programa seguinte:

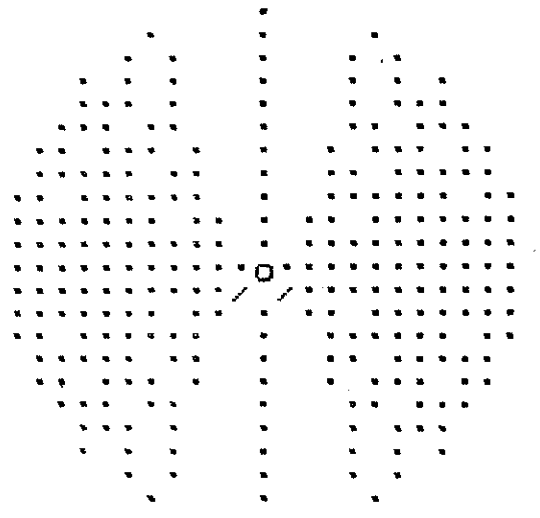
```
10 REM  DEFINIR UMA FUNÇÃO
20 DEF  FN  A(Z)=Z*Z
30 INPUT  Z
40 PRINT  Z,FN  A(Z)
50 GO  TO  30
```

```
12                    144
44                    1936
23.76                 564.5376
1111                  1234321
44                    1936
345                   119025
4                     16
11                    121
```

A linha 20 define uma função A, com o argumento Z igual a Z ao quadrado. Portanto, sempre que o programa encontra a expressão FN A(Z) calcula o quadrado do valor atribuído à variável Z. Pode-se verificar isto no programa que se segue.

No programa seguinte — "Morcego" — é definida uma função na linha 70. A função calcula a raiz quadrada da diferença entre o quadrado de duas variáveis, e na rotina 120 a 210 utiliza o valor H (ver linha 130) para determinar as posições de impressão dos pontos que desenham o morcego.

```
  10  REM  Morcego
  20  REM  Demonstração de DEF FN
  30  LET  L=0
  40  LET  P=10
  50  LET  Q=17
  60  LET  B=0
  70  DEF  FN  A(B)=SQR  (L*L-B*B)
  80  PAPER  6:  CLS
  90  INK  0:  BORDER  2
 100  PRINT  AT  P,Q;"O"
 110  LET  L=L+1
 120  FOR  B=0  TO  L
 130  LET  H=FN  A(B)
 140  PRINT  AT  P+B,Q+H;"."
 150  PRINT  AT  P+B,Q-H;"."
 160  PRINT  AT  P-B,Q-H;"."
 170  PRINT  AT  P-B,Q+H;"."
 180  BEEP  .02,50+H
 190  NEXT  B
 200  IF  L<11  THEN  GO  TO  110
 210  PRINT  INK  2;AT  9,16;"/  /"
```



19
DIM E QUADROS

A instrução DIM é usada para definir uma *lista* fácil de aceder. Pode tornar-se necessário, em certos programas, indicar elementos de uma comprida lista de números, tanto depois de lhes dar entrada por INPUT a fim de os usar por uma certa ordem como para imprimi-los de um determinado modo no visor.

Um quadro é um conjunto de espaços de memória reservados no interior do computador, e indicados pelo nome do quadro e por um subscrito. Para obter um quadro que guarde três elementos, escreve-se DIM A(3), o que produz um espaço livre para um quadro chamado A. Para guardar quatro elementos, escreve-se DIM B(4).

Escreva e faça executar o programa seguinte, que deve facilitar a compreensão do que dissemos.

```
 10 REM  ** Demonstração de quadros **
 20 DIM B(4)
 30 FOR A=1 TO 4
 40 LET B(A)=INT  (RND*9)+1
 50 NEXT A
 60 FOR A=1 TO 4
 70 PRINT TAB 6;"B(";A;")  IS
B(A)
 80 NEXT A

      B(1)  IS 6
      B(2)  IS 9
      B(3)  IS 7
      B(4)  IS 1
```

Como se pode verificar na linha 20 do programa que acabámos de executar, o computador só pode utilizar um quadro se o tivermos anteriormente dimensionado, usando uma instrução DIM. Esta é constituída por um número de linha seguido pela palavra DIM e pelo nome do quadro, sendo a dimensão do quadro encerrada entre parêntesis.

Os quadros de que estivemos a falar até agora têm apenas uma dimensão, o que é apropriado por exemplo para guardar uma lista de algarismos. No entanto, é possível utilizar quadros de mais dimensões. Estes quadros são designados, como é compreensível, por multi-dimensionais, e são obtidos utilizando uma instrução DIM com mais de um subscrito. Escreva e execute o programa que se segue:

```
 10 REM  Quadros multi-direccionais
 20 REM          ARRAYS
 30 DIM A(4,4)
 40 FOR B=1 TO 4
 50 FOR C=1 TO 4
 60 LET A(B,C)=INT  (RND*9)+1
 70 PRINT  "A(";B;",";C;")  IS  ";
A(B,C)
 80 NEXT C
 90 NEXT B
100 PRINT AT 16,15;"1 2 3 4"
105 PRINT
110 FOR B=1 TO 4
120 PRINT TAB 13;B;TAB 15;A(B,1
);"  ";A(B,2);"  ";A(B,3);"  ";A(B,
4)
130 NEXT B
```

Quando executar o programa verá no visor algo do seguinte tipo:

```
A(1,1)  é  9
A(1,2)  é  6
A(1,3)  é  7
A(1,4)  é  4
A(2,1)  é  3
A(2,2)  é  3
A(2,3)  é  4
A(2,4)  é  6
A(3,1)  é  8
A(3,2)  é  8
A(3,3)  é  2
A(3,4)  é  8
A(4,1)  é  7
A(4,2)  é  8
A(4,3)  é  6
A(4,4)  é  1
```

```
     1  2  3  4
1    9  6  7  4
2    3  3  4  6
3    9  8  2  8
4    7  8  6  1
```

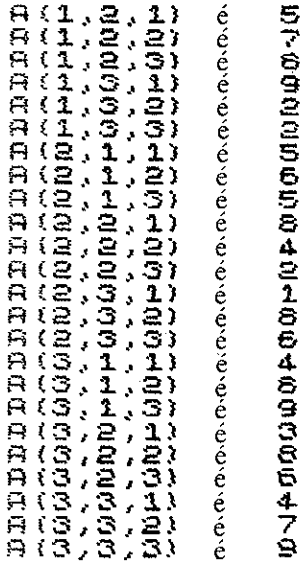
Em primeiro lugar preenchem-se os elementos do quadro com números entre um e nove, os quais são em seguida impressos pela linha 70 a fim de conhecermos o conteúdo de cada elemento do quadro. A pequena tabela impressa em baixo mostra o modo como os elementos do quadro se encontram organizados. Qualquer dos elementos pode ser acessado indicando as suas coordenadas no interior do quadro. Nestas consições, o elemento (3,3) deve encontrar-se no ponto onde se intersectam as linhas que partem dos dois números 3, sendo portanto o número 2.

Dimensionar um quadro consome bastante memória, pelo que se deve ter o cuidado de nunca definir um quadro de dimensões superiores às estritamente necessárias. O número de elementos de um quadro é igual ao primeiro número entre parêntesis multiplicado pelo segundo. Portanto, um quadro A(4,4) possui 16 (4 x 4) elementos. O leitor pode confirmar este resultado no exemplo anterior.

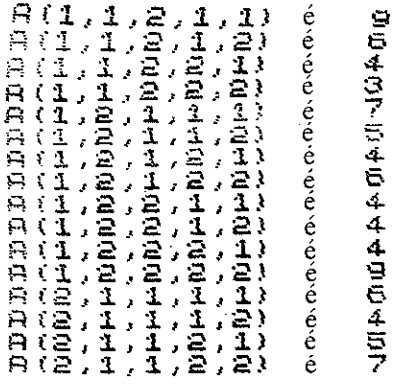
Não há qualquer razão para não utilizar quadros com mais de duas dimensões, excepto pelo facto de depressa se tornarem difíceis de manusear e de o número de elementos aumentar de modo alarmante (juntamente com a memória gasta). Vejamos um programa que utiliza um quadro de três dimensões. Se bem que se trate apenas de um quadro A(3,3,3), possui já um número de elementos bastante elevado (3 x 3 x 3).

```
 20  REM  Quadros multi-dimensionais
 30  DIM  A(3,3,3)
 40  FOR  B=1  TO  3
 50  FOR  C=1  TO  3
 55  FOR  D=1  TO  3
 60  LET  A(B,C,D)=INT  (RND*9)+1
 70  PRINT  "A(";B;",";C;",";D;")
is ";A(B,C,D)
 75  NEXT  D
 80  NEXT  C
 90  NEXT  B
```

```
A(1,1,1)  é  5
A(1,1,2)  é  3
A(1,1,3)  é  3
```

Aumente o número de dimensões para cinco, como acontece no exemplo seguinte, e mesmo que utilize apenas A(2,2,2,2,2) o quadro disporá já de 32 elementos (2 x 2 x 2 x 2 x 2).



```
A(2,1,2,1,1)    é  1
A(2,1,2,1,2)    é  7
A(2,1,2,2,1)    é  3
A(2,1,2,2,2)    é  6
A(2,2,1,1,1)    é  8
A(2,2,1,1,2)    é  2
```

```
 20 REM Quadros multi-dimensionais
 30 DIM A(2,2,2,2,2)
 40 FOR B=1 TO 2
 50 FOR C=1 TO 2
 60 FOR D=1 TO 2
 70 FOR E=1 TO 2
 80 FOR F=1 TO 2
 90 LET A(B,C,D,E,F)=INT (RND*9
)+1
100 PRINT "A(";B;",";C;",";D;",
";E;",";F;") IS ";A(B,C,D,E,F)
110 NEXT F
120 NEXT E
130 NEXT D
140 NEXT C
150 NEXT B
```

"DESCODIFICAÇÃO"

Vejamos agora um jogo, chamado "Descodificação", que nos mostra o modo de usar um quadro de uma única dimensão. O jogo é muito fácil. O computador "pensa" num número de quatro algarismos, e dispomos de dez tentativas para o adivinharmos. Um algarismo correcto numa posição errada no interior do código imprime "branco", e um algarismo correcto na posição igualmente correcta imprime um "preto".

```
 10 REM Descodificação
 20 DIM C(4)
 30 DIM G(4)
 40 PRINT '' PAPER 2; INK 6;"
    Estou a pensar num número de"
 50 PRINT PAPER 2; INK 6; "quatro
    algarismos que você"
```



```
 60 PRINT PAPER 2; INK 6;"DEVE DES
COBRIR EM DEZ TENTATIVAS. TODOS OS ALGARISMOS
SÃO"
 70 PRINT PAPER 2; INK 6; "DIFERENTES.
CARREGUE NUMA TECLA"
 80 PRINT PAPER 2; INK 6;"
    PARA CO
    MEÇAR
 90 PAUSE 4e4
100 CLS
110 LET C(1)=INT (RND*9)+1
120 FOR Z=2 TO 4
130 LET C(Z)=INT (RND*9)+1: BEE
P .5,C(Z)
140 FOR J=1 TO Z-1
150 IF C(J)=C(Z) THEN GO TO 130
160 NEXT J
170 NEXT Z
180 FOR G=1 TO 10: BEEP 1,2*G
200 PRINT PAPER 1; INK 7;TAB 6;
"Diga um algarismo";G;
210 INPUT A
220 LET A1=A
230 FOR Z=1 TO 4
240 LET G(Z)=A-10*INT (A/10)
250 LET A=INT (A/10)
260 NEXT Z
270 LET B=0
280 LET W=0
290 FOR Z=1 TO 4
300 IF C(Z)<>G(Z) THEN GO TO 33
0
310 LET B=B+1: BEEP .02,10*B
320 LET G(Z)=0
330 NEXT Z
340 FOR Z=1 TO 4
350 IF G(Z)=0 THEN GO TO 400
360 FOR J=1 TO 4
370 IF C(Z)<>G(J) THEN GO TO 39
0
380 LET W=W+1: BEEP .02,7*W
390 NEXT J
400 NEXT Z
420 PRINT INK 2;A1; "SCORED";INK 0;B;
"PRETO";
430 IF B<>1 THEN PRINT "S";
440 PRINT ", "; INK 2;W;" BRANCO";
450 IF W<>1 THEN PRINT "S";
460 PRINT
```

```
 470 IF B=4 THEN PRINT  "Adivinhou em": G;"tentativas!"
 480 IF B<>4 THEN NEXT G
 490 PRINT  "O código era"; INK 2; C(4); C(3); C(2); C(1)
2;C(4);C(3);C(2);C(1)
```

QUADROS DE CADEIAS ALFANUMÉRICAS

O leitor pode também usar cadeias alfanuméricas. que são muito semelhantes às cadeias numéricas. Escreva e execute o seguinte programa para compreender a actuação da cadeia alfanumérica. escrevendo depois quatro palavras (cada uma delas seguida de ENTER) quando o computador lhas pedir.

```
 10 REM Quadros alfanuméricos
 20 DIM A$(4,10)
 30 FOR B=1 TO 4
 40 INPUT A$(B)
 50 NEXT B
 60 FOR B=1 TO 4
 70 PRINT "A$(";B;")  IS ";A$(B)
 80 NEXT B
```

AS(1) É ÁGUA
AS(2) É UNIVERSO
AS(3) É MEDICINA
AS(4) É PORTUGAL

Se bem que o segundo número da instrução DIM (neste caso 10 deva ser tão comprido como a cadeia mais longa que se deseja utilizar. basta especificar a primeira dimensão (como acontece na linha 70) para que a máquina imprima toda a cadeia no visor.

Note que a principal diferença entre uma cadeia alfanumérica e uma numérica consiste no sinal "S" que é colocado imediatamente a seguir à letra. Isto indica ao computador que o nome se refere a uma cadeia alfanumérica.

Vejamos agora um curto programa que nos mostra o modo de utilizar os quadros alfanuméricos. O programa trata cinco palavras. Se quiser adaptá-lo para mais. altere o 5 nas linhas 20. 50 e 40. substituindo-o pelo número de palavras que lhe interessa.


```
 10 REM  Escolha de palavras.
 20 DIM W$(5,10)
 30 LET B=0
 40 LET G=5
 50 FOR A=1 TO 5
 60 INPUT W$(A)
 70 PRINT W$(A)
 80 NEXT A:  PRINT
 90 LET Z=1
100 LET B=Z+1
110 IF B>G THEN GO TO 190
120 IF W$(B)>W$(Z)  THEN GO TO 1
50
130 LET Z=Z+1
140 GO TO 100
150 LET Q$=W$(Z)
160 LET W$(Z)=W$(B)
170 LET W$(B)=Q$
180 GO TO 130
190 PRINT W$(G)
200 LET G=G-1
210 IF G>0 THEN GO TO 90
```

CORRER
CAIR
CAVAR
COMER
COSTURAR

CAVAR
COMER
CAIR
CORRER
COSTURAR

20
TRATAMENTO DE CADEIAS ALFANUMÉRICAS

A nossa discussão dos quadros alfanuméricos conduz-nos directamente ao estudo das cadeias que os constituem e à partição destas cadeias. Como o leitor já deve ter compreendido. uma cadeia é uma colecção de caracteres alfanuméricos colocados entre aspas

(incluindo símbolos e espaços, se se quiser). É designada por uma variável cujo nome é uma única letra seguida pelo sinal "$". As cadeias são identificadas com variáveis de um modo semelhante ao usado para as variáveis numéricas, através de uma instrução com a forma LET A$= "OLÁ".

Existem várias funções muito importantes que podem ser usadas para tratamento de cadeias alfanuméricas ou para separação de partes destas. Estas funções são:

CODE X$ — Dá o código do primeiro caracter de X$; se X$ for igual a "MICRO", CODE X$ será igual a 77 (consultar os códigos do conjunto de caracteres indicados no Manual).

CHR$ 77 — Podemos verificar se 77 é de facto o código do primeiro caracter de X$ (ou seja, se é o código de M) pedindo ao computador que PRINT CHR$ 77. A máquina imprimirá um M. De facto, CHR$ é o oposto de CODE, e transforma um código no caracter que lhe corresponde.

X$ (TO 3) — Esta função produz uma nova cadeia, contendo os *n* primeiros caracteres de X$; neste caso, os três primeiros: "MIC"

LEN X$ — Esta função dá o comprimento de uma cadeia; usando-a no nosso exemplo, X$= "MICRO", obteremos LEN X$= 5

X$(n TO m) — Esta função permite-nos obter uma cadeia contendo os caracteres de X$ entre o de ordem *n* e o de ordem *m*. X$(2 TO 4)= "ICR"

X$(3 TO) — Esta função é o contrário, como seria de esperar, de X$(TO 3), e dá os três (neste caso) últimos caracteres da cadeia original. X$(3 TO)= "CRO"

STR$ A — Esta função transforma a variável A, numérica, numa cadeia; se a variável tinha o valor 234, a cadeia resultante será "234". Isto pode não parecer muito útil, mas permite de facto certas manipulações de números que são bastante difíceis na sua forma numérica. Dentro em pouco estudaremos melhor esta função.

VAL X$ — Trata-se do "oposto" de STR$ A, transformando em número o conteúdo da cadeia se este for numérico. Se X$= "22+34", por exemplo, VAL X$= 56

Vejamos um programa que demonstra o tratamento de cadeias alfanuméricas, e um exemplo da saída por ele produzida:

```
 10 PRINT "SEJA X$= 'MICRO' "
 20 LET X$="MICRO"
 30 PRINT "CODE X$=";CODE X$
 40 PRINT "CHR$ 77=";CHR$ 77
 50 PRINT "X$(3 TO )=";X$(3 TO
)
 60 PRINT "X$( TO 3)=";X$( TO 3
)
 70 PRINT "LEN X$=";LEN X$
 80 PRINT "X$(2 TO 4)=";X$(2 TO
4)
 90 PRINT "SEJA X$="23+35""
100 LET X$="23+35"
110 PRINT "VAL X$=";VAL X$
120 PRINT "LET X=34"
130 LET X=34
140 PRINT "LET X$=STR$ X"
150 LET X$=STR$ X
160 PRINT "X$=";X$
```

```
LET X$="MICRO"
CODE X$=77
CHR$ 77=M
X$(3 TO )=CRO
X$( TO 3)=MIC
LEN X$=5
X$(2 TO 4)=ICR
LET X$="23+35"
VAL X$=58
LET X=34
LET X$=STR$ X
X$=34
```

USO DE LEN

Se deseja imprimir um certo número de caracteres iguais, por exemplo uma linha de "-" para sublinhar um título, pode utilizar dois métodos. Como é óbvio os títulos terão comprimentos diferen-

tes, pelo que você vê-se forçado a descobrir quantos traços deverá escrever para sublinhar toda a palavra. Se estiver a imprimir uma cadeia alfanumérica, por exemplo AS, pode usar a função LEN para determinar o seu comprimento, e portanto a quantidade de traços a imprimir.

```
(1)  10 FOR A= 1 TO LEN A$
     20 PRINT" — ";
     30 NEXT A
     40 PRINT
```

A linha 40 desloca a posição de impressão para a linha seguinte. Omita-o se desejar. O método seguinte é bastante mais rápido e utiliza apenas uma linha de programa.

```
(2)  10 PRINT"- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - "(TO LEN A$)
```

A única desvantagem neste caso é que se torna necessário escrever alguns traços a mais entre aspas, apesar de alguns deles nunca virem a ser utilizados. Ou seja, você deve conhecer à partida qual o maior comprimento possível dos títulos usados, prevendo esse comprimento máximo no número de traços incluídos na cadeia.

USO DE STRS

STRS é uma função muito útil e por vezes bastante esquecida. Como já mencionámos anteriormente, converte um número na cadeia equivalente. Experimente o programa seguinte:

```
10 PRINT 2
20 PRINT STR$ 2
30 PRINT 1/3
40 PRINT STR$ (1/3)
50 PRINT 9E15
60 PRINT STR$ 9E15
```


Ao executar este programa obterá os seguintes resultados:

```
 2
 .5
 0.33333333
 0.33333333
 9E+15
 9E+15
```

Podemos aprender bastante com estes exemplos. Em primeiro lugar, a cadeia produzida por STRS é igual ao que obteríamos imprimindo o número no visor. Em segundo lugar, os números inferiores a 1 são atribuídos a uma cadeia com um zero antes da vírgula decimal, desde que o primeiro algarismo depois da vírgula decimal seja diferente de 0 (isto é, o número deve ser igual ou maior do que 0.1 e menor do que 1), podendo existir até oito algarismos depois da vírgula (ou menos). É portanto possível que a cadeia possua um máximo de dez caracteres. No entanto, se o número a que se aplica a função STRS possuir mais de oito algarismos depois da vírgula decimal, é arredondado para oito casas decimais: STRS .3333333339 é "0.33333334". STRS é igualmente capaz de produzir números em notação científica (que já referimos no início do livro), por exemplo 9E + 15. Note que apesar de o computador aceitar 9E15, STRS aceita isto sob a forma 9E + 15 - isto é, o expoente é sempre indicado juntamente com o seu sinal. Os números muito pequenos, por exemplo 0.000009 são portanto escritos do seguinte modo: STRS.0000009="9E-7". Quando se usa STRS, é muitas vezes conveniente definir os limites do número de tal modo que STRS não comece a usar notação científica, o que pode causar problemas.

O leitor estará certamente a pensar: "óptimo, mas qual é o interesse disso?"

A principal utilidade consiste em converter números para cadeias de tal modo que possamos aplicar a estas as potencialidades da máquina em termos de tratamento de cadeias, formatando ou arredondando para um dado número de casas décimais, ou sempre que desejamos avaliar um número algarismo a algarismo. Vejamos alguns exemplos do uso de STRS.

1) Alinhamento de vírgulas decimais.

Suponhamos que o leitor dispõe de uma lista de números para imprimir. Experimente o programa seguinte:

```
10 LET a=RND*100
20 PRINT a
30 LET a=a*10
40 GO TO 20
```

Deverá obter algo do seguinte tipo:

```
8.581543
85.81543
858.1543
8581.543
85815.43
858154.3
8581543
85815430
```

Esta lista de números seria muito mais legível se fosse possível alinhar as vírgulas decimais. Experimente agora a seguinte rotina:

```
          79.025269
         790.25269
        7902.5269
       79025.269
      790252.69
     7902526.9

10 LET a=RND*100
20 PRINT TAB 15-LEN STR$ INT a
; a
30 LET a=a*10
40 GO TO 20
```

A rotina indicada dispõe os números de tal modo que as vírgulas decimais aparecem umas por baixo das outras - o que é muito útil para fazer uma listagem de números quando se deseja compará-los entre si. Conseguirá o leitor compreender o modo como o programa funciona? Suponhamos que o valor de A era 69.433594. O programa limita-se a determinar a parte inteira de A (INT A, 69), convertê-la numa cadeia (STRS INT A) e medir em seguida o comprimento desta (LEN STRS INT A), que neste caso é 2. Em seguida utiliza este número para descobrir em que ponto do visor deve começar a imprimir o valor de A. Note o modo como isto é feito:

TAB 15 - LEN STRS INT A

Isto significa que 15 é a coluna do visor onde é colocada a vírgula decimal: a máquina conta para trás o número de algarismos de STRS INT A.

2) Imprimir um dado número de casas decimais.

É muitas vezes necessário imprimir números com, por exemplo, três casas decimais. Note que o exemplo anterior imprimia números com todos os seus algarismo conhecidos. Podemos usar STRS para determinar a quantidade de números que são impressos depois da vírgula decimal. Consideremos a rotina seguinte:

```
10 LET a=RND
20 LET a$=STR$ a
22 IF a$(1)="." THEN LET a$="0
"+a$
25 LET b=LEN STR$ INT VAL a$
27 PRINT TAB 15-b;(a$+(" " AND
b=LEN a$)+"000")( TO b+4)
30 LET a=a*10
40 GO TO 20
```

```
             0.189
             1.893
            18.934
           189.346
          1893.463
```

Esta rotina imprime números com três casas decimais, acrescentando zeros no início ou no fim em caso de necessidade. Para levá-la a imprimir com Z casas decimais, fazem-se as seguintes alterações na linha 27: acrescentam-se tantos espaços a AS quantas forem as casas decimais pretendidas (ou seja, Z zeros); transforma-se a instrução de partição para (TO B+1+Z).

Expliquemos este programa linha a linha:

A linha 10 define o valor inicial de A.

A linha 20 converte A numa cadeia.

A linha 22 acrescenta um zero antes do primeiro algarismo no caso de se tratar de uma vírgula decimal. Infelizmente, a função STRS não é uniforme na sua acção, dado que por vezes acrescenta um zero em números inferiores a 1 e outras vezes não, conforme o primeiro algarismo depois da vírgula decimal é ou não maior que

zero. Torna-se portanto simples verificar se é ou não necessário um zero — se o primeiro caracter é uma vírgula. acrescenta-se um zero.

A linha 25 torna B igual ao número de algarismos da parte integral do número que estamos a imprimir medindo o comprimento da parte integral da cadeia A$. É necessário usar VAL A$ em vez de A porque o computador pode apresentar uma anomalia entre o último algarismo do valor de A e A$ tal como é determinado por STR$. o que em certos casos pode provocar problemas.

A linha 27. que espaça a posição de PRINT do modo indicado em (1), acima, começa em seguida a imprimir A$ com 3 casas decimais. Em primeiro lugar é impresso completamente A$. acrescentando-se uma vírgula decimal se A$ já representa um número inteiro e um número suficiente de zeros para obter três casas decimais. Talvez lhe pareça estranho acrescentar 4 a B - não é certo que estamos a utilizar três casas decimais? Mas não se esqueça da vírgula decimal — trata-se de um caracter extra. Para efeitos de tratamento da cadeia. a parte anterior é tratada como uma cadeia comprida desde que se encontre toda entre aspas. Limitámo-nos a acrescentar caracteres para preencher A$ até ficar com pelo menos três casas decimais. imprimindo em seguida 3 algarismos depois da vírgula. Esta rotina não arredonda a terceira casa decimal — mais adiante encontrará uma para este efeito.

3) Poupar memória.

E muitas vezes possível poupar memória usando cadeias para guardar números. em vez de variáveis numéricas. descodificando-os depois usando VAL. Pode-se guardar um número numa variável de cadeia usando STR$:

```
LET A$= STR$(1024)
```

descodificando-o mais tarde com VAL:

```
PRINT VAL A$
```

O leitor verificará muitas vezes que gasta mais memória a converter números deste modo do que no caso de usar variáveis numéricas: mas por vezes este método pode produzir maravilhas.

Experimente aplicar VAL a uma expressão como "ANT 1 § 4".

