Este livro destina-se a todos os utilizadores do Spectrum, quer sejam principiantes quer já programadores competentes.

Complementa o manual da Sinclair, explicando as características principais do BASIC do Spectrum e mostrando como podemos utilizá-las para criar programas de jogos ou programas de aplicação autênticos.

Incluem-se neste livro mais de 50 programas completos, acompanhados de explicações minuciosas, além de numerosas pequenas rotinas e três apêndices informativos

Quer os interesses do leitor se situem na aprendizagem do BASIC quer em programas de jogos ou de aplicação, decerto encontrará nestas páginas motivos de interesse e informações valiosas.

Todos os programas deste livro foram verificados e testados pelo Gabinete Verbo de Informática.

BIBLIOTECA VERBO DE INFORMÁTICA

2

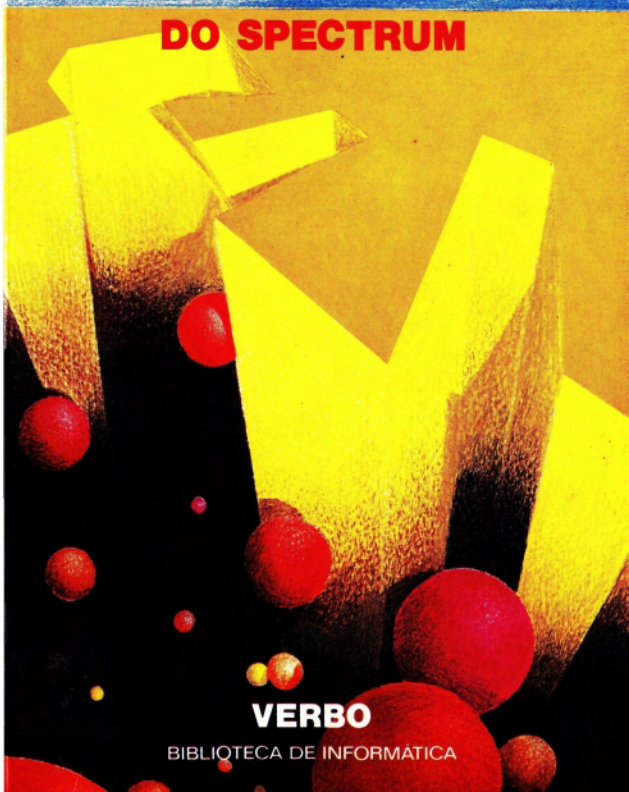
APROFUNDAR O BASIC

MIKE LORD

DO SPECTRUM

APROFUNDAR O BASIC M. LORD

Verbo



VERBO

BIBLIOTECA DE INFORMÁTICA

Aprofundar o BASIC do Spectrum

MIKE LORD

Aprofundar o BASIC do Spectrum

2.ª edição, revista

Verbo

Título original: *Exploring Spectrum Basic*
Tradução de Carlos Sebastião e Silva
Capa de Luís Anglin

Editorial VERBO. Lisboa / São Paulo
N.º Ed. 1595
Composto por Fotocompográfica, Lda.
Impresso por Empresa Litográfica do Sul
— Vila Real de St.º António,
em Março de 1985
Depósito legal 8043/85

ÍNDICE

Introdução

Bem-vindo, amigo explorador de microinformática, ao mundo do *Spectrum*.

E que mundo fascinante ele é! Um mundo sobre o qual terá controle absoluto. Onde a sua palavra é lei e as suas ordens serão executadas à letra — mesmo que, na realidade, pretenda que o computador faça uma coisa completamente diferente!

Este livro debruça-se sobre a linguagem de programação BASIC tal como ela é utilizada pelo *Sinclair Spectrum*. Foi escrito para complementar os excelentes manuais fornecidos com o *Spectrum*, nomeadamente para:

— Sublinhar alguns conceitos fundamentais da programação em BASIC que — na experiência do autor — os principiantes têm maior dificuldade em apreender.
— Explorar as características principais do BASIC do *Spectrum*, tais como os gráficos de alta definição, som e cor, em maior profundidade do que aquela que é possível nos manuais do *Spectrum*.
— Demonstrar como se pode usar o *Spectrum* para jogos, negócios e outras aplicações.
— Fornecer uma série de programas úteis para ilustrar estes aspectos.
— Constituir uma fonte de ideias para os seus próprios programas.

Quer os seus interesses se situem nos programas de jogos ou nas aplicações pessoais, de gestão ou de engenharia do *Spectrum*, certamente encontrará aqui qualquer coisa para si.

Mas esperamos que o que de mais valioso vai encontrar nestas páginas seja o incentivo para escrever os seus próprios programas e continuar as suas próprias investigações nos mistérios desta máquina maravilhosa.

Mike Lord

Algumas notas acerca dos programas deste livro:

— todos eles podem ser executados num *Spectrum* de 16 K, embora alguns se tornem mais úteis num de 48 K;
— o algarismo zero escreve-se ∅, de forma a distingui-lo da letra O nas listagens dos programas;
— é preciso cuidado em fazer a distinção entre o **algarismo** 1 e a **letra minúscula** l

Em caso de dúvida, repare-se que o algarismo "1" aparece como parte de, pelo menos, um número de linha em todas as listagens.

— Recordemos também que a introdução de cada um dos símbolos relacionais <=, >= e <> é igualmente efectuada por meio de uma única tecla.
— Os caracteres gráficos usuais *(standard)* foram desenhados tal como aparecem no *Spectrum* e, deste modo, ▣ representa o carácter gráfico na tecla "6". Onde houver dificuldade em reconhecer as formas ou onde foram utilizados caracteres gráficos definidos pelo utilizador, as notas que acompanham a listagem do programa indicarão quais as teclas a utilizar.
— O *Spectrum* ignora os espaços em branco, excepto se eles se encontrarem entre aspas; nos casos em que o número de espaços é importante, utilizámos o símbolo "•" para representar cada espaço.
— Para facilitar a sua compreensão, as listagens dos programas (quando não foram impressas pelo próprio *Spectrum*) foram impressas utilizando linhas com um número de caracteres superior aos 32 da impressora do ZX, e portanto,

```
9999 FOR d=1 TO 365: PRINT "Aman
ha e' ";: NEXT d
```

aparecerá neste livro na seguinte forma:

```
9999 FOR d=1 TO 365: PRINT "Amanha e';: NEXT d
```

O teclado do *Spectrum* não dispõe de acentos ou cedilhas, e por isso teremos de passar sem elas, a não ser que se definam esses símbolos em caracteres definíveis pelo utilizador, de que o *Spectrum* dispõe, mas isso complica bastante as listagens dos programas, o que é desaconselhável. Quando o acento é no fim de uma palavra ou sobre uma letra isolada (p. ex.: *é*), utilizou-se o símbolo de (') da tecla 7 em substituição do acento, imprimindo-o na posição imediatamente a seguir à do carácter em questão (p. ex.: e').

Capítulo 1

Início da viagem...

Aprender a trabalhar com um computador é um pouco como aprender a orientação numa cidade desconhecida. O que interessa, de início, são as avenidas e as ruas principais. Mais tarde, porém, exploram-se as ruas secundárias e outras áreas da cidade. Encontraremos certamente muitas coisas interessantes e provavelmente descobriremos alguns atalhos bastante úteis.

É essencial possuir um mapa ou um guia da cidade se não queremos perder-nos com frequência ou enfiar por alguns becos sem saída. Além disso, o mapa, muitas vezes, mostra aspectos que, sem a sua indicação, nos passariam despercebidos.

Mas a prática e o prazer da exploração residem no facto de as viagens serem orientadas por nós próprios.

UM "COMPUTADOR"?

A palavra "computador" aplica-se a uma vasta gama de dispositivos, desde o simples ábaco ou a máquina de somar mecânica ao paranóico computador *Hal* do filme *2001 — Odisseia do Espaço*. Todos eles têm em comum serem máquinas destinadas a manipular informação *(data)*. Tal como uma estância de madeiras "manipula" madeira — seleccionando-a, armazenando-a e até, por vezes, alterando-lhe as dimensões —, assim um computador manipula informação; seja ela o preço do arroz, uma lista de números de matrícula de carros roubados ou a probabilidade de um membro da tripulação de um submarino ser um agente inimigo.

Podemos definir o *Spectrum* de uma forma mais precisa dizendo que é um "computador electrónico, digital, com capacidade para armazenar programas". Isto diz-nos que o *Spectrum* é um dispositivo electrónico (e não mecânico ou biológico), que manipula informação sob a forma de números. Os computadores "analógicos" — em contraste — trabalham directamente com os valores das variáveis e não com os números que representam esses valores. Mas o *Spectrum* é um computador digital e, apesar de aceitar e reproduzir letras do alfabeto e outros símbolos, estes são todos representados internamente por números.

Ao descrever o *Spectrum* como um "computador com a capacidade para armazenar programas" *(stored program computer)*, estamos a focar um ponto muito importante. O "programa" é, obviamente, a lista das instruções que damos ao computador para que ele execute aquilo que queremos. No entanto, o mais importante é o *Spectrum* poder armazenar (ou "recordar") uma lista de instruções e, em seguida, executá-las.

Compare-se isto com o que uma calculadora vulgar pode fazer. É claro que podemos escrever um "programa" destinado a resolver um problema matemático numa calculadora mas, nesse caso, teríamos de introduzir as instruções na máquina uma a uma. De toda a vez que introduzíssemos uma instrução, a calculadora executá-la-ia e, depois, não faria mais nada até intruduzirmos o passo seguinte. Esta é, obviamente, uma forma muito lenta de resolver um problema. Se, pelo contrário, a lista das instruções estiver armazenada na memória do computador, este passa à fase seguinte logo que tiver terminado a execução da anterior.

Pelo facto de o programa ficar contido no computador, este pode repetir uma sequência de instruções muito rapidamente, enquanto, se utilizássemos uma calculadora, teríamos de ser nós próprios a introduzir repetidamente a sequência de instruções. Esta é uma característica muito poderosa do computador, e verificaremos que virtualmente todos os programas úteis se baseiam em ciclos *(loops)* de instruções que se repetem muitas vezes durante a execução do programa.

Do mesmo modo, um computador permite-nos fazer reexecutar um programa na sua totalidade sem termos de o introduzir novamente, e mesmo "guardar" *(save)* um programa num suporte de armazenamento "permanente" de informação tal como a *cassette* ou o disco magnético, de forma a ser, posteriormente, executado de novo, bastando para isso carregá-lo *(load)* de novo na memória do computador, em directo da *cassette* ou do disco.

Finalmente, há uma enorme vantagem em ter o programa armazenado em memória: a possibilidade de lhe efectuar correcções e alterações sem termos de o reescrever na totalidade. Por exemplo, se desejarmos modificar uma determinada instrução num programa do *Spectrum*, basta-nos escrever a nova "versão" dessa instrução. Essa nova "versão" substituirá automaticamente a "versão" anterior na memória do computador, sem afectar

qualquer das outras instruções do programa. Esta possibilidade, conhecida por *editing* (edição), é um factor essencial para o desenvolvimento de qualquer programa, à excepção dos muito curtos.

QUE MEMÓRIA?

Como se diz no manual de programação em BASIC, o *Spectrum* possui dois tipos de memória: ROM e RAM. Para compreender melhor as funções destas memórias, devemos dar uma olhadela rápida ao "coração" da máquina: o microprocessador *Z80*.

Não se deve confundir o termo "microcomputador", que designa um computador do tipo genérico do *Spectrum*, com o termo "microprocessador", que designa o "cérebro" do computador, ou seja, a unidade central de processamento (UCP), contida numa minúscula "pastilha" *(chip)* de silício.

O microprocessador *Z80* é um circuito integrado extremamente complexo, que efectua todos os cálculos necessários quando o *Spectrum* está a funcionar. Apesar de ser um calculador muitíssimo rápido — com capacidade para executar quase um milhão de instruções por segundo —, tem limitações em vários aspectos importantes. Por exemplo, é incapaz de compreender o BASIC por si próprio. Na verdade compreende apenas instruções escritas na linguagem máquina *Z80*, extremamente minuciosa.

A ROM *(Read Only Memory,* memória unicamente de leitura) contém muitas rotinas especiais escritas nesta linguagem máquina. Sempre que o *Spectrum* está ligado, o microprocessador *Z80* está a executar uma ou outra dessas rotinas, que servem nomeadamente para:

— Introduzir ou modificar e corrigir os programas.
— Executar o programa em BASIC. A ROM contém conjuntos de "regras" que o *Z80* utiliza para interpretar cada comando ou função BASIC de modo a executar aquilo que foi escrito no programa.
— Verificar se foi pressionada alguma tecla.
— Imprimir resultados, listagens, gráficos, etc., no *écran* de televisão ou por meio da impressora *ZX*.
— Passar os nossos programas para uma *cassette* ou voltar a carregá-los na memória do *Spectrum*.

A ROM contém, igualmente, as matrizes de pontos utilizadas para formar os caracteres no *écran* ou na impressora.

Toda esta informação é gravada no circuito integrado da ROM, no momento do fabrico. O utilizador lê com facilidade a informação contida na ROM, mas não consegue alterar nenhuma parte dela.

A RAM *(Random Access Memory* — memória de acesso aleatório, ou directo), por seu lado, é muito mais flexível. Além de ler o conteúdo da RAM, o utilizador (ou o microprocessador *Z80*) pode alterar o que nela está gravado, escrevendo no seu lugar nova informação. A grande desvantagem da RAM consiste em ela ser uma memória "volátil", o que significa que o seu conteúdo é "apagado" caso se verifique uma interrupção na alimentação de corrente ou, simplesmente, quando desligamos o *Spectrum*.

Como se descreve no Capítulo 24 do manual do *Spectrum*, a RAM está dividida em várias áreas. As mais importantes são as seguintes:

— o ficheiro de imagem *(display file)* e a área dos atributos. O *Spectrum* dispõe de *hardware* especial que efectua uma leitura contínua destas áreas de forma a produzir o sinal que envia ao *écran*. O ficheiro de imagem contém uma imagem dos caracteres e de outras formas que aparecem no *écran*, enquanto a área dos "atributos" contém informação acerca das cores e da luminosidade e define se alguma parte da imagem cintila ou não *(flash)*;
— a área das "variáveis de sistema" e as várias "pilhas" de elementos de memória *(stacks)* utilizadas pelo *Spectrum* para conter notas genéricas acerca do que o computador está a fazer em cada momento;
— a área de programas BASIC, onde o programa do utilizador é armazenado;
— a área das "variáveis", utilizada para armazenar as variáveis a empregar no programa do utilizador.

BITS, BYTES E K

Os recém-chegados ao mundo dos computadores ficam por vezes um pouco intrigados com a forma de medição da capacidade da

memória; todo este palavreado acerca de *bits*, de *bytes* e de *K* pode provocar inicialmente certa confusão. Contudo, a explicação é, na verdade, bastante simples e deriva do facto de a memória ser organizada de forma a compatibilizar-se com o microprocessador, o qual — sendo um dispositivo binário — dá o seu melhor rendimento quando processa números com potências de 2.

Além disso, o microprocessador necessita que a memória seja organizada de um modo uniforme. Podemos comparar a memória do computador a uma fila muito longa de caixas idênticas. As caixas são numeradas consecutivamente, de modo a que o processador utilize este número — ou "endereço" — para seleccionar uma determinada caixa.

Em cada posição ou célula de memória (ou seja, em cada "caixa") existe um número binário, constituído por 8 *"bits"* ou "dígitos binários" *(BInary digiTS)*, que tomam os valores "0" ou "1". Ao grupo de 8 *bits* dá-se o nome de *byte*.

Existem 256 (ou seja, 2^8) combinações possíveis de oito zeros e uns, de modo que podemos considerar que cada célula de memória contém um número decimal inteiro entre 0 e 255.

É claro que o *Spectrum* opera com números fora deste intervalo, assim como com fracções e, para o fazer, utiliza um grupo de cinco *bytes*, como se descreve no Capítulo 26 do manual do *Spectrum*.

Ao todo, o microprocessador *Z80* do *Spectrum* controla 65536 (2^{16}) células de memória diferentes. O primeiro quarto destas células é ocupado pela ROM e — segundo o modelo — as restantes são, parcial ou totalmente, preenchidas com a RAM.

Uma forma abreviada e cómoda de referir a capacidade da memória consiste em utilizar a abreviatura "K" para designar 1024 *bytes* ($1024=2^{10}$). Assim, dizemos que o *Spectrum* pode controlar 64 K *bytes* de memória, dos quais 16 K são ocupados pela ROM e um máximo de 48 K pela RAM.

BASIC

BASIC é um acrónimo de *Beginners All Purpose Symbolic Instruction Code*. É uma linguagem de alto nível desenvolvida na década de sessenta em Dartmouth College (E. U. A.), com a finalidade de proporcionar aos principiantes um meio relativamente simples de aprenderem programação. Existem outras

linguagens de programação, mas são, geralmente, demasiado incómodas ou demasiado complexas para o principiante. Eis algumas das mais popularizadas:

FORTRAN	Uma das principais linguagens evoluídas, muito eficiente em problemas matemáticos e ainda muito utilizada para escrever programas científicos ou de engenharia.
COBOL	Uma linguagem antiga mas ainda muito poderosa, destinada a aplicações correntes de gestão e negócios. Necessita de um grande computador.
Pascal	Linguagem semelhante ao *Algol*, que lhe é anterior. Começou por ser uma linguagem "académica", mas tem ganho rapidamente popularidade em aplicações importantes nos microcomputadores. É uma linguagem formal e estruturada, muito eficiente no tratamento de diversos tipos de informação.
LISP	Uma linguagem estranha, criada e desenvolvida com a finalidade de auxiliar as pesquisas na área da inteligência artificial. É, na verdade, uma linguagem de tratamento de cadeias *(strings)*, e diz-se que modifica a visão da realidade de quem a aprender.
FORTH	Muito apreciada por uns e simplesmente odiosa para os outros. As suas características principais são uma total incompreensibilidade para o programador que a ela não esteja habituado e uma tremenda capacidade de definir novos comandos.

Como todas as linguagens de computador evoluídas, o BASIC situa-se a meio caminho entre a linguagem natural humana e o mundo muito restrito do código máquina do processador. A tradução entre o programa do utilizador em BASIC e o código máquina é feita pelas rotinas do intérprete de BASIC residentes na ROM do *Spectrum;* a tradução entre o BASIC e a linguagem corrente do utilizador fica, evidentemente, a cargo deste último.

O BASIC está actualmente disponível em quase todos os

computadores e é, provavelmente, a linguagem de programação mais utilizada. Mas — curiosamente — todas as tentativas para a normalizar falharam. Parece que todos os computadores têm a sua própria versão de BASIC, com "melhoramentos" que são por vezes úteis e que são sempre largamente propagandeados pelos departamentos de *marketing*.

O BASIC do *Spectrum* é uma típica versão moderna da linguagem. Baseia-se num vocabulário de cerca de 90 palavras, as quais são utilizadas para formular "declarações" *(statements)*. Cada "declaração" constitui uma instrução para o computador efectuar uma determinada tarefa: um programa em BASIC consiste numa lista destas "declarações".

Capítulo 2

Impressão e programas

PRINT é a palavra mais importante em BASIC, e é um ponto de partida ideal para o principiante porque lhe mostra imediatamente aquilo que ele fez.

Ao desligar o *Spectrum* por uns segundos e ao voltar a ligá-lo, a frase

© 1982 Sinclair Research Ltd

aparece na parte inferior do *écran*. Vamos agora escrever o seguinte:

PRINT 123

(A palavra de comando PRINT é introduzida carregando unicamente na tecla P.) Nesta altura, aquilo que escrevemos deve ter aparecido na parte inferior do *écran*, seguido de um "cursor" intermitente. Enquanto o que introduzimos é apresentado na parte inferior do *écran*, o *Spectrum* está a funcionar no modo EDIT esperando a verificação de que está correcta essa informação.

Quando ficarmos satisfeitos com o que se apresenta no *écran*, carreguemos na tecla ENTER, levando o *Spectrum* a actuar sobre o que aparece na parte inferior do *écran*.

Se tudo estiver correcto, a imagem mudará para apresentar o número

123

no canto superior esquerdo do *écran*, exactamente aquilo que lhe pedimos para imprimir, e a mensagem em código

Ø OK, Ø:1

deve aparecer no canto inferior esquerdo. Este é o "relatório" do *Spectrum*, dizendo que executou com êxito o que lhe havia sido pedido e que não encontrou nada incorrecto.

Agora, antes de fazer qualquer outra coisa, damos entrada a outra declaração *(statement)* PRINT,

PRINT 456

e carregamos novamente na tecla ENTER, logo que tivermos verificado o que escrevemos. Se tudo estiver correcto, o número "456" deverá aparecer no *écran*, logo abaixo de "123".

Isto mostra-nos que tudo o que se tenha imprimido (PRINTed) no *écran* continuará lá, mesmo quando executado outro comando PRINT. A forma mais simples de "limpar" o *écran* consiste, unicamente, em voltar a pressionar a tecla ENTER.

Somos também levados a pensar que cada comando PRINT utiliza uma nova linha no *écran*. De facto, isto não é inteiramente verdade; a posição em que se imprime qualquer coisa é, na realidade, determinada de uma forma muito mais flexível, seguindo sempre o *Spectrum* a "pista" da "posição de impressão" *(print position)* corrente.

Esta "posição de impressão" é o local do *écran* onde será impresso o **próximo** carácter. Quando a metade superior do *écran* estiver "limpa", a posição de impressão é colocada no canto superior esquerdo, de modo a que o primeiro carácter a imprimir aí apareça.

Declarações PRINT simples, como as utilizadas nos dois exemplos anteriores, deslocam a posição de impressão para o início da linha seguinte logo que tenha sido efectuada toda a impressão necessária. Deste modo, a declaração PRINT seguinte irá começar a imprimir nessa linha.

;

Contudo, podemos evitar esta mudança de linha automática, alterando ligeiramente a declaração PRINT. Vamos, para exemplo, dar entrada aos dois comandos seguintes:

PRINT 12;
PRINT 34

Desta vez, o *écran* deve apresentar o resultado

1234

impresso no canto superior esquerdo. O ponto e vírgula (;) que aparece logo a seguir à primeira declaração PRINT fez com que o *Spectrum* deixasse a posição de impressão no lugar dele (ou seja, logo a seguir ao "2"), em vez de a deslocar para o início da linha seguinte. Faremos isto todas as vezes que o desejarmos e, para o verificar, vamos experimentar dar entrada aos quatro comandos seguintes:

```
PRINT Ø;
PRINT 12;
PRINT 34;
PRINT 5
```

Note-se que o ponto e vírgula no fim de cada declaração PRINT afecta a posição utilizada pela declaração PRINT seguinte.

,

Se, em vez de um ponto e vírgula, colocássemos uma vírgula (,) no fim de uma declaração PRINT, o resultado seria a deslocação da posição de impressão para:

— o meio da linha, se o último carácter impresso estivesse antes do meio da linha;
— o início da linha seguinte, se o último carácter impresso estivesse sobre o meio da linha ou depois dele.

Isto permite-nos imprimir em duas colunas bem ordenadas. Como exemplo, experimentemos dar entrada aos seguintes quatro comandos:

```
PRINT Ø,
PRINT 12,
PRINT 34,
PRINT 5
```


SEPARADORES

Podemos colocar mais do que um número a seguir à palavra de comando PRINT, desde que separemos os números por meio de uma vírgula ou de um ponto e vírgula. Relembremos que, e também neste caso, a vírgula significa "deslocar a posição de impressão para o meio da linha ou para o início da linha seguinte", como verificaremos introduzindo o seguinte:

```
PRINT 1,23,456,7
```

Relembremos igualmente que o ponto e vírgula significa que a posição de impressão não será deslocada. Experimente-se dar entrada a

```
PRINT 1;23;456;7
```

o que parece inútil, pois se faria o mesmo com

```
PRINT 1234567
```

mas, como veremos mais tarde, o ponto e vírgula é um separador extremamente útil quando queremos imprimir coisas mais complicadas do que simples números.

Um terceiro tipo de separador é o símbolo ' (impresso a vermelho na tecla "7") que desloca a posição de impressão para o início da linha seguinte. Experimente-se introduzir

```
PRINT 1'23''456'7
```

Observe-se que o *Spectrum* ignora uma única ' no fim de uma declaração PRINT.

TAB

Esta palavra de comando permite imprimir em qualquer ponto da linha.

De modo geral, TAB n desloca a posição de impressão ao longo da linha até à coluna n (as colunas de impressão são numeradas, da esquerda para a direita, desde Ø até 31). Se a

posição de impressão já ultrapassou a coluna n, será deslocada para a coluna n da linha seguinte.

Se n for superior a 31, o *Spectrum* subtrairá, repetidamente, 32 de n até o resultado se encontrar no intervalo Ø-31, antes de efectuar a TABulação (TAB vem de *tabular*, em inglês).

A ordem TAB n tem de ser incluída numa declaração PRINT e de ser seguida por um separador. O ponto e vírgula é geralmente o único separador que faz sentido utilizar, pois qualquer outro destruiria o efeito da utilização de TAB. Como exemplo, experimente-se o seguinte:

```
PRINT 1;TAB 2;2;TAB 1;3
```

AT

A palavra de comando TAB permite apenas deslocar a posição de impressão ao longo de uma linha — e, talvez, até à linha seguinte. Contudo, a ordem AT y,x é muito mais poderosa, pois desloca a posição de impressão para qualquer coluna (x) de qualquer linha (y). Os valores x e y devem ser separados por uma vírgula.

Como acontece com TAB, as colunas são numeradas de Ø a 31.

Apesar de o *Spectrum* poder apresentar 24 linhas de caracteres no *écran* as duas linhas do fundo são normalmente reservadas para mensagens do computador ou para entradas do utilizador. Por esse motivo, só podemos imprimir (PRINT) nas primeiras 22 linhas, que são numeradas desde Ø (linha do topo) a 21.

Como também acontece com TAB, a ordem (ou comando) AT y,x tem de ser incluída numa declaração PRINT, e seguida de um ponto e vírgula. Experimente-se dar entrada a

```
PRINT AT 1Ø,Ø;1Ø;AT 9,1;9
```

e a qualquer outra combinação sugerida pela nossa imaginação.

Podemos juntar quaisquer combinações de números, separadores, ordens TAB e AT numa única declaração PRINT; as únicas regras a observar são a necessidade de intercalar um separador entre cada par de elementos e a praticabilidade dos

valores associados a AT e TAB. Como exemplo, depois de "limpar" o *écran* (carregando em ENTER) podemos introduzir a declaração:

```
PRINT 1;AT 21,31;2;AT 1Ø,15;3;TAB 31;4,5
```

EDIÇÃO SIMPLES

O exemplo anterior apresentava uma linha bastante complicada, e é possível que, ao carregar em ENTER, o *Spectrum* nada faça além de colocar um quadrado negro intermitente, contendo um ponto de interrogação algures na linha. Isto significa que o *Spectrum* detectou algum erro na sintaxe (na gramática) da linha. Ele coloca o ponto de interrogação no local da linha onde crê existir o erro; contudo, o *Spectrum* nem sempre tem razão quanto a isto e, portanto, convém verificar a linha toda.

Se tivermos sido muito cuidadosos e, por conseguinte, o *Spectrum* tiver aceite a linha, experimentemos dar entrada ao seguinte:

```
PRINT AT 2;3;4
```

Ora este comando está manifestamente incorrecto!

Tendo detectado onde se situa o erro (no exemplo acima, deveria existir uma vírgula, e não um ponto e vírgula, a seguir ao "2"), o passo seguinte consiste em corrigi-lo.

É aqui que entra em cena o cursor intermitente "L". Qualquer carácter que o utilizador escreva será — como certamente já se observou — inserido logo à esquerda do cursor "L". Além disso, carregando simultaneamente nas teclas CAPS SHIFT e DELETE, o carácter ou a palavra de comando BASIC que esteja logo à esquerda do cursor "L" será "apagada" *(delected)* da linha. Deste modo, começando no extremo direito da linha, podem-se apagar os caracteres um a um, até eliminar o que se deseja e, seguidamente, escrever de novo o resto da linha.

Existe, contudo, um processo mais simples, que consiste em utilizar as funções de movimentação do cursor, oferecidas pelas teclas "5" e "8".

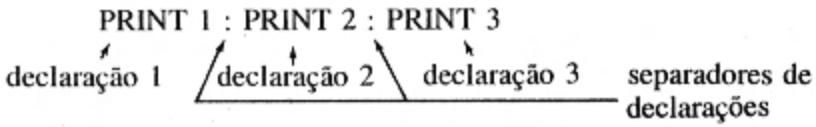
Carregando simultaneamente em CAPS SHIFT e na tecla "5" (◊), desloca-se o cursor "L" uma posição para a esquerda. Analogamente, pressionando CAPS SHIFT e a tecla "8" (◊), desloca-se o cursor uma posição para a direita. Deste modo, posiciona-se o cursor L imediatamente à direita do carácter (ou da palavra de comando) que se quer alterar (ou — no caso de este ter sido omitido — à direita da posição em que ele se devia situar), sem haver necessidade de apagar os restantes caracteres da linha à sua direita.

Note-se que o cursor "L" não tem de estar necessariamente no extremo da linha quando se carrega em ENTER.

Se a linha contiver muitos erros, carregue-se simultaneamente em CAPS SHIFT e EDIT (tecla "1") e a linha será totalmente apagada.

DECLARAÇÕES MÚLTIPLAS

Uma página ou duas atrás, usámos quatro declarações PRINT dando entrada a quatro linhas separadas. Em vez disso, podíamos ter dado entrada a todas elas, utilizando uma única linha e separando as declarações por dois pontos (:).



O *Spectrum* executará, primeiramente, a declaração inicial da linha, depois a segunda, e assim por diante até ao fim da linha.

Uma declaração bastante útil é a palavra de comando CLS (*Clear Screen*), a qual "limpa" o *écran* e desloca a posição de impressão para o canto superior esquerdo. Podemos combiná-la com uma declaração PRINT:

```
CLS : PRINT 123
```

Se, pelo contrário, tivermos dado entrada a

```
PRINT 123: CLS
```


é pouco provável que haja tempo para ver o número "123" aparecer no *écran* antes de ser apagado.

As declarações múltiplas levantam a questão "Qual o comprimento de uma linha?".

No que respeita à imagem no *écran*, cada linha está, obviamente, limitada a 32 caracteres.

No entanto, uma linha de comando (ou de programa) pode ter qualquer comprimento. De facto, ela estende-se desde o seu início até ao ponto em que se carrega na tecla ENTER, ainda que apareça no *écran* como várias linhas. Assim, é possível inserir um número ilimitado de declarações numa única linha de comando; tendo uma linha longa, a única desvantagem é ser mais difícil verificar se toda ela está correcta.

SOBREPONDO...

Ao imprimir mais do que um elemento, levanta-se a questão de um elemento que esteja a ser impresso no *écran* afectar ou não a imagem de um elemento previamente impresso. Para exemplificar o caso de a impressão de um elemento se sobrepor à de outro, experimentemos:

```
PRINT AT Ø,Ø;1234;AT Ø,Ø;Ø
```

donde resulta a apresentação, no *écran*, de

```
Ø234
```

e isto porque, apesar de o segundo elemento a ser impresso (Ø) se ter sobreposto e encoberto o primeiro carácter (1) do primeiro elemento, não afectou os outros caracteres.

No entanto, se colocarmos uma vírgula no fim da declaração, o resultado final será:

```
Ø
```

e isto porque a vírgula desloca a posição de impressão para o meio da linha e, ao fazê-lo, imprime espaços em branco. Isto fica

mais bem exemplificado introduzindo as duas linhas seguintes:

```
PRINT AT Ø,13;123456
PRINT AT Ø,13;Ø,
```

Depois de a segunda linha ter sido introduzida, o resultado, no *écran*, será:

```
Ø  456
```

Isto indica-nos uma forma eficiente de apagar linhas inteiras ou meias linhas. Por exemplo, a declaração

```
PRINT AT 16,Ø,,
```

apaga toda a linha 16 (notem-se as duas vírgulas no fim da declaração).

A ordem TAB faz uma coisa semelhante: imprime espaços em branco, à medida que desloca a posição de impressão, tal como se verifica introduzindo

```
PRINT AT Ø,Ø;123456
PRINT AT Ø,1;TAB 3;Ø
```

o que resultará na apresentação de

```
1  Ø56
```

Isto pode-se utilizar para apagar parte de uma linha. Por exemplo:

```
PRINT AT 16,1Ø;TAB 14;
```

apaga as colunas 10 e 13 da linha 16.

Repare-se que AT e o separador de "nova linha" (') não afectam em nada o que é apresentado no *écran*, excepto deslocarem a posição de impressão. O mesmo acontece com a mudança automática de linha, provocada por uma declaração PRINT que não termine com uma vírgula ou com um ponto e vírgula.

UM "PROGRAMA"

Até aqui, todas as vezes que pressionámos a tecla ENTER, o *Spectrum* executou imediatamente o comando ou comandos que nós tivéssemos escrito e a linha de comando desapareceu. Mas não dissemos nós que o *Spectrum* "armazenou" o nosso programa, de forma a que o possamos fazer executar novamente?

Assim é, mas aquilo que temos introduzido até agora não são linhas de programa. São, em vez disso, o que se chama "linhas de comando directo ou **imediato**", as quais são simplesmente executadas e seguidamente esquecidas pelo computador.

A diferença entre uma linha de comando directo e uma linha de programa está em que uma linha de programa começa com um **número de linha.** Este admite qualquer número de 1 a 9999, e é utilizado para:

— indicar ao *Spectrum* que se trata de uma linha de **programa;**
— servir de número de referência para uma determinada linha do programa.

Desliguemos o *Spectrum* por alguns segundos e voltemos a ligá-lo, de modo a que a frase inicial apareça de novo; podemos dar então entrada a:

```
2Ø PRINT 12345
```

Quando carregarmos desta vez na tecla ENTER, o *Spectrum* não imprimirá "12345", mas, em vez disso, apresentará uma cópia desta linha na parte superior do *écran*. O que fizemos foi introduzir um programa de uma só linha, e ainda não dissemos ao computador que o executasse.

Para que ele o faça, é necessário introduzir o comando directo

```
RUN
```

(tecla "R"), não esquecendo de carregar em ENTER.

A imagem no *écran* deverá então mudar, para apresentar "12345" na parte superior esquerda e

```
Ø OK, 2Ø:1
```

na parte inferior esquerda (o que significa "Programa executado

com êxito, terminando no fim da linha 20, declaração 1").

Se carregarmos novamente em ENTER, veremos a linha de programa reaparecer. Se dermos novamente entrada a RUN, o programa será executado mais uma vez — podemos fazê-lo quantas vezes quisermos.

Introduzamos agora uma nova linha:

```
1Ø PRINT Ø
```

Repare-se que, ao carregar em ENTER, a nova linha foi inserida antes da anterior. Isto acontece porque a nova linha tem um número de linha inferior. As linhas de programa são sempre dispostas pela ordem dos seus números de linha, ainda que sejam introduzidas por outra ordem qualquer. Introduzamos uma terceira linha:

```
3Ø PRINT 6
```

Se agora dermos ordem de execução (RUN) ao programa de três linhas resultante, a parte superior do *écran* deve apresentar o seguinte resultado:

```
Ø
12345
6
```

Isto demonstra que, para além de armazenar as linhas de um programa pela ordem dos seus números de linha, o *Spectrum* também as executa nessa ordem quando o programa é executado.

MONTAGEM DO PROGRAMA

Vimos anteriormente como modificar uma linha enquanto ela se encontra na parte de baixo do *écran*, e como acrescentar uma nova linha a um programa. Mas como proceder se desejarmos apagar ou alterar uma linha (de programa) já existente?

Partindo do princípio de que o programa anterior de três linhas


```
1Ø PRINT Ø
2Ø PRINT 12345
3Ø PRINT 6
```

ainda se encontra na memória do computador, vamos introduzir apenas o seguinte número de linha:

```
2Ø
```

Depois de se ter carregado na tecla ENTER, a nova versão do programa será apresentada no *écran* — e verificar-se-á que a linha 20 foi apagada. Deste modo, concluímos que introduzindo apenas um número de linha iremos apagar de um programa uma linha com esse número.

Voltemos a colocar essa linha no programa, introduzindo:

```
2Ø PRINT 3
```

O programa apresentado no *écran* deve ter agora, novamente, três linhas, mas — ui! — cometemos um erro, pois a linha 20 não é aquilo que era. Mas não há problema — basta introduzir a versão correcta da linha:

```
2Ø PRINT 12345
```

e ela substituirá automaticamente no programa a versão anterior.

Já se observou decerto que uma das linhas do programa tem um ">" depois do número de linha. É um cursor "editor de linha", e pode ser deslocado de linha para linha, carregando em:

— CAPS SHIFT e na tecla "6" (↓) para o deslocar para baixo, para a linha de programa seguinte;
— CAPS SHIFT e na tecla "7" (↑) para o deslocar para cima, para a linha de programa anterior.

Deslocando o cursor até à linha 10 e carregando simultaneamente em CAPS SHIFT e na tecla EDIT (tecla "1"), aparecerá uma cópia da linha na parte de baixo do *écran*. Pode-se modificar esta linha do mesmo modo que qualquer linha de entrada *(input)* que se encontre no extremo inferior do *écran* e, quando se carregar em ENTER, o *Spectrum* aceitará essa linha como se ela tivesse sido escrita e introduzida normalmente.

Se ficar com o mesmo número de linha, ela irá substituir a

versão anterior, mas, mudando-lhe o número de linha, a versão anterior continuará no programa e a nova versão será inserida no programa numa posição de acordo com o seu novo número de linha.

O cursor > pode também ser deslocado directamente para uma linha determinada, utilizando o comando directo LIST. Por exemplo, se introduzirmos

```
LIST 2Ø
```

o cursor será deslocado para a linha 20. Isto é bastante útil quando se tem um programa muito longo. Também se faz desaparecer o cursor LISTando um número de linha não existente, por exemplo:

```
LIST 13
```

Se o cursor não estiver visível, é possível fazê-lo reaparecer LISTando uma linha existente ou deslocando-o para cima e para baixo, utilizando CAPS SHIFT e a tecla "6" ou a tecla "7".

Finalmente, quando estivermos fartos do programa, podemos apagá-lo desligando o *Spectrum* por alguns segundos ou — de uma forma mais elegante — introduzindo o comando directo

```
NEW                    (tecla "A")
```

Quando nos tornarmos mais ambiciosos quereremos introduzir programas mais extensos do que é possível apresentar no *écran* de uma só vez. Não é problema: o *Spectrum* tem uma área de memória suficientemente vasta para armazenar o programa, e o que se vê no *écran* é apenas uma cópia de parte desse programa.