Dá resultado, e muitas vezes é até bastante útil. Pode colocar o nome de uma variável numérica entre aspas. e desde que tenha sido previamente definida será avaliada com êxito. De facto. VAL pode ser aplicada a todos os tipos de expressões numéricas. e é por vezes usada em vez de DEF FN.

Este tratamento pode ser também útil quando se deseja produzir números aleatórios várias vezes no mesmo programa. No início do programa inclui-se uma instrução do tipo LET A$= "RND * 6"; de cada vez que se deseja um número aleatório escreve-se então LET R= VAL A$.

21
INKEY$

Não é necessário carregar em ENTER depois de carregar numa tecla quando se usa uma instrução INKEY$. como o programa que se segue esclarece.

Experimente o seguinte. Escreva um número entre 1 e 9. carregando na tecla correspondente ao número. e verá a máquina imprimir CARREGOU EM 6. CARREGOU EM 1. etc. Toque na tecla 0. e o computador escreverá CARREGOU EM 0 e parará imediatamente.

```
10 REM **Demonstração de INKEY$ **
20 PAUSE 100
40 LET A$=INKEY$
50 PRINT "CARREGOU EM";A$
60 IF A$="0" THEN STOP
70 GO TO 20
```

```
CARREGOU EM 6
CARREGOU EM 7
CARREGOU EM 3
CARREGOU EM 5
CARREGOU EM 2
CARREGOU EM 6
CARREGOU EM 3
CARREGOU EM 6
CARREGOU EM 0
```

O programa seguinte — PREVISÕES — utiliza igualmente a declaração INKEYS. Neste jogo é necessário prever o número (entre um e nove) que o computador pensará em seguida. O número do computador é apresentado no visor, perto do centro, e o número menor é a pontuação. Quanto menor for a pontuação no final (quando se consegue prever com êxito o número do computador), melhor. O visor manter-se-á em branco até ser carregada uma tecla. A palavra "PONTUAÇÃO:" surgirá e desaparecerá continuamente no visor.

Ó SEU NÚMERO É 4

O MEU NÚMERO É 1

PONTUAÇÃO: 1

```
  10 REM **PREVISÃO**
  20 LET E=9
  40 LET 0=0
  50 PAUSE 100
  70 LET A$=INKEY$
  80 PRINT AT 8,5; INK RND*7; PA
PER 9;" O SEU NÚMERO É ";A$;"
   "
  90 LET 0=0+1: BEEP .02,0*3
 110 LET W$=STR$ (INT (RND*9)+1)
 120 PRINT AT 12,E;; INK RND*7;"
PAPER 9;"     O MEU NÚMERO É     ";W$;"
   "
 130 PRINT AT 14,E;; INK RND*7;
PAPER 9; FLASH 1; BRIGHT 1;"
A PONTUAÇÃO ";0;"              "
 135 STOP
 140 IF W$=A$ THEN BEEP .01,RND*
30: GO TO 130
 150 GO TO 50
```

O programa seguinte — LABIRINTO — permite-nos estudar também o funcionamento de INKEYS. Usando as teclas "A", "Z", "K" e "M" move-se o sinal S entre o canto inferior esquerdo do visor e o canto superior direito, sem atravessar nenhum dos pequenos quadrados brancos. Note que não existe nenhum percurso garantido, e que não há qualquer mecanismo para verificar se faz batota. No final é impresso o número de movimentos de que necessitámos.

```
  10 REM **LABIRINTO**
  20 REM USE AS TECLAS A Z M K
  30 REM PARA MOVER O SINAL $
  40 REM DO CANTO INFERIOR ESQUERDO
  50 REM ATÉ AO CANTO SUPERIOR
  60 REM      DIREITO
  70 BORDER 2: PAPER 7: CLS : BR
IGHT 1
  80 LET S=0
  90 FOR Y=1 TO 704
 100 LET T=INT (RND*3)
 110 PRINT INK (RND*5+1); ("■" AN
D T=0)+(" " AND T<>0);
 120 NEXT Y
 130 PRINT AT 0,0;"    ";AT 1,0;"
   "
 140 LET X=30: LET Y=19
 150 LET M=X
 160 LET N=Y
 170 PRINT AT Y,X; INK 2;"$"
 180 LET S=S+1
 190 LET A$=INKEY$
 200 IF A$="" THEN GO TO 190
 210 IF A$="A" AND Y>0 THEN LET
Y=Y-1
 220 IF A$="Z" AND Y<20 THEN LET
 Y=Y+1
 230 IF A$="K" AND X<31 THEN LET
 X=X+1
 240 IF A$="M" AND X>0 THEN LET
X=X-1
 250 IF X=0 AND Y=0 THEN GO TO 2
80
 260 PRINT AT N,M;" "
 270 GO TO 150
 280 PRINT AT 10,10; FLASH; BRIGHT; "CONSEGUIU"

 290 PRINT "";TAB 6; FLASH 1; BRIGHT 1; "PRECISOU
 DE"; S; "MOVIMENTOS"
```

Note na linha 110 o modo como utilizámos os métodos lógicos de avaliação do computador. Esta linha garante que no caso de T (produzido na linha anterior) ser zero, é impresso um quadrado negro; no caso de T ser maior do que zero é impresso um espaço em branco. É uma maneira bastante eficaz de copiar a instrução IF / /THEN ELSE existente noutros computadores. Esta técnica, e outras que lhe são semelhantes, são estudadas em maior detalhe noutro ponto do livro.

Se deseja limitar as respostas possíveis do utilizador, recorra a este método. Suponha que o utilizador dispõe apenas de alguns segundos para dicidir se quer ou não jogar novamente. Se demora muito tempo o programa pára. Para estudar esta rotina suponhamos que o utilizador deve carregar em R para fazer correr novamente o programa:

```
10 FOR F=1 TO 100
20 LET A$=INKEY$
30 IF A$="R" THEN GO TO 60
40 NEXT F
50 STOP
60 PRINT "Nova execução"
70 RUN
```

Alternativamente, pode-se utilizar o contador de imagens para temporizar entradas ou qualquer outra coisa.

Para usar o contador de imagens recorre-se à rotina seguinte para inicializar o relógio:

```
POKE 23673,255
POKE 23673,255
```

Para usar o número contido neste contador recorremos a:

```
LET= (256 * PEEK 23673 + PEEK 23672) /50
```

Conseguiremos assim uma leitura em segundos bastante rigorosa se fizermos PRINT R. Não esqueça que PAUSE e BEEP utilizam o contador de imagens, pelo que este não pode ser usado para temporização se o seu programa contém qualquer daquelas instruções. Experimente o seguinte para obter valores em segundos:


```
10 POKE 23673,255
20 POKE 23672,255
30 LET t=(256*PEEK 23673+PEEK
23672)/50
40 PRINT AT 0,0;t;"    "
50 GO TO 30
```

E o programa seguinte para obter um relógio digital:

```
10 REM Relógio digital
20 BORDER 1: PAPER 2: CLS
30 INK 7
40 INPUT "Indique horas":h
50 INPUT "Indique minutos":m
60 LET s=0
70 POKE 23673,255
80 POKE 23672,255
110 PRINT AT 4,12;h;" ";
120 IF m<10 THEN PRINT "0";
130 PRINT m;" ";
140 IF s<10 THEN PRINT "0";
150 PRINT s;" "
160 LET s=INT ((256*PEEK 23673+
PEEK 23672)/50)
170 IF s>59.9999 THEN LET m=m+1
: POKE 23673,255: POKE 23672,255
180 IF m=60 THEN LET h=h+1: LET
m=0
190 IF h=13 THEN LET h=1
200 GO TO 110
```

Se deseja definir o tempo rigorosamente, sem voltar a zero repetidamente, estude o capítulo sobre PEEKs e POKEs, mais adiante.

22
READ /DATA /RESTONE

READ e DATA constituem um modo muito prático de aceder a informação dentro de um programa, e são relativamente fáceis de usar. Escreva e faça executar o programa que se segue, o qual mostra o funcionamento destas instruções, voltando em seguida ao livro para uma sua explicação.

```
 10 REM READ/DATA
 20 REM ***********
 30 REM READ THE DATA
 40 REM ***********
 50 DIM B(5)
 60 FOR A=1 TO 5
 70 READ B(A)
 80 NEXT A
 90 REM ***********
100 REM READ IT BACK
110 REM ***********
120 FOR C=5 TO 1 STEP -1
130 PRINT B(C)
140 NEXT C
150 DATA 13,35241,88,2,199999

RUN

199999
2
88
35241
13
```

Na linha 70, o computador encontra a instrução READ...

Sempre que encontra esta instrução, a máquina passa para o primeiro elemento que se encontra à frente da palavra DATA, lendo-o neste caso para um quadro numérico. Os elementos DATA podem encontrar-se em qualquer ponto do programa (se bem que seja útil mantê-los relativamente perto da instrução READ que se refere a eles).

Voltemos ao programa "Foguetes" já apresentado neste livro.

No primeiro programa deste capítulo, os dados (DATA) são números, que são atribuídos a variáveis numéricas (os elementos do quadro numérico). Em "Foguetes", os dados são cadeias (linha 220) e podem ser atribuídos à variável de cadeia AS e impressos novamente na linha 120, durante a execução do ciclo designado por "foguete". Além de READ e DATA existe uma terceira palavra relacionada com este assunto: RESTORE. Esta declaração diz ao computador para voltar ao início da lista de dados e começar a lê-los novamente desde o primeiro. Vejamos um outro programa de exemplificação, que nos apresenta os dados sob a forma de uma cadeia alfanumérica e ilustra o funcionamento de RESTORE.


```
 10 REM READ/DATA
 20 REM ***********
 30 REM READ THE DATA
 40 REM ***********
 50 DIM B$(21,5)
 60 FOR A=1 TO 21
 70 READ B$(A)
 75 IF 3*INT (A/3)=A THEN RESTO
RE
 80 NEXT A
 90 REM ***********
100 REM Imprimir os dados
110 REM ***********
120 FOR C=21 TO 1 STEP -1
130 PRINT B$(C);"   ";
140 NEXT C
150 DATA "MATAR", "COMER", "LER"
```

```
LER         COMER       MATAR       LER
COMER       MATAR       LER         COMER
MATAR       LER         COMER       MATAR
LER         COMER       MATAR       LER
COMER       MATAR       LER         COMER
MATAR
```

Neste programa existem apenas três dados, pelo que é necessário utilizar RESTORE para permitir lê-los de novo. A linha 80 assegura que tal aconteça sempre que são lidos os três dados enquanto se executa o ciclo A entre 1 e 21. Note que os dados alfanuméricos devem ser colocados entre aspas.

Como já compreendeu, usa-se uma ordem READ para atribuir valores a variáveis a partir de uma sequência de elementos contidos numa instrução DATA. Cada um destes elementos encontra-se separado por vírgulas. A instrução READ é constituída pelo número de linha seguido da palavra READ e dos nomes que devem ser atribuídos aos elementos recolhidos na linha DATA.

Quando o programa encontra uma instrução READ, passa imediatamente à leitura do primeiro elemento de dados qualquer que seja a sua posição no interior do programa. O primeiro valor da instrução DATA será atribuído à primeira variável da instrução READ. Além de ler as instruções DATA o computador não lhe presta mais atenção, tratando-a para todos os outros efeitos como se fosse uma instrução REM. Altera o número da linha 150 para 25 e executa novamente o programa anterior. Verificará que (a), o com-

putador ignora a linha 25. e (b). continua a ler a mesma linha de dados.

Mesmo no caso de os dados estarem espalhados ao longo do programa. o computador procurá-los-á como se demonstra no programa seguinte.

```
  10 REM READ/DATA
  20 DATA "OLÁ",7
  30 DIM B$(21,4)
  40 DIM Z(21)
  60 FOR A=1 TO 21
  70 READ B$(A),Z(A)
  80 IF 3*INT (A/3)=A THEN RESTO
RE
  85 DATA "VIVA"
  90 NEXT A
  95 DATA 56,"JOÃO"
 130 FOR C=1 TO 21
 140 PRINT B$(C),Z(C)
 145 DATA 22
 150 NEXT C
```

É importante assegurar que se dispõe de suficientes elementos de dados para o número de vezes que o computador os irá ler. Retire a linha 145 do programa anterior e execute-o novamente. Obterá a mensagem de erro "E OUT OF DATA. 7Ø:1". O computador leu dois elementos da instrução DATA. mas depois não encontrou o terceiro porque ainda não tinha recebido nenhuma ordem RESTORE.

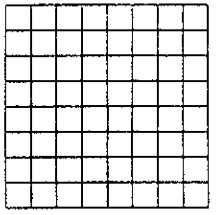
Não se esqueça de que apesar de não ser essencial dispor dos elementos de dados junto às instruções que os lêem. é muito provável que os seus programas se tornem mais fáceis de compreender se os dispuser deste modo. O método em causa facilita igualmente a descoberta das linhas que devem ser alteradas se estiver a trabalhar num programa.

23
GRÁFICOS DEFINIDOS PELO UTILIZADOR

Uma das características mais interessantes do ZX Spectrum é a possibilidade de definirmos os nossos próprios símbolos gráficos. É

muito simples fazê-lo. e permite-nos determinar a saída visual do nosso programa do modo que mais nos agrada. Neste capítulo. vamos desenvolver uma forma muito simplificada de um jogo de perícia bastante conhecido. o "Pacman". a fim de demonstrar o modo de usar gráficos produzidos pelo utilizador. O nosso jogo será chamado DOTMAN. e consistirá numa única criatura "Pacman" que tenta fugir a um único "Fantasma" ao mesmo tempo que tenta comer tantos pontos quanto possível.

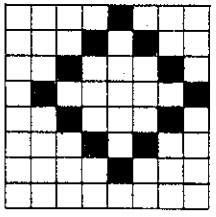
Os gráficos são definidos por uma rede oito por oito. como a seguinte:



Podemos escolher qualquer tecla. entre A e U. e impor o nosso próprio símbolo gráfico a essa tecla. de tal modo que quando é escolhida. em vez de imprimir a letra imprima esse símbolo. Se bem que. como é óbvio. estes símbolos gráficos sejam perdidos de cada vez que o computador é desligado. não são afectados por NEW e é portanto possível fazer NEW e manter os mesmos símbolos para um novo programa.

Altera-se o conteúdo de uma tecla executando um curto ciclo (LOOP). Vejamos um exemplo. Se quisermos o símbolo do naipe "ouros" na tecla corresponde a A. procede-se do seguinte modo.

Primeiramente. deve desenhar numa rede (existem várias no final deste capítulo. que pode fotocopiar para uso próprio) o padrão que deseja criar. O símbolo "ouros" pode ter a forma seguinte:



Introduzimos este símbolo fazendo um POKE em oito bytes, ou seja, usando oito instruções: o valor zero equivalerá a um "pixel" em branco, e um 1 produzirá um "pixel" a negro. A linha superior do nosso símbolo será portanto representada por 00001000, um número binário de oito algarismos; como se vê na figura, só o quinto "pixel" é negro. A segunda linha será 00010100. Compare-se isto com os quadrados negros (pixels) do diagrama. Indicamos ao computador que o número usado se encontra na base binária fazendo-o preceder pela palavra BIN, obtida carregando na tecla B. Vejamos agora a rotina que produz este símbolo:

```
  10 REM  OUROS
  20 FOR J=0 TO 7
  30 READ Q
  40 POKE USR "A"+J,Q
  50 NEXT J
  60 PRINT "A"
  70 DATA BIN 00001000,BIN 00010
100,BIN 00100010,BIN 01000001,BI
N 00100010,BIN 00010100,BIN 0000
1000,BIN 00000000
```

Note que o A da linha 60 é escrito em modo "gráficos". Faça executar o programa, e verá que o próprio A da linha 60 se altera, passando a apresentar a forma gráfica desejada:

```
60 PRINT "◇"
```

Não há dúvida de que criámos um símbolo bastante aceitável, como se pode ver na figura seguinte:

O leitor já terá compreendido que os zeros iniciais na linha DATA são redundantes. No entanto, preferimos deixá-los porque

se torna assim muito mais fácil verificar cada um dos elementos no caso de haver um erro. Experimentemos agora construir uma forma diferente. Escreva a seguinte linha de dados e execute em seguida o programa para ver o que obtém:

```
  70 DATA BIN 00110000,BIN 01110
000,BIN 01111110,BIN 01011111,BI
N 10011111,BIN 00010010,BIN 0001
0010,BIN 00000000
```

Pretendi desenhar um elefante, e como se pode observar no seu visor o resultado final parece bastante convincente.

Vamos tentar finalmente produzir o nosso jogo, DOTMAN. Começaremos por tentar pôr o programa em funcionamento, antes de definirmos os caracteres gráficos que dele fazem parte. Asseguramos assim que o programa propriamente dito adquire a importância que lhe cabe, não sendo substituído pela simples qualidade dos gráficos usados. É conveniente ver como funciona o programa utilizando exactamente o que irá representar cada uma dessas letras.

O nosso jogo será bastante simples. Numa rede de 14 x 14, encontra-se-á o nosso DOTMAN, obviamente sob o nosso comando, um fantasma que é comandado pelo computador e uma série de pontos que serão comidos por DOTMAN. Existirão na rede algumas paredes, constituindo um labirinto, e nem o DOTMAN nem o fantasma poderão passar por elas.

No final do livro, na secção intitulada "Como melhorar os seus programas", é sugerido que planeie as partes principais do seu programa antes de começar de tal modo que a estrutura do programa se torne suficientemente clara assim que começar a trabalhar nele. Se utilizar o processo que descrevemos, verá que tirará bastantes vantagens disso.

"DOTMAN"

No caso de DOTMAN, a estrutura do programa será a seguinte:

1-999: Enviar a execução para as partes do programa que definem a situação inicial.
1000-1999: Comando de DOTMAN pelo jogador
2000-3999: Movimentos do fantasma
4000-4999: Final do jogo se o fantasma apanha DOTMAN
5000-5999: Definir DOTMAN
6000-6999: Definir Fantasma
7000-7999: Imprimir o labirinto
8000-8999: Inicalizar variáveis

Depois de o programa começar em execução, efectuará um ciclo entre 1000 e 3999 (ou o maior número correspondente aos movimentos do fantasma) repetidas vezes até o fantasma apanhar DOTMAN, momento em que o programa passará à linha 4000 para executar a rotina de fim de jogo.

Escrevamos as primeiras rotinas de comando:

```
10 GO SUB 8000
20 GO SUB 7000
30 GO SUB 5000
40 GO SUB 6000
```

Basta-nos isto por agora. E, como vamos escrever todo o programa antes de definirmos o DOTMAN e o FANTASMA, podemos acrescentar 6999 RETURN e 5999 RETURN imediatamente, começando a construir o labirinto, depois de definirmos as variáveis. GA é a posição lateral do Fantasma, GD a coordenada "vertical" dele. EGA e EGD são as variáveis que "apagam" o fantasma quando este se move. Do mesmo modo, DD e DA são usadas para movimento lateral e vertical do DOTMAN, e EDD e EDA para o apagar.

Acrescentemos o labirinto e as rotinas para mover o DOTMAN e o Fantasma, e o programa ganha a aparência seguinte:


```
10 GO SUB 8000
20 GO SUB 7000
30 GO SUB 5000
40 GO SUB 6000
1000 REM mover dotman
1010 REM dotman definido por B,C,D,E
1020 PRINT AT EDD,EDA;" "
1030 PRINT AT DD,DA;A$
1035 LET EDD=DD: LET EDA=DA
1040 IF INKEY$="8" THEN IF DA<13
 THEN IF SCREEN$ (DD,DA+1)<>"X"
THEN LET DA=DA+1: LET A$="B"
1050 IF INKEY$="5" THEN IF DA>0
THEN IF SCREEN$ (DD,DA-1)<>"X" T
HEN LET DA=DA-1: LET A$="C"
1060 IF INKEY$="7" THEN IF DD>0
THEN IF SCREEN$ (DD-1,DA)<>"X" T
HEN LET DD=DD-1: LET A$="E"
1070 IF INKEY$="6" THEN IF DD<13
 THEN IF SCREEN$ (DD+1,DA)<>"X"
THEN LET DD=DD+1: LET A$="D"
1980 IF RND>.2 THEN GO TO 1000
1990 REM O fantasma é o gráfico G
2000 PRINT AT EGD,EGA;" "
2010 PRINT AT GD,GA;"G"
2015 LET EGD=GD: LET EGA=GA
2020 IF DD<GD THEN IF SCREEN$ (G
D-1,GA)<>"X" THEN LET GD=GD-1
2030 IF DD>GD THEN IF SCREEN$ (G
D+1,GA)<>"X" THEN LET GD=GD+1
2040 IF DA>GA THEN IF SCREEN$ (G
D,GA+1)<>"X" THEN LET GA=GA+1
2050 IF DA<GA THEN IF SCREEN$ (G
D,GA-1)<>"X" THEN LET GA=GA-1
2999 GO TO 1000
4999 STOP
5999 RETURN
6999 RETURN
7000 REM Construir labirinto
7010 FOR G=1 TO 14
7020 FOR H=1 TO 14
7030 PRINT ".";
7040 NEXT H
7050 PRINT
7060 NEXT G
7070 PRINT AT 2,7;"XXXXX"
7080 PRINT AT 3,3;"X";AT 3,11;"X
"
```

```
7090 PRINT AT 4,3;"X";AT 4,11;"X
"
7100 PRINT AT 5,3;"XXXX";AT 5,9;
"XXX"
7110 PRINT AT 6,6;"X";AT 6,9;"X"
7120 PRINT AT 7,6;"X";AT 7,9;"X"
7130 PRINT AT 8,3;"X";AT 8,6;"X"
;AT 8,9;"X"
7140 PRINT AT 10,3;"X"
7150 PRINT AT 11,3;"XXXX"
7160 PRINT AT 12,9;"XXX"
7170 RETURN
8000 REM Variáveis
8010 LET DA=0
8020 LET DD=0
8030 LET EDA=0
8040 LET EDD=0
8050 LET GA=13
8060 LET GD=13
8070 LET EGA=13
8080 LET EGD=13
8090 LET pontuação=0
8100 LET A$="B"
8999 RETURN
```

Se fizer executar este programa, verá que já "faz" alguma coisa. Não é muito brilhante, por agora, mas não se deixe abater. Ficará surpreendido com a óptima aparência do jogo depois de terminado. Pode ser melhorado muito facilmente, mesmo neste ponto, acrescentando:

```
7065 INVERSE 1
7165 INVERSE 0
```

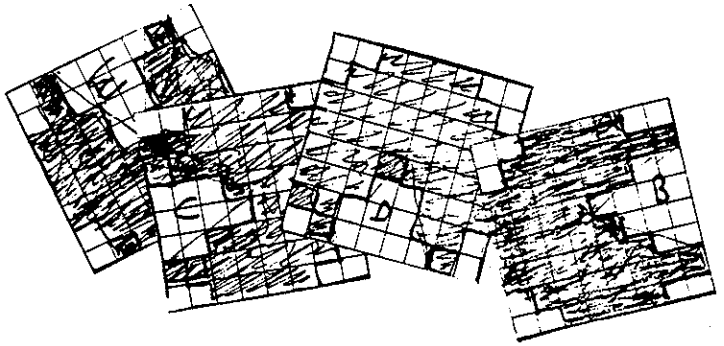
INVERSE funciona exactamente do mesmo modo que BRIGHT e FLASH: 1 significa "ligado" e 0 desligado. Execute novamente o programa, e veja o efeito que INVERSE tem sobre as paredes do labirinto. Basta isto para melhorar um pouco a aparência do programa. Devem existir quatro DOTMEN, um virado em cada direcção, para que a boca aponte no sentido em que se move: é por isso que a

variável A$, que corresponde ao DOTMAN, é alterada no fim das linhas 1040 e 1070.

Acrescente o seguinte, para definir os DOTMEN:

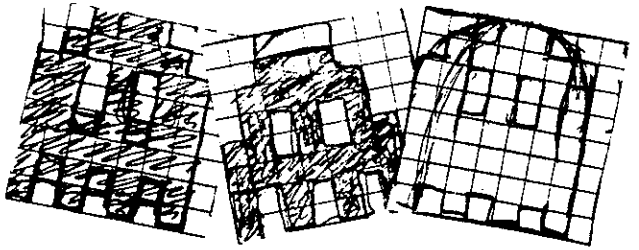
```
5000 REM Definir Dotman
5010 FOR J=0 TO 7
5020 READ Q
5030 POKE USR "B"+J,Q
5040 NEXT J
5050 DATA BIN 00111100,BIN 01111
111,BIN 11111100,BIN 11110000,BI
N 11111000,BIN 11111100,BIN 0111
1111,BIN 00111100
5060 FOR J=0 TO 7
5070 READ Q
5080 POKE USR "C"+J,Q
5090 NEXT J
5100 DATA BIN 00111100,BIN 11111
110,BIN 00111111,BIN 00011111,BI
N 00001111,BIN 00111111,BIN 1111
1110,BIN 00111100
5110 FOR J=0 TO 7
5120 READ Q
5130 POKE USR "D"+J,Q
5140 NEXT J
5150 DATA BIN 00111100,BIN 01111
110,BIN 11111111,BIN 11111111,BI
N 11110111,BIN 11100111,BIN 0100
0010,BIN 01000010
5160 FOR J=0 TO 7
5170 READ Q
5180 POKE USR "E"+J,Q
5190 NEXT J
5200 DATA BIN 01000010,BIN 01000
010,BIN 11100111,BIN 11101111,BI
N 11111111,BIN 11111111,BIN 0111
1110,BIN 00111100
```

Desenhámos algumas figuras possíveis para o DOTMEN antes de decidir por aquela que designámos por B. O diagrama em causa foi depois rodado para obter os valores binários para as quatro figuras a usar.



Acrescente alguma cor. acrescentando por exemplo INK 2 na linha 1030. Imprimirá assim DOTMEN vermelhos. INK 1. na linha 2010. produzirá fantasmas azuis.

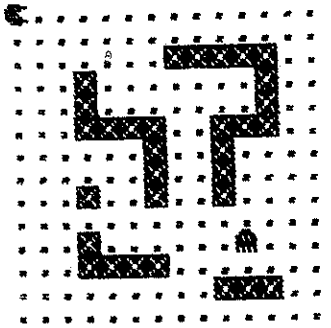
Vejamos agora quatro esboços para o fantasma.