ENCADEAR COISAS...

Até agora, só temos feito imprimir números, mas é claro que podemos fazer o *Spectrum* imprimir qualquer dos símbolos ou caracteres do teclado.

A única restrição é que o que queremos imprimir tem de ser

colocado entre duas aspas (" : o símbolo a vermelho na tecla "P"). A razão será explicada no próximo capítulo e, de momento, só nos interessa saber que tudo o que se escrever entre duas aspas será tratado como uma "cadeia" (*string*) de caracteres, e será impresso como entrada.

Deste modo, podemos introduzir o seguinte:

```
PRINT "REGRAS DO SPECTRUM"
```

Até se podem incluir números, como no exemplo seguinte, capaz de arruinar a fé de qualquer pessoa na integridade dos computadores:

```
PRINT "1 + 1 = 3"
```

Como já se deve ter observado, o *Spectrum* não imprime as aspas que enquadram a cadeia, o que levanta problemas quando se quer realmente fazer imprimir aspas. O segredo está em escrever um **par** de aspas sempre que se quer que **uma** seja impressa. Por exemplo:

```
PRINT "Ele disse ""TRETAS"" muitas vezes"
```

Como anteriormente, podemos incluir cadeias numa única declaração PRINT, desde que sejam separadas por (;), (,) ou (').

Podemos mesmo combinar cadeias com números, como por exemplo:

```
PRINT "Hoje é" ' "sexta-feira", "dia"; 13.
```

Podemos incluir caracteres gráficos numa cadeia ou fazer com que o *Spectrum* imprima o inverso (o "negativo") da imagem dos caracteres (caracteres em branco sobre fundo negro), utilizando para isso a ordem INV VIDEO (CAPS SHIFT e tecla "4"); se se deseja voltar ao modo chamado TRUE VIDEO, carrega-se em CAPS SHIFT e na tecla "3".

LPRINT, LLIST e COPY

Se o leitor dispuser da impressora *ZX*, pode experimentar os exemplos dados anteriormente utilizando a impressora, mas deverá substituir a palavra de comando PRINT por LPRINT (por cima da tecla "C") nos exemplos dados. A única diferença que encontrará é que a parte do número de linha de um AT é ignorada por LPRINT — apesar de se ter de incluir um número de linha válido a seguir ao AT.

LLIST (por cima da tecla "V") fará imprimir na impressora *ZX* todo um programa; LLIST n irá imprimir todas as linhas de um programa a partir da linha n, começando nela.

COPY é um comando útil, pois permite-nos obter uma cópia impressa do que quer que seja visível nas primeiras 22 linhas do *écran*, com uma excepção: se o que é visível for o resulado de uma listagem "automática" (listagem obtida carregando apenas em ENTER, em vez de ser obtida por um comando LIST), então o comando COPY não terá efeito. Repare-se que COPY obtém cópias impressas da listagem de um programa geralmente com melhor qualidade do que as obtidas com LLIST.

Capítulo 3

Variáveis, INPUT e expressões

Em todos os exemplos apresentados no capítulo anterior, sabíamos, na altura em que os escrevemos, exactamente aquilo que o *Spectrum* tinha de imprimir. Os elementos impressos eram aquilo a que se dá o nome de "constantes", e que podem ser de dois tipos:

— constantes numéricas. P. ex.: 123 ou 747
ou
— constantes de cadeia (*string constants*), tais como "Spectrum".

(A palavra *"literal"* é, por vezes, usada em vez de *"constant"*, em inglês.)

Contudo, quando escrevemos um programa, muitas vezes não sabemos exactamente como será determinado elemento. Por exemplo, se o nosso programa tiver a função, entre outras, de perguntar ao utilizador o seu nome e idade, não sabemos, na altura em que o escrevemos, quais serão as respostas. No entanto, temos de as representar de alguma forma no programa.

A solução é semelhante à utilizada em álgebra: representamos quaisquer incógnitas por uma letra ou por letras. Podemos, por exemplo, decidir utilizar a letra "a" para representar, no nosso programa, a idade (*age* em inglês) do utilizador.

Dizemos pois que "a" é uma "variável", e as variáveis em BASIC possuem duas características importantes, a saber:

— o seu "tipo": podemos ter uma variável numérica para representar um número, ou uma variável de cadeia (*string variable*) para representar uma cadeia;
— o "nome" utilizado para individualizar uma determinada variável. Existem regras rígidas especificando quais os "nomes" de variáveis que são aceitáveis em BASIC, e os diferentes tipos de variáveis são diferenciados por tipos diferentes de "nome".

Sempre que utilizamos uma variável pela primeira vez, o "intérprete" BASIC do *Spectrum* atribui-lhe automaticamente um espaço na RAM, de modo a que o valor aí seja armazenado. Sempre que essa variável é utilizada, o "intérprete" procura automaticamente o endereço da área da RAM atribuída a essa variável.

VARIÁVEIS NUMÉRICAS

As variáveis numéricas são as do tipo mais simples, e designam um número.

Uma variável númerica pode ser representada por qualquer combinação de letras, dos dígitos Ø a 9 e de espaços, desde que o primeiro carácter seja uma letra. Assim,

a
A
a2
Maior

são todas representações de variáveis, mas

2
2A

não o são.

O *Spectrum* não faz a distinção entre letras maiúsculas e letras minúsculas na representação de uma variável, assim como ignora os espaços em branco e, portanto, FRED, fred, Fred, fRED e mesmo FR ED são considerados pelo computador como o mesmo nome.

Para utilizar uma variável, temos primeiro de lhe dar um valor. Uma forma de o fazer é utilizar a declaração LET. Por exemplo:

```
LET A=1ØØ
```

faz com que o *Spectrum:*

— reserve um espaço na RAM para a variável A;
— armazene aí o valor "1ØØ".

No caso de termos as duas linhas de programa seguintes,

```
1Ø LET A=1ØØ
2Ø LET A=2ØØ
```

a variável A acabaria por ficar com o valor 2ØØ, pois a segunda declaração LET anularia a primeira, substituindo o anterior valor de A. Podemos verificar isto acrescentando uma terceira linha:

```
3Ø PRINT A
```

Vale a pena pensar acerca desta linha 3Ø. Não faz com que o *Spectrum* imprima a letra A; para isso seria necessário existirem aspas de ambos os lados da letra:

```
3Ø PRINT "A"
```

Na realidade o que faz é dizer ao *Spectrum* que imprima o valor da variável representada por A.

Podemos ver o que acontece se pedirmos ao computador que imprima o valor de uma variável que ainda não tenhamos utilizado (e portanto ainda não tenhamos definido), dando entrada ao comando directo

```
NEW
```

o qual apaga da memória do *Spectrum* quaisquer linhas de programa ou variáveis e, seguidamente, introduzindo e fazendo executar o programa:

```
1Ø PRINT B
```

O *Spectrum* responderá com o seguinte:

```
variable not found   (variável não encontrada)
```

SOMAS SIMPLES

O *Spectrum* adiciona dois números sem a menor dificuldade. Experimente-se introduzir

```
10 LET a=5
20 LET b=7
30 LET c=a+b+2
40 PRINT c
```

A resposta impressa no *écran* deverá — é claro — ser "14" (5+7+2). Podemos experimentar versões diferentes da linha 30 e, depois, a subtracção:

```
30 LET c=b-a
```

ou a multiplicação:

```
30 LET c=a*b
```

ou a divisão:

```
30 LET c=b/2
```

Quem estiver habituado à álgebra pode ficar surpreendido com declarações BASIC do género

```
LET a=a+1
```

Mas o sinal de igualdade em BASIC não exprime a mesma coisa que o sinal de igualdade em álgebra. Em BASIC este sinal não significa que os dois membros da "igualdade" sejam iguais: é, antes, uma instrução para que o computador

— calcule primeiramente o valor da expressão a direita do sinal de igualdade;
— seguidamente, atribua este valor à variável mencionada à esquerda do sinal de igualdade.

Portanto, o exemplo dado algumas linhas acima apenas adiciona "um" ao valor da variável a.

O *Spectrum* também tem o operador aritmético ↑ (na tecla "H"). Este sinal significa "elevar à potência de". É, pois, um operador que permite a operação de exponenciação. Assim,

```
5 ↑ 2 é o mesmo que 5*5
5 ↑ 4 é o mesmo que 5*5*5*5
```

Note-se que o *Spectrum* não permite que se coloque um valor negativo à esquerda do sinal ↑; ou seja, não permite a exponenciação de um número negativo. Deste modo,

```
(-3) ↑ 2
```

irá provocar a impressão de uma mensagem de erro.

EXPRESSÕES GRANDES

Não existe limite para a complicação que a expressão à direita do sinal de igualdade possa ter. No entanto, é preciso lembrar que o *Spectrum* executa primeiro as operações ↑, depois as multiplicações ou divisões, e só no fim se ocupa das adições e subtracções.

Por exemplo, para calcular

```
LET c=a+b/7
```

o computador irá, primeiramente, efectuar a divisão de b por 7, e, seguidamente, adicionar o resultado a a. De modo semelhante,

```
LET c=5+6*2-1
```

irá atribuir a c o valor 16:

```
  5
+ 6 *2
- 1
```

Podemos contudo fazer o *Spectrum* calcular uma determinada parte de uma expressão em primeiro lugar, bastando para isso

que se ponha essa parte entre parênteses. Por exemplo:

```
LET c=(5+6)*2-1
```

atribui à variável c o valor 21, e

```
LET c=(5+6)*(2-1)
```

atribui-lhe o valor 11.

Apesar de haver vários símbolos com a forma de parênteses no teclado do *Spectrum*, os únicos a utilizar em expressões aritméticas são os que se encontram impressos a vermelho nas teclas "8" e "9".

Podem-se colocar parênteses dentro de parênteses:

```
(1+(2+3)*4)*5
```

será calculado como

```
  (1+  5  *4)*5
=(1+  2Ø    )*5
 =     21     *5
 =       1Ø5
```

Quem já tiver trabalhado com álgebra, estará provavelmente familiarizado com expressões semelhantes à seguinte:

```
2 (a+b)
```

onde está implícito um sinal de multiplicação entre o "2" e o parêntese esquerdo. Isto não pode ser feito em BASIC; o sinal de multiplicação deve ser sempre apresentado, como na expressão

```
2*(a+b)
```

INFORMAÇÕES GERAIS

O BASIC do *Spectrum* é bastante "inteligente" num aspecto: onde a sintaxe de uma declaração exija um **número,** pode-se utilizar

qualquer **expressão numérica válida.** Ou seja, qualquer expressão que dê um resultado numérico.

Portanto, e recordando do capítulo anterior que a palavra PRINT pode ser seguida de um número, devemos poder fazer seguir PRINT de uma expressão numérica, como por exemplo:

```
PRINT 22+23
```

Experimente introduzir isto e verá que resulta, assim como resultam as instruções de programa seguinte:

```
1Ø LET min=4
2Ø LET max=1Ø
3Ø PRINT "a medida e' ";(min+max)/2
```

INPUT

Até aqui, o computador só trabalhou com os valores que incluímos no próprio programa. O comando INPUT permite ao programa obter um valor do utilizador, enquanto está a ser executado.

Na sua forma mais simples, uma declaração de INPUT consiste na palavra de comando INPUT (tecla "I") seguida do nome de uma variável como em

```
1Ø INPUT a
2Ø PRINT a
```

Ao executar este programa, verifica-se que parece parar com o *écran* vazio, à excepção de um cursor "L" intermitente na sua parte inferior esquerda. Isto é provocado pela declaração INPUT: o computador está à espera que se introduza um número.

Vamos experimentá-lo, escrevendo um número e carregando em ENTER. O número que escrevemos desaparecerá da parte de baixo do *écran* para ser impresso no canto superior esquerdo. A linha 10, na verdade, tomou o valor introduzido e atribuiu-o à variável a.

Podíamos ter introduzido uma expressão numérica em vez de um número simples. Vamos fazer executar novamente o pro-

grama mas, desta vez, introduzindo uma expressão como a seguinte:

```
22/2
```

Verificaremos que o programa aceita a expressão e imprime o resultado.

O único problema deste pequeno programa é que ele não é muito "simpático" para com o utilizador. De facto, apresenta-lhe um *écran* praticamente vazio e espera do utilizador que faça a coisa certa (p. ex.: introduzir um número). Isto não traz embaraços num programa pequeno, mas que aconteceria num programa que requeresse muitos INPUTs? O utilizador perder-se-ia certamente, a não ser que alguma espécie de "lembrete" ou "deixa" fosse apresentada no *écran*, sempre que se tivesse de introduzir algum valor.

Poder-se-ia utilizar, é claro, uma declaração PRINT:

```
1Ø PRINT "Introduza um numero"
2Ø INPUT n
3Ø PRINT "O numero introduzido e'";n
```

Mas o comando INPUT do *Spectrum* também pode ser utilizado para imprimir um "lembrete". As regras são as seguintes:

— Uma palavra de comando INPUT pode ser seguida de qualquer número de elementos desde que eles sejam separados da mesma forma que os elementos numa declaração PRINT.
— Se o primeiro carácter de um elemento for uma letra, então esse elemento será tomado como a representação de uma variável, e o computador ficará à espera que o utilizador introduza um valor adequado.
— Qualquer outra coisa será impressa exactamente como se fosse parte de uma declaração PRINT, mas, neste caso, na parte inferior do *écran*. Pode-se mesmo utilizar TAB e AT, mas o número de linha de uma instrução AT será o número da última linha de *écran*.
— Depois de completamente executada a declaração de INPUT, o que quer que tenha sido impresso por essa declaração será apagado do *écran*.

Como demonstração, experimente-se o seguinte:

```
1Ø INPUT "Introduza um numero ";a
2Ø PRINT "O numero introduzido e' ";a
```

Uma declaração de INPUT pode mesmo obter valores para mais de uma variável:

```
1Ø INPUT "Introduza a ";a' "Introduza b";b' "Introduza c
";c
2Ø PRINT a'b'c
```

Executando este exemplo, veremos como o *Spectrum* avança passo a passo na execução da declaração de INPUT, à medida que introduzirmos cada valor.

VARIÁVEIS DE CADEIA

A uma "variável" também se pode atribuir uma cadeia, sendo uma "cadeia" constituída por um conjunto de caracteres ou qualquer outro tipo de símbolo existente no teclado do *Spectrum*.

No entanto, para que o *Spectrum* possa saber se o "nome" de uma variável se destina a identificar uma cadeia ou um número, utiliza um tipo especial de "nome" para representar as variáveis de cadeia. Os "nomes" desse tipo são constituídos por uma única letra seguida do símbolo $; por exemplo:

```
A$
x$
```

são nomes válidos de variáveis de cadeia.

Como sabemos, o *Spectrum* não faz a distinção entre letras minúsculas e letras maiúsculas e, portanto, A$ é a mesma coisa que a$. Contudo, distingue perfeitamente a variável de cadeia A$ da variável numérica A.

Podemos utilizar LET para atribuir um valor a uma variável de cadeia, exactamente como fizemos para as variáveis numéricas ou, ainda, incluí-las em declarações PRINT:

```
1Ø LET a$="Spectrum"
2Ø LET b$=a$
3Ø PRINT b$
```

Note-se que, se a variável à esquerda do sinal de igualdade for uma variável de cadeia, a expressão à direita do sinal deve dar como resultado uma cadeia, e se a variável à esquerda do sinal for uma variável numérica, a expressão à direita também tem de ser numérica. Os dois tipos não podem ser misturados. Por exemplo, escrever

```
LET a="Spectrum"
LET a$=12+34
```

não é permitido.

Pode-se igualmente utilizar uma variável de cadeia numa declaração INPUT:

```
1Ø INPUT a$
2Ø PRINT a$
```

Quando fizermos executar este programa, veremos que desta vez a declaração de INPUT coloca aspas de ambos os lados do cursor "L", no canto inferior esquerdo do *écran*, e que tudo aquilo que introduzirmos é impresso exactamente como o escrevemos, sem ser feita qualquer tentativa de cálculo por parte do *Spectrum*. Por exemplo, se escrevemos

```
1+2
```

a linha 20 deste programa imprimirá

```
1+2
```

e não

```
3
```


Estas aspas são por vezes incomodativas, mas eliminamo-las incluindo a palavra de comando LINE (tecla 3) antes do nome da variável de cadeia. Por exemplo:

```
1Ø INPUT "Introduza o seu nome"; LINE n$
2Ø INPUT "Viva! ";(n$)'"Quantos anos tem? ";y
3Ø PRINT n$;" tem ";y;" anos de idade"
```

A linha 20 é interessante, pois desejávamos fazer imprimir o valor de n$ (o seu nome) como parte do "lembrete" de INPUT, mas o INPUT toma qualquer elemento que comece por uma letra como uma variável a introduzir *(input)*. Contudo, colocando o nome da variável (n$) entre parênteses, obtemos um elemento que já não começa por uma letra! Deste modo, imprime-se o valor de n$ (isto é, o seu nome).

EXPRESSÕES DE CADEIA

Da mesma forma que podemos colocar qualquer expressão numérica à direita do sinal de igualdade quando exista o nome de uma variável numérica à esquerda do sinal, podemos escrever qualquer expressão de cadeia à direita do sinal se houver o nome de uma variável de cadeia à esquerda do dito sinal.

Há duas formas principais de construir uma expressão de cadeia. A primeira delas utiliza o sinal "+" para unir duas cadeias:

```
LET a$="TIME"+"DATA"
```

fará a$ ser igual a "TIMEDATA", e o programa

```
1Ø LET a$="DIS"
2Ø LET b$=a$+"PARATE"
3Ø PRINT b$
```

irá, é claro, imprimir DISPARATE, tal como o fará o programa seguinte:

```
1Ø LET a$="DIS"
2Ø LET a$=a$+"PARATE": PRINT a$
```

O BASIC do *Spectrum* também permite transcrever parte de uma cadeia, utilizando para isso o operador TO (tecla "F"). De um modo geral,

```
LET a$=b$(xTO y)
```

irá igualar a$ do x-ésimo ao y-ésimo carácter de b$. Assim,

```
LET m$="11-SET-85"(4 TO 6)
```

fará m$ igual a "SET".

Se o número à esquerda do TO é 1 significa que se deseja que a transcrição comece com o primeiro carácter; podemos então omitir esse 1. Por exemplo:

```
"SINCLAIR"( TO 3)          é  "SIN"
```

De forma semelhante, se desejarmos transcrever os caracteres de uma cadeia desde um ponto determinado até ao fim dessa cadeia, sendo portanto o número a seguir ao operador TO igual ao comprimento em caracteres da cadeia, podemos também omitir esse número. Assim,

```
"SINCLAIR"(5 TO  )         é  "LAIR"
```

O utilizador deverá ter o cuidado de verificar se existem realmente tantos caracteres quantos aqueles que indicou, pois, por exemplo,

```
"ABC"( TO 4)
```

daria origem à mensagem de erro "subscript wrong", porque "ABC" só tem três caracteres.

Retirar um único carácter de uma cadeia pode igualmente ser feito com TO:

```
"OXO"(2 TO 2)
```


Mas é mais simples escrever:

```
"OXO"(2)
```

O operador TO serve também para modificar parte de uma cadeia. Como exemplo, se experimentarmos introduzir

```
1Ø LET a$="123456"
2Ø LET a$(3 TO 4)="AB"
3Ø PRINT a$
```

este programa imprimirá

```
12AB56
```

Neste caso, a linha 20 "diz" ao *Spectrum* para substituir certos caracteres da cadeia a$, mas deixa inalterados os restantes caracteres da cadeia. É claro que os caracteres a substituir têm de existir, caso contrário, se a linha 2Ø fosse, por exemplo

```
2Ø LET a$(6 TO 7)="AB"
```

o programa pararia e apareceria no *écran* a mensagem "subscript wrong", pois não existe um sétimo carácter em a$.