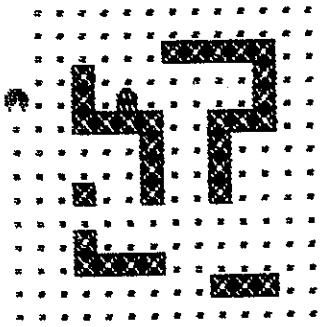
O Fantasma finalmente utilizado é um primo directo do que se encontra à esquerda. Se bem que não pareça aqui particularmente prometedor. deu óptimos resultados no visor. O leitor verificará provavelmente. de vez em quando. que preferirá alterar o gráfico ao vê-lo no visor. Sentirá que é mais fácil trabalhar directamente a partir deste. depressa aprendendo quais os bits das instruções DATA que devem ser alterados. De qualquer modo. passemos à nossa versão final de DOTMAN. completa com algumas saídas. Como acontece tantas vezes. as imagens que apresentamos não fazem

justiça à aparência do jogo. Decerto gostará de jogar este jogo. e de descobrir modos de o melhorar para o tornar mais semelhante ao jogo de perícia vendido comercialmente.

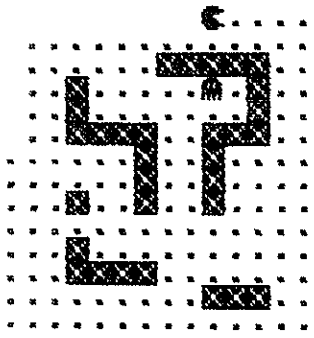
Talvez o leitor queira acrescentar uma rotina para produzir um labirinto variável (colocando X's aleatoriamente na grelha na subrotina da linha 7000). Pode igualmente levar o DOTMAN e o Fantasma a partirem de posições diferentes do labirinto. Pode também incluir uma rotina que calcule as maiores pontuações e as guarde em memória.



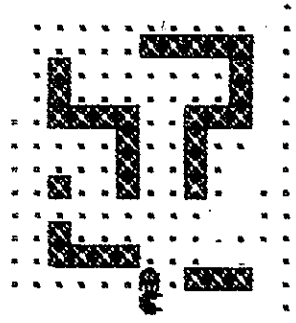
SCORE IS 0



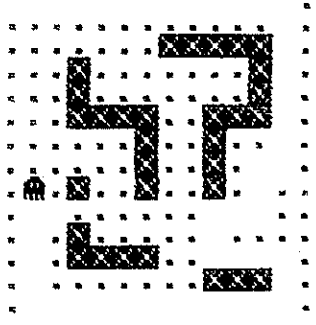
SCORE IS 9428



SCORE IS 21213



SCORE IS 47140



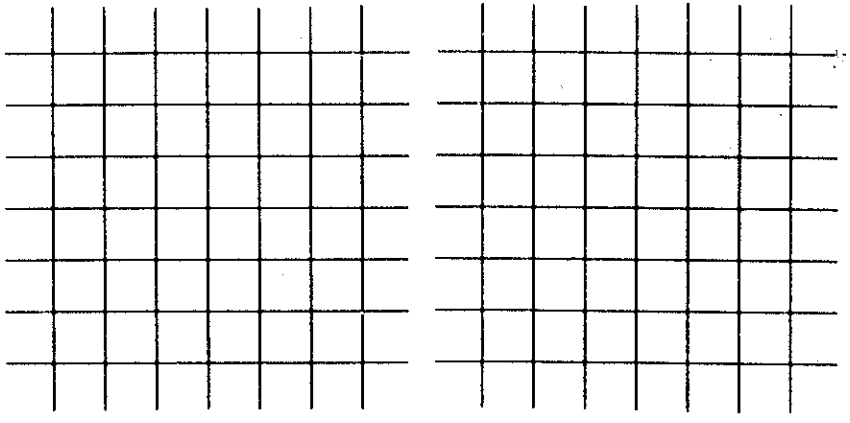
SCORE IS 108422

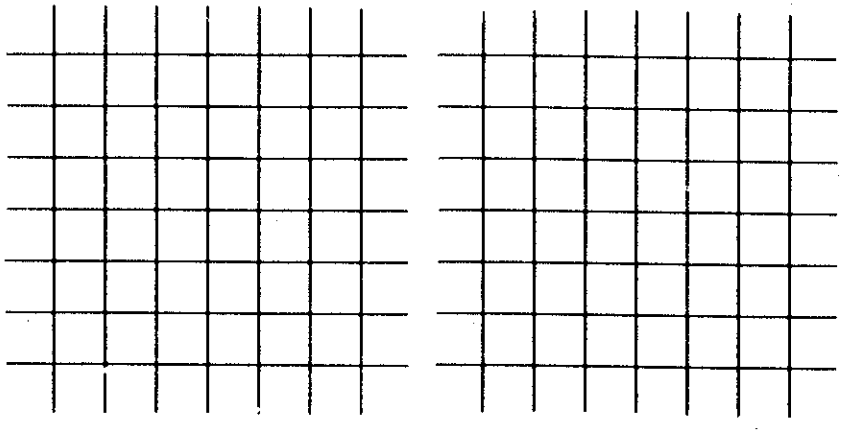
```
  10  GO  SUB  6000
  20  GO  SUB  7000
  30  GO  SUB  5000
  40  GO  SUB  6000
1000  REM  mover dotman
1010  REM  dotman é gráfico B.C.D.E
1020  PRINT  AT  EDD,EDA;"  "
1025  IF  SCREEN$  (DD,DA)="." THEN
 LET  SCORE=SCORE+2357
1030  PRINT  AT  DD,DA;  INK  2;A$
1032  IF  GA=DA  AND  GD=DD  THEN  GO
TO  4000
1035  LET  EDD=DD:  LET  EDA=DA
1040  IF  INKEY$="8"  THEN  IF  DA<13
 THEN  IF  SCREEN$  (DD,DA+1)<>"X"
THEN  LET  DA=DA+1:  LET  A$="C"
1050  IF  INKEY$="5"  THEN  IF  DA>0
THEN  IF  SCREEN$  (DD,DA-1)<>"X"  T
HEN  LET  DA=DA-1:  LET  A$="D"
1060  IF  INKEY$="7"  THEN  IF  DD>0
THEN  IF  SCREEN$  (DD-1,DA)<>"X"  T
HEN  LET  DD=DD-1:  LET  A$="B"
1070  IF  INKEY$="6"  THEN  IF  DD<13
 THEN  IF  SCREEN$  (DD+1,DA)<>"X"
THEN  LET  DD=DD+1:  LET  A$="E"
1980  IF  RND>.2  THEN  GO  TO  1000
1985  PRINT AT 7,16;FLASH 1; "Pontuação;" ;Pontuação

1990  REM  : Fantasma é gráfico G
2000  PRINT  AT  EGD,EGA;  INK  RND*6
;":"
2010  PRINT  AT  GD,GA;  INK  1;"G"
2012  IF  GA=DA  AND  GD=DD  THEN  GO
TO  4000
2015  LET  EGD=GD:  LET  EGA=GA
2020  IF  DD<GD  THEN  IF  SCREEN$  (G
D-1,GA)<>"X"  THEN  LET  GD=GD-1
2030  IF  DD>GD  THEN  IF  SCREEN$  (G
D+1,GA)<>"X"  THEN  LET  GD=GD+1
2040  IF  DA>GA  THEN  IF  SCREEN$  (G
D,GA+1)<>"X"  THEN  LET  GA=GA+1
2050  IF  DA<GA  THEN  IF  SCREEN$  (G
D,GA-1)<>"X"  THEN  LET  GA=GA-1
2999  GO  TO  1000
4000  PRINT AT 17: 0; INK RND * 6; PAPER 9; FLASH
1; "Fim do jogo"
4010  BEEP  .01,RND*60
4020  PRINT  INK  RND*6;  FLASH  1;AT
```

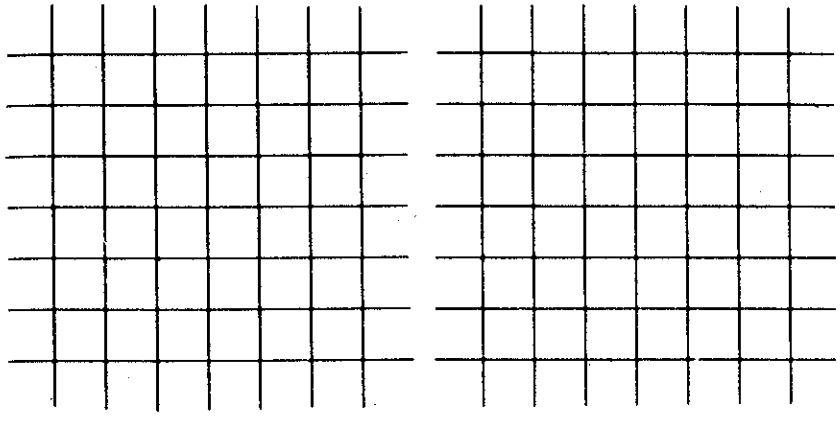
```
 EGD,EGA;" ";AT GD,GA;"■"
4030 PRINT INK RND*6; FLASH 1;AT
 EDD,EDA;" ";AT DD,DA;"■"
4040 GO TO 4000
5000 REM Definir DOTMEN
5010 FOR J=0 TO 7
5020 READ 0
5030 POKE USR "8"+J,0
5040 NEXT J
5050 DATA BIN 00111100,BIN 01111
111,BIN 11111100,BIN 11110000,BI
N 11111000,BIN 11111100,BIN 0111
1111,BIN 00111100
5060 FOR J=0 TO 7
5070 READ 0
5080 POKE USR "C"+J,0
5090 NEXT J
5100 DATA BIN 00111100,BIN 11111
110,BIN 00111111,BIN 00011111,BI
N 00001111,BIN 00111111,BIN 1111
1110,BIN 00111100
5110 FOR J=0 TO 7
5120 READ 0
5130 POKE USR "D"+J,0
5140 NEXT J
5150 DATA BIN 00111100,BIN 01111
110,BIN 11111111,BIN 11111111,BI
N 11110111,BIN 11100111,BIN 0100
0010,BIN 01000010
5160 FOR J=0 TO 7
5170 READ 0
5180 POKE USR "E"+J,0
5190 NEXT J
5200 DATA BIN 01000010,BIN 01000
010,BIN 11100111,BIN 11101111,BI
N 11111111,BIN 11111111,BIN 0111
1110,BIN 00111100
5999 RETURN
6000 REM Definir fantasma
6010 FOR J=0 TO 7
6020 READ 0
6030 POKE USR "G"+J,0
6040 NEXT J
6050 DATA BIN 00111000,BIN 01111
100,BIN 11010110,BIN 11010110,BI
N 11111110,BIN 11111110,BIN 1010
1010,BIN 10101010
6999 RETURN
7000 REM Construir labirinto
```



```
7010 FOR G=1 TO 14
7020 FOR H=1 TO 14
7030 PRINT INK RND*6;".";
7040 NEXT H
7050 PRINT
7060 NEXT G
7065 INVERSE 1
7070 PRINT AT 2,7;"XXXXX"
7080 PRINT AT 3,3;"X";AT 3,11;"X
"
7090 PRINT AT 4,3;"X";AT 4,11;"X
"
7100 PRINT AT 5,3;"XXXX";AT 5,9;
"XXX"
7110 PRINT AT 6,6;"X";AT 6,9;"X"
7120 PRINT AT 7,6;"X";AT 7,9;"X"
7130 PRINT AT 8,3;"X";AT 8,6;"X"
;AT 8,9;"X"
7140 PRINT AT 10,3;"X"
7150 PRINT AT 11,3;"XXXX"
7160 PRINT AT 12,9;"XXX"
7155 INVERSE 0
7170 RETURN
8000 REM Variáveis
8010 LET DD=0
8020 LET DA=0
8030 LET EDD=0
8040 LET EDA=0
8050 LET GA=13
8060 LET GD=13
8070 LET EGA=13
8080 LET EGD=13
8090 LET Pontuação= 0
8100 REM Gráfico B em 8110
8110 LET A$="C"
8999 RETURN
```







21
LIMPAR PARTE DO VISOR

Vamos tratar de uma subrotina bastante útil que nos permite limpar qualquer número de linhas a partir da base do visor. Possibilita manter instruções nas primeiras linhas do visor durante o jogo. ou a pontuação ou quaisquer outros dados especiais enquanto tudo o resto é limpo. A subrotina em causa deve ser chamada por GO SUB 8010.

```
8010 INPUT "Quantas linhas para limpar?":C
8020 IF C<0 OR C>21 THEN GO TO 8
010
8030 FOR F=21 TO 21-C STEP -1
8040 PRINT AT F,0;"                "
8050 NEXT F
8060 PRINT AT F+1,0;
8070 RETURN
```

A linha 8010 pergunta quantas linhas deseja limpar. a partir da linha 21 na parte inferior do visor. A instrução INPUT não está validada: talvez lhe interesse experimentá-la. A instrução da linha

8060 desloca a posição de impressão para o início da parte do visor que foi limpa.

25
ROLAMENTO DA IMAGEM EM BASIC

É fácil escrever programas em código-máquina para carregar blocos de memória de uma posição para outra dentro da RAM, por exemplo para carregar uma imagem numa dada posição. O BASIC é geralmente demasiado lento para nos permitir fazer isto. O único método que pode ser usado realisticamente para rolar a imagem consiste em armazenar uma imagem do visor num quadro de cadeias e imprimi-la. Vejamos quatro exemplos de programas que nos permitem rolar todo o visor para cima, para baixo, para a esquerda ou para a direita.

Estas rotinas são notoriamente mais lentas do que a ordem SCROLL, particularmente as que executam o rolamento para a direita e para a esquerda porque são impressas uma linha de cada vez.

ROLAMENTO PARA CIMA

```
 5 REM Para cima
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$=a$(33 TO )+"
                     "
50 GO TO 30
```

ROLAMENTO PARA BAIXO

```
 5 REM Para baixo
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$="
          "+a$( TO 672)
50 GO TO 30
```


ROLAMENTO PARA A ESQUERDA

```
 5 REM Para a esquerda
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$(f TO f+31)=a$(f+1 TO
f+31)+" "
60 NEXT f
70 GO TO 30
```

ROLAMENTO PARA A DIREITA

```
 5 REM Para a direita
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$(f TO f+31)="  "+a$(f
TO f+30)
60 NEXT f
70 GO TO 30
```

26
LINHAS ESTACIONÁRIAS

O modo mais fácil de conseguir isto consiste em usar a partição de cadeias para "SCROLL" apenas certas partes. Antes do mais, o rolamento para cima. L é o número de linhas que não devem ser roladas.

```
10 DIM a$(704)
20 INPUT a$
25 INPUT L
30 PRINT AT 0,0;a$
40 LET a$(L*32+1 TO )=a$((L+1)
*32+1 TO )
50 GO TO 30
```

Vejamos em seguida o rolamento da imagem para baixo. L é o número de linhas que não devem ser deslocadas na parte inferior do visor.

```
10 DIM a$(704)
20 INPUT a$
25 INPUT L
30 PRINT AT 0,0;a$
40 LET a$( TO 704-L*32)="
                                    "+a$(
TO 704-(L+1)*32)
50 GO TO 30
```

Pode-se aplicar a mesma técnica aos rolamentos laterais, mas como estes são já bastante lentos não parece muito útil fazê-lo. Pode-se desenvolver esta ideia de modo a manter estacionárias quaisquer linhas do visor enquanto as que se encontram acima ou abaixo delas são deslocadas, modificando muito simplesmente a partição da cadeia alfanumérica; mas se for necessária uma aritmética complexa para o conseguir a rotina pode-se tornar extremamente lenta.

Uma outra técnica que se pode aplicar a estas rotinas consiste em "enrolar" a imagem de tal modo que tudo o que desaparece de um dos lados do visor torna a aparecer no lado oposto. Vejamos como é possível conseguir isto num rolamento para cima:

```
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$=a$(33 TO )+a$( TO 32
)
50 GO TO 30
```

E num rolamento para baixo:

```
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$=a$(673 TO )+a$( TO 6
72)
50 GO TO 30
```

Num rolamento para esquerda:


```
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$(f TO f+31)=a$(f+1 TO
f+31)+a$(f)
60 NEXT f
70 GO TO 30
```

Para um rolamento para a direita, altere a linha 50 para:

```
50 LET a$(f TO f+31)=a$(f+31)+
a$(f TO f+30)
```

Finalmente, recorde que em vez de usar PRINT para colocar seja o que for no visor, pode usar

LET AS (x)= "X"

dado o que tudo o que é impresso com PRINT não é rolado — só aparece no visor o conteúdo do quadro AS.

XXVII

GRÁFICOS MÓVEIS

A técnica que se aplica nos gráficos móveis consiste em desenhar primeiro um caracter numa dada posição durante um curto período de tempo, apagando-o depois e voltando a desenhá-lo noutra posição. São usadas variáveis para recordar a posição do caracter. Vejamos um exemplo desta técnica:

```
10 LET x=0
20 PRINT AT 5,x;"■"
30 PRINT AT 5,x;" "
40 LET x=x+1
50 GO TO 20
```

Este programa não é propriamente óptimo — o ponto negro parece cintilar enquanto se move no visor, e o programa pára com uma mensagem de erro sempre que o ponto atinge a margem direita do

visor. Acontece que X é a variável que diz ao computador em que ponto do visor deve imprimir o ponto negro, e quando o seu valor é superior a 31 a máquina deixa de poder imprimir porque os limites do visor são 0 e 31: qualquer tentativa de usar um número maior do que 31 equivale portanto a querer imprimir o ponto fora do visor, do lado direito. Ora o Spectrum, sendo uma máquina bastante esperta, decide que isto não é possível e portanto pára a execução do programa e diz-nos o que está mal a fim de podermos fazer a correcção necessária. Façamo-la. O modo mais simples consiste em garantir que, no caso de o valor de X ultrapassar os limites aceitáveis, a máquina os corrija automaticamente. Um modo de o conseguir obriga a acrescentar uma nova linha:

```
45 IF x>31 THEN LET x=0
```

Mas como já dispomos de uma linha dizendo LET x = 0 (linha 10), podemos conseguir o mesmo enviando a execução para essa linha, por exemplo:

```
45 IF x>31 THEN GO TO 10
```

Esta técnica é muito usada em programas — reenviar a execução para o princípio de modo a redefinir algumas variáveis, levando-as a assumirem os seus valores iniciais. Neste exemplo ambos os métodos produzem os mesmos resultados, mas em certos programas só um deles poderá ser apropriado.

O passo seguinte consiste em melhorar a apresentação no visor, retirando a cintilação ao caracter. Um modo de o conseguir é manter o ponto visível durante um tempo mais prolongado do que aquele durante o qual é limpo:

```
10 LET x=0
20 PRINT AT 5,x;"■"
30 FOR f=1 TO 10
40 NEXT f
50 PRINT AT 5,x;" "
60 LET x=x+1
70 IF x>31 THEN LET x=0
80 GO TO 20
```


Esta técnica parece dar resultados bastante bons, mas na maior parte dos programas é necessário executar outros cálculos que tornarão a execução mais lenta. Nessas condições o ciclo das linhas 30 e 40 pode aumentar desnecessáriamente o tempo necessário para a execução. Experimentemos agora reduzir a cintilação diminuindo o tempo entre a impressão de um espaço e a impressão do ponto:

```
10 LET x=0
20 LET p=x
30 LET x=x+1
40 IF x>31 THEN LET x=0
50 PRINT AT 5,p;" ";AT 5,x;"■"
60 GO TO 20
```

O resultado parece agora bastante melhor.

Aqui, X é a variável que recorda a posição actual do ponto. P recorda a posição anterior a fim de permitir apagá-la desenhando um espaço por cima. A linha 10 inicializa X para um valor de 0. A linha 20 determina o valor P tornando-o igual a X *antes de* X ser aumentado na linha 30. Este aumento pode ser qualquer — experimente alterar o número antes do sinal + para ver o efeito. Parece mover-se mais rapidamente, o que pode ser ou não vantajoso. Por agora fiquemos com o valor 1 depois do sinal +.

A linha 40 serve para garantir que quando o ponto negro atinge o lado direito do visor é reenviado para o lado esquerdo atribuindo novamente o valor 0 a X. Assegura-se assim o abastecimento costante de pontos negros... A linha 50 executa a impressão — note o modo como é possível incluir duas funções AT na mesma linha juntando-as com um ponto e vírgula. A propósito, pode conseguir--se o mesmo usando TAB. Tente escrever estas situações em duas linhas separadas para ver se introduzem alguma diferença no programa:

```
50 PRINT AT 5,p;" "
55 PRINT AT 5,x;"■"
```

Pode encurtar-se o programa em uma linha alterando a linha 30 para:

```
30 LET x=x+1 AND x<31
```

O modo como esta linha funciona é bastante complexo, sendo melhor explicada nos capítulos sobre instruções condicionais. Simplificando, significa "se X é inferior a 31 acrescentar 1 ao valor de X; mas se X não é inferior a 31 fazer X igual a zero".

Eliminemos agora a linha 40 (a que tornámos redundante) e renumeremos as linhas em passos de 10. Ficaremos com o seguinte programa:

```
10 LET x=0
20 LET p=x
30 LET x=x+1 AND x<31
40 PRINT AT 5,p;" ";AT 5,x;"■"
50 GO TO 20
```

Talvez tenha notado que o espaço é sistematicamente impresso uma coluna atrás do ponto, sendo portanto possível usar muito simplesmente PRINT AT 5,X; "espaço□", isto é, imprimir o espaço e o símbolo sráfico juntos e dispensar a variável P. Isto permite melhorar a apresentação, mas provoca problemas quando se atinge o fim da linha e se torna necessário empregar uma outra linha de programação para apagar a posição anterior. Para compreender melhor o que estamos a dizer, experimente o seguinte:

```
10 LET x=0
20 LET x=x+1 AND x<30
30 PRINT AT 5,x;"■"
40 GO TO 20
```

Compreende o que pretendemos dizer? De cada vez que surge um novo ponto no visor o antigo mantém-se à direita. Tentemos agora apagar estes pontos:

```
10 LET x=0
20 LET x=x+1 AND x<30
30 PRINT AT 5,x;"■"
35 IF x=0 THEN PRINT AT 5,31;"
"
40 GO TO 20
```


Podemos usar novamente AND para encurtar um pouco o programa:

```
10 LET x=0
20 LET x=x+1 AND x<30
30 PRINT AT 5,x;" ■";AT 5,31;"
" AND x=0
40 GO TO 20
```

A linha 35 do primeiro programa e a segunda parte da linha 30 do segundo garantem que só é impresso um espaço no caso de o ponto ter atingido o fim da linha. Vejamos um outro modo de fazer isto, usando a variável de controlo de um ciclo FOR / NEXT em vez da variável convencional X. Este método utiliza uma quantidade de memória um pouco inferior e é um pouco mais rápido de executar.

```
10 FOR x=0 TO 30
20 PRINT AT 5,x;" ■";AT 5,31;"
" AND x=0
30 NEXT x
40 GO TO 10
```

Um outro modo de usar este método consiste em usar PRINT AT 5,31;" espaço□" fora do ciclo FOR/NEXT. Isto tem a vantagem de não obrigar o computador a examinar tantas vezes a expressão, pelo que o programa se torna consideravelmente mais rápido:

```
10 FOR x=0 TO 30
20 PRINT AT 5,x;" ■"
30 NEXT x
40 PRINT AT 5,31;" "
50 GO TO 10
```

Acabámos portanto por dispor de uma rotina que é bastante rápida de executar e muito económica em termos de memória usada. Vimos por outro lado alguns dos problemas que ocorrem no desenvolvimento deste tipo de programa. Os exemplos que estudámos até agora têm a ver com a obtenção de um movimento constante no visor. Mas necessitaremos certamente de mover caracteres no visor sujeitos a um comando por teclado. Para o conseguir, será necessário voltar à declaração INKEYS, muito útil para a movi-

mentação de símbolos gráficos. Se o leitor leu o capítulo anterior, terá verificado que INKEYS é o caracter que corresponde à tecla que é premida. Se se carrega em K, INKEYS é "K", isto é, INKEYS= "K" Se não carregamos em nada, INKEYS passa a ser uma cadeia vazia, ou "" (INKEYS não pode ser "espaço" porque carregar em SPACE actua como BREAK quando um programa está a correr, e pára o programa BASIC). Veremos agora alguns outros modos de utilizar INKEYS, porque se trata de uma função muito potente e extremamente útil para movimentação de gráficos. A maior parte dos jogos de perícia ("Arcade") obrigam o utilizador a carregar em botões, accionar interruptores ou mover púnhos ("joysticks"). Como o computador não dispõe originalmente de joysticks, se bem que os leitores os possa comprar à parte, todo o comando dos gráficos deve ser realizado através do teclado.

Para comandar o movimento no visor, existem certas teclas mais vantajosas do que outras. Consideremos por exemplo as teclas Z e M da fiada inferior do teclado. Podem ser usadas para mover para a esquerda (Z) ou para a direita (M), para comandar o movimento esquerda / direita no visor, por exemplo num jogo do tipo "invasores". A vantagem destas teclas consiste em encontrarem-se colocadas logicamente, tornando-se bastante práticas. Uma desvantagem consiste porém em a tecla M se encontrar perto da tecla BREAK, podendo o utilizador parar acidentalmente o programa ao fazer um último esforço para se libertar da Ameaça Verde!

Se deseja comandar o movimento para a esquerda, para a direita, para cima e para baixo (por exemplo para mover o cursor no programa de processamento de texto no final deste livro) talvez seja melhor escolher as teclas 5, 6, 7 e 8 dado que estão assinaladas por quatro setas apontando nas direcções e sentidos requeridos. Estão juntas para uma maior facilidade de uso, e afastadas da tecla BREAK.

O modo mais comum de usar INKEYS num programa com gráficos móveis (quer se trate de um jogo ou de uma aplicação mais séria) consiste em incluí-la numa instrução condicional comandando o valor de uma variável. Por exemplo:

```
IF INKEY$= "8" THEN LET X=X+ 1
```

Neste caso, acrescenta-se 1 ao valor de X no caso de o utilizador carregar na tecla 8: mas X mantém-se no caso de não se carregar nessa tecla, independentemente de ser ou não premida qualquer

outra. Em seguida a variável pode ser usada para comandar, por exemplo, o ponto onde se deve imprimir um símbolo no visor:

```
PRINT AT 5,X; "□"
```

Se adoptarmos a convenção de definir com X a posição horizontal no visor e com Y a posição vertical, então quanto maior for o valor de X mais para a direita será impresso o caracter, e quanto maior for Y mais em baixo será impresso.

Sabendo isto podemos escrever um curto programa que comande o movimento de um caracter (por exemplo um asterisco) usando as teclas 5,6,7 e 8 (as teclas com setas):

```
100 IF INKEY$="5" THEN LET X=X-
1
110 IF INKEY$="8" THEN LET X=X+
1
120 IF INKEY$="7" THEN LET Y=Y-
1
130 IF INKEY$="6" THEN LET Y=Y+
1
140 PRINT AT Y,X;"*"
```

Antes de podermos executar este programa teremos de definir X e Y, senão nada acontecerá. Acrescentemos portanto as seguintes linhas:

```
10 LET X= 0
20 LET Y= 0
```

Podemos em seguida executar o programa. Conseguimos assim imprimir um asterisco no canto superior esquerdo do visor, e o programa pára depois da linha 140. Para impedir que isto aconteça, podemos acrescentar uma linha como a seguinte:

```
200 GO TO 100
```

Garantimos assim que o computador continue a fazer o seu trabalho. Faça executar o programa e experimente carregar nas teclas 5,6,7 e 8, uma de cada vez. Verificará que o asterisco se desloca, deixando uma cauda de asteriscos atrás de si. Continue a

mover-se em direcção à margem, carregando na tecla apropriada. Começam a acontecer coisas estranhas. Se sair pela base do visor ou pelo seu lado direito, o programa pára com uma mensagem de erro. Isto deve-se ao facto de o valor de X se ter tornado superior a 31 ou de o de Y superior a 21, sendo o computador solicitado para imprimir um asterisco fora do visor, o que não pode acontecer.

No entanto se tentar sair pela parte superior do visor ou pelo seu lado esquerdo, acontecem coisas ainda mais estranhas — o asterisco começa a mover-se na direcção oposta! Existe uma explicação simples para isto. Quando o valor X ou Y é negativo (como acontece nestes dois últimos casos), PRINT ignora o sinal (aceita o valor ABS da variável, se quiser) e portanto obriga algumas teclas a trocarem de funções! Isto não é muito útil, obrigando a trazer novamente o asterisco à mesma margem para recuperar um funcionamento normal. Necessitamos portanto de um método que permita que o asterisco páre ao atingir uma das margens, não tentando sair para fora do visor.

O método mais simples consiste em impedir X e Y de assumirem valores que provocam estes problemas. Os valores permissíveis são 0 a 31 para X e 0 a 21 para Y. Vejamos como é possível obter estes resultados. Alteremos as linhas 100, 110, 120 e 130 do seguinte modo:

```
 100 IF INKEY$="5" AND x>0 THEN
LET x=x-1
 110 IF INKEY$="8" AND x<31 THEN
 LET x=x+1
 120 IF INKEY$="7" AND y>0 THEN
LET y=y-1
 130 IF INKEY$="6" AND y<21 THEN
 LET y=y+1
```

Isto leva o asterisco a manter-se convenientemente no visor, mas enfrentamos o problema de eliminar a cauda de asteriscos visíveis. Temos de incluir um modo de apagar o asterisco quando sai de uma posição. O melhor modo de o fazer consiste em usar um segundo conjunto de variáveis que recordem a antiga posição do caracter, e se esta for diferente da actual será impresso um espaço na posição abandonada.

Para tal acrescentemos as seguintes linhas:


```
  50 LET a=x
  60 LET b=y
 135 IF a<>x OR b<>y THEN PRINT
AT b,a;" "
 200 GO TO 50
```

O programa que agora temos no computador terá a aparência seguinte:

```
  10 LET x=0
  20 LET y=0
  50 LET a=x
  60 LET b=y
 100 IF INKEY$="5" AND x>0 THEN
LET x=x-1
 110 IF INKEY$="8" AND x<31 THEN
 LET x=x+1
 120 IF INKEY$="7" AND y>0 THEN
LET y=y-1
 130 IF INKEY$="6" AND y<21 THEN
 LET y=y+1
 135 IF a<>x OR b<>y THEN PRINT
AT b,a;" "
 140 PRINT AT y,x;"*"
 200 GO TO 50
```

Dispomos agora do módulo básico de um programa de gráficos, móveis. Quando resolvermos conceber um jogo utilizando esta rotina seremos provavelmente obrigados a alterar alguns pormenores ou a ordem das instruções, mas os princípios aplicados continuarão a ser os mesmos. Tal como aconteceu anteriormente, podemos ainda encurtar um pouco esta rotina, por exemplo:

```
  10 LET x=0
  20 LET y=0
  30 LET a=x
  40 LET b=y
 100 LET x=x-(INKEY$="5" AND x>0
)+(INKEY$="8" AND x<31)
 110 LET y=y-(INKEY$="7" AND y>0
)+(INKEY$="6" AND y<21)
 135 IF a<>x OR b<>y THEN PRINT
AT b,a;" "
 140 PRINT AT y,x;"*"
 200 GO TO 30
```

Esta versão ocupa aproximadamente 50 bytes a menos do que a versão anterior. É ainda possível combinar a linha 140 com a linha 135 numa única declaração condicional, dado que não é necessária qualquer instrução PRINT no caso de o asterisco não se ter deslocado. Isto significa que podemos tirar a linha 140. Vejamos como se deve nesse caso alterar a 135:

```
135 IF a<>x OR b<>y THEN PRINT
AT b,a;" ";AT y,x;"*"
```

Não se esqueça de retirar a linha 140. A desvantagem disto é que apesar de poupar 5 bytes o asterisco não é inicialmente impresso no visor, só começando a sê-lo depois de movimentado. Talvez seja portanto melhor manter a versão anterior.

28
SCREEN$ E ROLAMENTO DE IMAGEM

Vejamos agora um outro método bastante útil para movimentar gráficos, o rolamento ("scrolling") da imagem. Este, que pode ser executado automaticamente incluindo a linha POKE 23692,-1, desloca tudo o que se encontra no visor de uma linha para cima, e se existia alguma coisa impressa na primeira linha desaparece.

O programa que se segue — ROAD RUNNER — mostra o funcionamento de "scrolling". Neste programa tentaremos conduzir uma longa fila de pequenas "viaturas" por um caminho tortuoso demarcado por asteriscos vermelhos. Os comandos são "Z" e "M", que servem para deslocar os carros para a esquerda e a direita respectivamente.

As linhas 80 e 90 movem o caminho aleatoriamente, garantido que não saia por uma das margens laterais do visor. A linha 110 imprime o "carro", que é deslocado para cima (tal como o caminho) pelas linhas 130 e 140.

A linha 160 introduz uma nova função do Spectrum, a SCREENS, que é usada para indicar o estado do visor no local especificado à frente da palavra SCREENS. Na linha 160, o computador utiliza esta função para observar a posição imediatamente à frente da última posição do carro. Se descobre aí um asterisco, sabe que o

carro está quase a chocar, enviando portanto a execução para a linha 200, onde se inicia a rotina "Chocou!".

```
  10 REM ROAD RUNNER
  20 LET C=0
  30 LET T=0
  40 GO SUB 250
  50 LET A=10
  60 LET X=13
  70 LET Y=12
  80 LET K=INT (RND*2)
  90 LET A=A-(K=1 AND A>1)+(K=0
AND A<24)
 100 REM Nas linhas 110 e 200 "C" é um caracter gráfico.
 110 PRINT AT Y,X-1; INK 1;"C"
 120 PRINT AT 20,A; INK 2;"*";TA
B A+5;"*"
 130 PRINT
 140 POKE 23692,-1: PRINT
 150 PRINT INK 6; PAPER 2;AT 0,10;
"A pontuação é";T;" "
 160 IF SCREEN$ (Y+1,X-1)="*" TH
EN GO TO 200
 170 LET X=X-(INKEY$="Z")+(INKEY
$="M")
 180 LET T=T+1
 190 GO TO 80
 200 PRINT AT Y,X-1; INK 1;"C"
 210 PRINT AT 8,8; FLASH 1; BRIG
HT 1;"   Você chocou!!!   "
 220 PRINT AT 8,10; FLASH 1; BRI
GHT 1; INK RND*7; PAPER 5;"
"Você pontuou";T;" "
 230 BEEP .01,RND*20-RND*20
 240 GO TO 210
 250 FOR J=0 TO 7
 260 READ Z
 270 POKE USR "C"+J,Z
 280 NEXT J
 290 RETURN
 300 DATA BIN 00110110,BIN 00110
110,BIN 00111110,BIN 00010100,B
IN 00111110,BIN 00110110,BIN 00
011100,0
```

Vejamos alguns outros programas que ilustram as ideias de movimentação de gráficos que acabámos de descrever.

"READ ARROW"

O objectivo deste jogo é disparar, carregando em qualquer tecla excepto em BREAK, quando RED ARROW (Seta Vermelha) voa sobre a sua base. Dispõe de um número limitado de tiros, pelo que convém acertar tantas vezes quanto possível antes de o jogo acabar.

Vejamos a listagem:

```
  5 REM Os caracteres A, B e C da linha 10
  7 REM são caracteres gráficos
  8 GO SUB 200
  9 PAPER 7: CLS
 10 PRINT AT 15,14; INK 1;"  ";AT 4,0; PAPER 7; INK 2;"          H
its:0          ";AT 16,0; INK 3;"
 20 LET h=0
 30 LET m=h
 40 LET y=RND*6+7
 50 FOR x=0 TO 19
 60 IF h+m=250 THEN GO TO 1000
 70 REM Na linha 80, D é caracter gráfico
 80 PRINT AT y,x; INK 2;" D"
 90 IF INKEY$<>"" AND x=14 THEN GO TO 140
100 IF INKEY$<>"" THEN LET m=m+1
110 NEXT x
120 PRINT AT y,20;" "
130 GO TO 40
140 LET h=h+1
150 FOR f=1 TO 30
160 PRINT AT y,x; INK 2; PAPER 4; BRIGHT 1; FLASH 1;"Acertou"; FOR z=1 TO 20: BORDER RND*7: BEEP .01,39-3*z: PRINT AT y,x;"       "
170 NEXT f
180 PRINT AT 4,11; PAPER 7; INK 2;h
190 GO TO 40
200 FOR j=0 TO 7
210 READ q
220 POKE USR "A"+j,q
230 NEXT j
240 FOR j=0 TO 7
250 READ q
260 POKE USR "B"+j,q
270 NEXT j
280 FOR j=0 TO 7
290 READ q
300 POKE USR "C"+j,q
310 NEXT j
320 FOR j=0 TO 7
330 READ q
340 POKE USR "D"+j,q
350 NEXT j
400 RETURN
500 DATA BIN 00000001,BIN 00000011,BIN 00000110,BIN 00001100,BIN 00011000,BIN 00110000,BIN 01100000,BIN 11000000
510 DATA BIN 11111111,BIN 11111111,0,0,0,0,0,0
520 DATA BIN 11000000,BIN 01100000,BIN 00110000,BIN 00011000,BIN 00001100,BIN 00000110,BIN 00000011,1
530 DATA BIN 00001000,BIN 00010100,BIN 00100010,BIN 01000010,BIN 00111111,BIN 01000010,BIN 10000100,BIN 00001000,0
1000 FOR g=1 TO 70
1010 BEEP .01,g/2: BEEP .01,40-g
1020 NEXT g
1030 RUN
```

A linha 10 define a parte principal do visor, aquela que não se move durante o programa, H é a variável que recorda a quantidade de vezes que se acerta e M. a quantidade de falhas (tiros que não atingem o OVNI...), estando ambas originalmente em zero. A linha 40 define a coordenada Y da RED ARROW, dando-lhe um valor aleatório entre 7 e 13. PRINT aceita o valor inteiro mais próximo se uma das coordenadas não for inteiro, pelo que RND * 6 + 7 pode assumir valores entre 7 e 13. Isto dá-nos a posição da RED ARROW no visor. O ciclo principal para controlo do voo do OVNI inicia-se na linha 50. 0 é a posição de partida, 19 a posição final. A linha 60 verifica se você já utilizou todos os seus tiros, enviando depois a execução para a linha 1000 se já o tiver feito. A linha 80 apaga a posição anterior imprimindo um espaço numa posição atrás da RED ARROW. A linha 90 verifica se o utilizador está a carregar

em alguma tecla e se a RED ARROW se encontra directamente por cima da base, enviando depois a execução para a rotina da linha 140. Esta acrescenta um ao número de pontos e fornece uma imagem "explosiva" no ponto onde o avião foi atingido, reenviando a execução de novo para a linha 40. No entanto, se estava a ser premida uma tecla e a RED ARROW não se encontrava directamente sobre a base, é somado um ao número de tiros falhados. Se a RED ARROW atinge o final do seu movimento, o programa sai do ciclo FOR / NEXT e alinha 120 apaga a sua posição final antes de a linha 130 reenviar o programa de novo para a linha 40, a fim de imprimir o ojecto na sua nova posição.

"GARBAGE GOBBLER"

O objectivo deste jogo é limpar o lixo que aparece no visor sob a forma de manchas de cor magenta. Comanda-se o "gobbler" usando as teclas 5,6,7 e 8 sendo o movimento executado na direcção e sentido das setas impressas em cada tecla. O lixo é limpo passando sobre ele. O jogador dispõe de um tempo limitado, sendo-lhe indicado a quantidade de lixo limpo... O ciclo das linhas 40 a 60 distribui aleatoriamente o lixo.

```
  10 REM GARBAGE GOBBLER
  20 GO SUB 1000
  30 REM Na linha 50. M é caracter gráfico
  40 FOR G=1 TO 50
  50 PRINT AT RND*16+3,RND*26+4;
 INK 1;"▒"
  60 NEXT G
  70 LET X=0
  80 LET Y=0
  85 FOR G=1 TO 500
  90 LET A=X
 100 LET B=Y
 110 LET Y=Y-(INKEY$="7" AND Y>1
)+(INKEY$="6" AND Y<20)
 120 LET X=X-(INKEY$="5" AND X>1
)+(INKEY$="8" AND X<31)
 130 IF ATTR (Y,X)=49 THEN LET
pontuação= pontuação + 1 :PRINT AT 0,20; INK
 RND*7; PAPER 9; FLASH 1;  "Pontua-
ção;" ;pontuação: AT 20,0; "Tempo;";500-g;" "
```



```
 140 IF ATTR (Y,X)=49 THEN BEEP
.01,40
 150 REM Na linha 160. D é caracter gráfico
 160 PRINT AT B,A;" ";AT Y,X; IN
K 2;"◀"
 170 NEXT G
 180 PRINT AT 0,20; INK RND*7; P
APER 9; FLASH 1;"Pontuação:" ;pontuação;
 AT 20,0; "Tempo;";0;"  "
 190 BEEP .1,RND*50: GO TO 190
 900 STOP
1000 FOR A=0 TO 7
1010 READ G
1020 POKE USR "M"+A,G
1030 NEXT A
1040 DATA BIN 10100101,BIN 01011
010,BIN 10100101,BIN 01011010,BI
N 01011010,BIN 10100101,BIN 0101
1010,BIN 10100101
1050 FOR A=0 TO 7
1060 READ G
1070 POKE USR "D"+A,G
1080 NEXT A
1090 DATA BIN 00011111,BIN 01111
111,BIN 11111100,BIN 11111000,BI
N 11100000,BIN 11111000,BIN 0111
1100,BIN 00111111
1100 BORDER 2: PAPER 6: CLS
1120 LET pontuação= 0
1500 RETURN
```

Vamos explicar agora como se pode descobrir o que está no visor, de tal modo que a máquina saiba que o "gobbler" se prepara para passar por cima de alguma coisa. Este método será usado bastantes vezes nos seus programas. A função aqui mais importante é ATTR que, como pode verificar nas linhas 130 e 140, é semelhante em forma a PRINT AT. As linhas 130 e 140 verificam os atributos do quadrado Y, X que se encontra à frente do "gobbler". Se encontra um valor de 49, sabe que esse quadrado contém lixo, actualizando portanto a pontuação e o tempo. O número de facto produzido pela função ATTR depende do caracter estar ou não a cintilar, de ser ou não brilhante e das cores de INK e PAPER que lhe correspondem.

Como ATTR é um pouco difícil de usar, é melhor construir uma pequena rotina para imprimir os resultados de ATTR antes de decidir definitivamente qual o valor que deseja verificar. A "ausência" de um valor quando o "gobbler" passa por cima de uma posição em branco não é recomendada; em vez disto, convirá verificar a presença de um valor bem definido. Foi isso que fizémos ao escrever este programa, deixando inicialmente em branco a linha 140, e escrevendo na linha 130 PRINT AT 0,0; ATTR (Y,X); em seguida vimos o que acontecia quando o "gobbler" se preparava para passar sobre um quadrado com lixo. Depois de estudarmos os valores que então eram impressos no visor pela linha 130, determinámos o valor mais seguro para utilizar na instrução da linha 140 para verificar a passagem do "gobbler" sobre o lixo e actualizar a pontuação.

Este método de trabalho é o único seguro, pelo que aconselhamos o leitor a preferí-lo.

29
UTILIZAÇÃO DE CADEIAS ALFANUMÉRICAS

Finalmente, existe um outro método para produzir movimentos no visor utilizando a linguagem BASIC, que no entanto não é muito usado — recorrendo ao uso de cadeias alfanuméricas. As cadeias podem ser impressas com grande rapidez, e as grandes potencialidades do computador em termos de partição ("slicingg") destas cadeias permitem-nos dispor de uma ferramenta poderosa. O método básico consiste em definir um quadro de cadeias alfanuméricas suficientemente grande para cobrir toda a área do visor utilizada em movimento, imprimindo esse quadro numa dada posição. Para simular o movimento podemos imprimir diferentes partes do quadro ou alterar o seu conteúdo.

1) Imprimir diferentes partes do quadro alfanumérico.
Experimente o programa bastante curto que se segue:

```
10 DIM A$(32)
20 LET A$(1)="+"
30 FOR A=32 TO 1 STEP -1
40 PRINT AT 20,0;A$(A TO )+A$(
TO A-1)
50 NEXT A
60 GO TO 30
```

O leitor consegue descobrir porque razão é necessário obrigar a linha 30 a contar *para trás*, de 32 para 1? O que aconteceria se a linha 30 contasse de 1 até 32 (30 FOR A= 1 TO 32)? Desenhe num pedaço de papel (ou use a impressora se tiver uma) cada uma das fases da construção da imagem pela máquina. Note a elevada velocidade possível, e o modo como a posição anterior é limpa e a nova posição impressa de uma só vez. É possível obter um efeito interessante alterando a linha 20 para 20 INPUT A$ e dando entrada a uma mensagem com um máximo de 32 caracteres. Isto é semelhante a um tipo de imagem muito usado em lojas para fins publicitários, se bem que não seja o tipo de efeito que normalmente se deseja.

Este método de produção de gráficos móveis utilizando cadeias alfanuméricas é muito útil porque não altera o conteúdo destas cadeias, limitando-se a apresentá-las por uma ordem diferente; é portanto fácil recuperar a informação inicial em qualquer momento. Pode ser usado para um efeito publicitário como o do programa seguinte.

```
20 INPUT "Indique mensagem";AS
30 IF A$="" THEN GO TO 20
40 LET S=50
50 FOR B=1 TO LEN A$+33
70 PRINT AT 5,0;("                "+A$+"
                ")(B T
O B+31)
80 LET S=S-(S AND INKEY$="F" A
ND S>1)+(S AND INKEY$="D")
90 IF INKEY$="A" THEN GO SUB 1
40
100 FOR A=1 TO 5
110 NEXT A
```

```
120 NEXT B
130 GO TO 50
140 FOR A=1 TO 200
150 NEXT A
160 RETURN
```

O programa pede-nos que indiquemos uma mensagem que começará a aparecer pelo lado direito do visor e se desloca para a esquerda, acabando por desaparecer e recomeçando de novo. A sequência de "rotação" da mensagem inicia-se a velocidade lenta mas pode ser apressada carregando na tecla F, ou tornada ainda mais lenta carregando na tecla D. A velocidade maior é muito rápida, e a menor extremamente lenta. Pode-se "congelar" a imagem durante curtos períodos de tempo carregando na tecla A. Pode-se parar o programa em qualquer momento carregando em BREAK.

Desde que o computador tenha suficiente memória disponível, a dimensão da imagem é apenas limitada pelas maiores dimensões de cadeia que a máquina é capaz de utilizar. Na prática, no entanto, se enche o visor com a mensagem quando se dá entrada a esta (isto é, quando tem mais de 24 linhas de 32 caracteres cada), todos os caracteres subsequentes parecem estar por baixo do visor — experimente para compreender o que quero dizer. E aviso o leitor mais curioso de que vai ficar com dores no dedo...

A mensagem começa a surgir do lado direito do visor a cerca de um quarto da altura, e move-se sobre o visor para a esquerda. Depois de ter desaparecido na margem esquerda, volta a surgir do mesmo modo que inicialmente e repete este ciclo indefinidamente. A velocidade a que o faz pode ser modificada do modo já descrito; e a imagem pode ser congelada também como já referimos.

A linha 70 é bastante complexa: para impedir a alteração do conteúdo da cadeia AS, todo o conteúdo do primeiro par de parêntesis é tratado como uma única cadeia bastante comprida consistindo em 32 espaços seguidos pela mensagem e outros 32 espaços. No segundo par de parêntesis é indicada a partição que deve ser realizada sobre a cadeia e que define a parte desta que é impressa. Note que AS ainda retém a sua identidade. Quaisquer que sejam as partes escolhidas, a cadeia impressa tem sempre um comprimento de 32 caracteres.

Tal como o apresentamos, o programa não permite alterar a mensagem depois de estar em execução — o leitor necessitará de usar BREAK e em seguida RUN novamente o programa. Um modo de fazer no entanto aquelas alterações consiste em acrescentar a linha que se segue, de tal modo que ao carregar em "1" (EDIT) o programa retome a execução:

```
65 IF INKEY$="1" THEN RUN
```

2) Deslocar os elementos do quadro alfanumérico. Experimente o programa seguinte:

```
10 DIM a$(32)
20 FOR a=1 TO 32
30 LET a$(a)="+"
40 PRINT AT 20,0;a$
50 LET a$(a)=" "
60 NEXT a
70 GO TO 20
```

Este método permite-nos obter uma maior flexibilidade. Podemos tratar cadeias alfanuméricas com rapidez e eficiência. Estas cadeias são muito úteis para armazenar informações porque é possível nesse caso aceder a elas rapidamente e de um modo bastante mais prático do que ao introduzi-las em declarações REM, por exemplo, e a velocidade com que é possível imprimi-las transforma-as num método muito simples de produzir imagens. A principal desvantagem é que gastam bastante memória dado que a informação é guardada tanto no ficheiro de imagem ("display file") como nas cadeias em causa, possivelmente também no próprio programa. Vamos apresentar um programa que move uma imagem baseando-se em informações contidas nas cadeias alfanuméricas a imprimir.

O programa é chamado "Invasores" porque se trata de uma versão de um jogo de perícia com o mesmo nome. É uma versão muito simplificada, sendo incluído aqui apenas com o objectivo de demonstrar o uso das cadeias para efeito de movimento de imagem.

Podemos usar as teclas 5 e 8 para executar movimentos para a esquerda ou direita respectivamente. Dispara-se sobre os invasores carregando na tecla 7. Se nos encontramos directamente por baixo de um invasor ao disparar este é destruído e desaparece.

Existem sete ondas de invasores e é necessário destrui-los a todos para ganhar.

```
  1 REM Invasores
  2 REM Na linha 40. F é caracter gráfico
  3 REM Na linha 120 B é caracter gráfico.
  4 BORDER 2: PAPER 0: CLS
  5 GO SUB 290
 10 DIM a$(32)
 20 DIM b$(32)
 30 FOR d=1 TO 7
 35 REM Na linha 40 deixam-se 6 espaços antes do primeiro
F, e 3 espaços entre cada 2 caracteres
 40 LET a$="      F   F   F   F
    F   F"
 50 LET X=INT (RND*32)
 60 LET C=X
 70 PRINT AT 5,12; INK X/5; PAP
ER 9; FLASH 1; BRIGHT 1; "ONDA:
";D;" "
 80 LET C=0
 90 FOR B=D+9 TO 19 STEP 2
100 FOR A=0 TO 31
110 LET X=X+(INKEY$="8" AND X<3
1)-(INKEY$="5" AND X>0)
120 PRINT AT B,0; INK RND*2;A$;
AT 20,C;" " AND C<>X;AT 20,X; IN
K 4;"A"
130 IF A$=B$ THEN GO TO 200+(60
AND D=7)
140 LET C=X
150 IF INKEY$="7" THEN BEEP .01
,20: IF A$(X+1)="F" THEN LET A$(
X+1)="■": PRINT AT B,0; INK 6;A$
: BEEP .01,50: LET A$(X+1)=" ":
LET SC=SC+1: PRINT AT 2,10; INK
2; PAPER 0; BRIGHT 1; FLASH 1;"Pon
tuação: > ";SC*27187
160 NEXT A
165 LET A$=A$(4 TO )+A$( TO 3)
170 PRINT AT B,0;B$
180 NEXT B
190 GO TO 240
200 FOR B=1 TO 10
210 NEXT B
220 PRINT AT 20,C;" "
230 NEXT D
240 PRINT FLASH 1; INK RND*7; P
APER 9;"ATERRARAM!"
250 BEEP .1,-RND*30: POKE 23692
,-1: GO TO 240
260 PRINT INK RND*7; PAPER 9;
"TODOS OS INVASORES DESTRUIDOS"
265 POKE 23692,-1
270 BEEP .01,RND*50: GO TO 260
280 REM Invasores
290 FOR J=0 TO 7
300 READ Q
310 POKE USR "F"+J,Q
320 NEXT J
330 DATA BIN 00111110,BIN 00101
010,BIN 00111110,BIN 00011100,BI
N 00001000,BIN 01110111,BIN 0100
0001,BIN 01000001
340 REM Base
350 FOR J=0 TO 7
360 READ Q
370 POKE USR "B"+J,Q
380 NEXT J
390 DATA BIN 01111110,BIN 01111
110,BIN 01111110,BIN 01111110,BI
N 01100110,BIN 11100011,BIN 1100
0011,BIN 10000001,BIN 10000001
400 LET SC=0
410 RETURN
```

Este programa gasta bastante memória. Não tendo sido feita qualquer tentativa para a economizar. O leitor pode torná-lo mais rápido fazendo algumas alterações. As duas cadeias que nos interessam são AS e BS. Esta última é simplesmente definida com 32 espaços, sendo usada para impedir a escrita de 32 espaços entre aspas em vários pontos do programa. AS é a cadeia que representa os invasores. Inicialmente possui 32 elementos, o número de caracteres existentes numa única linha de imagem. A linha 40 define o estado inicial dos caracteres e pode consistir em qualquer combina-

ção de espaços e caracteres gráficos - devendo ter o comprimento de 32 caracteres. X é a variável que controla a nossa posição e o seu valor é alterado na linha 110. A linha 120 realiza a impressão, actualizando a posição dos invasores e a quantidade destes.