Mas que acontece se a expressão à direita do sinal de igualdade for demasiadamente longa ou demasiadamente curta? Com este caso o *Spectrum* pode lidar. Assim, se a cadeia à direita do sinal é longa demais, o computador transcreverá apenas tantos caracteres quantos os necessários. Modificando a linha 20 para

```
2Ø LET a$(3 TO 4)="ABCD"
```

ainda se obtém

```
12AB56
```

Se, por outro lado, a cadeia à direita do sinal for demasiado curta, o *Spectrum* juntará tantos espaços em branco quantos

forem necessários para obter o comprimento necessário. Assim, de

```
2Ø LET a$(3 TO 4)="A"
```

resultará

```
12A  56
```

Também neste caso, se desejarmos substituir um único carácter, podemos utilizar a forma simplificada, apresentada a seguir:

```
LET a$(x)
```

em vez de

```
LET a$(x TO x)
```

FUNÇÕES

O BASIC do *Spectrum* inclui um certo número de palavras de comando conhecidas por "funções". Estas "funções" são operações pré-programadas que dão um resultado numérico ou uma cadeia e que, portanto, podem ser utilizadas em expressões numéricas ou de cadeia.

SQR é uma dessas "funções". Como o nome sugere, permite calcular a raiz quadrada (*SQuare Root*, em inglês) de um número. Assim, por exemplo,

```
PRINT SQR 4
```

imprimirá

```
2
```

Conhecemos por "argumento" da função o valor sobre o qual ela opera. No exemplo apresentado, "4" é o "argumento" da função SQR. Algumas funções operam apenas sobre um argumento, outras necessitam de dois e outras há que não necessitam de argumento algum.

Os argumentos podem também ser variáveis, como em

```
PRINT SQR a
```

ou ser expressões, como, por exemplo,

```
PRINT SQR (9+16)
```

Foram colocados parênteses em torno da expressão anterior, de modo a que a função se aplique à totalidade da expressão.

```
PRINT SQR 9+16
```

daria um resultado diferente.

O manual apresenta uma lista completa das funções disponíveis do *Spectrum*, e deve-se reparar que os nomes das funções que dão uma cadeia como resultado (e não um resultado numérico) terminam sempre com o sinal $ (p. ex.: CHR$).

Muitas dessas funções (COS, EXP, etc.) são matemáticas, e a sua utilidade deve ser evidente se se estiver a escrever um programa que delas necessite. As outras funções — mais "esotéricas" — serão tratadas em capítulos posteriores. Convém, contudo, lembrar que, caso não se tenha a certeza acerca do que faz determinada função, é muito fácil de verificá-lo, geralmente incluindo-a numa declaração PRINT, de modo a ver-se o resultado que dá a aplicação da função.

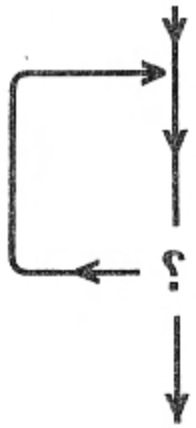
Capítulo 4

Ciclos e decisões

Quando se olha para um programa escrito em BASIC vê-se uma única lista de linhas de instrução numeradas. Essas linhas dispõem-se segundo a ordem dos respectivos números de linha, sendo a linha com o número mais baixo impressa em primeiro lugar e a que tiver o número mais elevado impressa em último.

Analogamente, quando se faz executar (RUN) o programa, o computador executa primeiramente a instrução contida na linha de número mais baixo; seguidamente, passa em geral para a linha de número imediatamente superior e executa a instrução nela contida. O processo repete-se até o computador chegar ao fim da linha de número mais elevado do programa.

Mas um computador pode fazer muito mais do que começar na primeira linha de um programa e ir gradualmente executando linha a linha até chegar ao fim. Pode-se fazer com que ele repita a execução de uma linha ou de um grupo de linhas quantas vezes



desejarmos, e mude de "rumo" em resposta a um determinado conjunto de circunstâncias, saltando (*skipping*) sobre várias linhas ou voltando a uma linha de número inferior. São estas capacidades que dão a um computador todo o seu poder e flexibilidade, e nenhum programa digno desse nome se conseguirá escrever sem fazer uso delas.

GO TO

Usa-se a palavra de comando GO TO (na tecla "G") para alterar a ordem pela qual se executam as declarações de programa, fazendo o *Spectrum* "saltar" para outra linha.

Na sua forma mais simples, uma declaração GO TO consiste apenas na palavra GO TO seguida de um número de linha:

```
1Ø PRINT  "Spectrum.";
2Ø GO TO 1Ø
```

Se a experimentarmos, veremos que é uma pequena rotina surpreendentemente eficiente.

Quando o computador chega à linha 20, encontra a declaração GO TO a mandá-lo voltar ao início da linha 10, que é o que ele faz, e assim executa a linha 10 outra vez. E mais outra... e outra... até que o *écran* fica cheio e a mensagem

```
scroll?
```

aparece no canto inferior esquerdo. Esta mensagem é uma pergunta que o computador faz ao utilizador para saber se ele deseja que a imagem "role" ou seja "enrolada", tal como se estivesse escrita num rolo de papel, passando a imagem apresentada para "fora" do *écran,* deixando-o limpo, sem que no entanto essa imagem seja apagada definitivamente, ficando guardada na memória do *Spectrum*. Carregando numa tecla qualquer, à excepção de N ou de BREAK, faz-se com que o programa continue a ser executado (após a imagem ter "rolado") e volte a preencher o *écran*.

Se juntarmos uma terceira linha,

```
15 POKE 23692,Ø
```

a imagem será então "enrolada" de vez, sem mesmo nos pedir licença (quem não compreender esta terceira linha não se preocupe — por enquanto vamos tratá-la como se ela fosse uma fórmula mágica infalível). Contudo, só se conseguirá agora fazer parar o programa desligando o *Spectrum* ou carregando simultaneamente nas teclas CAPS SHIFT e BREAK, pois o programa tem uma estrutura de "ciclo ininterrupto".

Pode fazer-se com que o computador efectue "saltos" para a frente como para trás, como se exemplifica pelo programa seguinte, cuja linha 30 nunca é executada:

```
1Ø PRINT "O SPECTRUM E' 'O'"
2Ø GO TO 4Ø
3Ø PRINT "PIOR"
4Ø PRINT "COMPUTADOR"
```

DIA DE SEMANA

O número a seguir ao GO TO pode ser substituído por uma expressão numérica. Isto torna-se útil quando se pretende introduzir um elemento de escolha no salto.

```
  10 INPUT "Introduza o dia da s
emana sob a forma de um numero d
e 1 a 7: ";d
  20 CLS : PRINT "O dia ";d;" e
";
  30 GO TO 100+d
 100 GO TO 10
 101 PRINT "Segunda": GO TO 10
 102 PRINT "Terca": GO TO 10
 103 PRINT "Quarta": GO TO 10
 104 PRINT "Quinta": GO TO 10
 105 PRINT "Sexta": GO TO 10
 106 PRINT "Sabado": GO TO 10
 107 PRINT "Domingo": GO TO 10
```

Neste programa, a linha 3Ø vai provocar um salto para a linha 1Ø1 se for introduzido o número "1", para a linha 1Ø2 se for introduzido o número "2", e assim por diante. O principal problema deste tipo de GO TO "computado" reside no facto de ele aceitar valores inesperados. Por exemplo, neste programa é pedido ao utilizador que introduza um número de 1 a 7, mas há uma probabilidade razoável de alguém introduzir um número fora deste intervalo. Se esse alguém introduzir "Ø", a linha 3Ø provocará um salto para a linha 1ØØ, que foi incluída no programa de forma a lidar com essa possibilidade. Se, por outro lado, alguém introduzir um número superior a 7, então a linha 3Ø provocará um salto para uma linha com um número superior a

qualquer número de linha do programa, o que é impossível. Felizmente, o BASIC do *Spectrum* é bastante tolerante a este respeito: se lhe pedirem que salte para um número de linha que não existe, ele saltará para a linha de maior número seguinte ou — se não existir nenhuma — o programa parará simplesmente.

IF - - THEN

As palavras de comando IF e THEN proporcionam uma forma muito mais genérica de permitir ao programa escolher entre efectuar ou não determinada acção. Estas duas palavras são sempre associadas numa declaração da forma

IF expressão lógica THEN declaração (declarações)

Uma "expressão lógica" é toda a expressão que dê um resultado "verdadeiro" ou "falso". São exemplos disto as seguintes expressões:

```
A=B
Preto=Branco
2<=5
a$<>"SIM"
```

A declaração ou declarações que se seguem ao THEN podem ser quaisquer declarações válidas em BASIC, têm de fazer parte da mesma linha de programa de IF THEN e são executadas apenas se a expressão lógica for "verdadeira". Se a expressão lógica der um resultado "falso", o *Spectrum* "saltará" para o início da linha de programa seguinte. Se houver mais do que uma declaração depois do THEN, essas declarações têm de ser separadas por sinais de dois pontos (:), tal como em qualquer outra linha de programa.

Já basta de teoria; vamos, pois, ver como isto "funciona" na prática:

```
IF ano = 2Ø85 THEN PRINT "Centenario"
```

Esta instrução apenas imprimirá "Centenário" se a variável "ano" tiver o valor de "2Ø85"...

```
IF score>hiscore THEN PRINT "Bravo!": LET
hiscore=score
```

onde "score" quer dizer "pontuação" e "hiscore" se pode traduzir por "pontuação máxima" (atingida anteriormente). Preferiu-se manter aqui os vocábulos ingleses pelo facto, já mencionado, de o *Spectrum* não possuir acentos ou cedilhas e por a tradução exigir, neste caso, um comprimento de linha excessivo.

Quanto à rotinazinha apresentada acima, pode dizer-se que ela é bastante útil quando colocada no final de um programa de jogos.

IF THEN pode ser seguido de outros IF THEN. Por exemplo:

```
IF mes=4 THEN IF dia = 23 THEN PRINT "Feliz
Aniversario"
```

IF THEN é muitas vezes seguido de GO TO, tal como na rotina seguinte, que permite imprimir os números de 1 a 10:

```
1Ø LET A=1
2Ø PRINT A
3Ø LET A=A+1
4Ø IF A<=1Ø THEN GO TO 2Ø
```

No caso de o leitor estar confuso acerca dos valores "verdadeiro" (*True*) e "falso" (*False*), talvez seja melhor explicar que o *Spectrum* gera um resultado numérico sempre que opera sobre uma expressão lógica; sendo esse resultado "1" se a expressão é "verdadeira" ou "Ø" se é "falsa". Pode obter a demonstração disto introduzindo o seguinte comando directo:

```
PRINT 2=2,2=3
```

o qual irá imprimir 1 Ø

Analogamente, a linha


```
LET a=expressao logica
```

irá atribuir à variável a o valor um se a expressão lógica for verdadeira, ou zero se for falsa.

Na prática, o *Spectrum* considera **qualquer** valor, excepto o zero, como "verdadeiro", e portanto a linha

```
IF a THEN PRINT "VERDADEIRO"
```

imprimirá "VERDADEIRO" para qualquer valor diferente de zero que se dê a "a"

Note-se que os símbolos<> (diferente de), <= (menor ou igual a) e >= (maior ou igual a) devem ser introduzidos como palavras de comando **individuais** e **não** como combinações das palavras de comando <,> e =.

NOT

Esta é uma função bastante útil, que converte "verdadeiro" em "falso" e vice-versa.

```
IF NOT (2=3) THEN PRINT "2 NAO E' IGUAL A 3"
```

AND

A palavra de comando AND é um "operador lógico" e pode ser utilizada como parte de uma expressão lógica. Se escrevermos

```
p AND q
```

onde p e q representam ambos expressões lógicas, o resultado só terá o valor "verdadeiro" se p e q forem ambos verdadeiros. Caso contrário, o resultado terá o valor "falso". Deste modo,

```
IF (mes=4) AND (dia=23) THEN PRINT "Feliz
Aniversário"
```

é uma forma alternativa de escrever a rotina apresentada anteriormente para exemplificar o uso de dois IF numa única linha.

Também se utiliza o operador AND dentro de uma expressão **numérica,** pois

```
A AND B
```

dá o valor

```
A se B não for zero
Ø se B for zero
```

Como B pode ser uma expressão lógica seria lícito escrever uma declaração do género

```
LET altura=altura − (1Ø AND altura>=1Ø)
```

a qual iria subtrair 10 da variável "altura", mas só na condição de a variável "altura" tomar um valor maior ou igual a 10.

AND pode mesmo ser utilizado numa expressão de cadeia. Por exemplo:

```
A$ AND B
```

dá como resultado A& se B não for zero, ou uma cadeia nula (vazia) "" se B for zero

Isto torna-se útil muitas vezes em declarações PRINT, tais como:

```
PRINT ("BOM" AND X)+("MAU" AND NOT X)
```

Esta declaração imprime "BOM" ou "MAU" consoante o valor de X.

Complicando ainda um pouco mais as coisas, podemos até colocar uma expressão numérica/lógica a seguir a um GO TO, como por exemplo:

```
GO TO (1ØØ AND x<Ø)+(2ØØ AND x=Ø)+(3ØØ AND
x>Ø)
```


o que faria o computador saltar para a linha 100, 200 ou 300, dependendo do valor de **x**.

NOTA: AND é a operação lógica 'E' da Álgebra de Boole, conhecida por 'Conjunção'.

OR

A palavra de comando OR (na tecla "U") é um outro operador lógico:

```
p OR q
```

(onde tanto p como q são expressões lógicas) dará o resultado "verdadeiro" se p ou q (ou ambas) forem verdadeiras. Só dará o resultado "falso" se p e q forem ambas falsas. Um exemplo adequado da sua utilização seria o seguinte:

```
IF (energia=Ø) OR (ar=Ø) THEN PRINT "VOCE
ESTA' MORTO"
```

Tal como AND, o operador OR usa-se também numa expressão numérica:

```
A OR B
```

irá dar 1 se B for diferente de zero ("verdadeiro")
A se B for zero ("falso")

de modo que

```
LET A=A OR (A<1)
```

asseguraria que o valor da variável A nunca descerá abaixo de 1.

Uma expressão de cadeia não admite o uso do operador OR

NOTA: OR é a operação lógica 'OU' da Álgebra de Boole, conhecida por 'Disjunção'.

FOR-TO-STEP--NEXT

Algumas páginas atrás, apresentámos um programa simples com o qual se imprimiram os números de 1 a 10 utilizando uma

declaração IF THEN GO TO para que o programa voltasse atrás se ainda não tivesse sido impresso o décimo número.

Mas existe uma outra forma pela qual poderíamos ter feito isto: utilizando um ciclo *(loop)* FOR-NEXT. É uma estrutura de BASIC muito útil, que faz repetir todo o programa ou parte dele um determinado número de vezes antes de continuar a sua execução. Esta estrutura consiste em duas declarações:

```
FOR a=x TO y STEP z
```

e

```
NEXT a
```

(FOR, TO, STEP e NEXT são todas palavras de comando, dadas pelas teclas "F", outra vez "F", "D" e "N", respectivamente.)

A parte do programa a ser repetida coloca-se entre as declarações FOR e NEXT.

Diz-se que "a" é a "variável de controle" do ciclo. Na realidade, esta variável é precisamente como qualquer outra variável numérica, como excepção de o seu "nome" apenas poder ser constituído por uma única letra. "x", "y" e "z" são expressões numéricas, na primeira declaração apresentada acima.

Quando encontra a declaração FOR, o *Spectrum* calcula primeiramente as expressões "x", "y" e "z" e memoriza os resultados obtidos. Então, iguala a variável de controle "a" ao valor "inicial" *(start value)* "x".

Seguidamente, executa as declarações que se seguem à declaração FOR até chegar a uma declaração NEXT que tenha a mesma variável de controle que a declaração FOR. Adiciona então o "valor de passo" *(step value)* ou "incremento" (z) à variável de controle (a). Caso o valor da variável de controle seja agora superior ao "valor limite" ou "valor de teste" (y) (ou menor do que y se o incremento z for negativo), o programa continua a ser executado com a declaração que se segue à declaração NEXT. Mas, se o valor da variável de controle ainda não tiver excedido o valor limite, o programa voltará à declaração a seguir a FOR.

Há, contudo, uma excepção a estas regras: quando se encontra a declaração FOR e se calculam os valores limite e incremento,

— se o valor da variável de controle é maior do que o valor limite y, e o incremento z é positivo,
— se o valor da variável de controle é menor do que o valor limite y, e o incremento z é negativo,

o *Spectrum* passará imediatamente para a declaração a seguir a NEXT, sem executar as declarações entre FOR e NEXT.

A palavra de comando STEP e a expressão do incremento (z) podem ser omitidas; neste caso considera-se que o incremento é igual a um.

Talvez isto pareça muito complicado, mas os princípios básicos de um ciclo FOR NEXT apreendem-se com bastante facilidade por meio de alguns exemplos:

```
10 FOR a=1 TO 10
20 PRINT a
30 NEXT a
```

Este exemplo faz imprimir os números de 1 a 10, enquanto que

```
10 FOR a=10 TO 1 STEP -1
20 PRINT a
30 NEXT a
```

os faz imprimir na ordem inversa.

Juntando a linha

```
40 PRINT a
```

a estes dois exemplos imprime-se o valor da variável de controle depois de se completar a execução do ciclo.

Mudando a linha 10 para

```
10 FOR a=1 TO 0
```

ou para

```
10 FOR a=0 TO 1 STEP -1
```

ilustram-se as condições em que as declarações dentro do ciclo não são executadas.

Se desejarmos, alteraremos o valor da variável de controle da mesma forma que o de qualquer outra variável, e assim,

```
10 FOR f=1 TO 5
20 IF f=13 THEN LET f=14
30 PRINT f
40 NEXT f
```

fará imprimir o número dos andares de um hotel americano, por exemplo.

TABUADAS

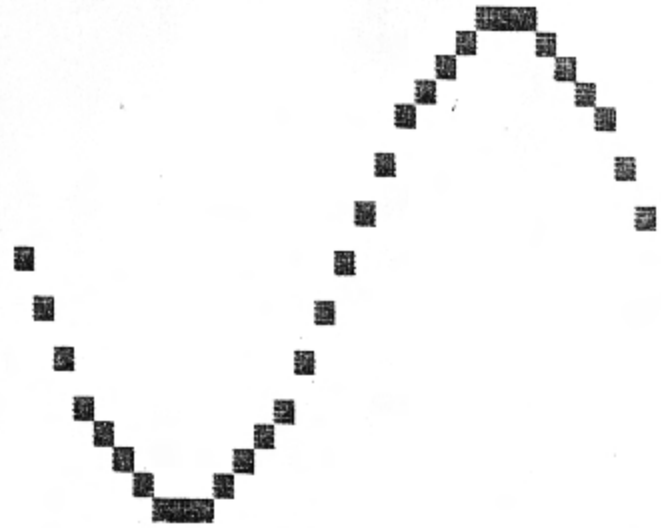
Como aplicação prática de um ciclo FOR NEXT, este programa fará imprimir tabuadas de multiplicação:

```
  10 INPUT "Tabuada do numero ?
";t
  20 FOR a=1 TO 12
  30 PRINT a;" VEZES ";t;" = ";a
*t
  40 NEXT a
```

SENO APROXIMADO

Como mais um exemplo de um programa simples baseado no ciclo FOR NEXT, este programa apresenta um desenho tosco de uma onda sinusoidal; quem não souber o que é uma onda sinusoidal, não se preocupe — limite-se a admirá-la.

```
  10 FOR a=0 TO 31
  20 PRINT AT (11+10*SIN (a*PI/1
6)),a;"■"
  30 NEXT a
```

Introduz-se o quadrado negro apresentado entre aspas na linha 20 da seguinte forma:

— carregando na tecla INV VIDEO enquanto se mantém pressionada a tecla CAPS SHIFT;
— introduzindo seguidamente um espaço em branco;
— carregando por fim na tecla TRUE VIDEO enquanto se mantém pressionada a tecla CAPS SHIFT.

quem preferir ver os espaços na curva preenchidos, altere a linha 10 de modo a utilizar um incremento menor, por exemplo:

```
10 FOR a=0 TO 31 STEP .5
```

Fazendo-se o incremento igual a zero obter-se-ia um ciclo sem fim, mas com este programa isso não traria qualquer vantagem.

NÚMEROS PRIMOS

Como exemplo de um programa que utilize mais do que um ciclo FOR NEXT, este programa calcula os 80 primeiros números primos depois de 2.

Como tem uma estrutura bastante complexa, desenharam-se linhas de modo a mostrar onde o "fluxo" do programa se pode desviar da sua progressão normal até à linha de número mais elevado. Esta técnica é muito útil quando se pretende compreender o funcionamento de um programa.

O ciclo principal do programa (linhas 20 a 80) apenas conta os números primos encontrados à medida que o programa avança, e a variável de controle do ciclo, "a", é também utilizada pela linha 70 de modo a apresentar um resultado com uma formatação bem ordenada.

```
10 LET t=1
20 FOR a=0 TO 79
30 LET t=t+2
40 FOR f=3 TO SQR t STEP 2
50 IF t=f*INT (t/f) THEN GO TO
30
60 NEXT f
70 PRINT TAB 8*a;t;
80 NEXT a
```

Dentro deste ciclo exterior, a variável t é o número que é testado para se ver se é primo. Começando com um número ímpar, e incrementando-o de cada vez de 2 (linha 30), o programa evita testar números pares, que não podem ser primos.

O teste para verificar se t é primo é efectuado pelas linhas 40-60, as quais verificam se qualquer número ímpar entre 3 e a raiz quadrada de t é divisor de t. Se o é, então t não é primo, e o programa volta à linha 30 para obter o próximo valor de t a ser testado. Note-se que, enquanto algumas versões de *Basic* não permitem programas que "saltem fora" de ciclos FOR-NEXT, o *Basic* do *Spectrum* não tem tal limitação. Se não forem encontrados quaisquer divisores, t é um número primo e é impresso por meio da linha 70.

ENCAIXE

Colocar um ciclo FOR NEXT dentro de outro, tal como

```
FOR A=B TO C
FOR X=Y TO Z
  . . .
NEXT X
NEXT A
```


é um processo, conhecido por *nesting* (encaixe, anichamento), perfeitamente aceitável. Ciclos nestas condições denominam-se "ciclos anichados" ou "encaixados" *(nested loops)*.

O que o *Spectrum* não pode aceitar, todavia, são dois ciclos FOR NEXT que se cruzem, tal como no exemplo abaixo:

```
FOR A=B TO C
FOR X=Y TO Z
  . . .
NEXT A
NEXT X
```

BINÁRIOS

O programa seguinte foi escrito para demonstrar o processo de encaixe de ciclos FOR NEXT. Este programa utiliza oito ciclos para gerar todos os 256 números binários de 8 *bits* possíveis, e fá-los imprimir acompanhados dos seus equivalentes decimais.

Quem dispuser da impressora *ZX* pode criar a sua própria tabela de conversão binário-decimal, alterando simplesmente a palavra PRINT na linha 110 para LPRINT.

```
100 FOR a=0 TO 1
101 FOR b=0 TO 1
102 FOR c=0 TO 1
103 FOR d=0 TO 1
104 FOR e=0 TO 1
105 FOR f=0 TO 1
106 FOR g=0 TO 1
107 FOR h=0 TO 1
110 PRINT a*128+b*64+c*32+d*16+
e*8+f*4+g*2+h;TAB 4;"= ";a;b;c;d
;e;f;g;h
120 NEXT h
121 NEXT g
122 NEXT f
123 NEXT e
124 NEXT d
125 NEXT c
126 NEXT b
127 NEXT a
```

```
0   = 00000000
1   = 00000001
2   = 00000010
```

```
3     =  00000011
4     =  00000100
5     =  00000101
6     =  00000110
7     =  00000111
8     =  00001000
9     =  00001001
10    =  00001010
11    =  00001011
12    =  00001100
13    =  00001101
14    =  00001110
15    =  00001111
16    =  00010000
17    =  00010001
18    =  00010010
19    =  00010011
20    =  00010100
21    =  00010101
```

```
234   =  11101010
235   =  11101011
236   =  11101100
237   =  11101101
238   =  11101110
239   =  11101111
240   =  11110000
241   =  11110001
242   =  11110010
243   =  11110011
244   =  11110100
245   =  11110101
246   =  11110110
247   =  11110111
248   =  11111000
249   =  11111001
250   =  11111010
251   =  11111011
252   =  11111100
253   =  11111101
254   =  11111110
255   =  11111111
```


Capítulo 5

Quadros e dados

Como vimos, é bastante fácil programar um computador de modo a que ele repita a mesma tarefa, tantas vezes quantas quisermos, utilizando um ciclo FOR NEXT ou uma declaração IF THEN GO TO.

Portanto, quando quisermos que o programa lide com mais do que pequenas quantidades de informação ou dados *(data)*, vale a pena organizar essa informação de um modo uniforme, de forma a que todos os cálculos sejam executados por uma única secção do programa, a qual é repetida tantas vezes quantas as necessárias.

As simples variáveis numéricas e de cadeia que temos considerado até aqui apenas são adequadas para representar elementos individuais de informação. A linguagem BASIC, contudo, fornece-nos também as "variáveis de quadro" ou de "matriz" *(array variables)*, mais correctamente denominadas "variáveis indexadas", para o tratamento de conjuntos ou blocos de informação.

QUADROS NUMÉRICOS

Suponha o leitor que é um professor e quer guardar num computador as notas de fim de período dos seus alunos. Para simplificar por enquanto o problema, admitiremos que apenas tem de armazenar uma nota por aluno — por exemplo, a média global de avaliação do aluno no período em causa.

Poderia utilizar-se uma nova variável para cada aluno — e, como o *Spectrum* não põe quaisquer restrições aos nomes atribuídos a variáveis numéricas, introduzir-se-ia qualquer coisa como

```
LET JAIME BOAVIDA = 4
LET MARGARIDA ANDRADE = 15
LET JOSE NATA = 2Ø
```

o que iria certamente armazenar os resultados, e o leitor poderia

até encontrar a nota de um determinado aluno por meio de uma declaração como a seguinte:

```
PRINT Artur Ferreira
```

Contudo, seria extremamente difícil — senão impossível — a um programa tratar esta informação. Mesmo para fazer uma coisa simples, como calcular a nota média da turma, o programa teria de conhecer de antemão todos os nomes. Para se compreender o problema que assim se levanta, experimente-se escrever um programa para adicionar as notas de uma turma de 30 alunos quando essas notas estão contidas em variáveis individuais como no exemplo acima, tendo em consideração que na altura de escrever o programa não se sabem os nomes dos alunos!

A forma de ultrapassar este problema é não considerar — por enquanto — o nome dos alunos e, como alternativa, representá-los por números. Com trinta alunos numa turma, utilizar-se-iam os números 1 a 30.

Usar-se-ia então o comando DIM do *Spectrum*, com uma declaração da forma

```
DIM n(3Ø)
```

(DIM encontra-se na tecla "D".) Esta declaração faz com que o *Spectrum* reserve espaço de memória para um conjunto de 30 variáveis numéricas n(1) a n(3Ø).

Estas variáveis usam-se da mesma forma que qualquer outra variável numérica, com a única restrição de não poderem ser utilizadas como variável de controle de um ciclo FOR NEXT. Mas o importante é que o número entre parênteses (o "índice") pode ser qualquer expressão numérica ou variável. Deste modo,

```
n(7)
n(8-1)
n((p-1)/2)      onde p é 15
```

são, todas, representações válidas da mesma variável.

Isto permite a um programa tratar sucessivmente todas as 30 variáveis recorrendo a um simples ciclo, como por exemplo a rotinazita seguinte, utilizável para introduzir as notas dos alunos:

```
1ØØ FOR a=1 TO 3Ø
11Ø INPUT "Nota do aluno ";(a);n(a)
12Ø NEXT a
```

O conjunto das 30 variáveis n(1) a n(3Ø) é denominado "matriz", "quadro" ou *array*. Há contudo alguns pontos a focar acerca da utilização de um quadro:

— deve-se reservar espaço de memória para o quadro por meio de uma declaração DIM antes de qualquer dos elementos do quadro ser utilizado;
— o nome do quadro (n neste exemplo) não precisa de ter mais de uma letra; como acontece normalmente com nomes de variáveis, o *Spectrum* não faz a distinção entre letras minúsculas e maiúsculas;
— além de reservar espaço de memória, a declaração DIM também fixa em zero o valor de todos os elementos;
— o elemento do quadro ou variável indexada n(p) é uma variável diferente da simples variável numérica n;
— se existirem duas declarações DIM com o mesmo nome num programa, quando a segunda for executada o *Spectrum* retirará da memória o quadro anterior;
— os índices começam sempre em "1". O único limite ao número de elementos de um quadro é o espaço de memória disponível. Como cada elemento ocupa 5 *bytes*, com um programa muito curto podem-se ter até 1600 elementos de quadro num *Spectrum* de 16 K, ou 8200 num de 48 K.

QUADROS NUMÉRICOS MULTIDIMENSIONAIS

Voltando ao nosso papel de professor, vimos como lidar com uma nota por aluno, ou o que, na realidade, constitui uma única coluna de números. Portanto, será compreensível que queiramos agora expandir o nosso programa de forma a que trate outras colunas de resultados — suponhamos, as notas de diversas disciplinas.

Poderíamos utilizar um quadro diferente para cada disciplina, de modo a que

o quadro f(3ϕ) contivesse as notas de Francês
o quadro m(3ϕ) contivesse as notas de Matemática
o quadro i(3ϕ) contivesse as notas de Informática

Mas, para não complicar o programa, é conveniente utilizar um único quadro que tenha dois índices. Se tivéssemos — por exemplo — cinco disciplinas, poderíamos elaborar um quadro desse tipo por meio da declaração

DIM n(5,3ϕ)

Este quadro pode ser considerado como uma folha de papel pautada de forma a ter 5 colunas, cada uma com 30 linhas. O elemento individual do quadro

n(d,a)

corresponde ao quadrado no papel que se encontra na intersecção da coluna d com a linha a ou — no nosso exemplo — à nota do aluno a na disciplina d.

Como temos agora dois índices, a rotina utilizada para introduzir as notas necessita de dois ciclos encaixados:

```
100 FOR d=1 TO 5
110 CLS : PRINT "DISCIPLINA ";d
120 FOR a=1 TO 30
130 INPUT "Nota do aluno ";(a);
n(d,a)
140 NEXT a
150 NEXT d
```

Repare-se que poderíamos igualmente elaborar um quadro com

DIM a(3ϕ,5)

mas, ao localizarmos um determinado elemento deste quadro, teríamos de colocar o número do aluno, a, como primeiro índice, e o número da disciplina, d, como segundo índice, ou seja:

n(a,d)

Poderíamos ir mesmo mais longe, acrescentando um terceiro índice de modo a fazer a distinção entre as notas dos três períodos do ano escolar:

DIM (3,5,3ϕ)

o que poderíamos comparar a três folhas de papel, uma para cada período.

Ou até — se o leitor fosse promovido a presidente do conselho directivo da escola — poderia acrescentar um quarto índice para fazer a distinção entre as várias turmas da sua escola:

DIM (6,3,5,3ϕ)

O exemplo acima aplicar-se-ia, é claro, a uma escola que tivesse 6 turmas, cada uma delas com 30 alunos no máximo.

Quem quiser "impressionar", classifique o quadro anterior como um quadro "a quatro dimensões" ou "quadridimensional". Podem-se utilizar tantas dimensões quantas se quiser com o BASIC do *Spectrum*, e a única restrição existente é o espaço de memória disponível para armazenar todos os elementos.

QUADROS DE CADEIAS ALFANUMÉRICAS

O que tem faltado até aqui ao nosso sistema de registo de notas da escola é a lista dos nomes dos alunos. Não devemos continuar a representá-los por números — ainda estamos relativamente longe do *1984* de George Orwell...

Precisamos, pois, de elaborar um quadro que contenha os nomes dos alunos. Teria de ser um quadro de **cadeias** (*string array*), ou seja, um quadro que contenha cadeias alfanuméricas, o que levanta imediatamente a questão:

Qual o comprimento de uma cadeia?

A declaração DIM não tem qualquer problema em reservar espaço para um quadro numérico, pois cada elemento de um quadro numérico ocupa sempre 5 *bytes*. Contudo, uma cadeia pode ter qualquer comprimento.

O *Spectrum* contorna este problema aceitando que a declaração DIM estabeleça um quadro de **caracteres** se o nome da variável for seguido por um sinal de "$". Cada carácter ocupa um *byte* de memória. Assim,

```
DIM a$(1Ø)
```

define um quadro unidimensional de 10 caracteres, e

```
DIM a$(5,1Ø)
```

define um quadro de 5×10 caracteres. Como nos casos precedentes, utilizaremos quantas dimensões desejarmos, sujeitos apenas às limitações decorrentes do espaço de memória disponível.

Quando se executa uma declaração DIM referente a caracteres, apaga-se qualquer quadro alfanumérico existente que tenha o mesmo nome. Não se pode ter simultaneamente a$ e a$(3) Note-se que, como acontece com um quadro numérico, o nome de um quadro de caracteres tem de ser constituído por uma única letra (embora seguida por um $).

O comando DIM também serve para preencher o quadro alfanumérico (ou de caracteres) com espaços em branco.

Geralmente, temos maior interesse em lidar com cadeias do que com caracteres isolados, e portanto, o *Spectrum* permite-nos tratar partes de um quadro alfanumérico como se fossem cadeias. As regras para o fazer são as seguintes:

— se o último índice de um elemento de um quadro alfanumérico for omitido, o *Spectrum* assume que se está a fazer referência a uma cadeia;
— esta cadeia tem o número de caracteres indicado pelo último índice da declaração DIM.

Por exemplo: se a declaração DIM fosse:


```
DIM a$(3,5)
```

poderia definir três cadeias A$(1) A$(2) e A$(3), cada uma das quais teria cinco caracteres de comprimento.

Um caso especial e bastante útil ocorre quando uma declaração DIM apenas possui um índice; por exemplo

```
DIM a$(32)
```

define um quadro alfanumérico (vazio) de 32 espaços em branco, o qual pode ser usado como uma cadeia de 32 espaços fazendo apenas referência a a$.

Talvez se compreenda melhor o conceito de quadros de cadeias alfanuméricas se se atentar na seguinte rotina:

```
1Ø DIM n$(2,3)
2Ø LET n$(1)="EU"
3Ø LET n$(2)="ELA"
```

O quadro alfanumérico resultante será:

E	U	
E	L	A

O comprimento de cada cadeia é determinado pela declaração DIM. Se lhe atribuirmos um novo valor (ou conjunto de caracteres) que seja demasiado curto, o *Spectrum* preencherá o espaço restante com espaços em branco. Se, pelo contrário, tentarmos atribuir-lhe um valor demasiado longo, o *Spectrum* tomará apenas o número de caracteres indicado pela declaração DIM, ignorando os restantes. Por exemplo, se a linha 20 da rotina acima fosse

```
2Ø LET n$(1)="ELES"
```

então o quadro alfanumérico resultante seria o seguinte:

E	L	E
E	L	A

Voltando às pautas das notas dos nossos alunos, se desejássemos registar os seus nomes no computador, teríamos primeiro de decidir qual o número máximo de caracteres que cada nome poderia utilizar — 20, por exemplo — e depois definir um quadro alfanumérico contendo cadeias de 20 caracteres, como

```
DIM n$(3Ø,2Ø)
```

para registar os nomes dos alunos de uma turma, na forma de cadeias n$(1) a n$(3Ø); ou,

```
DIM n$(6,3Ø,2Ø)
```

no caso de desejarmos registar os nomes dos alunos de seis turmas.

Comparações de cadeias DIMensionadas

O facto de todas as cadeias num quadro de cadeias alfanuméricas terem um comprimento fixo introduz uma complicação quando se deseja comparar duas cadeias, fazendo uma delas parte de um quadro. Isto é consequência de o *Spectrum* não dizer que duas cadeias são iguais a não ser que tenham o mesmo comprimento.

Demonstrando o que isto significa, a rotina

```
1Ø DIM n$(3,5)
2Ø LET x$="OK"
3Ø LET n$(1)=x$
4Ø IF n$(1)=x$ THEN PRINT "COMPREENDIDO"
```

não irá imprimir "COMPREENDIDO" porque as duas cadeias que são comparadas pela linha 40 têm comprimentos diferentes. Mostramo-lo acrescentando a linha seguinte:

```
35 PRINT LEN n$(1), LEN x$
```

(A "função" LEN — por cima da tecla "K" — determina o número de caracteres numa cadeia.)

A forma mais perfeita de ultrapassar este problema consiste em DIMensionar x$ como sendo um quadro de 5 caracteres, acrescentando neste caso a linha

```
15 DIM x$(5)
```

ORDENAÇÃO SIMPLES

Colocada a informação num quadro, tem-se logo, em geral, o desejo de ordená-la (*sort it*). O pequeno programa apresentado a seguir estabelece um quadro de dez cadeias de 12 caracteres, pede ao utilizador dez palavras para armazenar no quadro e, finalmente, apresenta-as ordenadas por ordem alfabética. O conteúdo do quadro vai sendo impresso em cada fase de execução do programa e, portanto, é bastante interessante observar o processo de ordenação.

```
  10 DIM n$(10,12)
 100 FOR a=1 TO 10
 110 INPUT "Palavra ";(a);".";  L
INE n$(a)
 120 PRINT AT a,0;n$(a)
 130 NEXT a
 200 LET c=0
 210 FOR a=1 TO 9
 220 IF n$(a)<=n$(a+1) THEN GO T
O 260
 230 LET t$=n$(a): LET n$(a)=n$(
a+1): LET n$(a+1)=t$
 240 PRINT AT a,0;n$(a);n$(a+1)
 250 LET c=1
 260 NEXT a
 270 IF c=1 THEN GO TO 200
```

O algoritmo utilizado para a ordenação é bastante rudimentar: apenas corre a lista de palavras, analisando um par de cada vez e permutando-as se a sua ordem se encontra trocada (linha 230). Quando chega ao fim da lista, o programa verifica o indicador (*flag*) 'c' de modo a ver se efectuou alguma alteração e, em caso afirmativo, volta a correr a lista de palavras.