A linha 130 compara os invasores com uma cadeia de 32 espaços (BS), e se verifica que AS não contém invasores (ou seja, é apenas constituída por espaços) provoca um salto para a onda seguinte de invasores ou para a mensagem de vitória na linha 260. A linha 150 tem um interesse especial, dado que procura na cadeia o caracter que se encontra acima do jogador na zona dos invasores, se encontra um destes converte-o num espaço. Isto só é feito porém no caso de se carregar na tecla 7.

O resto do programa serve essencialmente para temporizar os ciclos executados e escolher as diferentes ondas de invasores.

Se se está a guardar toda a imagem num quadro de cadeias alfanuméricas (ou uma parte da imagem, envolvendo linhas que se seguem umas às outras), é possível usar dois métodos diferentes recorrendo a diferentes tipos de quadros. Consideremos o caso da imagem completa, de 22 linhas por 32 caracteres. Será necessário um quadro de 22X32 elementos, o que pode ser conseguido de uma das formas seguintes:

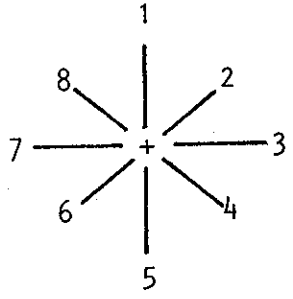
1) Usando um quadro bi-dimensional, construído pela instrução DIM AS(22,32). Pode-se então usar a instrução PRINT AT para aceder estes elementos, recordando que o primeiro elemento do quadro tem o número 1 e que as coordenadas de impressão partem do 0. Por exemplo, para PRINT AT y,x: CHRS T é necessário escrever LET AS(y + 1,x + 1)= CHRS T. O problema deste método consiste em que se torna necessário uma instrução PRINT bastante longa para colocar todo o quadro no visor, isto é, PRINT AT 0,0:AS(1), AS(2),AS(3)...AS(21),AS(22). No entanto, como a principal razão para usar um quadro alfanumérico para a impressão de um imagem é a facilidade em aceder à informação, isto só é necessário no início de um programa dado que daí em diante necessitamos apenas de imprimir as partes do quadro que estamos a tratar. Consideremos o exemplo de um jogo de damas. Necessitamos de imprimir todo o tabuleiro no início do jogo, e de examiná-lo completamente quando necessário: no entanto, quando se trata de imprimir os movimentos feitos, bastam-nos as partes do quadro

envolvidas no movimento — a posição de onde sai a peça, a posição para onde vai e talvez a parte do quadro onde é capturada uma peça inimiga.

2) Usando um quadro de uma só dimensão, com 704 elementos correspondentes à imagem de 22 x 32, construída com a instrução DIM AS(704). É possível imprimir todo o quadro de um só vez com a instrução PRINT AT 0,0:AS, sendo muito rápido fazê-lo. Os elementos são fáceis de aceder. Para deslocar um caracter no visor teremos de deslocá-lo no próprio quadro de tal modo que o seu movimento no visor pareça convincente. Para compreendermos isto é melhor observarmos antes do mais a disposição do quadro no visor. Consultemos a seguinte figura:

X / Y	0	1	2	29	30	31
0	A$(1)	A$(2)	A$(3)	A$(30)	A$(31)	A$(32)
1	A$(33)	A$(34)	A$(35)	A$(62)	A$(63)	A$(64)
2	A$(65)	A$(66)	A$(67)	A$(94)	A$(95)	A$(96)
3	A$(97)	A$(98)	A$(99)	A$(126)	A$(127)	A$(128)

O diagrama apresentado mostra um fragmento da parte superior do visor. Y e X são as coordenadas da função PRINT AT y,x. O leitor consegue compreender a relação entre as coordenadas y e x e os parâmetros de AS? Existem 32 elementos de AS em cada linha do visor. As coordenadas X partem de 0 enquanto que os parâmetros do quadro partem de 1. Nestas condições, y,x corresponde a AS(y * 32+x+1). Quando se move um caracter este pode deslocar-se para uma das oito posições à sua volta, indicadas no diagrama seguinte:



Suponhamos agora que o invasor se encontra em AS(A). Apresentamos aqui uma tabela mostrando a diferença que existe entre os parâmetros dos elementos que representam as posições possíveis (quanto deve ser acrescentado ao valor anterior do parâmetro para o alterar em cada caso):

Direcção do movimento	Quantidade a acrescentar a A
1	−32
2	−31
3	1
4	33
5	32
6	31
7	−1
8	−33

MOVIMENTO EM DIAGONAL

Temos de evitar passar para além dos limites do quadro alfanumérico, tal como é necessário ter cuidado ao usar PRINT, a fim de evitar que a execução seja interrompida devido a um erro nos parâmetros. Podemos utilizar as teclas de cursor para comandar o movimento vertical e o movimento horizontal, e usar as mesmas teclas, mas agora juntamente com a tecla de CAPS SHIFT para realizar movimentos em diagonal. Vamos estudar um curto programa que desloca um ponto através do visor sob o comando destas teclas, a fim de ilustrar o modo como se pode conseguir este resultado.

Carregando em SHIFT e 5, SHIFT e 6, SHIFT e 7 ou SHIFT e 8 desloca-se o ponto numa direcção que faz 45° com a indicada nas teclas de cursor. O programa não contém quaisquer instruções capazes de impedirem erros nos parâmetros tendentes a produzir um movimento fora dos limites do quadro.

```
  10 DIM a$(704)
  20 LET a=INT (RND*704)+1
  30 LET a$(a)=" "
  40 LET a=a-(32 AND INKEY$="7")
-(31 AND INKEY$=CHR$ 11)+(1 AND
INKEY$="8")+(33 AND INKEY$=CHR$
9)+(32 AND INKEY$="6")+(31 AND I
NKEY$=CHR$ 10)-(1 AND INKEY$="5"
)-(33 AND INKEY$=CHR$ 8)
  50 LET a$(a)="■"
  60 PRINT AT 0,0;a$
  70 GO TO 30
```

A razão de as combinações de CAPS SHIFT e cada uma das teclas de cursor, terem sido representadas por CHRS 8, CHRS 9, CHRS 10 e CHRS 11 consiste em não poderem ser indicadas directamente no teclado (actuam como comando de cursor), pelo que o modo mais simples de as obter consiste em usar CHRS.

Deve-se sublinhar que o uso de cadeias para criar gráficos móveis é limitado às aplicações em que a velocidade não constitui um factor importante, ao contrário do acesso à informação. Um bom exemplo são os jogos de tabuleiro, como as damas, onde as peças se movem ocasionalmente mas é necessário verificar rapidamente a informação existente.

30

OPERAÇÕES ARITMÉTICAS

Aconselho o leitor a fazer uma leitura rápida d' te capítulo antes de o estudar em pormenor. Talvez descubra que não contém quaisquer dados novos para si. Se for este o caso, passe ao capítulo seguinte.

Os símbolos das várias operações em BASIC são provavelmente já seus conhecidos. Consistem no de multiplicação (*), divisão (/), subtracção (-), soma (+) e potenciação (↑). O computador respeita uma prioridade bem definida na execução destas operações.

O termo prioridade designa aqui a ordem pela qual as expressões são avaliadas. O computador pode não avaliar tudo pela ordem em que se encontra escrito. Por exemplo, se o leitor colocar uma expressão entre parêntesis verifica que obtém muitas vezes um resultado diferente do obtido sem os parêntesis(aliás, a omissão de parêntesis pode levar o computador a recusar-se a fazer o cálculo devido à existência de um erro de sintaxe). Atribuimos portanto a cada operação uma *prioridade*, variável entre 1 e 16.

As operações de maior prioridade são as primeiras a ser executadas, e quando duas operações têm a mesma prioridade são executadas por ordem, começando pela esquerda. Com efeito, o computador observa a expressão, descobre a sua parte com maior prioridade e "diz" para si mesmo: "Espera um pouco, há aqui qualquer coisa com maior prioridade — volto aqui quando for a tua vez".

Operação	Prioridade
()	16
Subscritos /partições	12
todas as funções	11
↑	10
− (negação)	9
*	8
/	8
+	6
− (subtracção)	6
=,>,<,<=,>=,<>	5
NOT	4
AND	3
OR	2

Note que se considera sempre que um número é positivo a menos que esteja precedido do sinal menos. Do mesmo modo, e a menos que o número contenha uma vírgula decimal, o computador parte sempre do princípio de que se trata de um inteiro. Se bem que se possa usar pontos em representação de vírgulas decimais quando se trabalha com o computador, não são permitidas vírgulas (que em inglês representam milhares). O uso de *notação científica* para números muito grandes ou muito pequenos foi já explicado no capítulo sobre variáveis. Consulte-o novamente se deseja recordá-la.

A versão de BASIC utilizada no Spectrum é relativamente rápida, como se pode ver executando os programas que se seguem. O primeiro programa calcula progressões aritméticas. É necessário indicar o primeiro termo, o valor da diferença e o número de termos, e o computador dará rapidamente o resultado.

```
 10 REM Progressões aritméticas
 20 PRINT "Vou determinar o valor"
 30 PRINT "da progressão aritmética"
 40 PRINT "cujos dados me vai indicar."
 50 PRINT "Escreva o primeiro termo."
 60 INPUT primeiro
 70 PRINT primeiro
 80 PRINT "Qual é o valor da diferença?"
 90 INPUT diff
100 PRINT ,diff
```

```
110 PRINT' "Quantos termos?"
120 INPUT termos
130 LET termos= INT (termos +.5)
140 CLS
150 POKE 23692,-1
160 PRINT INK 6; PAPER 0;"Progressão
aritmética"
180 PRINT INK 2; '"Termo"; TAB 13; "Valor"

190 LET count=0
200 FOR L=0 TO termos-1
210 LET w=L+1
220 LET q= primeiro + (L * diff)
230 LET count=count+q
240 PRINT TAB 4;w;TAB 13;q
250 NEXT L
260 PRINT INK 2;TAB 4; '"A soma é"; count
```

Vou determinar o valor
da progressão aritmética
cujos dados me vai indicar.

Escreva o primeiro termo

234

Qual é o valor da diferença?

13.5

Quantos termos?

12

Progressão aritmética

Termo	Valor
1	234
2	247.5
3	261
4	274.5
5	288
6	301.5
7	315
8	328.5
9	342
10	355.5
11	369
12	382.5

A soma é 3699

Como o leitor pode verificar, o programa determina igualmente a soma dos termos.

NÚMEROS PRIMOS

Os números primos são fáceis de determinar.

```
 10 REM Números primos
 20 PRINT "Indique valor do maior"
 30 PRINT "número primo que deseja"
 50 INPUT a: PRINT a
 60 DIM z(a): LET k1=a
 70 FOR j=1 TO a: LET z(j)=j
 80 NEXT j: IF a<4 THEN GO TO 2
00
 90 LET z(4)=5: LET k1=4
100 LET iz=5
110 LET iz=iz+2
120 IF iz>a THEN GO TO 200
125 LET jo=3
130 LET ex=iz/z(jo)
140 IF ex=INT ex THEN GO TO 110
150 IF ex<z(jo) THEN GO TO 180
160 LET jo=jo+1
170 GO TO 130
180 LET k1=k1+1: LET z(k1)=iz
190 GO TO 110
200 POKE 23692,-1
210 PRINT "Números primos até ";a
230 PRINT TAB 4; INK 2; PAPER 6
"Número de ordem", "Número primo"
240 FOR c=1 TO k1
250 PRINT TAB 4;c,z(c)
260 NEXT c
```

Indique valor do maior

número primo que deseja

Números primos até 20

Número de ordem	Número Primo
1	1
2	2
3	3
4	5
5	7
6	9
7	11
8	13
9	17
10	19

As potencialidades do computador em termos de cálculos matemáticos podem também, como é óbvio, ser aplicadas à obtenção de outros tipos de informação.

ESTATÍSTICAS

O programa que se segue, constituído por uma série de rotinas de cálculo estatístico, pode ser facilmente dividido em quatro programas mais curtos, iniciando-se nas linhas 1000, 2000, 3000 e 4000. É necessário ao leitor indicar o valor de COUNT e TOTAL para cada um deles.

Os quatro programas são:

Média Aritmética — Trata-se simplesmente do cálculo da média de um conjunto de números.

Média Geométrica — A média geométrica é a raíz de ordem *n* do produto dos números, onde *n* designa a quantidade de números indicados.

Média Harmónica — Esta média é obtida a partir dos recíprocos dos números indicados.

Factorial — O factorial é a progressão A*(A-1)*(A-2)*(A-3)... até (3)*(2)*(1), onde A é o inteiro indicado na linha 4030. Como esta operação só pode ser executada sobre inteiros, a linha 4040 transforma em inteiro qualquer número indicado.

A rotina que se inicia na linha 9000 apresenta um menú.

Note o uso de GO TO A * 1000 na linha 9600. É um modo resumido de dizer:

```
IF A = 1 THEN GOTO 1000
IF A = 2 THEN GOTO 2000
IF A = 3 THEN GOTO 3000
IF A = 4 THEN GOTO 4000
```

O leitor poderá usar muitas vezes esta técnica em programas sujeitos a menú.

```
  20 GO TO 9000
  30 REM Nas linhas 1040, 2040, 3040, usar a aspa
da tecla 7

 900 REM ************************
1000 REM Média aritmética
1010 PRINT "MÉDIA ARITMÉTICA"
1020 PRINT "Indique os números cuja média"
1030 PRINT TAB 5;"pretende obter,"
1040 PRINT "Escreva 'E' para terminar"
1060 INPUT Q$: IF Q$="" THEN GO
TO 1060
1070 IF Q$="E" THEN GO TO 1120
1080 POKE 23692,-1: PRINT Q$
1090 LET TOTAL=TOTAL+VAL Q$
1100 LET COUNT=COUNT+1
1110 GO TO 1060
1120 PRINT '"A média aritmética é";
TOTAL /COUNT
1130 GO TO 9000
1500 REM ************************
2000 REM Média geometrica
2010 PRINT "MÉDIA GEOMÉTRICA"
2020 PRINT "Indique os números cuja média"
2030 PRINT "geometrica pretende obter"
2040 PRINT "Escreva 'E' para terminar"
2050 LET TOTAL=1
```

```
2060 INPUT Q$: IF Q$="" THEN GO
TO 2060
2070 IF Q$="E" THEN GO TO 2120
2080 LET COUNT=COUNT+1
2090 LET TOTAL=TOTAL+VAL Q$
2100 POKE 23692,-1: PRINT Q$
2110 GO TO 2060
2120 PRINT '"A média geométrica é"; TOTAL ↑
(1 /COUNT)
2130 GO TO 9000
2900 REM ************************
3000 REM Média Harmónica
3010 PRINT "MÉDIA HARMONICA"
3020 PRINT "Indique os números cuja média"
3030 PRINT "harmónica pretende obter."
3040 PRINT "Escreva 'E' para terminar"
3050 INPUT Q$: IF Q$="" THEN GO
TO 3050
3060 IF Q$="E" THEN GO TO 3120
3080 POKE 23692,-1: PRINT Q$
3090 LET TOTAL=TOTAL+(1/VAL Q$)
3100 LET COUNT=COUNT+1
3110 GO TO 3050
3120 PRINT '"A média Harmónica é":
1 /(TOTAL /COUNT)
3130 GO TO 9000
3900 REM ************************
4000 REM FACTORIAL
4010 PRINT "FACTORIAL"
4020 PRINT "Indique um inteiro", "inferior
a 34."
4030 INPUT NUM: IF NUM>=34 THEN
GO TO 4030
4040 LET NUM=INT (NUM)
4050 LET A=1
4060 FOR B=1 TO NUM
4070 LET A=A*B
4080 NEXT B
4100 PRINT '"O factorial de ";NUM;" é";A

8900 REM ************************
9000 PRINT "Escolha o programa que pretende"
9010 PRINT "1 — Média Aritmética"
9020 PRINT "2 — Média Geométrica"
9030 PRINT "3 — Média Harmónica"
9040 PRINT "4 — Factorial"
9050 PRINT "5 — Para terminar": PRINT
9060 LET A$=INKEY$
```



```
9070 IF A$<"1" OR A$>"5" THEN GO
TO 9060
9080 LET A=VAL A$
9090 IF A=5 THEN STOP
9100 LET TOTAL=0
9110 LET COUNT=0
9120 PRINT "**********************
**********"
9130 GO TO A*1000
```

É possível fazer cálculos muito mais complexos. O programa que se segue, por exemplo, calcula a média, o desvio, o erro da média e a variância de 20 distribuições de frequências com um máximo de 240 elementos cada. Pode igualmente calcular qualquer diferença significativa entre duas distribuições recorrendo ao "Student's T= teste".

```
 10 REM T-TEST
 12 REM Convertido de um programa
de Allan Norlin
 13 REM por Anders Lund.
 20 DIM X(240)
 22 DIM N(20)
 24 DIM M(20)
 26 DIM S(20)
 28 DIM E(20)
 30 DIM V(20)
 32 DIM G(20)
110 PRINT "Este programa calcula a média."
115 PRINT "O desvio, o erro standard e"
122 PRINT "A variância de um máximo de"
123 PRINT "20 distribuições de frequência"
130 PRINT "e 'Student's T-Test' "
135 PRINT "verificando se existe uma"
140 PRINT "diferença significativa"
141 PRINT "quaisquer duas distribuições"
142 PRINT "de frequências"
144 PRINT
146 PRINT "Deseja os resultados apenas"
147 PRINT "no visor (D) ou também"
148 PRINT "na impressora (P)?"
149 INPUT W$
150 CLS
```

```
160 LET G=0
161 LET Q=0
162 LET S=0
170 PRINT "Quantas distribuições?"
180 INPUT M
200 LET G=G+1
205 PRINT
220 PRINT "Distribuição número";G
245 PRINT "Quantos elementos?"
250 INPUT N
252 PRINT
255 LET N(G)=N
260 LET S1=0
262 LET S2=0
265 PRINT "Escreve dados um a um"
270 FOR I=1 TO N
271 CLS
280 PRINT "Número";I
290 INPUT X
295 CLS
300 LET S1=S1+X
310 LET S2=S2+X*X
320 LET X(I)=X
330 NEXT I
340 LET M(G)=S1/N
345 LET V(G)=((S2-S1*S1/N)/(N-1
))
350 LET S(G)=SQR V(G)
360 LET E(G)=S(G)/SQR (N)
380 PRINT"Deseja os dados impressos (s /n)?"
390 INPUT Q$
395 CLS
400 IF Q$="N" THEN GOTO 550
405 PRINT "Número da distribuição: ";G
410 LET Q=Q+1
415 LET R=1
417 LET K=0
420 FOR I=1 TO N
425 PRINT AT R,K;I;" ";X(I)
430 IF I/20=INT (I/20) THEN LET
K=K+10
431 IF K>21 AND W$="P" THEN COP
Y
432 IF K>21 THEN GOSUB 3000
435 IF K>21 THEN LET K=K-30
460 IF I/20=INT (I/20) THEN LET
R=R-20
465 LET R=R+1
480 NEXT I
490 IF W$="P" THEN COPY
```



```
500 GOSUB 3000
550 IF G<M THEN GOTO 200
740 FOR I=1 TO M
750 PRINT "Número da distribuição:";
752 PRINT
755 PRINT "Número de dados= ";N(I)
758 PRINT
760 PRINT "Média= ";M(I)
762 PRINT
765 PRINT "Desvio standard= ";S(I)
767 PRINT
770 PRINT "Erro standard= ";E(I)
772 PRINT
775 PRINT "Variância= ";V(I)
776 IF W$="P" THEN COPY
777 GOSUB 3000
780 NEXT I
790 PRINT
800 PRINT "Deseja realizar 'T - Test' * (s
/n)?"
810 INPUT Q$
815 CLS
820 IF Q$="N" THEN GOTO 1200
830 PRINT "T - test entre duas distribuições"
835 PRINT
840 PRINT "Indique número da distribuição"
845 INPUT S1
847 PRINT "Número: ";S1
849 PRINT
850 PRINT "Indique número da distri
buição"
852 INPUT S2
854 PRINT "Número: ";S2
890 LET A=N(S1)+N(S2)
900 LET B=N(S1)*N(S2)
910 LET D=A-2
920 LET T=ABS (M(S1)-M(S2))/SQR
(((N(S1)-1)*S(S1)↑2+(N(S2)-1)*
S(S2)↑2)*A/(B*D))
930 GOSUB 2000
935 CLS
940 PRINT "T - Test para distribuições número;";S1;
" e número;";S2
S2
945 PRINT
950 PRINT "T=";T
955 PRINT
960 PRINT "DF=";D
965 PRINT
```

```
 970 PRINT "P=";P
 975 PRINT
 980 IF P>0.001 THEN GOTO 990
 982 PRINT "P<0.001   ***"
 984 GOTO 1050
 990 IF P>0.01 THEN GOTO 1000
 992 PRINT "P<0.01    **"
 994 GOTO 1050
1000 IF P>0.05 THEN GOTO 1010
1002 PRINT "P<0.05    *"
1004 GOTO 1050
1010 PRINT "P>0.05  NS."
1050 IF U$="P" THEN COPY
1055 PRINT
1056 PRINT
1057 PRINT
1058 PRINT
1060 PRINT "Deseja testar mais"
1065 PRINT "distribuiçoes (s / n)?"
1070 INPUT Q$
1075 CLS
1080 IF Q$="N" THEN GOTO 1200
1090 GOTO 540
1200 PRINT "Deseja analisar mais". "dis
tribuições (s /n)?"
1210 INPUT Q$
1215 CLS
1220 IF Q$="Y" THEN GOTO 120
1240 PRINT "****  FIM     ****"
1245 GOTO 9999
2000 REM Calcular valor T, valor P
(probabilidade)  e valor DF (graus de liberdade)
2020 LET P=1
2025 LET M=1
2030 IF T=0 THEN GOTO 2210
2040 LET U=T*T
2050 IF U<1 THEN GOTO 2100
2060 LET I=M
2070 LET J=D
2080 LET T1=U
2090 GOTO 2130
2100 LET I=D
2110 LET J=M
2120 LET T1=1/U
2130 LET A1=2/9/I
2140 LET B1=2/9/J
2150 LET Z=ABS ((1-B1)*T1 ↑ (1/3)
-1+A1)/SQR (B1*T1 ↑ (2/3)+A1)
2160 IF J>3 THEN GOTO 2180
```



```
2170 LET Z=Z*(1+.08*Z ↑ 4/J ↑ 3)
2180 LET P=.5/(1+Z*(.196854+Z*(.
115194+Z*(.000344+Z*.019527))))
↑4
2190 IF U>1 THEN GOTO 2210
2200 LET P=1-P
2210 RETURN
3000 PRINT "Carregue em C para continuar"
3004 IF INKEY$<>"C" THEN GOTO 30
04
3007 CLS
3010 RETURN
9999 STOP
```

ESPÉCIES VIVAS

O último programa deste capítulo utiliza o computador para simular os ciclos de vida de duas espécies. uma das quais se alimenta da outra. e para construir um gráfico com as respectivas populações relativas. A relação entre ambas as espécies é controlada por uma equação diferencial. Indica-se as populações iniciais, sob a forma de números entre um e cinco. Aceitam-se fracções. é fascinante indicar uma população muito baixa para uma das espécies e outra bastante elevada para a segunda. observando o modo como ambas evoluem em seguida. Quando o programa calcula um certo número de gerações. volta ao início e pede novos valores iniciais para as duas populações. O desenvolvimento desta relação é então apresentado num gráfico. que se sobrepõe ao gráfico anterior. de tal modo é possível construir vários gráficos capazes de compararem os efeitos da diferente quantidade de cada um dos animais na evolução de ambas as espécies.

```
  5 BORDER 2: PAPER 6: CLS
 10 REM espécies
 20 PRINT "Quantos da espécie um (1 /5)?"
 30 INPUT X: IF X<1 OR X>5 THEN
GO TO 30
 40 PRINT "E da espécie 2 (1 /5)?"
 50 INPUT Y: IF Y<1 OR Y>5 THEN
GO TO 50
```

```
  55 LET X=X+RND: CLS
  60 FOR Z=1 TO 20
  70 FOR T=1 TO 7 STEP .5
  80 PRINT AT 1,1; "Espécie um:
"; FLASH 1; BRIGHT 1;"   "; INT (X
*10000);"   "
  90 PRINT AT 2,1; INK 1; "Espécie
   dois:  "; FLASH 1; BRIGHT 1; INK
2;"   "; INT (Y*10000);"   "
 100 LET X=X+(4*X-2*X*Y)*0.01
 110 LET Y=Y+(X*Y-3*Y)*0.01
 120 PLOT 30*X,30*Y
 130 NEXT T
 135 BEEP .05,Z+T
 140 NEXT Z
 150 LET X=RND*6
 160 LET Y=RND*6
 180 GO TO 60
```

31
FUNÇÕES

O dialecto do BASIC usado no Spectrum, tal como acontece noutras versões, contém um certo número de funções pré-programadas que podem ser usadas num programa ou em ordens directas. A discussão que se segue inclui uma "função definida pelo utilizador" que desenha a imagem de um vampiro!

Funções gerais:

ABS - Esta função dá-nos o valor de X ignorando o seu sinal: se X for -10, ABS(X) será 10. Do mesmo modo, se X for 10, ABS(X) continuará a ser 10.

INT - A função INT dá-nos um número inteiro, ou a parte inteira de um número, não superior a ele. Se X for 2,42, INT (X) será 2.

INT arredonda sempre para o inteiro inferior, isto é, 2,2 arredonda para 2 e 2,9 arredonda igualmente para 2. Muitas vezes é necessário arredondar os números para o inteiro superior, de tal

modo que 2,6 seja arredondado para 3, etc (certas instruções fazem isto automaticamente, por exemplo PRINT AT, POKE, etc). É muito fácil fazer um arredondamento deste tipo. Suponhamos que o número a arredondar é A. Se primeiro somarmos 0,5 a A, e em seguida lhe aplicarmos a função INT, obteremos o resultado desejado. Como exemplo consideremos que A é igual a 2,6: PRINT INT (2.6 + 0.5) dará 3, enquanto que PRINT INT (2.3 + 0.5) dará ainda 2.PRINT INT (2.5 + 0.5) é ainda arredondado para 3.

É muitas vezes necessário, quando se fazem cálculos contabilísticos, dispor de dados com duas casas decimais a fim de ter em conta os centavos e os tostões. Podemos fazê-lo recorrendo à rotina seguinte. A rotina em causa inclui por outro lado um 0 antes da vírgula decimal no caso de a resposta ser inferior a 1.00. A quantia é atribuída a A na linha 10, em dólares neste caso, mas sem o sinal S - este sinal é acrescentado na rotina da linha 50.

```
  10 INPUT A
  20 LET A$=STR$ (INT (A*100+0.5
)/100)
  25 IF A$(1)="." THEN LET A$="0.
"+A$
  30 LET B=LEN A$-LEN STR$ INT V
AL A$
  40 LET A$=A$+(".00" AND B=0)+(
"0" AND B=2)
  50 PRINT "$";A$
  60 GO TO 10
```

```
11.245      $11.25
33          $33.00
33.1        $33.10
333.888     $333.89
0.9999      $1.00
22.009      $22.01
```

RND - Esta função é usada para produzir um número aleatório ("random"). Produz um número entre zero e um.

SGN - Esta função permite obter o sinal da variável entre parêntesis, o SGN do *argumento*, nome que é dado a

essa variável. Se X é igual a 20, isto é se X é um número positivo. SGN(X)= 1. SGN(-20)= -1. SGN(0)= 0.

TAB - Como já se referiu anteriormente, trata-se da função de tabulação, que desloca o ponto de impressão o número de espaços indicado pelo argumento. Nestas condições PRINT TAB (6);"S" imprimirá o sinal S na sétima posição a partir da margem esquerda, enquanto que PRINT TAB (13);"S" imprimi-lo-á na décima quarta posição.

A linha também pode ser indicada, acrescentando um segundo argumento dentro do parêntesis, depois de uma vírgula. Nestas condições PRINT TAB (4,9);"S" imprimirá um sinal de escudos cinco espaços (linhas) abaixo, e a dez de distância da margem. No caso de o número indicado ser superior a 31, TAB diminui-lhe o equivalente a uma linha: toma um valor que varia apenas entre 0 e 31, e desloca a posição de impressão sempre para um ponto na mesma linha a menos que isto envolva mover-se para trás, caso em que passa à linha seguinte. Isto significa que o argumento de TAB é é dividido por 32 (o número de colunas existentes no visor), aceitando-se apenas o resto desta divisão. Este pormenor pode ser aproveitado quando a posição de impressão é determinada por um cálculo, não sendo então necessário que o resultado desse cálculo esteja contido na gama 0 a 31.

EXP - Esta função produz o valor de *e* elevado à pontência indicada em argumento, pelo que PRINT EXP 5 dará 148,41316.

LN - LN X produz o logaritmo natural do número, pelo que PRINT LN 5 dará 1,6094379.

SQR - Esta função produz a raíz quadrada de um número: se X for igual a 5, PRINT SQR X dará 2,236068.

FUNÇÕES TRIGONOMÉTRICAS

SIN - Dá-nos o seno de um ângulo em radianos. SIN 5 produz -0,95892428.

COS - Produz o co-seno de um ângulo em radianos. PRINT COS X, com X igual a 5, dá 0,28366219.

TAN - Produz a tangente do ângulo X em radianos.

Note que o computador mede sempre os ângulos em radianos. PI radianos é igual a 180 graus.

A função RANDOMIZE funciona do seguinte modo:

O número que é colocado depois da palavra RANDOMIZE é armazenado nas variáveis de sistema depois de ter sido arredondado para o inteiro mais próximo. Se se escreve apenas RANDOMIZE ou RANDOMIZE 0 é usado o valor do contador de imagens. Este valor *não* é afectado por CLEAR ou RUN, mas é reposto em zero pela função NEW, tal como ao ligar a máquina. O número altera-se de cada vez que se utiliza RND.

32
CONVERSÃO DE OUTROS PROGRAMAS BASIC

É possível encontrar muitos programas BASIC em livros e revistas; mas como todas as versões desta linguagem diferem entre si é improvável que um programa escrito para outro computador possa ser executado no Spectrum sem modificações. A extensão e natureza destas alterações depende bastante da estrutura do programa em causa e do modo como trata os dados, mas é possível indicar algumas regras gerais sobre o que interessa ter em conta ao tentar converter um programa para execução no Spectrum.

INTEIROS

Em geral, deve-se acrescentar a função ao INT antes de uma divisão num programa concebido para um computador com aritmética de inteiros. Pode ser necessário incluir a divisão entre parêntesis de modo a que INT actue apenas sobre o resultado da divisão.

DIM

Certas versões BASIC permitem-nos escrever várias instruções DIM numa única linha sem repetição da palavra-chave DIM, por exemplo DIM A$(9), B$(8), C$(7). Estas declarações devem ser substituídas no Spectrum, por instruções DIM individuais. Se o programa exige quadros com nomes de comprimento superior a uma letra, é necessário substituí-los por nomes de uma só letra

como AS e B. Se não dispõe de um número de letras suficiente talvez seja possível acrescentar dimensões a um quadro já existente, usando essas dimensões extra para substituir o novo quadro. Tenha cuidado com parâmetros 0.

GET, GETS

Trata-se de uma função que lê caracteres ou valores a partir das teclas premidas. Assume várias formas em diferentes computadores, mas em geral espera até ser premida uma tecla antes de continuar a execução, atribuindo a uma variável o caracter correspondente à tecla premida ou o código desse caracter. Por exemplo, GET AS ou LET AS= GETAS. Tais instruções podem ser substituídas no Spectrum por outras do seguinte tipo:

```
1000 LET a$=INKEY$
1010 IF a$="" THEN GO TO 1000
```

Estas duas instruções dão como resultado o caracter correspondente à tecla premida no teclado. Se a função deve produzir o código do caracter (que será provavelmente escrito sob a forma ASC) convirá usar a rotina seguinte:

```
1000 LET a$=INKEY$
1010 IF a$="" THEN GO TO 1000
1020 LET a=CODE a$
1030 PRINT a
```

A versão que produz um valor numérico e não um código é um pouco diferente. É necessário garantir que o caracter lido do teclado se encontra na gama 0 a 9, a fim de podermos aplicar VAL para converter o caracter em número. Vejamos um modo de o fazer:

```
1000 LET a$=INKEY$
1010 IF a$<"0" OR a$>"9" THEN GO
 TO 1000
1020 LET a=VAL a$
1030 PRINT a
```


Talvez o leitor encontre ainda uma versão de INKEYS que permite especificar um limite de tempo para a resposta do utilizador, por exemplo

100 LET AS= INKEYS(X)

onde X especifica o limite de tempo. Isto pode ser convertido de dois modos, a saber:

```
  10 LET x=50
 100 PAUSE x
 110 LET a$=INKEY$
```

ou ainda, como se demostra neste simples jogo,

```
   5 REM Passar primeiro para maiúsculas
  10 LET B$=CHR$ (INT (RND*26)+C
ODE "A")
  20 PRINT AT 10,0; "Carregue rapidamente numa
tecla";B$
 100 FOR a=0 TO 100
 110 LET a$=INKEY$
 120 IF a$<>"" THEN GO TO 140
 125 NEXT a
 130 PRINT "Acabou o tempo"; STOP
 140 IF A$=B$ THEN PRINT "Acertou":
STOP
 150 PRINT "Falhou"
```

VAL

Se o argumento de VAL não é válido, obtém-se uma mensagem de erro. Outras versões de BASIC dão nestas condições o resultado 0.

SET, RESET.

Estas instruções são usadas para tornar branco ou negro um determinado ponto. Devem ser substituídas por um PLOT /OVER / /PRINT AT.

ELSE

Esta instrução é uma extensão da instrução condicional IF...

THEN e permite obter mais de um resultado em função do caracter verdadeiro ou falso da condição. Pode ser substituída por suas expressões condicionais no seu computador. Por exemplo:

```
20 IF X=1 THEN LET Y=7 ELSE GOTO 80
```

pode ser substituído por:

```
20 IF X=1 THEN LET Y=7
21 IF X<> 1 THEN GOTO 80
```

Se a acção consiste em atribuir um de vários valores alternativos a uma variável, é possível substituí-la por uma única linha de instruções. Por exemplo.

```
50 IF X=1 THEN LET Y=7 ELSE LET Y=8
```

pode ser substituída por:

```
50 LET Y=(7 AND X=1) + (8 AND X<>1)
```

Certas expressões como a anterior podem ser até substituídas por outras ainda mais simples:

```
50 LET Y=7 + (1 AND X <>1)
```

Não é possível fornecer um guia geral dado que o método usado variará de um exemplo para outro - mas os casos anteriores darão uma ideia das possíveis soluções.

O leitor pode encontrar também uma das instrução onde a acção de ELSE seja ela própria condicional:

```
10 IF X=1 THEN LET Y=1 ELSE IF X=5 THEN GOTO
100
```

Isto deverá ser re-escrito do seguinte modo:


```
10 IF X =1 THEN LET Y=1
11 IF X<> 1 THEN IF X=5 THEN GOTO 100
```

ou ainda

```
10 IF X=1 THEN LET Y=1'
11 IF X<> 1 AND X=5 THEN GOTO 100
```

É também possível encontrar muitos tipos de ELSEs condicionais, e as versões que lhe correspondem no Spectrum dependerão ainda aqui da variante em causa.

REPEAT... UNTIL

Trata-se de um ciclo que realiza continuamente uma operação, terminando apenas quando é satisfeita uma determinada condição. A sua utilidade é tão vasta que se torna particularmente difícil especificar um método universal de conversão para o BASIC do Spectrum, sendo talvez melhor substituída pela declaração condicional IF... THEN GO TO. Vejamos um exemplo:

```
10 PRINT "ESCREVA SIM OU NÃO"
20 REPEAT
30 INPUT A$
40 UNTIL A$="SIM" OR A$="NÃO"
```

pode ser substituído por:

```
10 INPUT "ESCREVA SIM OU NÃO";a$
20 IF a$<> "Sim" AND a$< > "SIM" AND a$
< > "não" AND a$ < > "NÃO" THEN GO TO 10
```

As estruturas REPEAT... UNTIL são geralmente muito mais complexas do que o exemplo dado, e pode tornar-se necessário descobrir um meio de conversão diferente de IF... THEN GO TO. Por exemplo, quando o valor de uma variável constitua o factor determinante, pode-se usar por vezes um ciclo FOR/NEXT. No entanto, a possibilidade de usar uma instrução condicional IF...THEN

GO TO deve ser sempre considerada, constituindo muitas vezes o único método de conversão aceitável.

VARIÁVEIS NÃO DEFINIDAS

Se tentar usar uma variável antes de ter sido definida ou atribuída num programa, alguns computadores atribuir-lhe-ão o valor. No caso do Spectrum apenas se consegue uma mensagem de erro. É portanto necessário definir ou atribuir valores a todas as variáveis usadas no progrma.

MATRIZES

Certas versões BASIC possuem funções de matriciação que realizam operações sobre quadros numéricos. O Spectrum não dispõe destas funções, pelo que é necessário executá-las individualmente sobre os elementos do quadro, talvez recorrendo a um ciclo.

```
10 DIM X(Y)
20 DIM P(Y)
30 MAT X=P
```

Neste caso particular é possível substituir a operação em causa por:

```
10 LET n=0
20 DIM x(y)
30 DIM p(y)
40 LET n=n+1
50 IF n<y THEN GO TO 40
```

PROC, ENDPROC

Trata-se de um método de utilização se subrotinas concebido de tal modo que entre outras coisas torna os programas e as listagens muito mais fáceis de compreender e ler (alguns chamam a istc programação estruturada). Permite o uso de subrotinas especificamente para realizar certas tarefas e aproxima-se de facto de uma subrotina de muitos pontos de vista, apresentando no entanto a importante diferença de ser chamada por um nome e não por um número de linha. Vejamos o exemplo seguinte, que imprime uma pontuação no visor:


```
100 PROCpontuação
.....
1000 DEF PROC pontuação
1010 PRINT "PONTUAÇÃO= ";S
1020 ENDPROC
```

ENDROC é de certo modo semelhante a RETURN, dado que provoca o final da rotina e obriga a execução a voltar à linha que se segue àquela que a chamou, neste caso à linha que se segue a 100. O nome da subrotina não é usado no Spectrum, se bem que seja possível adaptar o BASIC usado por este, como verificaremos no exemplo seguinte. O modo mais simples de converter para o ZX BASIC consiste em escrever na linha 100 GO SUB linha 1000, talvez incluindo uma declaração REM em algum ponto da rotina a fim de a identificar, e terminando a subrotina do modo habitual, isto é, com uma instrução RETURN.

```
 100 GOSUB 1000
.....
1000 REM  subrotina pontuação
1010 PRINT "PONTUAÇÃO= ";S
1020 RETURN
```

Se quiser reter a possibilidade de nomear a subrotina, pode usar uma variável para designar a subrotina e utilizá-la como destino da ordem GO SUB. Pode incluir uma declaração REM na subrotina para a identificar e relacionar com o variável usada. É útil utilizar caracteres em aberto ("inverse") nestas declarações REM a fim de se notarem melhor ao observar a listagem. Consegue-se assim dar uma razoável aparência de estruturação aos programas.

```
  50 LET  pontuação=1000
.....
 100 GOSUB  pontuação
.....
1000 REM  subrotina pontuação
1010 PRINT "pontuação= ;
1020 RETURN
```

Se bem que as PROCs possam ser complexas, uma vulgar subrotina constitui o melhor método de conversão para o BASIC do Spectrum.

INSTR(AS,BS)

Trata-se de uma função que verifica se existe uma cópia de BS em AS, e se existir diz-nos onde se inicia a cópia. Por exemplo, se BS for "PUT" e AS for "COMPUTADOR", então o valor de INSTR(AS,BS) será 4 porque a parte de AS que contém as letras "PUT" se inicia no quarto elemento de AS. Se a função não encontra uma cópia de BS em AS, INSTR(AS,BS) tem o valor 0. É necessário escrever uma rotina especial para obter esta função no Spectrum.

Vejamos um método de converter esta função para execução no Spectrum:

```
AMANTE A
INSTR (AS,BS)= 1
AMANTE N
INSTR(AS,BS)= 4
AMANTE RISO
INSTR(AS,BS)= 0
AMANTE FELIZ
INSTR(AS,BS)= 0
AMANTE AMA
INSTR(AS,BS)= 1
```

```
  10 REM -- subrotina para 'INSTR'
  20 INPUT A$
  30 INPUT B$
  40 PRINT A$;" ";B$
  50 GO SUB 1000
  60 PRINT "INSTR(A$,B$) =";Y
  70 GO TO 20
1000 REM SUBROUTINE FOR 'INSTR'
1010 LET Y=0
1020 IF LEN A$=0 OR LEN B$=0 OR
LEN B$>LEN A$ THEN RETURN
1030 FOR Y=1 TO LEN A$-LEN B$+1
```


Note que se quiser detectar palavras inteiras em vez de cadeias simples terá de examinar AS procurando espaços ou sinais de pontuação que indiquem o início e final de palavras. A rotina acima limita-se a descobrir cadeias semelhantes, pelo que no caso de o leitor querer encontrar a palavra AQUI numa frase contendo a palavra AQUILO, a máquina indicaria as quatro primeiras letras desta última. No entanto, este problema também existe normalmente nas máquinas que usam INSTR...

DIV

DIV produz a parte inteira do resultado de uma divisão, por exemplo 17 DIV 5 dá 3. No nosso computador pode-se aplicar INT ao resultado da divisão com o mesmo efeito; portanto, A DIV B deve ser substituído no Spectrum por INT (A /B).

MOD

MOD dá o *resto* de uma divisão, por exemplo 17 MOD 5 é 2. A MOD B é A - (INT(A /B * B)) no Spectrum. Note que TAB está já associado a uma acção MOD (módulo 32) neste computador.

TAB

Alguns computadores podem ter dois argumentos na função TAB, indicando a linha e a coluna onde a máquina deve imprimir. Isto corresponde no caso do Spectrum ao uso de AT. Por exemplo, TAB (X,Y) corresponde a AT Y,X no Spectrum. As coordenadas X e Y podem apresentar-se pela ordem inversa em alguns computadores.

GRAUS E RADIANOS

O Spectrum processa funções trigonométricas em radianos usando a expressão:

```
LET RADIANOS= (PI * GRAUS) /180
```

Os radianos podem ser convertidos em graus do seguinte modo:

```
LET GRAUS= (180 * RADIANOS) /PI
```

LOGARITMOS NA BASE 10

Como o Spectrum funciona com logaritmos naturais, na base, se necessitar de logaritmos de base 10 por alguma razão pode obtê-los usando a expressão:

```
LET LOGBASE10 (X) = (LN(X)/(LN(10))
```

Pode utilizar isto para determinar logaritmos em qualquer base: suponhamos que deseja o logaritmo de X na base B:

```
LET LOGBASEB (X) = (LN(X))/(LN(B))
```

O símbolo de percentagem é geralmente usado para especificar uma variável inteira, por exemplo A%. São usados principalmente por pouparem memória ou porque podem ser processados mais rapidamente do que as variáveis convencionais. Em geral não há qualquer problema ao usar uma variável vulgar, se bem que se deva ter consciência destas variáveis inteiras porque cortam automaticamente o quociente aceitando apenas a sua parte inteira. Neste caso usa-se LET A= INT (A /2), por exemplo, para tornar inteiro o resultado de uma divisão.

?

Na maior parte dos computadores o símbolo? é usado como abreviatura da ordem PRINT.

33

PEEK E POKE

Estas duas ordens são extremamente potentes, permitindo ao utilizador fazer coisas que não conseguiria de outro modo. Comecemos por definir os dois termos PEEK e POKE.

1) PEEK m indica-nos o número armazenado no endereço m da memória.
2) POKE m,n coloca o número n no endereço m da memória.

Quando é aceite, apaga o valor que aí se encontrava anteriormente.

O termo endereço necessita de algumas explicações. Qualquer computador pensa e memoriza tudo sob a forma de números, e não de palavras ao contrário das pessoas. Certos arranjos de algarismos levam o computador a realizar tarefas específicas. Chama-se a isto um programa. Ora o computador necessita de um modo de armazenar todos estes números a fim de os "recordar" quando necessitar deles.

Certos arranjos de algarismos podem levar o computador a imprimir qualquer coisa no visor, a somar dois números ou até a "estoirar" ("crash") se lhe for pedido que faça qualquer coisa impossível.

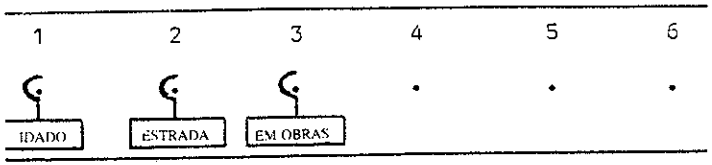
Acontece no entanto que o computador não pode guardar os número num sítio qualquer - isto provocaria o caos, e a máquina não saberia onde deveria procurar os números quando necessitasse deles. Existe portanto um método que permite à máquina manter as suas informações convenientemente organizadas.

Imagine que pretende apresentar uma mensagem a um público e que para esse efeito dispõe de alguns cartões com palavras impressas. Os cartões em causa dispõem de um pequeno gancho que permite fixá-los em qualquer posição a fim de obter a mensagem necessária. Se por exemplo quiser apresentar a mensagem seguinte:

CUIDADO! ESTRADA EM OBRAS

necessitará dos seguintes cartões.

Estas cartões devem ser colocados sobre uma placa. Se pendurarmos o primeiro à esquerda da placa, no prego número 1, obteremos o seguinte resultado:



Os números dos pregos existentes na placa indicar-nos-ão onde é pendurado cada um dos cartões. Isto serve-nos como termo de comparação com aquilo que se passa num computador. Existem na memória deste 65536 "pregos" (posições de memória) onde podemos "pendurar" números, divididos em grupos para diferentes fins. Estas "posições de memória" são designadas por endereços.

Um dos modos de uso da ordem POKE a que já recorremos neste livro consiste em produzir gráficos definidos pelo utilizador. Estes são "POKEd" em endereços a partir do 32600 no caso de um Spectrum de 16 K.

Se observarmos o primeiro prego da figura anterior (PEEK 1), verificamos que guarda a palavra CUIDADO!. Se observarmos o segundo, (PEEK 2), descobriremos a palavra ESTRADA, etc. O leitor compreendeu certamente a analogia. Não se esqueça de que o computador guarda números e não palavras, mas que o método de armazenamento das informações é em ambos os casos o mesmo. Podemos igualmente alterar o conteúdo das posições de memória usando POKE para colocar nelas um novo número. Podemos fazer o equivalente a POKE prego 2,EDIFÍCIO, substituindo a palavra ESTRADA pela palavra EDIFÍCIO na mensagem e alterando todo o seu significado. O grande segredo destas duas ordens, PEEK e POKE, não é de facto aquilo que fazem mas sim o modo como as usamos. À primeira vista não parece muito divertido ver quais os números que se encontram em cada endereço ou colocar nestes novos números. Interessa-nos porém utilizar estas funções num programa para obter resultados. Através da experiência e do estudo dos programas de outras pessoas, à medida que o nosso conhecimento sobre o computador aumenta, descobrimos novos usos para PEEK e POKE. Antes de vermos alguns exemplos, vejamos no entanto o modo como se devem construir as instruções PEEK e POKE.

PEEK m, m é o endereço que queremos observar; pode ser um valor qualquer entre 0 e 65535 (na máquina de 48 K), podendo constituir o resultado de um cálculo.

POKE m, n, é o endereço onde desejamos colocar o novo valor, n. Este último, escrito depois da vírgula, pode variar entre 0 e 255 ou ser o resultado de um cálculo (desde que esteja compreendido entre estes valores). Pode-se utilizar valores negativos, entre 0 e -255, mas isto raramente é feito e não é de qualquer modo particularmente útil.

Observemos alguns usos de PEEK e POKE.

1) Declarações REM.

Muitos programas utilizam informações guardadas em instruções REM colocadas nas primeiras linhas do programa BASIC. Este método baseia-se na facilidade de acesso destas informações, e permite um uso bastante económico da memória. O aspecto importante a ter em conta é que o endereço do primeiro caracter escrito a seguir à palavra REM quando esta se encontra na primeira linha de programa é 23760 (todos os programas BASIC começam a ser guardados na memória a partir do endereço 23755; esta posição e a seguinte guardam o número da linha, as duas posições seguintes o comprimento da linha, sendo depois guardado o texto da linha; em 23759 é neste caso guardada a palavra-chave REM). Se portanto tivermos o programa:

```
1 REM ABCDEF
2 PRINT PEEK 23560
```

o programa imprimirá ao ser executado o número 65. Este número é o código do caracter A, pelo que de facto o endereço 23760 contém este caracter em código.

Pode-se facilmente alterar este valor usando a função POKE. Se por exemplo quisermos substituir A por Z, podemos procurar na tabela existente no manual da máquina o código correspondente a Z, que é 90. Em seguida escrevemos:

POKE 23760,90

Poderíamos também ter escrito POKE 23760, CODE "Z", usando a função CODE e obter o mesmo resultado. O endereço seguinte, 23761, guarda o código de B, 23762 o código de C, etc.

A técnica de guardar e recuperar valores em posições de memória definidas por declarações REM reveste uma particular importância para armazenamento de programas em código-máquina em computadores como o ZX81.

Se bem que esta técnica também possa ser usada no Spectrum, o manual deste (no capítulo intitulado "Uso de Código Máquina) explica um modo de reservar memória para este fim e de introduzir o programa em código (usando POKEs) na área assim reservada da memória.

2) Uso do contador de imagens.

Nos endereços 23672, 23673 e 23674 é contado o número de imagens enviadas para o receptor de televisão desde que o computador foi ligado. Podem-se usar as funções PEEK e POKE sobre estes endereços.

Este contador pode ser colocado em zero recorrendo às instruções seguintes:

```
POKE 23674, 255
POKE 23673, 255
POKE 23672, 255
```

E para ler o valor neles contido usa-se a seguinte expressão:

```
65536* PEEK 23674 + PEEK 23672 + 256* PEEK 23673
```

Obtemos deste modo uma resposta em número de imagens: e como em Portugal são enviadas par o receptor de televisão 50 imagens por segundo (equivalente à frequência de 50 Hertz ou ciclos por segundo da tensão de alimentação) teremos de dividir aquele total por 50 a fim de obtermos uma resposta em segundos:

```
LET Tempo=(65536*PEEK23674 + PEEK23672 +256*
PEEK 23673)/50
```

Inclui-se a seguir um programa capaz de actuar como cronómetro:


```
  5 POKE 23674,255
 10 POKE 23673,255
 20 POKE 23672,255
 30 LET time=(65536*PEEK 23674+
PEEK 23672+256*PEEK 23673)/50
 40 PRINT AT 11,14;INT (time*10
)/10;"  "
 50 GO TO 30
```

A função INT da linha 40 foi acrescentada a fim de impedir a impressão de fracções inferiores a um décimo de segundo. Este cronómetro é razoavelmente rigoroso porque o contador de imagens é controlado por circuitos especiais, pelo que a menos que o programa actue deliberadamente sobre ele a contagem que realiza é independente da velocidade a que qualquer programa é executado. O contador permite atingir tempos até aproximadamente quatro dias. Se o leitor quiser obter uma resposta em minutos e segundos pode utilizar a rotina seguinte:

```
  5 POKE 23674,255
 10 POKE 23673,255
 20 POKE 23672,255
 30 LET time=(65536*PEEK 23674+
PEEK 23672+256*PEEK 23673)/50
 40 PRINT AT 11,12;INT (time/60
);":";INT (time-INT (time/60)*60
);"  "
 50 GO TO 30
```

34
USOS COMERCIAIS

O computador pode ser usado para várias aplicações comerciais em pequenas empresas. Existe no comércio uma grande variadade de programas que exploram as potencialidades do computador neste campo. Neste capítulo iremos estudar algumas aplicações práticas muito simples do seu computador Spectrum.

O primeiro programa utilitário que apresentamos serve para calcular juros compostos. Foi escrito por James Walsh. As perguntas são feitas ao utilizador de modo bastante claro, e é um programa fácil de seguir.

```
 90 LET A$="Ano Juro Total"
100 INPUT "Número de anos?";Y
110 PRINT "Juro composto"
120 PRINT "" Ao longo de ";Y;" anos"
130 INPUT "Quantia?";A; LET T= A
140 PRINT "Capital a juro:";A
150 INPUT "Juros ao ano? ";IN
160 CLS : PRINT AT 1,0;
170 FOR N=1 TO Y
180 POKE 23692,-1
200 GO SUB 340
210 PRINT N;TAB 8;"$";INT (U+.5
);TAB 19;"$";INT (T+.5)
260 NEXT N
270 PRINT "Total=$";INT (T+.5)
290 PRINT "juro - "; IN; "%"
310 PRINT "Quantia original= ";a
320 PRINT AT 0,0;A$
330 STOP
340 LET U=1+(IN/100)*T
350 LET T=(IN/100)*T+T
360 RETURN
```

PROCESSAMENTO DE TEXTO

Este programa para processamento de texto disporá as palavras do modo mais apropriado antes de as imprimir - permite-nos emendar quaisquer erros usando um cursor móvel. Escreve-se o texto (até 17 linhas), que é atribuído a uma única cadeia alfanumérica, XS. Quando todo o texto foi introduzido carrega-se em ENTER, e o computador distribui as palavras de modo a assegurar que nenhuma delas é cortada no final de cada linha.