Pode-se retardar o ritmo de execução do programa de modo a tornar o processo de ordenação mais visível, acrescentando a linha

```
255 PAUSE 2Ø
```

PAUSA

A palavra de comando PAUSE, que acabámos de utilizar, não deve oferecer dúvidas quanto ao seu significado, e a declaração

```
PAUSE n
```

faz o computador esperar n/50 segundos (n/60 nos Estados Unidos) antes de passar à declaração seguinte do programa, desde que n se encontre no intervalo de 1 a 65535. Se for introduzido um valor não inteiro, o computador arredondá-lo-á para o número inteiro mais aproximado — da mesma forma que procede com outros comandos, como TAB, AT e PLOT, os quais apenas funcionam correctamente com valores inteiros (números inteiros).

E — como diz o manual — a "pausa" terá um fim prematuro se pressionarmos qualquer das teclas (excepto CAPS SHIFT ou SYMBOL SHIFT).

Sendo um pouco mais elaborada, a declaração PAUSE Ø faz com que o computador espere por tempo indeterminado ou até que pressionemos alguma tecla.

Pelo facto de a pausa originada pelo comando PAUSE terminar se premirmos uma tecla, é muitas vezes preferível utilizar um ciclo FOR-NEXT vazio, tal como

```
FOR a=1 TO 1ØØ: NEXT a
```

quando se deseja apenas um retardamento e se quer evitar o risco de alguém precipitar as coisas deixando o dedo sobre uma tecla. De modo aproximado, deve-se fazer o ciclo repetir-se umas 200 vezes por cada segundo de pausa desejado — embora a execução

do ciclo seja mais lenta se ele não se encontrar próximo do início da listagem.

Paradoxalmente, a PAUSE pode também terminar pelo pressionamento de uma tecla que tenha ocorrido antes de a declaração PAUSE ter sido alcançada. Para o verificar basta executar a rotina seguinte

```
1Ø PRINT 1
2Ø FOR a=1 TO 5ØØ: NEXT a
3Ø PRINT 2
4Ø PAUSE 1ØØ
5Ø PRINT 3
```

a qual deverá imprimir "1", esperar uns dois segundos (linha 20) antes de imprimir "2", e esperar mais uns dois segundos (linha 40) antes de imprimir o "3". E esta rotina fará exactamente isto desde que não se toque no teclado enquanto está a ser executada. Mas, se pressionarmos uma tecla depois de o "1" ter sido impresso e antes do "2", ou seja, enquanto o computador está a executar o ciclo da linha 20, a PAUSE seguinte, na linha 40, será anulada e o "3" impresso imediatamente a seguir ao "2".

Uma forma de contornar tal problema consiste em inserir uma curta declaração PAUSE adicional, de modo a "absorver" o efeito de qualquer pressionamento de tecla anterior. Na rotina apresentada acima, conseguimos este efeito alterando a linha 40 para:

```
4Ø PAUSE 1: PAUSE 1ØØ
```

DATA, READ E RESTORE

Se desejarmos preencher um quadro com informação fixa (constante), com as cores do espectro, por exemplo, poderíamos utilizar uma série de declarações LET:

```
1Ø DIM c$(7,8)
2Ø LET c$(1)="violeta"
3Ø LET c$(2)="anil"
etc.
```

Contudo, isto torna-se fastidioso, e o BASIC do *Spectrum* proporciona uma forma melhor de resolver o problema. Esta forma consiste em incluir os elementos de informação (dados) em declarações DATA e depois fazê-los ler (READ) no quadro.

Uma declaração DATA consiste na palavra de comando DATA (por cima da tecla "A") seguida de um ou mais elementos de informação (ou dados) separados por vírgulas, por exemplo:

```
DATA 1,2,"APERTO O CINTO"
```

As declarações DATA colocam-se em qualquer ponto do programa.

Quando fazemos executá-lo, o "indicador de dados" *(data pointer)* é deslocado de modo a indicar a primeira declaração DATA na listagem.

A declaração READ consiste na palavra de comando READ (por cima da tecla "A"), seguida de um ou mais nomes de variáveis, separados por vírgulas:

```
READ a,b,c$
```

Quando o programa executa a declaração READ, atribui a cada variável o valor seguinte da lista de dados, deslocando o indicador de dados de cada vez que o faz. Assim, as duas linhas dadas como exemplo acima resultariam em que

a variável "a" tivesse o valor "1"
a variável "b" tivesse o valor "2"
a variável " c$" tivesse o valor "APERTO O CINTO"

Os elementos de informação, ou dados, podem ser incluídos em qualquer número de declarações DATA, pois o comando READ apenas toma um elemento de cada vez, à medida que aparece na listagem do programa.

Portanto, para preencher o nosso quadro de cadeias com as cores do arco-íris, poderíamos utilizar o seguinte:

```
1Ø DIM c$(7,8)
2Ø DATA "violeta","anil","azul", "verde"
3Ø DATA "amarelo","laranja","vermelho"
4Ø FOR a=1 TO 7: READ c$(a): NEXT a
```


Os elementos de informação que aparecem numa declaração DATA não têm necessariamente de ser constantes, podendo ser expressões de cadeia ou expressões numéricas, como por exemplo:

```
DATA a/a,SQR 4,CHR$ 66 +"APERTO O CINTO"
```

As únicas condições existentes estabelecem que as expressões utilizadas sejam válidas e que uma variável numérica numa declaração READ seja acompanhada de uma expressão numérica compatível na declaração DATA ou que uma expressão de cadeia acompanhe, nas mesmas condições, uma variável de cadeia.

Quando o indicador de dados tenha passado para além do último elemento de informação (ou seja, quando todos os elementos de informação, ou dados, tenham sido lidos), qualquer tentativa para fazer ler outro elemento dará lugar à seguinte mensagem de erro:

```
Out of data
```

a qual se pode traduzir aproximadamente por "dados esgotados".

A declaração RESTORE utiliza-se em qualquer altura, com o fim de deslocar o indicador de dados, e assim afectar o elemento a ler em seguida.

```
RESTORE
```

por si só desloca o indicador de dados para o primeiro elemento de informação da primeira declaração DATA do programa.

```
RESTORE n
```

desloca o indicador de dados para o primeiro elemento de informação contido na linha n ou para o primeiro elemento de informação da primeira declaração DATA que se encontre depois da linha n, no caso de não existir qualquer linha n ou se a linha n não contiver nenhuma declaração DATA.

Ao contrário do que diz o manual do *Spectrum*, CLEAR não efectua qualquer RESTORE automático.

ÁRABE – ROMANO

Constituindo um exemplo da utilização de quadros e de uma declaração DATA, o programa seguinte efectua a conversão de números representados no nosso habitual sistema de numeração "árabe" nos seus equivalentes "romanos". Por exemplo:

1982 em MCMLXXXII

As linhas 100 a 130 determinam os dados fixos a utilizar pelo programa (r$ serve como um quadro de sete caracteres). O ciclo da linha 230 à linha 270 exerce seguidamente uma verificação para determinar qual dos sete dígitos romanos deverá ser impresso de cada vez que se executa o ciclo.

```
  50 REM "ARABE-ROMANO"
 100 DIM r(9)
 110 DATA 1000,500,100,50,10,5,1
,0,0
 120 FOR x=1 TO 9: READ r(x): NE
XT x
 130 LET r$="MDCLXVI"
 200 INPUT "Numero (0 para parar
) ? ";a
 210 IF a=0 THEN STOP
 220 PRINT a;" = ";
 230 FOR x=1 TO 7
 240 IF a>=r(x) THEN PRINT r$(x)
;: LET a=a-r(x): GO TO 240
 250 LET y=1+2*INT ((x+1)/2)
 260 IF a>=r(x)-r(y) THEN PRINT
r$(y);: LET a=a+r(y): GO TO 240
 270 NEXT x
 300 GO TO 200
```


Capítulo 6

Criação de um programa

A CANÇÃO DE TRIUNFO DO PROGRAMADOR

A primeira versão que escrevi foi a que pus no mercado.
A segunda já funcionava. Mas eu era então arrojado.
E reformulei-a outra vez. E mais outra. E ainda outra vez.
Cada versão era melhor do que a anterior de forma drástica.
E agora é tão curta, tão elegante e tão prática
Que não é por dólares que se pode comprar, mas por Yen *talvez.*

Apesar de ser divertido passar programas escritos por outras pessoas, elaborar os nossos próprios programas é bem mais compensador. É imenso o sentimento de satisfação que experimentamos quando, por fim, as nossas primeiras e vagas ideias se tornam num programa sólido e funcional. Para facilitar um pouco o processo ao iniciado na "arte", este capítulo debruça-se sobre alguns aspectos da criação de um bom programa.

PLANEAMENTO

O primeiro ponto, e talvez o mais importante, é que os programas bem sucedidos não são *escritos* na forma em que se escreve uma carta a um amigo, começando por "Caro Fulano" e continuando a carta cobrindo os tópicos à medida que eles vêm à cabeça, colocando talvez um *P. S.* no fim, se nos esquecemos de mencionar qualquer coisa no texto principal da carta.

Os programas devem ser *criados* de uma forma muito semelhante à de uma obra de arte ou de um grande edifício: começando por um plano geral e aperfeiçoando-o até haver a

certeza de que a estrutura básica está correcta antes de avançar com trabalho mais minucioso. A tentação de introduzir imediatamente algumas linhas de programa no computador deve ser contrariada até que se tenha um plano do que se quer escrever.

A razão de ser deste procedimento reside no facto de qualquer programa útil e eficiente incluir tantos pormenores que, uma vez que se comece a trabalhar nesses pormenores, é provável que se verifique ser impossível manter em mente o conjunto. Assemelha-se um pouco à impossibilidade de ver uma floresta por causa das árvores. O computador avança através da floresta do programa deslocando-se simplesmente de uma árvore para a outra, mas, a não ser que o utilizador veja a floresta em conjunto, globalmente, como pode ele ter a certeza de que o computador vai na direcção correcta?

O ALGORITMO E OS DADOS

"Algoritmo" é uma forma elaborada de designar o meio pelo qual se efectua uma tarefa. Existem geralmente várias formas de resolver um problema, e o leitor deverá tentar encontrar uma que se adapte à maneira pela qual os computadores funcionam "naturalmente". Por exemplo, o ciclo FOR NEXT é uma ferramenta muito poderosa em BASIC, e portanto vale a pena tentar encontrar um algoritmo que funcione fazendo repetir muitas vezes uma operação simples.

Analogamente, a informação ou, mais precisamente, os "dados", devem ser tratados de uma forma muito "regular". Devem procurar-se analogias entre diferentes tipos de dados e seguidamente tirar partido delas de modo a simplificar o programa.

Em programas que lidem com uma quantidade apreciável de informação, vale sempre a pena dedicar algum tempo a estudar a melhor forma de a ordenar. De facto, o sucesso da maior parte dos programas comerciais de gestão "reais" reside, quase inteiramente, na forma por que a informação é organizada, de modo a ser manipulada com a maior facilidade.

FLUÊNCIA

É uma coisa que só aparece com a prática. Deve-se praticar escrevendo os nossos próprios programas, estudando os que outras pessoas escreveram e vendo como elas resolvem os problemas e pensando como os resolveríamos nós.

VELOCIDADE CONTRA ESPAÇO CONTRA SENSO

Estes três objectivos entram geralmente em conflito. A maior parte dos programas elaborados de modo a obter uma execução rápida ocupam em geral grandes áreas de memória, e os que se destinam a caber num espaço limitado da RAM são normalmente mais lentos. Aumentar a velocidade ou diminuir o espaço de memória necessário a um programa envolve, muitas, vezes, a utilização de certos artifícios de programação que tornam a listagem de compreensão difícil.

De modo geral, o *Spectrum* tem rapidez e capacidade de memória suficientes para a maior parte das aplicações; é portanto conveniente começar por procurar a clareza e só deixar esse rumo quando se é forçado por imposições de espaço ou de velocidade.

A TERCEIRA VERSÃO

Na prática, escrever um programa revela-se uma tarefa bem mais complicada do que começar simplesmente com um esquema genérico do que se pretende e ir, gradualmente, acrescentando os pormenores até se obter um programa plenamente operacional.

É vulgar acontecer chegarmos a meio da elaboração de um programa e aparecer um problema qualquer grave, o que significa termos de voltar ao princípio, à "estaca zero". E, quando por fim completamos o programa, descobrimos geralmente que o podíamos ter escrito de maneira melhor.

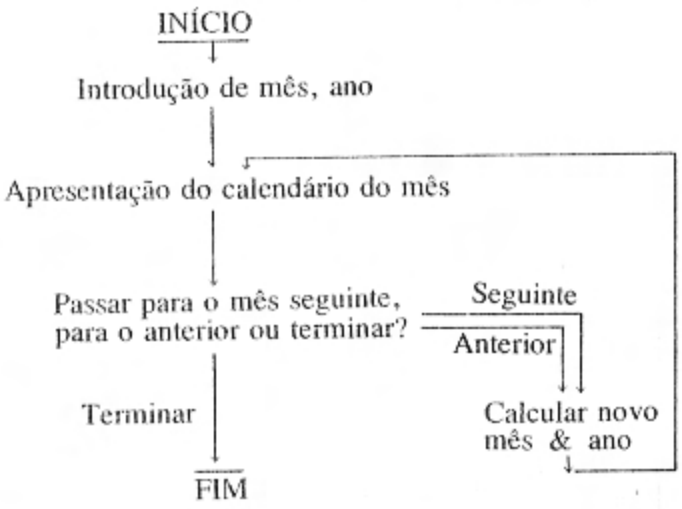
Por esta razão, os melhores elementos (ou "peças", segundo certas terminologias) de *software* foram geralmente escritos e revistos duas ou três vezes, tendo passado por outras tantas versões e levado várias vezes o tempo previsto para os completar.

Em grande parte dos casos isto é inevitável, pois o projecto de programas é um processo iterativo (e aqui está uma frase para baralhar quem o criticar), apesar de se evitar uma quantidade apreciável de trabalho de revisão e correcção se traçarmos de início um plano global e adequado.

CALENDÁRIO: UM EXEMPLO

Para ilustrar as ideias expostas na secção anterior, vamos escrever um programa para imprimir um calendário.

Em primeiro lugar, pensemos um pouco acerca do que o programa se destina a fazer. Não poderíamos certamente apresentar o calendário para um ano inteiro, de uma só vez, no *écran* (não há espaço para tal), mas há espaço suficiente para apresentar o calendário de um mês. O programa, portanto, deve "perguntar" ao utilizador qual o mês e de que ano deseja ver apresentado no *écran*. Para facilitar a utilização, podemos incluir um "dispositivo" que — depois de apresentado um mês — permita ao utilizador fazer avançar ou recuar um mês de cada vez a imagem apresentada. Para evitar uma estrutura de "ciclo sem fim", este "dispositivo de escalonamento" (*stepping feature*) deverá também permitir que o utilizador pare o programa de uma forma elegante.



A estrutura geral do programa terá, portanto, um aspecto semelhante ao da gravura da página anterior.

O "bloco" "introdução de mês, ano" é suficientemente simples para não necessitar de qualquer outro desenvolvimento antes de escrevermos as linhas de programa definitivas. Para simplificar as coisas, o mês pode ser introduzido sob a forma de um número, e o programa verificará se esse número se encontra entre 1 e 12.

Para tornar o programa tão útil quanto possível, deveríamos concebê-lo para funcionar para qualquer ano, mas a mudança do calendário juliano para o calendário gregoriano complica o assunto. Esta mudança, que determinou a "perda" de vários dias, ocorreu em datas diferentes, consoante o país, desde 1572 até 1917, data em que todos os países que aceitaram a mudança "acertaram" o calendário. Em Portugal essa mudança ocorreu em 1582 e, portanto, o programa foi elaborado de modo a aceitar qualquer ano a partir de 1582.

A apresentação no *écran* do calendário de um mês passa por três fases:

Limpar o *écran* e imprimir cabeçalhos
↓
Calcular o dia da semana correspondente ao primeiro dia do mês
↓
Imprimir os dias do mês

Para calcular o dia da semana que corresponde ao primeiro dia do mês, podemos utilizar o seguinte algoritmo:

```
LET D=3+Y+INT((Y-1)/4) - INT((Y-1)/100)
LET D=D + número total de dias dos anteriores meses do ano
LET D= D-7*INT(D/7)
```

onde o valor final de D é o dia da semana (0 - 6) e Y é o ano (*Year*).

O número total de dias dos anteriores meses do ano pode ser calculado por um único ciclo FOR-NEXT:

FOR a=1 TO M−1: LET D=D + dias do mês **a** : NEXT a

onde M é o número (1 - 12) do mês que queremos ver apresentado no *écran*. Repare-se que este algoritmo se baseia no facto de o *Spectrum* não executar declarações dentro de um ciclo FOR NEXT se o valor inicial da variável de controle for superior ao valor final, tal como acontece aqui quando M=1 (Janeiro).

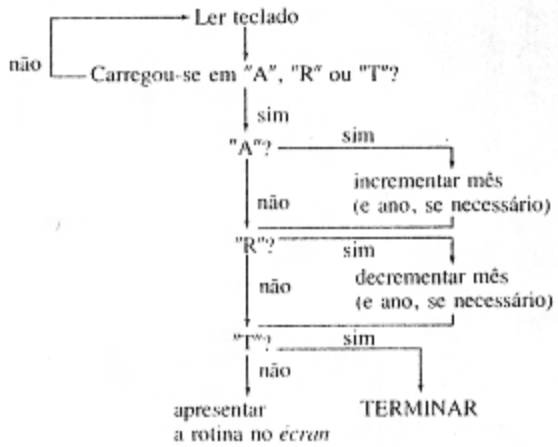
Consegue-se fazer imprimir os dias do mês utilizando um outro ciclo FOR-NEXT destinado a imprimir os números de 1 ao número de dias do mês, com uma ordem TAB incluída na declaração PRINT para imprimi-los nas posições correctas:

Voltando ao nosso plano inicial, os elementos (ordens, etc.) restantes são os que permitem recuar ou avançar um mês de cada vez ou terminar a execução do programa.

Vamos utilizar a função INKEY$ para isto, de modo a que o utilizador apenas tenha de pressionar uma tecla. As teclas que escolhemos foram:

A para *a*vançar para o mês seguinte
R para *r*ecuar para o mês anterior
T para *t*erminar

Temos de utilizar algumas declarações IF para fazer com que o programa reaja correctamente ao que escrevermos, e uma solução possível originará um fluxograma de forma semelhante à seguinte:



Por fim, devemos considerar a forma pela qual a informação vai ser incluída no programa. Além das variáveis simples, como as que representam o dia, o mês e o ano, o programa necessita de uma lista do número de dias de cada mês. Além disso, seria interessante se o programa apresentasse no *écran* o nome do mês (em vez de o representar simplesmente por um número) — ou, pelo menos, apresentar uma abreviatura de três letras do dito nome — portanto, o programa necessitará também de uma lista desses nomes.