No início aparece um menú com três opções: 1 - corrigir o texto: 2 - LPRINT o texto: e 3 - recomeçar. Se decidir corrigir um texto, este aparecerá de novo no visor, com as palavras "Escreva 1 para voltar ao menú" escritas em cima. Deve usar em seguida as teclas 5, 6, 7 e 8 para mover o cursor na direcção indicada pelas setas nelas existentes, e o cursor desloca-se ao longo do texto colocando um ponto negro sobre a letra pela qual passa. Quando encontra uma letra errada carrega em "A" e na parte inferior do visor surgem as palavras "Escreva letra". Você escreve a letra certa e carrega em ENTER, sendo a letra errada substituída pela nova. Se carregar em "1" em qualquer momento voltará ao menú original, e a partir deste modo pode escrever "2" para LPRINT o texto (imprimi-lo numa impressora).

Depois de passar o texto para a impressora observa no visor um novo menú, que nos permite recomeçar, voltar a imprimir, ou abandonar o programa.

```
10 REM  processador de texto
20 PRINT "Escreva o texto"
30 INPUT X$
32 LET X$=X$+"
                "
35 CLS
40 GO SUB 1000
45 GO SUB 1000
50 PRINT X$
70 PRINT "Escreva 1 para corrigir o texto, 2 para imprimir,
          3 para começar"
 80 INPUT Q
100 IF Q=3 THEN RUN
110 IF Q=2 THEN GO TO 4000
120 IF Q=1 THEN GO TO 2000
130 GO TO 80

1000 REM Pára divisão de palavras
1010 LET N=1
1020 GO SUB 1180
1030 LET N=N+33
1040 IF N>=LEN X$ THEN RETURN
1045 REM Um espaço na linha seguinte
1050 IF X$(N)=" " THEN GO TO 116
0
1065 REM  Um espaço na linha seguinte
1070 IF X$(N)=" " THEN GO TO 103
0
1080 LET J=0
1090 GO SUB 1180
1100 LET J=J+1
1105 REM  Um espaço na linha seguinte
          NEXT LINE
1110 IF X$(N)>" " THEN GO TO 109
0
1120 FOR N=N TO N+J-1
1125 REM  Um espaço na linha seguinte
```

```
1130 LET X$=X$(1 TO N)+" "+X$(N+
1 TO )
1140 NEXT N
1150 GO TO 1030
1160 LET X$=X$(1 TO N-1)+X$(N+1
TO )
1170 GO TO 1020
1180 LET N=N-1
1190 RETURN
2000 REM *  Correcções  *
2010 CLS
2020 PRINT "Escreva 1 para voltar ao
 menu"
2030 LET A=1: LET L=LEN X$
2032 LET Z$=A$(1)
2035 PRINT AT 2,0;X$
2037 LET X$(A)=Z$
2050 IF INKEY$="8" AND A<L THEN
LET A=A+1
2055 IF INKEY$="6" AND A<L+32 TH
EN LET A=A+32
2060 IF INKEY$="5" AND A>1 THEN
LET A=A-1
2065 IF INKEY$="7" AND A>32 THEN
 LET A=A-32
2070 IF INKEY$="1" THEN GO TO 70
2080 IF INKEY$="A" THEN GO SUB 3
000
2090 PRINT AT 1,0;A;" ";X$(A);"
"
2100 LET Z$=X$(A)
2110 LET X$(A)="■"
2120 GO TO 2035
3000 INPUT INK 2; FLASH 1; BRIGH
T 1;"  Escreva letra  ";H$
3010 LET X$(A)=H$
3030 RETURN
4000 LPRINT X$
4010 PRINT "Escreva 1 para imprimir de novo"
4020 PRINT TAB 5;"2 para recomeçar
"
4030 PRINT TAB 5;"3 para parar"
4040 INPUT U
4050 IF U=1 THEN GO TO 4000
4060 IF U=2 THEN RUN
4070 IF U=3 THEN STOP
4080 GO TO 4040
```


O último programa deste capítulo serve para colocar indicações e referências de páginas por ordem alfabética, permitindo a construção de indíces (por exemplo de livros ou artigos em revistas) sendo fácil adaptá-lo para listas de stocks, etc.

O programa é em três partes. A primeira (até à linha 100) aceita as entradas, a segunda (linhas 200 a 300) ordena-as e a terceira (linhas 310 a 530) apresenta os dados.

O programa pede-nos que escrevamos um título (TS) e o nome do autor (NS), indicando em seguida os temas e as páginas, uma a uma, terminando com um "E". O programa aceitará até 400 entradas (linha 20). Este máximo foi pensado para uma máquina de 16K. Num computador de 48K podem-se utilizar 2000 entradas ou mais. No fim pode-se escolher entre imprimir o resultado no visor ou na impressora.

Vejamos um exemplo de execução:

Jogos Mentais
João Silva
Aprendizagem — 189
Atenção — 7
Compreensão — 7
Equipamento — 32
Memória — 92
Percepção — 187
Personalidade — 9

```
 10 REM  Indice de livros
 20 DIM A$(400,16)
 30 INPUT "Escreva título";T$
 40 INPUT "Escreva autor";N$
 50 FOR G=1 TO 500
 60 INPUT  "Escreva palavra e página
                Escreva "E" para terminar
    ";A$(G)
 70 REM  15  espaços na linha seguinte
 80 IF A$(G)="E
THEN GO TO 200
 90 PRINT A$(G)
100 NEXT G
200 CLS
210 PRINT  "Um momento por favor"
220 FOR B=1 TO G-1
230 FOR C=B+1 TO G-1
```

```
240 IF A$(B)<=A$(C) THEN GO TO
250
250 LET D$=A$(B)
260 LET A$(B)=A$(C)
270 LET A$(C)=D$
280 NEXT C
290 NEXT B
300 PRINT "Terminado"
310 PRINT "Escreva 1 para LPRINT"
320 PRINT "Escreva 2 para DISPLAY"
330 IF INKEY$="2" THEN CLS : GO
TO 440
340 IF INKEY$="1" THEN GO TO 36
0
350 GO TO 330
360 LPRINT T$
370 LPRINT
380 LPRINT N$
390 LPRINT
400 FOR A=1 TO G-1
410 LPRINT A$(A)
420 NEXT A
430 STOP
440 PRINT T$
450 PRINT N$
460 PRINT
500 FOR A=1 TO G-1
520 PRINT A$(A)
530 NEXT A
```

35
MELHORE OS SEUS PROGRAMAS

O leitor passou provavelmente por várias fases no desenvolvimento das suas capacidades como programador. Após uma primeira e breve luta com o BASIC, descobriu subitamente que é facil escrever programas com êxito. Talvez sejam bastante deselegantes em termos de listagem, e talvez os seus amigos necessitem de uma explicação detalhada antes de perceberem o que deveriam fazer ao usar o seu programa, mas pelo menos a máquina aceitava-os e executava-os.

Acabamos sempre por chegar a uma fase em que resolvemos

fazer melhor do que isto. Mas muitas vezes sentimo-nos apenas vagamente insatisfeitos com os nossos programas, sem sabermos muito bem como deveremos melhorá-los. Vamos apresentar algumas ideias básicas que talvez o possam guiar.

Em primeiro lugar, observe a listagem do seu programa. Os programas com declarações REM têm uma melhor aparência, e são mais fáceis de entender quando voltamos a estudá-los ao fim de algum tempo. Como é óbvio, a falta de memória pode impedir este luxo, mas se dispuser de memória suficiente deve sempre incluí-las. Simples declarações REM seguidas de uma linha de asteriscos, para se notarem mais facilmente, podem ser muito úteis para separar as diferentes partes de um programa. Examine criticamente qualquer GO TO incondicional. Um excesso de GO TO enviando para outras partes do programa demonstram um modo confuso de pensar, tornam os programas mais lentos, e podem impossibilitar a sua decifração.

É muito boa prática ter cada uma das secções principais de um programa (como a que define as variáveis no início da execução, a que imprime o tabuleiro de jogo, a que determina quem ganhou, etc) em subrotinas separadas. O início do seu programa pode assemelhar-se um pouco ao seguinte:

```
10 REM * Nome do Programa *
20 REM VARIAVEIS
30 GOSUB 9000
40 REM COMPOR QUADRO
50 GOSUB 8000
60 REM LANCE DO JOGADOR
70 GOSUB 7000
80 REM LANCE DO COMPUTADOR
90 GOSUB 6000
100 REM VERIFICAR SE O JOGO ACABOU
110 GOSUB 5000
120 GOTO 50
```

Como pode verificar, assegura assim que o programa execute um ciclo contínuo até terminar na subrotina 5000. Pode escrever uma série de linhas como estas antes de escrever qualquer outra coisa, e mesmo antes de saber como vai programar algumas das tarefas executadas pelas subrotinas.

Em seguida pode escrever o programa módulo a módulo, verificando se cada um destes funciona antes de passar ao seguinte. É relativamente fácil descobrir os erros num programa actuando deste modo, e é sempre mais simples manter uma ideia de onde se encontra tudo. Evite deixar que o programa "se escreva a si mesmo".

A listagem deve ser portanto tão transparente quanto possível, tanto para facilitar a descoberta de erros como para permitir mais tarde uma compreensão imediata da utilidade de cada uma das suas partes. As saídas do programa devem por outro lado manter uma boa aparência: se a quantidade de memória disponível não constituir problema, procure melhorar a imagem no visor até ser bastante clara e simples. Use linhas em branco para espaçar as frases, traços para dividir o visor de um modo lógico, etc. Depois de o programa trabalhar satisfatoriamente, vale a pena perder mais algum tempo com a subrotina que controla a imagem. Constatará uma vez mais a vantagem de ter uma única subrotina a tratar da imagem, dado que é assim fácil descobrir o modo de o melhorar.

Evidentemente, como vivemos num mundo que não é ideal, é improvável que seja impossível incluir todas as instruções destinadas ao visor numa única subrotina, mas se apontar para este fim verificará que se torna muito mais fácil desenvolver depois o programa até lhe parecer satisfatório.

O método "estruturado" que esboçámos ajuda-o também a realizar um outro objectivo de um bom programa — fazer exactamente aquilo que esperamos que faça, de cada vez que é executado. Deve escrever um programa de tal modo que, mesmo que não esteja presente quando um amigo resolve executá-lo, se comporte exactamente do modo previsto. Isto significa não só, como é óbvio, que não deve conter quaisquer erros, mas também que contém instruções (que podem ser incluídas na subrotina que define as variáveis) simples, claras e completas.

As perguntas feitas ao utilizador devem ser claras, de tal modo que o operador saiba quando deve escrever um número, uma série de números, uma palavra, uma data, uma mistura de letras e números, etc. O programa deve partir do princípio de que o operador é um idiota acabado, e que apesar da clareza das instruções ou perguntas o utilizador acabará sempre por fazer algo de errado. Um exemplo clássico disto é a indicação de datas. As validações de entradas devem ser pensadas de tal modo que garantam a rejeição

de uma data (ou qualquer outra coisa) escrita de um modo que a máquina não percebe (por exemplo indicando o mês antes do dia) ou que esteja errada (por exemplo referindo o dia 32 de Fevereiro). Deve garantir que, independentemente daquilo que o operador fizer, o programa não estoire ou actue de algum modo imprevisto. Isto pode acontecer por exemplo quando a máquina espera uma entrada numérica e o operador se engana e carrega na tecla correspondente a uma letra, ou ainda quando carrega em ENTER sem ter escrito nada. Pode-se evitar isto admitindo apenas entradas alfanuméricas, verificando-se o conteúdo destas é correcto e devolvendo a execução à mesma instrução de INPUT no caso de não o ser. Se a entrada for correcta, pode-se transformá-la facilmente em valor numérico utilizando uma das instruções VAL ou CODE.

A documentação é um aspecto da programação que muitas vezes se tende a negligenciar. É praticamente essencial realizá-las num programa que é destinado a publicação, e é sempre aconselhável proceder do mesmo modo mesmo em programas para uso exclusivo do autor. No mínimo, a documentação deve incluir uma lista de variáveis, uma explicação da estrutura do programa (que será fácil de fazer se o programa tiver sido construído do modo "modular" indicado mais atrás) e algumas instruções breves, em particular no caso do próprio programa não as conter. Um exemplo de execução mostrando o tipo de entradas e a natureza e apresentação das saídas do programa é também bastante útil.

O seu programa deve ser executado tão rapidamente quanto possível. De cada vez que é executado um GO TO ou é chamada uma subrotina, o computador deve procurar em todo o programa, linha a linha, aquela que lhe interessa: nestas condições, a colocação das subrotinas muito usadas perto do início do programa permite acelerá-lo um pouco. É por isso que as instruções de programa estão muitas vezes colocadas no final deste. Certamente que o leitor não deseja que o computador passe novamente pelas linhas de inicialização e de instruções de cada vez que lhe é dito que procure uma subrotina ou saia dela.

Defina antes de mais as variáveis muito usadas, de modo a ocuparem as primeiras linhas de definição de variáveis. O computador só procurará até encontrar a linha que deseja, pelo que não há qualquer necessidade de o forçar a ler mais números de linha do que os estritamente necessários.

Finalmente, e tenha presente que este é de longe o melhor modo de verificar um programa escrito por si, peça a um amigo ou a uma amiga que se sente à frente da televisão e carregue em RUN, sem conhecer absolutamente nada sobre o programa. Se houver qualquer hesitação da sua parte, ou o programa não actuar do modo esperado, vai ter que perder mais algum tempo com ele.

Em resumo, portanto:

— Utilize instruções REM.

— Faça programas com listagens simples e lógicas.

— Utilize técnicas de programação estruturadas, controlando o programa por um ciclo de chamamentos a subrotinas.

— Examine as instruções de salto incondicional de modo crítico.

— Torne as saídas em visor atraentes e claras.

— Garanta a maior clareza possível nas perguntas feitas ao utilizador.

— Valide todas as entradas.

— Documente os seus programas, mesmo que apenas construa uma lista de variáveis.

— Tente dar a maior velocidade possível aos programas.

— Verifique os programas pedindo a alguém que os execute.

36
PROGRAMAS, PROGRAMAS, PROGRAMAS

Finalmente, vamos apresentar alguns programas que talvez lhe agradem.

```
  1 REM Corrida de galgos
  2 REM @ Gourlay,Hartnell 1982
  5 RANDOMIZE
  7 GO SUB 200
 10 FOR x=1 TO 22
 20 PRINT INK 4;TAB 30;"▗▀▖"
 30 NEXT x
 35 PRINT AT 0,6; "Aposta no número";w
 40 DIM a(9)
 50 FOR x=1 TO 9
 60 PRINT AT 2*x,a(x);" "
 70 LET a(x)=a(x)+RND*2
 80 PRINT AT 2*x,a(x); INK x/2;
x
 85 BEEP .01,3*x
 90 IF a(x)>30 THEN GO TO 115
100 NEXT x
110 GO TO 50
115 FOR g=1 TO 50 STEP 2
120 PRINT AT 18,6; INK RND*7;x;
" é o vencedor!"
123 BEEP .02,g
125 IF w=x THEN PRINT AT 20,3;
INK RND * 7;"E você também vence!!"
130 FLASH RND
140 NEXT g
145 FLASH 0
150 RUN
200 BORDER 0
205 PRINT AT 3,1; INK 2; "Bem-vindos
às corridas de galgos"
210 PRINT AT 5,6; INK 4; "Correm
nove cães."
220 INPUT ( INK 2;"Faça as suas apostas");w
230 IF w<1 OR w>9 THEN GO TO 22
0
235 BORDER 2
240 CLS : RETURN
```

```
 10 REM Star bouncer
 15 REM @ Hartnell, 1982
 20 BORDER 0
 30 LET a=1: LET b=1: LET c=RND
*20: LET d=RND*20
 40 PRINT AT c,d;"▗▀▖";AT c,d; IN
K RND*7;"*"
 50 IF c+b>21 OR c+b<-1 THEN LE
T b=-b: BEEP 0.2,-RND*15
 60 IF d+a>31 OR d+a<0 OR RND>.
95 THEN LET a=-a: BEEP 0.1,RND*2
5
 70 LET c=c+b: LET d=d+a
 80 GO TO 40
```

COLOURTHELLO

Desafie o seu Spectrum para um jogo de Reversi com este programa. Colourthello, escrito por Graham Charlton. Colourthello foi pensado para ilustrar as potencialidades em termos de som e cor do Spectrum. Quando fizer executar o programa, verificará a extensão destas. Os movimentos são realizados indicando o número escrito de lado e em seguida o que se encontra na linha superior ou inferior, sob a forma de um número com dois algarismos.

```
    1 REM _____Colourthello____
    5 PRINT AT 0,12; INK 2;"C"; I
NK 1;"o"; INK 6;"l"; INK 3;"o";
INK 4;"u"; INK 5;"r"; INK 2;"t";
 INK 1;"h"; INK 6;"e"; INK 3;"l"
; INK 4;"l"; INK 2;"o"
   10 DIM a(10,10): FOR b=1 TO 10
: FOR c=1 TO 10
   20 BEEP .01,b*c/10
   40 IF b<>1 AND c<>1 AND b<>10
AND c<>10 THEN LET a(b,c)=CODE "
*"
   50 NEXT c: NEXT b: LET p=0: LE
T r=0
   70 LET a(5,5)=CODE "x": LET a(
5,6)=CODE "x": LET a(6,5)=CODE "
o": LET a(6,6)=CODE "o"
  120 INPUT (INK; "Quer jogar primeiro?"); q$
  125 CLS : GO SUB 3000
  127 PRINT AT 0,12; INK 2;"C"; I
NK 1;"o"; INK 6;"l"; INK 3;"o";
INK 4;"u"; INK 5;"r"; INK 2;"t";
 INK 1;"h"; INK 6;"e"; INK 3;"l"
; INK 4;"l"; INK 2;"o"
  130 IF CODE q$<>CODE "n" AND CO
DE q$<>CODE "N" THEN GO TO 2000
1000 PRINT INK 2;AT 10,16;"Meu
movimento"
1010 LET s=CODE "o": LET t=CODE
"x": LET h=0
1040 FOR a=2 TO 9: FOR b=2 TO 9
1060 IF a(a,b)<>CODE "." THEN GO
 TO 1320
1070 LET q=0: FOR c=-1 TO 1: FOR
 d=-1 TO 1: LET k=0: LET f=a: LE
T g=b
1130 IF a(f+c,g+d)<>s THEN GO TO
 1180
1140 LET k=k+1: LET f=f+c: LET g
=g+d: GO TO 1130
1180 IF a(f+c,g+d)<>t THEN GO TO
 1200
1190 LET q=q+k
1200 NEXT d
1210 NEXT c
1220 IF f=2 OR f=9 OR g=2 OR g=9
 THEN LET q=q*2
1230 IF f=3 OR f=8 OR g=3 OR g=8
 THEN LET q=q/2
1250 IF (f=2 OR f=9) AND (g=3 OR
 g=8) OR (f=3 OR f=8) AND (g=2 O
R g=9) THEN LET q=q/2
1280 IF q<h OR q=0 OR (RND>.3 AN
D q=h) THEN GO TO 1320
1290 LET h=q: LET m=a: LET n=b
1320 NEXT b
1330 NEXT a
1340 IF h=0 AND r=0 THEN GO TO 5
000
1350 IF h=0 THEN GO TO 1370
1360 GO SUB 4000
1370 GO SUB 3000
2000 PRINT INK 1;AT 10,16;"Seu
movimento"
2010 LET s=CODE "x": LET t=CODE
"o"
2030 INPUT r
2040 IF r=0 THEN GO TO 2090
2050 IF r<11 OR r>88 THEN GO TO
2030
2060 LET m=INT (r/10)+1: LET n=r
-10*INT (r/10)+1
2080 GO SUB 4000
2090 GO SUB 3000: GO TO 1000
3000 PRINT AT 5,0;: BEEP .25,RND
*5
3010 LET c=0: LET h=0
3030 PRINT INK 4;" 12345678 "
3040 FOR b=2 TO 9: PRINT INK 4;b
-1;
3060 FOR d=2 TO 9
3070 IF a(b,d)=CODE "x" THEN PRI
NT INK 2;"x";
```

```
3075 IF a(b,d)=CODE "o" THEN PRI
NT INK 1;"o";
3077 IF a(b,d)=CODE "." THEN PRI
NT INK 5;".";
3080 IF a(b,d)=CODE "x" THEN LET
 c=c+1
3090 IF a(b,d)=CODE "o" THEN LET
 h=h+1
3100 NEXT d
3110 PRINT INK 4;b-1
3120 NEXT b
3130 PRINT INK 4;"  12345678  "
3150 PRINT 'INK 3; "Tenho" ;INK 2;C; INK 3;"
"Você tem"; INK 1;h;" ""

3170 RETURN
4000 FOR c=-1 TO 1
4010 FOR d=-1 TO 1
4020 LET f=m: LET g=n
4040 IF a(f+c,g+d)<>s THEN GO TO
 4080
4050 LET f=f+c: LET g=g+d: GO TO
 4040
4080 IF a(f+c,g+d)<>t THEN GO TO
 4140
4090 LET a(f,g)=t: IF m=f AND n=
g THEN GO TO 4140
4110 LET f=f-c: LET g=g-d: GO TO
 4090
4140 NEXT d: NEXT c: RETURN
5000 IF c > h THEN PRINT "Ganhei;";C;"-";h

5010 IF h>c THEN PRINT "Você ganhou
 ";h;" - ";c
5050 © Charlton 1982
```

```
  12345678
1....x...1
2....x...2
3...xox...3
4....ox...4
5...xox...5
6....o....6
7........7
8........8
  12345678
```

Tenho 7

```
Você ganhou
```

Você tem 4

VIDA

Apresentamos agora duas versões do jogo da VIDA de John Conway, que simula o nascimento, crescimento e morte de uma colónia de células. Estas evoluem de acordo com as seguintes regras:

— Cada célula possui oito vizinhas.
— Todas as células com duas ou três vizinhas, sobrevive até à geração seguinte.
— Se existirem três, e apenas três células vizinhas, nasce uma nova célula.
— Qualquer célula com quatro ou mais vizinhas morre devido a excesso de população.

```
   5 REM LIFE - © ANNE MARSHALL
  10 DIM A(145): DIM L(145): DIM
E(8)
  15 LET G=0
  20 FOR T=1 TO 8
  25 READ Z: LET E(T)=Z: NEXT T
  30 LET C=CODE "o": LET Z=128
  35 BORDER 1: PAPER 0: CLS
  40 FOR B=1 TO 12
  50 FOR D=1 TO 12
  60 LET A(B+10*D)=Z
  70 IF RND>.45 THEN LET A(B+10*
D)=C
  80 LET L(B+10*D)=A(B+10*D)
  90 NEXT D: NEXT B
 100 LET G=G+1
 120 FOR U=1 TO 12
 130 FOR B=1 TO 12
 140 LET F=U+10*B
 180 IF G=1 THEN GO TO 250
 190 LET H=0
 200 FOR T=1 TO 8
 210 IF A(F+E(T)+1)=C THEN LET H
=H+1
 220 NEXT T
 230 IF A(F)=C AND H<>3 AND H<>2
 THEN LET L(F)=Z
 235 IF A(F)=Z AND H=3 THEN LET
L(F)=C
```

```
 240 NEXT B: BORDER RND*7: NEXT
U
 245 BORDER 1
 250 FOR M=11 TO 144: LET A(M)=L
(M): BEEP .005,M/3: NEXT M
 255 PRINT AT 5,0;
 260 FOR U=1 TO 12: PRINT TAB 4;
 270 FOR B=1 TO 12: LET F=U+10*B
 280 PRINT INK 6;CHR$ A(F);" ";:
 NEXT B: PRINT : NEXT U
 285 PRINT AT 3,10; PAPER 2; INK
 6;"Generation ";G: BEEP .3,50
 290 GO TO 100
 300 DATA 11,10,9,1,-1,-9,-10,-1
1


  10 REM Conway's Colony
  15 REM © Hartnell, 1982
  20 GO SUB 90
  30 LET colónia=200
  40 LET   actualizaçao =320
  45 REM ****************
  50 GO SUB colónia
  60 GO SUB actualização
  70 GO TO 50
  80 REM ******************
  90 REM  inicializar
 100 CLS
 110 LET    células =0
 120 DIM a(11,11): DIM b(11,11)
 130 FOR x=2 TO 10: FOR y=2 TO 1
0
 135 BORDER RND*7
 140 IF RND>.35 THEN LET a(x,y)=
1: BEEP .02,x*y/2: LET  células= células
  +1
 150 LET b(x,y)=a(x,y)
 160 NEXT y: NEXT x
 170 LET year=0
 175 BORDER 7: CLS
 180 RETURN
 190 REM ******************
 200 REM Print  colónia
 210 LET    ano = ano+1   1
 220 BEEP .02,RND*20
 250 PRINT AT 1,0; INK RND*6;"[redacted]
[redacted]    ";ano;AT 3,8;
 255 LET  células= 0
 260 FOR x=2 TO 10: FOR y=2 TO 1
```



```
0
 270 LET a(x,y)=b(x,y)
 280 IF a(x,y)=0 THEN PRINT "   "
;
 285 IF a(x,y)=1 THEN PRINT INK
RND*6;"[redacted]";: LET   células = células+1   1
 290 NEXT y: PRINT : PRINT : PRI
NT TAB 8;: NEXT x
 292 PRINT AT 21,0; INK RND*6;"cé
lulas     "; células ;"  "
 293 IF células < 6 THEN RUN
 295 RETURN
 300 REM ****************
 310 REM actualização
 340 FOR x=2 TO 10: FOR y=2 TO 1
0
 345 BORDER RND*6
 350 LET c=0
 360 IF a(x-1,y-1)=1 THEN LET c=
c+1
 370 IF a(x-1,y)=1 THEN LET c=c+
1
 380 IF a(x-1,y+1)=1 THEN LET c=
c+1
 390 IF a(x,y-1)=1 THEN LET c=c+
1
 400 IF a(x,y+1)=1 THEN LET c=c+
1
 410 IF a(x+1,y-1)=1 THEN LET c=
c+1
 420 IF a(x+1,y)=1 THEN LET c=c+
1
 430 IF a(x+1,y+1)=1 THEN LET c=
c+1
 440 IF a(x,y)=1 AND c<>2 AND c<
>3 THEN LET b(x,y)=0
 450 IF a(x,y)=0 AND c=3 THEN LE
T b(x,y)=1
 460 NEXT y: NEXT x
 470 RETURN
```

FÓSFOROS

Neste jogo considera-se que existe um determinado número de fósforos, e que o jogador e a máquina tiram alternadamente um ou mais fósforos; a maior quantidade que se pode tirar de cada vez é indicada na parte superior do visor. O jogador que tira o último fósforo perde. O computador não é infalível.

```
  5 REM * Fósforos *
 10 REM  Texto branco sobre azul
 15 PAPER 1: INK 7: BORDER 1: C
LS
 20 LET E=0: LET Z=16+INT (RND*
9)
 30 IF 2*(Z/2)=Z THEN LET Z=Z+1
 40 LET H=INT (RND*4)+2
 50 PRINT PAPER RND*5+2; INK 0;
AT 0,6;" Máximo ";H
 60 IF E>0 THEN PRINT AT 7,2;
 "Você tirou"; E; TAB 20;"Eu tirei"; Q'
 70 FOR K=1 TO Z: BEEP .01,K
 80 PRINT INK RND*5+2;K;"■ ";
 90 IF RND>.85 THEN PRINT : PRI
NT
 100 NEXT K
 105 LET K=7: IF RND>.5 THEN LET
 K=4
 110 INPUT INK K; "Quantos quer tirar?";E
 120 IF E>H OR E<1 THEN GO TO 11
0
 130 CLS : LET Z=Z-E
 140 IF Z=0 THEN BORDER RND*7: P
RINT PAPER RND*5;AT 10,12; "Venci"
": BEEP .05,RND*30+30: GO TO 140
 150 LET Q=Z-1-INT ((Z-1)/(H+1))
*(H+1)+INT (RND*3)-1
 160 IF Q>Z OR Q<1 OR Q>H THEN G
O TO 150
 170 LET Z=Z-Q
 180 IF Z=0 THEN BORDER RND*7:PRINT
PAPER RND * 6; AT 10,5 "Eu tirei";Q;" e portanto GANHOU!"
: BEEP. 05, RND * 40: GO TO 180.
 190 GO TO 50
```


Máximo: 5

Você tirou 3 Eu tirei 1

```
1■  2■

3■

4■  5■  6■  7■  8■

9■

10■  11■

12■  13■  14■
```

CASINO

O programa que se segue custa-lhe 1550 por cada rotação.