Poderíamos definir quadros de 12 elementos para colocar os 12 números e os 12 nomes. Teríamos então de preencher os elementos do quadro se o programa começasse por declarações DATA e READ, ou mesmo por uma enfiada de declarações LET. Mas, para quantidades de informação relativamente pequenas, é mais fácil armazenar tudo na forma de uma cadeia e utilizar as funções de partição de cadeias para extrair os bocados desejados. Assim, na listagem do programa definitivo apresentada a seguir, m$ contém 12 abreviaturas de três letras representando os nomes dos meses e n$ os 12 números de dois dígitos que dão o número de dias de cada mês.

```
  5 REM "CALENDARIO"
 10 LET m$="JanFevMarAbrMaiJunJ
ulAgoSetOutNovDez"
 20 LET n$="312831303130313130303
13031"
 100 INPUT "Ano (>1582)? ";y: IF
 y<1583 THEN GO TO 100
 110 INPUT "Mes (1-12)? ";m: IF
m<1 OR m>12 THEN GO TO 110
 200 CLS : PRINT AT 4,8;m$(3*m-2
 TO 3*m);TAB 18;y
 210 PRINT AT 7,2;"Dom Seg Ter Q
ua Qui Sex Sab"
 220 LET n$(3 TO 4)="28"
 230 IF y=4*INT (y/4) AND y<>100
*INT (y/100) THEN LET n$(3 TO 4)
="29"
 240 LET d=3+y+INT ((y-1)/4)-INT
 ((y-1)/100)
 250 FOR a=1 TO m-1: LET d=d+VAL
 n$(2*a-1 TO 2*a): NEXT a
 260 LET d=d-7*INT (d/7)
 300 PRINT AT 9,0;
 310 FOR a=1 TO VAL n$(2*m-1 TO
2*m)
```

```
 320 LET r=2+4*(a+d-7*INT ((a+d)
/7))
 330 PRINT TAB r;a;
 340 NEXT a
 400 PRINT AT 20,0;"Carregue na
tecla 'A' 'R' ou 'T'"
 410 LET i$=INKEY$
 420 IF i$<>"a" AND i$<>"r" AND
i$<>"t" THEN GO TO 410
 430 IF i$="a" THEN LET m=m+1
 440 IF m>12 THEN LET m=1: LET y
=y+1
 450 IF i$="r" THEN LET m=m-1
 460 IF m<1 THEN LET m=12: LET y
=y-1
 470 IF i$<>"t" THEN GO TO 200
```

As linhas 10 e 20 definem os dados constantes (fixos).

As linhas 100 e 110 obtêm do utilizador o ano e o mês e verificam a sua validade (isto é: se estão dentro dos limites aceites pelo programa).

As linhas 200 - 260 imprimem os cabeçalhos correspondentes a um mês e calculam o dia da semana que corresponde ao primeiro dia desse mês.

As linhas 300 - 340 imprimem o número de dias do mês.

As linhas 400 - 470 permitem ao utilizador passar ao mês seguinte ou ao mês anterior, ou então terminar o programa (quando se quer passar a outro ano, por exemplo, carrega-se em "T"; o programa termina e faz-se executá-lo novamente).

```
          Dez        1999

  Dom  Seg  Ter  Qua  Qui  Sex  Sab

                    1    2    3    4
    5    6    7    8    9   10   11
   12   13   14   15   16   17   18
   19   20   21   22   23   24   25
   26   27   28   29   30   31

Carregue na tecla 'A' 'R' ou 'T'
```


DEPURAÇÃO

Ocasionalmente, verifica-se que um programa elaborado por nós não funciona correctamente ao fazer executá-lo pela primeira vez. Não é caso para preocupações, pois é provável que a culpa não seja nossa mas, sim, desses perniciosos "parasitas" (*bugs*, em inglês, cuja tradução mais literal seria "percevejos") que tenham invadido o programa. O processo destinado a encontrar e eliminar esses "parasitas" ou *bugs* é conhecido genericamente por *de-bugging*, que deveremos traduzir por "depuração".

É principalmente um processo de dedução lógica, começando por se investigar a partir do erro, recuando no programa para se descobrir o que levou a esse erro. As técnicas destinadas a efectuá-lo incluem nomeadamente:

— acrescentar declarações PRINT adicionais em locais judiciosamente escolhidos de forma a permitir-nos saber como as variáveis principais estão a "funcionar" ou que caminho o programa está a levar (esta é uma boa razão para deixar intervalos entre os números de linha);
— acrescentar declarações STOP em pontos "estratégicos" do programa (há programadores com bons conhecimentos que usam habilmente declarações IF – THEN STOP, dispostas de forma a fazer parar o programa quando necessário);
— retirar temporariamente linhas de programa críticas; a melhor maneira de o fazer consiste geralmente em acrescentar uma palavra de comando REM logo a seguir ao número de linha.

Uma vez que o programa tenha parado — seja por que razão for —, as declarações LET e PRINT utilizadas em modo imediato (ou seja, sem um número de linha) são muito úteis para examinar e modificar os valores das variáveis. Feito isto, uma palavra de comando CONT permitirá continuar a execução do programa a partir do ponto em que este tenha parado, ou então um GOTO fá-lo-á arrancar de novo em qualquer outra linha.

Capítulo 7

A sorte é uma bela coisa

Os computadores são fundamentalmente máquinas precisas. Desprovidas de imaginação. Se perguntarmos a um deles quanto é seis vezes sete, obteremos sempre a resposta 42. Esta é a sua grande força.

Mas isto também é uma fraqueza, especialmente quando se utilizam em interacção com seres humanos numa base "criativa". Quer o utilizemos em jogos ou para criar "arte", a coisa que menos queremos obter do computador é uma resposta monótona e mecânica. E no campo da inteligência artificial é, muitas vezes, útil conseguir que computador actue de uma forma imprevisível.

Por isso, o BASIC do *Spectrum* inclui a função RND. Esta função permite obter uma fracção decimal aleatória *(random)* entre zero e, muito aproximadamente, um. Ou seja, de 0,00000... a 0.99999...

É claro que, na realidade, o *Spectrum* não possui um cérebro próprio, e não fornece efectivamente números na verdade aleatórios. Em vez disso — tal como se descreve no manual do *Spectrum* —, gera uma longa sequência de números, de tal forma que não existe qualquer relação aparente entre um número e o seguinte. Sempre que utilizamos a função RND, o computador apresenta o número seguinte da lista. Rigorosamente falando, deveríamos chamar-lhes "números pseudo-aleatórios". São, contudo, suficientemente aleatórios para todas as aplicações práticas.

CLIVE

O computador que faz a extracção dos números premiados na lotaria inglesa, a British Premium Bond, chama-se *Ernie*. Por o programa apresentado a seguir efectuar uma função semelhante, decidimos dar-lhe igualmente um nome de sonância humana.

Este programa extrai um número premiado dos números situados entre (e possivelmente incluindo) um limite inferior e um limite superior, escolhidos e introduzidos pelo utilizador. Se

desejarmos apresentar o *Spectrum* em alguma festa escolar ou acontecimento semelhante, deveremos levar connosco uma *cassette* com este programa; certamente será útil.

Extrair um número é fácil — veja-se a linha 220 — e a maior parte do programa destina-se a apresentar uma imagem interessante que retenha a atenção dos espectadores e a criar um ambiente de emoção crescente enquanto o número está a ser escolhido. Para ajudar a produzir este efeito, o programa apresenta uma sequência de 100 números aleatórios, abrandando gradualmente até que a escolha final é assinalada por uma margem *(border)* intermitente. Se o leitor já domina os comandos de cor, de som, BRIGHT e FLASH, porque não incluí-los no programa de modo a provocar um entusiasmo ainda maior?

```
  50 REM "CLIVE"
 100 RANDOMIZE
 110 CLS
 120 INPUT "Numero menor ";a;" M
aior ";b
 130 IF a>=b THEN GO TO 120
 200 FOR c=1 TO 100
 210 CLS
 220 PRINT AT 10,12;INT (a+(1+b-
a)*RND)
 230 PAUSE 1+c/4
 240 NEXT c
 300 FOR c=1 TO 10
 310 BORDER 0: PAUSE 10
 320 BORDER 7: PAUSE 10
 330 NEXT c
 400 PRINT AT 21,0;"Outro jogo ?
"
 410 LET d$=INKEY$: IF d$="" THE
N GO TO 410
 420 IF d$="s" OR d$="S" THEN RU
N
```

100-130 Inicializam, obtêm os limites superior e inferior e verificam se esses números são praticáveis.

200-240 Apresentam 100 números aleatórios.

300-340 Anunciam o número premiado.

400-420 Permitem ao utilizador fazer executar o programa novamente, extrair outro número premiado e, além disso, evitam que a mensagem "Ø/ - -", que talvez causasse certa confusão, apareça na parte de baixo do *écran* antes de o programa deixar de ser utilizado.

COMO RND

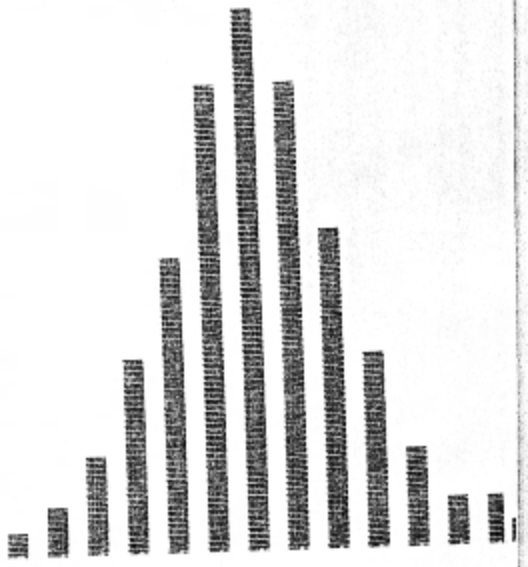
Se vamos contar com a função RND do *Spectrum*, valerá a pena ver quão aleatórios são os números por ela obtidos.

Este programa efectua um teste bastante simples, utilizando RND para apresentar uma série de números inteiros aleatórios de 1 a 16, fazendo um registo de quantas vezes cada determinado número inteiro é apresentado e traçando um gráfico de barras (histograma) dos resultados.

```
 10 DIM a(16)
 20 LET s=.1
 30 RANDOMIZE
100 LET b=1+INT (16*RND)
110 LET a(b)=a(b)+s
120 IF a(b)>21 THEN STOP
130 PRINT AT 21-a(b),2*b-1;"■"
140 GO TO 100
```

A variável "s" é utilizada como um «factor de escala»: quanto menor for o valor dado a s na linha 20, maior é a quantidade de números aleatórios que podem ser acumulados antes de o programa parar por uma das barras do gráfico ter chegado ao extremo superior do *écran*. Verifique-se em que medida a variação de s afecta o resultado.

O histograma apresentado à direita foi conseguido por uma versão modificada do programa, a qual obteve números aleatórios de 1 a 16 segundo uma distribuição "gaussiana". Esta versão foi conseguida pela substituição da linha 100 do programa anterior pelas seguintes quatro linhas:




```
100 LET c=0: FOR d=1 TO 12: LET
c=c+RND: NEXT d
102 LET sdev=2: LET mean=7.5: L
ET b=INT (1+(c-6)*sdev+mean)
104 IF b<1 THEN LET b=1
106 IF b>16 THEN LET b=16
```

As linhas 104 e 106 evitam que o programa falhe *(crash)* se forem gerados valores extremos, e esta versão modificada do programa baseia-se na fórmula seguinte:

$$GRND = (SRND - 6)*SDEV - MEAN$$

onde GRND *(Gaussian RaNDom number)* é um número aleatório com distribuição gaussiana (mais correctamente: uma variável aleatória com distribuição normal ou gaussiana), SDEV *(Standard DEViation* — desvio padrão) e MEAN (média) são valores parametrizantes para a obtenção dos valores do GRND, e SRND é a soma de 12 números aleatórios situ- los entre 0 e 1.

PALAVR?? CRUZA???

Como um exemplo da utilização de RND para a extracção aleatória de letras experimente-se este programa, o qual foi elaborado para o apaixonado pelas palavras cruzadas ou para alguém que procure um nome para um novo programa.

Basta introduzir uma palavra (ou uma mnemónica) com as letras incertas substituídas por pontos de interrogação, e o programa apresentará uma coluna repleta de palavras sugeridas — substituindo os pontos de interrogação por letras escolhidas aleatoriamente. Quando o *écran* ficar cheio (em altura), o inquiridor 'scroll?' aparecerá no canto inferior esquerdo; carregue-se pois na tecla "N" ou BREAK se se quiser terminar, ou então em qualquer outra tecla se se desejar outra coluna de palavras.

No entanto, deve ter-se cuidado quanto à pessoa que utiliza o novo programa. Muitas crianças egotistas divertem-se simplesmente a observar variações do seu próprio nome apresentadas no *écran*, mas tem-se conhecimento de outras que utilizam o

programa para apresentar "acidentalmente" palavras "pouco educadas"!

```
100 INPUT "Palavra ? ";p$
110 FOR a=1 TO LEN p$
120 LET c$=p$(a)
130 IF c$="?" THEN LET c$=CHR$
(65+25*RND)
140 PRINT c$;
150 NEXT a
160 PRINT
170 GO TO 110
```

ROLETA RUSSA

Em alguns programas, desejamos que determinado acontecimento ocorra apenas ocasionalmente, não como resultado de alguns cálculos ou em resposta a um determinado *input*, mas simplesmente ao acaso, aleatoriamente.

Um exemplo disto é o "jogo" tradicional da roleta russa, no qual um grupo de jogadores passa de um a outro um revólver de seis tiros carregado com uma única bala no tambor. Quando cada um dos jogadores recebe o revólver, faz rodar o tambor (como uma roleta), aponta o revólver à cabeça e aperta o gatilho. Há portanto uma probabilidade em seis de o revólver disparar. O jogo termina — dependendo das regras acordadas — quando resta apenas um ou mesmo nenhum dos jogadores.

Na versão (computorizada) deste "jogo", o *Spectrum* decide se o premir do gatilho se revelará ou não fatal. A linha 340 dá um resultado inofensivo quando o número aleatório gerado é superior a um sexto ou, pondo a coisa de outro modo, ela "mata" em média um jogador por cada seis tiros (o que está de acordo com a probabilidade indicada atrás).

Este programa ilustra igualmente a utilização de *flags* ("bandeiras", "etiquetas", mas, neste caso mais precisamente, "indicadores") em programação. Estes indicadores são variáveis utilizadas, não para guardar um valor determinado, mas para indicar se um determinado acontecimento ocorreu ou não anteriormente. Por exemplo, o quadro p() regista quais os jogadores que foram já "mortos" durante o jogo: todos os elementos de p() são inicialmente passados a zero pela declaração DIM na linha 150 e,

à medida que cada jogador "morre", o elemento correspondente do quadro passa a 1 pela linha 360. Isto permite ao programa evitar o embaraço resultante de pedir a jogadores "mortos" que joguem na sua vez (linha 220).

Analogamente, a variável f é utilizada para decidir se vale ou não a pena continuar o jogo, verificando se existe ainda algum jogador "vivo" (linhas 200, 230, 410).

```
 50 REM "ROL.RUSSA"
100 RANDOMIZE
110 DIM z$(15)
120 PRINT AT 10,8;"<<ROLETA RUS
SA>>"
130 INPUT "Numero de jogadores
? ";n
140 IF n<1 THEN STOP
150 DIM p(n)
200 LET f=0
210 FOR a=1 TO n
220 IF p(a)=1 THEN GO TO 390
230 LET f=1
300 CLS : PRINT AT 6,2;"Jogador
 ";a;"'"E'a sua vez de"'"enfrent
ar"'"a morte !"
310 PRINT AT 8,27;"▗▄▄▖";AT 9,
30;"▝▀▘"
320 PRINT '"Pressione ENTER"'"p
ara disparar": INPUT LINE i$
330 PRINT AT 8,0;z$'z$'z$'z$'z$
340 IF RND>1/6 THEN PRINT AT 8,
20;"click": GO TO 380
350 PRINT AT 8,20;"▗ ▖";AT 9,1
9;"▝▄▄▘";AT 9,0;"<< BANG !!>>"
360 LET p(a)=1
380 PAUSE 50
390 NEXT a
400 IF f=1 THEN GO TO 200
405 LET d$="os": IF n=1 THEN LE
T d$="o"
410 PRINT AT 14,1;"--Tive muito
 gosto em ";d$;" ter--";AT 16,11
;"conhecido"
```

100-150 Inicializam, obtêm o número de jogadores. A linha 110 é uma forma prática de gerar uma cadeia de 15 espaços, a qual pode ser utilizada posteriormente pela linha 330 para apagar parte da imagem.

200-390 Fazem passar uma vez o revólver por todos os jogadores, dando a cada um a vez de jogar.

400-410 Voltam a fazer passar o revólver uma outra vez, se ainda houver jogadores "vivos"; caso contrário, imprimem uma educada e cínica despedida.

Nota: os caracteres "invulgares" entre aspas, nas linhas 310 e 350, são introduzidos utilizando as seguintes sequências de teclas:

Linha 310, primeira parte:
SHIFT/GRAPHICS 5 SHIFT/3 SHIFT/3 SHIFT/8 SHIFT/8 9
segunda parte:
SYMBOL SHIFT/7 SHIFT/GRAPHICS SHIFT/8 9

Linha 350, primeira parte:
SHIFT/GRAPHICS 4 SPACE SPACE SHIFT/3 9
segunda parte:
SHIFT/GRAPHICS SHIFT/8 SHIFT/2 SHIFT/3 SHIFT/3 SHIFT/2 9

CARTA ALTA; CARTA BAIXA

Vimos anteriormente como utilizar RND para extrair um número de um intervalo limitado mas, frequentemente, deseja-se que o computador seleccione um elemento de uma lista de coisas, não de números. Por exemplo, neste programa, o *Spectrum* tem de extrair uma carta de jogar e, embora algumas cartas sejam numeradas, outras (valete, dama, rei) não o são.

A forma de efectuar este tipo de selecção é elaborar uma lista dos elementos, fazer depois com que o programa extraia um número, e utilizar o elemento que ocupe na lista a posição com esse número. Neste programa, a linha 10 organiza uma lista de 13 cartas sob a forma de uma cadeia de 13 caracteres (23456789DJQKA, sendo D: 10; J: valete; Q: dama ou rainha; K: rei; e A: ás), e o computador pode então seleccionar uma carta gerando um número aleatório entre 1 e 13 e extrair da cadeia o caracter que representa a carta, como na linha 210.

A pessoa joga contra o computador, o qual tira duas cartas do conjunto das 13 e mostra uma delas. O jogador tem então de adivinhar se a segunda carta (não apresentada) é mais alta ou

mais baixa do que a primeira e fazer uma aposta. Começa o jogo com 1000$00, e este termina quando o jogador está falido ou, pelo contrário, leva a banca à falência por ganhar um total superior a 99 999$00.

```
  5 REM "CARTAS"
 10 LET c$="23456789DJQKA"
 20 RANDOMIZE
 30 LET m=1000
100 LET x=4+INT (8*RND)
110 LET y=1+INT (13*RND): IF y=
x THEN GO TO 110
120 CLS : PRINT AT 2,0;"Eu esco
lhi duas cartas :"
130 PRINT AT 5,6;c$(x);TAB 16;"
"
140 INPUT "A segunda carta e'ma
is alta (a)"'"ou mais baixa (b)
do que a"'"primeira ? "; LINE i$
150 IF i$<>"a" AND i$<>"b" THEN
GO TO 140
160 LET d$="mais alta": IF i$="
b" THEN LET d$="mais baixa"
170 PRINT AT 8,0;"V. pensa que
a segunda e'"'d$''"O seu saldo e
 de ";m;"$00"
180 INPUT "Quanto e'que aposta
? ";e: IF e<0 OR e>m THEN GO TO
160
190 PRINT "Aposta ";e;"$00": IF
 e=0 THEN GO TO 300
200 FOR p=1 TO 200: NEXT p
210 PRINT AT 5,16; FLASH 1;c$(y
)
220 IF (i$="a" AND y>x) OR (i$=
"b" AND y<x) THEN GO TO 250
230 LET m=m-e: PRINT AT 13,0;'"
Pouca sorte!": IF m>0 THEN GO TO
 270
240 PRINT '"Esta'falido !!": ST
OP
250 LET m=m+e: PRINT AT 13,0;'"
Bom palpite!"
260 IF m>99999 THEN PRINT "Levo
u a banca a falencia !!": STOP
270 PRINT '"O seu saldo e'agora
 de ";m;"$00"
300 PRINT '"Carregue em ENTER"
'"para nova aposta": INPUT LINE
i$: GO TO 100
```

10- 30 Fixam os valores iniciais.
100-130 O computador escolhe duas cartas (x e y) e apresenta a carta x.
140-190 Recebe a aposta do jogador e verifica a sua validade (se é maior que zero e inferior ou igual ao saldo presente do jogador).
200-270 Apresenta a carta y, calcula e imprime os resultados. A linha 200 apenas introduz um ligeiro atraso de modo a aumentar a "tensão".