De vez em quando surge a opção PARAR. Pode-se parar as quatro rodas se se quiser. Quando surge a palavra PARAR, indica-se cada um dos números que se deseja parar, carregando em ENTER depois de cada um. Para passar à rotação seguinte carrega-se em 5" e em ENTER.

```
 20 POKE 23609,100
 30 GO SUB 9000
 40 POKE 23692,-1
 50 PAPER 0: CLS : BORDER 0: IN
K 7
 60 PRINT "'PAPER 2; TAB2; "Esta é a volta"; VOLTA;
TAB2;"Você possui $"; dinheiro"' TAB2; "Carregue em qualquer
tecla para rolar"
 70 IF INKEY$<>"" THEN GO TO 70
 80 IF INKEY$="" THEN GO TO 80
 85 POKE 23692,-1
 90 FOR G=1 TO 50: BORDER RND*7
: BEEP .01,50-G: NEXT G: BORDER
0
 100 FOR J=1 TO 4
```

```
  110  IF M(J)=J THEN GO TO 150
  120  LET A(J)=INT (RND*4)+1
  140  BEEP .1,50/J
  150  NEXT J
  155  LET      volta=volta+1
  160  GO SUB 5000
  165  GO SUB 4000
  170  IF RND>.7 THEN GO SUB 6000
  175  FOR T=1 TO 40: PRINT AT 1,2
5; INK RND*7;"■ ■";AT 1,25; INK
RND*7;" ■ ": NEXT T
  177  FOR T=1 TO 25: PRINT : NEXT
T
  180  IF dinheiro > 0 THEN GO TO 60
  190  PRINT ''''TAB 5;"sobreviveu"; volta;
" voltas"
  195  BORDER RND*7
  200  PRINT "Mas agora está falido
e o"
  205  BORDER RND*7
  210  PRINT "C A S I N O   FECHOU!
  215  BORDER RND*7
  220  POKE 23692,-1
  230  PAUSE 10
  240  GO TO 190
4000  REM ** Dinheiro    **
4005  PRINT ''': POKE 23692,-1
4010  LET    dinheiro=dinheiro-1.5
4020  IF A(1)=A(2) AND A(2)=A(3)
AND A(3)=A(4) THEN PRINT INK 6;"
£££££   JACKPOT!!!!   $$$$$$$$$$$$$$$$$$
 BEEP 2,10: PRINT     "Voce ganhou"
;" ": LET    dinheiro = dinheiro +10: GO TO 4
100
4030  IF (A(1)=A(2) AND (A(3)=A(2
) OR A(4)=A(2))) OR A(1)=A(2) AN
D A(2)=A(4) OR A(2)=A(3) AND A(3
)=A(4) THEN PRINT INK 6; PAPER
;"$$$$$$$$$    TRIS IGUAIS!          $$$$
$": BEEP 2,20: PRINT ;"Ganhou
$5": LET    dinheiro=dinheiro +5: GO TO 41
00
4040  IF A(3)=A(2) AND A(3)=A(4)
THEN PRINT INK 6; PAPER 2;"$$$$$
$$$ TRIO $$ TRIO $$ $$$$$": BEEP
 2,40: PRINT ''''"Ganhou          $7.50":
LET    dinheiro=dinheiro +7.5: GO TO 4100
4050  IF A(1)+A(2)+A(3)+A(4)=10 T
```



```
HEN PRINT PAPER 2;">>>>>>>>>"; PA
PER 0;" SMASHEROO!!": BEEP 2,-30
: PRINT ''" GANHOU   $7.50!!": LET
   dinheiro=dinheiro +7.5
4100  FOR T=1 TO 20: BORDER RND*7
: BEEP .01,T: NEXT T: BORDER 0
4105  PRINT
4110  FOR T=1 TO 64: PRINT INK RN
D*7;"■";: NEXT T
4120  PRINT ''TAB 8;"AGORA VOCE TEM"
; dinheiro'
4130  FOR T=1 TO 64: PRINT INK RN
D*7;"■";: NEXT T
4140  PRINT
4150  POKE 23692,-1: PRINT : PRIN
T
4160  DIM M(4)
4170  RETURN
5000  REM **  rolar    **
5010  FOR T=1 TO 50: BORDER RND*7
: BEEP .01,50/T/2: NEXT T: BORDE
R 0
5020  PRINT ''''TAB 4;
5030  FOR J=1 TO 4
5040  IF A(J)=1 THEN PRINT INK 2;
"■<>■ ";: BEEP .1,10
5050  IF A(J)=2 THEN PRINT INK 7;
"■$$■ ";: BEEP .1,20
5060  IF A(J)=3 THEN PRINT INK 4;
"■**■ ";: BEEP .1,30
5070  IF A(J)=4 THEN PRINT INK 5;
"■■■■ ";: BEEP .1,40
5080  PAUSE 70
5090  NEXT J
5100  RETURN
6000  REM **  parar    **
6010  DIM M(5)
6020  BEEP .5,1
6025  POKE 23692,-1
6030  PRINT '' INK 6; "Indique qualquer nú
mero que"
6040  PRINT ' INK 6;    "queira parar:
    escreva 5"
6050  PRINT ' INK 6; "quando tiver
acabado."
6060  INPUT Q
6070  IF Q<>5 THEN PRINT INK 2;Q
6080  LET M(Q)=Q
```

```
6090 IF 0<>5 THEN GO TO 6060
6100 RETURN
9000 REM **   variáveis   **
9010 DIM A(5): DIM M(5)
9020 LET dinheiro=7.5
9030 LET voltas=1
9040 FOR T=1 TO 20
9050 BEEP .2,2*T
9060 NEXT T
9070 BORDER 7: PAPER 7: CLS
9080 BORDER 0: PAPER 0: CLS
9090 RETURN
```

CIRCUITO

Este programa foi adaptado de um outro, para o ZX80 (2K RACETRACK) publicado originalmente na revista mensal do ZX USER'S CLUB inglês, INTERFACE. A versão original foi escrita por Alan Gunnell.

É fácil jogá-lo, e como termina fornecendo uma pontuação depois de cada "corrida", actua como um desafio, levando-nos a insistir tentando melhorar os resultados obtidos. O programa inclui vários graus de dificuldade.

Ao longo da corrida é-nos pedido que indiquemos a aceleração e a velocidade utilizada. O leitor depressa aprenderá os efeitos que ambas têm. A sua pontuação é indicada continuamente (linha 220), sendo o seu valor final fornecido ao terminar o jogo. Ao longo da corrida são fornecidas indicações ao jogador, por exemplo sobre o comportamento do piloto traseiro. Verificará que existe uma certa tendência para chocar com outros veículos, mas que o seu tende a sobreviver a um número infinito de acidentes... A linha 290 tem um interesse especial, substituindo cinco instruções IF /THEN do tipo IF H= 5 THEN let BS= ...

```
  5 REM CIRCUITO
 10 REM Programa adaptado por
 12 REM Alan Gunnell
 14 REM Originalmente publicado
 16 REM por INTERFACE
```



```
 18 REM
 20 LET a=5: LET g=1: LET b=3
 22 BORDER 1: PAPER 7: INK 0
 25 INPUT "Qual a pista (3 a 5)?
";v
 27 IF v<3 OR v>5 THEN GO TO 25
 30 LET x=0
 40 LET l=100+v*v
 50 LET s=0
 60 IF x=10 THEN STOP
 80 LET x=x+1
 90 IF x=10 THEN PRINT INK RND
* 6; TAB 8; "A corrida terminou"';TAB 4; "Pontuação:";L;
"num total de"; 100 + V * V; POKE 23692,- 1: BORDER RND
* 7: BEEP. 02, RND* 30;GO TO 90

110 GO SUB 180
112 FOR t=1 TO 50: BEEP .02,t:
NEXT t
115 GO SUB 270
120 PRINT INK RND*6;b$
125 GO SUB 145
130 GO SUB 350
135 PAUSE 50
140 GO SUB 270
142 GO TO 60
145 FOR t=1 TO 50: BORDER RND*7
 NEXT t: BORDER 1
149 LET s=ABS (s+(a*a)-(b*15)+(
2*g))
150 PRINT '''' PAPER 2; INK 6;"
Velocidade engrenada ";g;" Velocidade ";s
160 GO SUB b*1000
180 BORDER 1: INPUT INK 7;"Sele
ct a gear (1 TO 10) ";g
190 IF g<1 OR g>10 THEN GO TO 1
80
200 INPUT INK 7;"Aceleração (0 a 10)?";a
210 IF a<1 OR a>10 THEN GO TO 2
00
220 PRINT INK RND * 6; "Pontuação actual; ": INK 2;L
240 INPUT PAPER 2; INK 6; "Travagem
(0 a 10)?";b
250 IF b<0 OR b>10 THEN GO TO 2
40
260 RETURN
270 LET h=INT (RND*v)+1
```

```
  290 LET b$="óleo na estrada" AND
 h=5) + ("pregos" AND          h=4) + ("esqui
na" " AND h=3) + ("curva" " AND h=2)
 + ("recta" " " AND h=1)
  340 RETURN
  350 IF a=0 THEN LET a=1
  360 LET s=ABS (s+(a*a)-(b*15)+(
2*g))
  370 IF s<10 THEN LET s=10
  380 IF s<15 THEN PRINT "    IN
K 2;"O condutor atrás está a apinhar"; PRINT INK 1;
"APRESSE-SE""""
  400 RETURN
1000 IF s>90 THEN PRINT INK 2;"V
ai muito depressa desacelere!": LET l=l-5
1010 RETURN
2000 IF s>40 THEN BEEP 3,50: FOR
 q=1 TO 20: BORDER RND*6: NEXT q
: PRINT INK 2;"Choque"
2010 IF b>8 THEN PRINT INK 2;"Cr
ash": BEEP .5,20: BORDER RND*6:
PAUSE 20: LET l=l-9+INT (RND*10)
2020 RETURN
3000 IF s>25 THEN FOR r=1 TO 10:
 PRINT INK RND*6;"Choque!!!!!!!!!":
NEXT r: LET l=l-10
3010 RETURN
4000 IF s>35 THEN PRINT INK 2;"*
******Crash*******************":
 LET l=l-10
4010 RETURN
5000 IF s>20 THEN PRINT INK 2;
          "CHOOOOOOOOQUEEE!!!!!!": LE
T l=l-10
5005 FOR t=1 TO 50: PAPER RND*7:
 CLS : NEXT t: PAPER 7
5010 IF b>3 THEN PRINT "Crash!!"
: LET l=l-10
5020 RETURN
```

BREAKOUT

Neste jogo, baseado num outro escrito por Eric Thompson, controla-se o movimento da pequena raquette na parte inferior com as teclas "1" e "0". A sua bola é uma letra "o" minúscula.


```
    3 POKE 23609,100
    5 REM BREAKOUT
    6 REM BASEADO EM PROGRAMA ZX81
    7 REM POR ERIC THOMPSON
    9 GO SUB 500
   15 PAPER 6: CLS : BORDER 2
   20 FOR L=0 TO 7
   30 PRINT AT L,8; INK RND*3;"▚▚
▚▚▚▚▚▚▚▚▚▚▚▚▚▚"
   40 NEXT L
   55 LET S=0
   60 LET X=16
   70 LET B=INT (RND*10)+8
   80 LET Q=1-INT (RND*3)
   90 IF Q=0 THEN GO TO 80
  100 FOR P=8 TO -8 STEP -1
  110 IF ABS B>18 THEN LET B=18
  115 LET X=X+(INKEY$="0")-(INKEY
$="1")
  120 PRINT AT 9,X-2; INK 2;"  ▀▀
  ";AT P,B; INK 1;"o"
  140 LET B1=B
  150 IF B=8 OR B=18 THEN LET Q=-
Q: BEEP .005,30
  160 LET B=B+Q
  170 LET X=X+(INKEY$="0")-(INKEY
$="1")
  175 FOR G=1 TO Z: NEXT G
  180 PRINT AT P,B1;" "
  190 NEXT P
  195 LET X=X+(INKEY$="0")-(INKEY
$="1")
  200 IF ABS (B-X)>2 THEN GO TO 2
30
  205 LET S=S+1
  210 PRINT AT 16,8; INK 1;"Your
score is "; INK 2;S: BEEP .25,1
  220 GO TO 60
  230 FOR G=1 TO 400: NEXT G
  240 LET S=0
  250 GO TO 15
  500 INPUT "GRAU DE DIFICULDADE
 (1 TO 20)?";Z
  510 IF Z<1 OR Z>20 THEN GO TO 5
00
  520 LET Z=2*Z
  530 FOR G=1 TO Z
  540 BEEP .02,G
```

```
550 NEXT S
560 RETURN
```

"GALAXIAN"

Este programa foi adaptado por Tim Hartnell de um outro para o ZX81 escrito por James Walsh e Paul Holmes, primeiramente publicado na revista DATABUS.

```
  1 REM      Galxian
    Holmes, Walsh, Hartnell
  4 GO SUB 1000
  5 BORDER 0
  6 PAPER 0: CLS
  7 INK 6
 10 LET X=10
 20 LET S=0
 30 LET P=1
 40 LET S=S+P
 50 CLS
 60 PRINT AT 0,0;S
 70 LET P=0
 80 LET N=10
 90 LET Y=13
100 LET C=0
110 PRINT AT 15,X;"    "
120 IF INKEY$="5" THEN LET X=X-
1
130 IF INKEY$="8" THEN LET X=X+
1
140 PRINT AT 15,X; INK 2;A$
145 BEEP .004,10
150 PRINT AT P,N;" "
160 IF RND>.65 THEN LET N=N+INT
(RND*3-1)
170 IF RND>.8 THEN LET P=P+1
190 PRINT AT P,N; INK 4;"Ω"
200 IF P>14 THEN GO TO 5
210 IF C=0 AND INKEY$="1" THEN
LET C=X+1
220 IF C=0 THEN GO TO 110
230 PRINT AT Y,C;" "
240 LET Y=Y-1
250 IF Y<4 THEN GO TO 90
260 PRINT AT Y,C;"↑": BEEP .009
,50
```



```
270 IF C=N AND Y=P THEN FOR W=1
TO 3: PRINT AT Y,C; INK RND*8;"
*": BEEP .15,15: BORDER RND*7: B
EEP .15,20: BORDER RND*7: BEEP .
15,30: NEXT W: GO TO 40
280 GO TO 110
1000 LET A$="[illegible]"
1005 FOR J=0 TO 7
1010 READ N
1020 POKE USR "A"+J,N
1030 NEXT J
1040 FOR J=0 TO 7
1050 READ N
1060 POKE USR "B"+J,N
1070 NEXT J
1080 FOR J=0 TO 7
1090 READ N
1100 POKE USR "C"+J,N
1110 NEXT J
1120 FOR J=0 TO 7
1130 READ N
1140 POKE USR "M"+J,N
1150 NEXT J
1160 FOR J=0 TO 7
1170 READ N
1180 POKE USR "E"+J,N
1190 NEXT J
2000 DATA BIN 11111110,BIN 11111
100,BIN 11111000,BIN 11110001,BI
N 01100001,BIN 10010001,BIN 1011
1000,BIN 10111100
2010 DATA BIN 01111110,BIN 10111
101,BIN 11000011,BIN 01000010,BI
N 00000000,BIN 01000010,BIN 0011
1100,BIN 00111100
2020 DATA BIN 01111111,BIN 00111
111,BIN 00011111,BIN 10001111,BI
N 10000110,BIN 10000001,BIN 0001
1101,BIN 00111101
2030 DATA BIN 01111110,BIN 10011
001,BIN 11000011,BIN 01100110,BI
N 01100110,BIN 10000001,BIN 1100
0011,BIN 11100111
2040 DATA BIN 11110111,BIN 11100
011,BIN 11010101,BIN 10110110,BI
N 11110111,BIN 11110111,BIN 1110
0011,BIN 11001001
5000 RETURN
```

APÊNDICES

MICRODRIVE

Trata-se de um sistema de memória externa equivalente a um disco em miniatura. Cada Microdrive dispõe de cerca de 85 K de programa e dados, podendo ser ligados simultaneamente 8 ao Spectrum. A velocidade de transferência de informação do Microdrive para o Spectrum é de 16 K por segundo, e o tempo total para varrer toda a cassette a fim de descobrir um dado registo é cerca de sete segundos se bem que muitos tempos de acesso possam ser inferiores a esse. Certas ordens que não foram mencionadas neste livro foram concebidas para uso com o Microdrive e a interface RS232.

INTERFACE RS232

O interface RS232 permite ligar o Spectrum a um qualquer dos periféricos (como impressoras, terminais ou outros computadores) compatíveis com ela. O RS232 é uma norma industrial, pelo que o Spectrum pode ter uma vasta gama de utilidades com este interface. O sistema de utilização do interface encontra-se no monitor do Spectrum.

OUTRAS ORDENS

OPEN#, CLOSE#, MOVE, ERASE, CAT e FORMAT são concebidas para uso com o interface e o Microdrive. O Microdrive aceita não apenas SAVE, VERIFY, LOAD e MERGE, mas também PRINT, LIST, INPUT e INKEYS.

IN e OUT são instruções que afectam os "ports" de entrada e saída do computador, sendo usadas para controlar ou obter informações do teclado, da impressora, etc.

CONVERSÃO BINÁRIO PARA DECIMAL

Talvez o leitor prefira usar números decimais em vez de binários nas suas instruções de criação de gráficos definidos por si. Se assim for, a lista que se segue poderá ser-lhe útil: no caso de estar interessado, o programa usado para a imprimir é indicado no final.

00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
00000101	5
00000110	6
00000111	7
00001000	8
00001001	9
00001010	10
00001011	11
00001100	12
00001101	13
00001110	14
00001111	15
00010000	16
00010001	17
00010010	18
00010011	19
00010100	20
00010101	21
00010110	22
00010111	23
00011000	24
00011001	25
00011010	26
00011011	27
00011100	28
00011101	29
00011110	30
00011111	31
00100000	32
00100001	33
00100010	34
00100011	35
00100100	36
00100101	37
00100110	38
00100111	39
00101000	40
00101001	41
00101010	42
00101011	43
00101100	44
00101101	45
00101110	46
00101111	47
00110000	48

```
00110001
00110010
00110011
00110100
00110101
00110110
00110111
00111000
00111001
00111010
00111011
00111100
00111101
00111110
00111111
01000000
01000001
01000010
01000011
01000100
01000101
01000110
01000111
01001000
01001001
01001010
01001011
01001100
01001101
01001110
01001111
01010000
01010001
01010010
01010011
01010100
01010101
01010110
01010111
01011000
01011001
01011010
01011011
01011100
01011101
01011110
01011111
01100000
01100001
```

```
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```



```
01100010
01100011
01100100
01100101
01100110
01100111
01101000
01101001
01101010
01101011
01101100
01101101
01101110
01101111
01110000
01110001
01110010
01110011
01110100
01110101
01110110
01110111
01111000
01111001
01111010
01111011
01111100
01111101
01111110
01111111
10000000
10000001
10000010
10000011
10000100
10000101
10000110
10000111
10001000
10001001
10001010
10001011
10001100
10001101
10001110
10001111
10010000
10010001
```

```
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
```

```
10010010
10010011
10010100
10010101
10010110
10010111
10011000
10011001
10011010
10011011
10011100
10011101
10011110
10011111
10100000
10100001
10100010
10100011
10100100
10100101
10100110
10100111
10101000
10101001
10101010
10101011
10101100
10101101
10101110
10101111
10110000
10110001
10110010
10110011
10110100
10110101
10110110
10110111
10111000
10111001
10111010
10111011
10111100
10111101
10111110
10111111
11000000
11000001
11000010
```

```
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
```



```
11000011
11000100
11000101
11000110
11000111
11001000
11001001
11001010
11001011
11001100
11001101
11001110
11001111
11010000
11010001
11010010
11010011
11010100
11010101
11010110
11010111
11011000
11011001
11011010
11011011
11011100
11011101
11011110
11011111
11100000
11100001
11100010
11100011
11100100
11100101
11100110
11100111
11101000
11101001
11101010
11101011
11101100
11101101
11101110
11101111
11110000
11110001
11110010
```

```
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
```

```
11110011          243
11110100          244
11110101          245
11110110          246
11110111          247
11111000          248
11111001          249
11111010          250
11111011          251
11111100          252
11111101          253
11111110          254
11111111          255
```

```
 10 FOR B=0 TO 255
 20 LET J=B
 30 LET A$=""
 40 FOR N=0 TO 7
 50 LET T=J-INT (J/2)*2
 60 IF T=0 THEN LET A$="0"+A$
 70 IF T<>0 THEN LET A$="1"+A$
 80 LET J=INT (J/2)
 90 NEXT N
100 PRINT A$,B
110 NEXT B
```

MENSAGENS DE ERRO

Estas mensagens são impressas na parte inferior do visor sempre que o computador pára a execução de algum programa BASIC, servindo para explicar a razão dessa paragem.

Em geral CONTINUE reenvia para a linha e a instrução especificadas na última mensagem, mas existem excepções a isto no caso das mensagens 0, 9 e D.

Significado das mensagens

0 OK
Cumprimento com êxito de uma ordem, ou salto para um número de linha superior a qualquer existente.

1 NEXT without FOR
O programa encontra uma instrução NEXT sem antes ter encontrado a instrução FOR correspondente.

2 Variable not found
A variável usada na instrução não foi definida anteriormente.

3 Subscript wrong
Os parâmetros de uma variável de quadro ultrapassam as dimensões desse quadro, ou possui um número errado de parâmetros.

4 Out of memory
O programa requer uma quantidade de memória superior à disponível para uso em BASIC.

5 Out of screen
Uma instrução INPUT tentou produzir mais de 23 linhas no visor. Ocorre igualmente devido à instrução PRINT AT 22...

6 Number: too big
Os cálculos realizados conduziram a um resultado superior a cerca de 10^{38}.

7 RETURN without GO SUB
O programa encontra uma instrução RETURN sem ter passado pela GO SUB correspondente.

8 End of File (operações com Microdrive, etc)
Final do ficheiro.

9 STOP statement

A Invalid argument
Argumento formulado incorrectamente

B Integer out of range
Número inteiro fora da gama aceite como válida.

C Nonsense in BASIC
Ocorre numa variedade de situações. Indica erros de lógica entre instruções.

D BRAEK — CONT repets
BREAK pára a execução: se não forem cumpridas outras ordens entretanto, a tecla CONTINUE permite voltar à instrução onde o programa parou.

E Out of DATA
Paragem do programa por existência de uma instrução READ à qual não correspondem instruções DATA, ou quando estas não foram revalidadas por uma instrução RESTORE.

F Invalid File name
Mensagem apresentada quando o nome dado a um programa ou ficheiro em instruções SAVE / LOAD não é correcto.

G No room for line
Não existe espaço para nova linha.

H STOP in INPUT
Cumprimento de ordem de paragem quando a máquina espera dados.

I FOR without NEXT
Existência de um ciclo FOR que não deve ser executado nenhuma vez e a máquina não encontra a NEXT correspondente.

J Invalid I / O device (operações com microdrive, etc).

K Invalid colour
Número incorrecto na especificação da cor. Ocorre também numa variedade de situações ao encontrar códigos de controlo de cor ou outros atributos.

L BREAK into program
Detecção entre duas instruções de tecla BREAK premida.

M RAMTOP no good
O valor indicado RAMTOP (final da área BASIC) é demasiado grande ou pequeno.

N Statement lost
Salto para uma instrução que já não existe.

O Invalid stream (operações com microdrive, etc.)

P FN without DEF
A máquina não encontra em nenhum ponto do programa a definição da função do utilizador cuja execução é pedida.

Parameter error
Número de argumentos errados, ou um deles de tipo incorrecto (cadeia em vez de número, etc.)

R Tape loading error
Impossibilidade de ler programa ou ficheiro em dia.

ÍNDICE

Este livro acabou de se imprimir
em 1984
para a
EDITORIAL PRESENÇA, LDA.
na
Empresa Gráfica Feirense, Lda.
Vila da Feira