"HAIKU" DE PÉ-QUEBRADO

Haiku é um género de poesia japonesa com uma estrutura essencialmente metafórica e que consiste em geral num certo número fixo de sílabas (normalmente 17) dispostas em 3 linhas (5, 7, 5) ou mesmo em 4 linhas, e cujo significado é apenas dado por sugestão.

O programa apresentado a seguir produz uma cadeia interminável da pior espécie de poesia *Haiku* de 4 linhas, imprimindo uma frase aleatória extraída, à vez, de cada um de quatro grupos de frases.

Este programa ilustra outra forma de extrair aleatoriamente um elemento não numérico de um grupo, colocando os elementos em declarações DATA, lendo um número aleatório de elementos (como, p. ex., na linha 120) e utilizando o que tiver sido lido em último lugar.

Os quatro grupos de frases estão contidos em quatro grupos de declarações DATA, cada um deles começando, respectivamente, na linha 1000, 1100, 1200 e 1300. O grupo específico a utilizar é seleccionado pelas linhas 100 e 110.

Pode tirar-se bom partido de um programa deste tipo, compondo as nossas próprias listas de frases e divertindo-nos a observar os resultados. Estes resultados não têm de ser necessariamente "poesia": podem perfeitamente ser constituídos por cabeçalhos jornalísticos, jargão pseudotécnico ou mesmo discursos políticos. Quanto mais longas forem as listas, de maior efeito será o resultado, mas, para que o programa funcione correctamente, cada um dos quatro grupos deve conter o mesmo número

de frases, e a linha 10 deve ser alterada de acordo com esse número.

```
  5 REM "HAIKU"
 10 LET m=6
 20 RANDOMIZE
100 FOR a=1000 TO 1300 STEP 100
110 RESTORE a
120 FOR b=1 TO 1+m*RND: READ c$
: NEXT b
130 PRINT c$
140 NEXT a
150 POKE 23692,0
160 PAUSE 200
170 PRINT ''
180 GO TO 100
1000 DATA "Gotas de orvalho","Ce
u anil","Passaros em bando","Doi
s homens"
1010 DATA "Seara ondulante","Gat
os"
1100 DATA "Junto ao lago","O ven
to suspira","Cantam cigarras"
1110 DATA "Nuvens","Borboletas",
"Ao longo do caminho"
1200 DATA "Com a alvorada","Fumo
 ao longe","Musica distante"
1210 DATA "O sol desponta","Bamb
u verdejante","Uma pequena flor"
1300 DATA "O dia sobe lentamente
","Por sobre as montanhas"
1310 DATA "Enquanto o passado se
 vai","As criancas brincam na lu
z"
1320 DATA "Arados esquecidos na
erva","As arvores florescem por
fim"
```

10- 20 Fixam os valores iniciais.
100-140 Fazem imprimir uma frase extraída à vez de cada um dos grupos.
150-180 Permitem à imagem "rolar" *(scroll)*, parar um momento à espera de aplauso e começar a criação seguinte.

CRIPTOGRAFIA

Este programa é indispensável para espiões de Cambridge, Omsk ou de qualquer grande multinacional, pois codifica qualquer mensagem num código virtualmente inviolável. Para a descodificar, é necessário conhecer a "chave" *(key)* utilizada quando a mensagem foi codificada, assim como ter acesso a um *Spectrum*.

Cada carácter da mensagem é codificado (e descodificado) utilizando uma chave diferente tirada de uma sequência gerada pela função RND. O operador introduz um número inicial (que deve estar compreendido entre 1 e 65535), e esse valor é utilizado pela declaração RANDOMIZE na linha 140 para determinar a sequência correcta quando se codifica ou descodifica uma mensagem.

```
  50 REM "CRYPTO"
 100 INPUT "Introduza D para des
codificar, C para codificar "; L
INE e$
 110 IF e$<>"c" AND e$<>"C" AND
e$<>"d" AND e$<>"D" THEN GO TO 1
00
 120 INPUT "Introduza a chave"'"
do codigo (1-65535) ";k
 130 IF k<1 OR k>65535 THEN GO T
O 120
 140 RANDOMIZE k
 150 INPUT "Introduza a mensagem
:"' LINE i$: PRINT i$
 200 FOR a=1 TO LEN i$
 210 LET c=CODE  i$(a): IF c<32
OR c>127 THEN GO TO 250
 220 LET x=96*RND: IF e$="d" OR
e$="D" THEN LET x=-x
 230 LET c=c+x: IF c>127 THEN LE
T c=c-96
 240 IF c<32 THEN LET c=c+96
 250 PRINT CHR$ c;
 260 NEXT a
```

100-130 Recebem a instrução para codificar ou descodificar, assim como a chave inicial *(starting key)* do código.

140 Esta linha introduz o valor inicial do código no gerador de números aleatórios.

150 Recebe a mensagem (i$).

200-260 Descodificam/codificam e fazem imprimir um carácter de cada vez.

O número chave para cada carácter é gerado pela linha 220 e é seguidamente adicionado ou subtraído ao valor (c) do CODE (código) desse carácter. Se o número resultante estiver fora do intervalo de números CODE para caracteres "normais" (32-127), adiciona-se ou subtrai-se 96 a esse número, de forma a colocá-lo dentro do intervalo aceitável.

Exemplo: h64Q9:hT (chave 123)

BARALHADA

Neste programa, utilizamos a função RND para criar um problema que o utilizador terá de pôr em ordem. BARALHADA é uma versão computadorizada dos conhecidos *puzzles* de quadrados com letras deslizantes, nos quais se tem de recriar um padrão de 24 letras num quadro de 5 × 5 deslocando uma letra de cada vez para o quadrado (ou posição) que está vazio de momento.

Quando o executamos, o programa apresenta-nos primeiramente a imagem do quadro com todas as letras nas posições correctas. Em seguida, baralha-as de acordo com o nível de "perícia"que o utilizador tenha escolhido e espera então pacientemente enquanto este tenta desenredar a baralhada. Se para isso levar mais do que 9999 "passos", o *Spectrum* desistirá, com profunda consternação!

O programa trabalha directamente com a imagem do *écran*, utilizando a função SCREEN$ para saber o que se encontra impresso numa determinada posição. Repare-se que a linha traçada sobre a orla que delimita a área do jogo serve para que essas posições de caracteres não sejam confundidas com o quadrado vazio para o qual se pode deslocar uma letra: a função SCREEN$ devolve uma cadeia nula (vazia) se não conseguir identificar o que está no *écran* como um elemento do conjunto de caracteres normais.

```
  5 REM "BARALHADA"
 10 DATA 0,-1,0,1,-1,0,1,0
 20 DIM h(4): DIM v(4)
 30 FOR a=1 TO 4: READ h(a),v(a
): NEXT a
```

```
 100 PLOT 101,69: DRAW 45,0: DRA
W 0,45: DRAW -45,0: DRAW 0,-45
 110 FOR y=8 TO 12: FOR x=13 TO
17
 120 PRINT AT y,x;CHR$ (12+x+5*y
)
 130 NEXT x: NEXT y
 140 LET x=17: LET y=12: PRINT A
T y,x;"."
 200 RANDOMIZE : LET x1=0: LET y
1=0
 210 INPUT TAB 8;"Pericia (1-9)
? ";s: IF s<1 OR s>9 THEN GO TO
210
 220 FOR a=1 TO 5*s
 230 LET c=1+INT (4*RND): LET x2
=x+h(c): LET y2=y+v(c)
 240 IF SCREEN$ (y2,x2)="" OR (x
1=x2 AND y1=y2) THEN GO TO 230
 250 PRINT AT y,x;SCREEN$ (y2,x2
);AT y2,x2;" "
 260 LET x1=x: LET y1=y: LET x=x
2: LET y=y2
 270 NEXT a
 300 FOR m=1 TO 9999: PRINT AT 1
8,12;"Passo ";m
 310 PRINT AT 20,11;"Letra ? "
 320 LET i$=INKEY$: IF i$="" THE
N GO TO 320
 330 IF CODE i$>96 THEN LET i$=C
HR$ (CODE i$-32)
 340 FOR a=1 TO 4: IF SCREEN$ (y
+v(a),x+h(a))=i$ THEN GO TO 400
 350 NEXT a: GO TO 320
 400 PRINT AT 20,0,,
 410 PRINT AT y,x;SCREEN$ (y+v(a
),x+h(a))
 420 LET x=x+h(a): LET y=y+v(a):
 PRINT AT y,x;"."
 500 FOR p=8 TO 12: FOR q=13 TO
17
 510 IF p=12 AND q=17 THEN GO TO
 600
 520 IF SCREEN$ (p,q)<>CHR$ (12+
q+5*p) THEN GO TO 540
 530 NEXT q: NEXT p
 540 NEXT m
 600 PRINT AT 20,9;"* TERMINADO
*"
```

"·" Este sinal representa espaço em branco.

X	A	B	C	D
E	F	G	H	I
J	K		L	M
N	O	P	Q	R
S	T	U	V	W

10- 30 Definem os quadros *(arrays)* h() e v() destinados a conterem as componentes horizontais e verticais das quatro direcções de movimento possíveis.

100-140 Fazem imprimir o quadro *(array)* com as 24 letras todas em ordem alfabética.

200-210 Formam um ciclo destinado a baralhar o quadro *(array)* um certo número de vezes, dependente do valor de s (perícia—*skill*, em inglês). x e y são as coordenadas da posição do quadrado vazio num dado momento, x2 e y2 as coordenadas da posição seguinte e x1 e y1 as coordenadas da posição anterior, as quais são memorizadas pelo computador de modo a evitar que o programa desloque (ao baralhar o quadro) uma letra várias vezes entre dois mesmos quadrados.

230-240 Seleccionam uma das quatro posições circundantes do quadrado vazio, e verificam se ela não foi utilizada na vez anterior.

250-260 Deslocam a letra para o quadrado vazio, actualizam x1 e y1 de acordo com o deslocamento, etc.

300-540 Ciclo executado uma vez por cada passo do jogador.

300-330 Apresentam o número do passo, recebem a letra que o jogador deseja deslocar e convertem-na em maiúscula, caso seja necessário.

340-350 Verificam se a letra escolhida é contígua ao quadrado vazio. Caso isso não se verifique, o programa volta atrás para obter outra letra escolhida pelo jogador.

400 Elimina do *écran* a linha que pede ao jogador a letra por ele escolhida ("Letra ?").

410-420 Deslocam a letra escolhida para o quadrado vazio.

500-530 Verificam se as letras estão já todas na ordem correcta; no caso de isso acontecer, o programa salta para a linha 600; caso contrário, regressa à linha 300.

600 Fim do jogo.

O símbolo " ␣ " representa um espaço em branco que deve ser incluído nessa posição para que o programa funcione correctamente.

Capítulo 8

Peek, poke in e out

O *Spectrum* possui um máximo de 65 536 (64 K) células (ou posições) de memória endereçáveis. As primeiras 16 384 são ocupadas pela ROM, e as restantes 16 384 (16 K) ou 49 152 (48 K), dependendo do modelo, pela RAM.

Como o manual de programação BASIC *(BASIC Programming Manual)* do *Spectrum* tão claramente descreve, uma declaração PEEK permite ler o conteúdo de qualquer dessas 65 536 possíveis células de memória. E — se o endereço corresponder à RAM — uma declaração POKE permite alterar o conteúdo da célula.

Cada célula, quer pertença à RAM ou à ROM, contém 8 dígitos binários *(bits)*, que constituem um *byte*. Dependendo do contexto, os *bytes* podem ser considerados como:

— um número binário de 8 *bits* situado entre ØØØØØØØØ e 11111111;
ou — um número hexadecimal situado entre ØØ e FF;
ou — qualquer dos caracteres do "conjunto de caracteres" *(character set)* do *Spectrum*, o qual é apresentado no Apêndice A do manual do *Spectrum* (pág. 135). Este conjunto de caracteres inclui palavras de comando, símbolos gráficos e caracteres de controle de impressão, assim como caracteres alfanuméricos;
ou — um número decimal entre Ø e 255;
ou — uma instrução em código máquina para o microprocessador *Z80*.

PEEK e POKE são ordens que utilizam as representações numéricas decimais.

Ter a possibilidade de ler ou alterar o conteúdo de uma célula de memória individual é por vezes extremamente útil. Esta possibilidade é a maior parte das vezes aproveitada para definir caracteres gráficos definíveis pelo utilizador, introduzindo (PO-

KEing) na RAM valores calculados de forma a produzirem os padrões desejados. Usa-se, igualmente, para examinar ou alterar os valores das variáveis de sistema, as quais determinam a forma como o *Spectrum* reage a diversos acontecimentos e situações. Por exemplo, no programa *"Haiku* de pé-quebrado", apresentado anteriormente, utilizámos

POKE 23692,ϕ

para permitir que a imagem no *écran* "rolasse" sem pedir autorização para tal ao utilizador. Ao longo deste livro damos outros exemplos da utilização de PEEK e POKE com as variáveis de sistema, sintetizados para referência no Apêndice 1.

PEEK pode ser utilizada para examinar o conteúdo da ROM do *Spectrum*. Contudo, trata-se em geral de uma actividade razoavelmente inútil (a não ser que o utilizador seja um entusiasta do código máquina *Z80*), excepção feita ao que respeita a área da ROM que contém as matrizes de pontos que formam os caracteres normais do *Spectrum* (códigos 32 a 127).

Cada carácter no *écran* ou enviado para a impressora *ZX* consiste num quadro *(array)* ou matriz de 8×8 elementos (pontos) chamados *pixels*. O padrão particular de cada carácter é contido em 8 *bytes* (8×8 *bits*) da ROM.

A variável de sistema CHARS (23606,7) contém um número inferior em 256 ao endereço do início da tabela de padrões de pontos incluída na ROM. Assim, o endereço do primeiro *byte* do padrão de pontos da ROM para qualquer carácter é dado, simplesmente, por:

Conteúdo de CHARS+8 * CODE (código) do carácter

Tomemos, por exemplo, o carácter ϕ. Podemos fazer com que o *Spectrum* imprima o conteúdo dos 8 *bytes* que definem o carácter, utilizando a rotina seguinte:

```
  10 PRINT "ENDER.","CONTEUDO"''
  20 LET s=PEEK 23606+256*PEEK 2
3607+8*CODE "0"
  30 FOR a=s TO s+7: PRINT a,PEE
K a: NEXT a
```



```
ENDER.          CONTEUDO

15744           0       00000000
15745           60      00111100
15746           70      01000110
15747           74      01001010
15748           82      01010010
15749           98      01100010
15750           60      00111100
15751           0       00000000
```

A coluna da direita foi acrescentada de modo a apresentar o conteúdo de cada célula de memória (cujos endereços respectivos se encontram na coluna da esquerda) sob a forma de um número binário de 8 *bits*. Olhando para esta "versão" binária, e imaginando que o "1" é sempre negro e o "ϕ" sempre branco, pode distinguir-se a forma do carácter.

ESTANDARTE

Este programa utiliza os padrões de pontos contidos na ROM para imprimir, através da impressora *ZX*, uma mensagem de grandes dimensões. A impressão é feita de lado (cada carácter "deitado" na tira de papel) de modo a que possa ser impressa uma mensagem com qualquer comprimento, e a altura de cada carácter é igual à largura total do papel da impressora.

A largura dos caracteres pode ser controlada, e introduzindo o valor 4 obtêm-se caracteres com a relação largura/altura correcta. A figura apresentada como exemplo acima foi impressa utilizando o valor 2.

Como se observa na figura, o programa lida igualmente com caracteres gráficos definidos pelo utilizador. Se o CODE do carácter se situa entre 144 e 164, então a variável de sistema UDG (23675,6) é lida de modo a encontrar o início da área da RAM destinada aos padrões de pontos dos caracteres gráficos definidos pelo utilizador.

```
  50 REM "ESTANDARTE"
 100 INPUT "Introduza a mensagem
:";m$: PRINT m$
 110 INPUT "Largura do caracter?
";w: PRINT "Largura ";w
 120 FOR a=1 TO LEN m$
 130 LET b=PEEK 23606+256*PEEK 2
3607+8*CODE m$(a)
 140 IF CODE m$(a)>143 AND CODE
m$(a)<155 THEN LET b=PEEK 23675+
256*PEEK 23676+8*(CODE m$(a)-144
)
 150 DIM x(8): FOR c=0 TO 7: LET
 x(8-c)=PEEK (b+c): NEXT c
 200 LET d=128
 210 FOR c=0 TO 7
 220 FOR e=1 TO w
 230 FOR f=1 TO 8: IF x(f)<d THE
N GO TO 250
 240 LPRINT TAB 4*f-4;"    ";: I
F e=w THEN LET x(f)=x(f)-d
 250 NEXT f: LPRINT
 260 NEXT e: LET d=d/2
 270 NEXT c
 280 NEXT a
```

Devem ser incluídos quatro espaços (vazios) em *inverse video* entre as aspas na linha 240.

100-110 Recebem do utilizador a mensagem e a largura do carácter.

120-280 Constituem o ciclo principal do programa, executado uma vez para cada carácter.

130-140 Atribuem a b o endereço do primeiro *byte* dos que definem o padrão de pontos do carácter a ser impresso.

150 Coloca 8 *bytes* do padrão de pontos do carácter nos 8 elementos de x ( ).

200-270 Rotina para imprimir um carácter.

210-270 Ciclo executado uma vez por cada coluna do padrão de pontos.

220-260 Ciclo destinado a repetir a impressão de uma coluna um número de vezes igual ao valor de w (*width*—largura).

230-250 Ciclo para imprimir uma coluna.

O algoritmo utilizado para traduzir o número decimal (0-255) obtido através de PEEK nos respectivos dígitos binários individuais é dado nas seguintes linhas:

```
LET d=128
FOR c=0 TO 7
IF x>=d THEN PRINT "O bit ";7-c;" e' 1": LET
x=x-d
LET d=d/2
NEXT c
```

onde x é o valor obtido pela declaração PEEK. Foi experimentada uma versão que utilizava o operador "↑", mas revelou-se extraordinariamente lenta.

RELÓGIO DIGITAL

Este programa utiliza a variável de sistema FRAMES (células RAM com os endereços 23672/3/4) para obter um referencial de tempo suficientemente preciso para um relógio digital. A variável FRAMES, de três *bytes*, é incrementada em 1 todos os 20 milissegundos ou 1/50 de segundo (todos os 1/60 de segundo nos modelos do *Spectrum* para os Estados Unidos).

Uma grande parte do programa é utilizada para elaborar uma imagem com caracteres de dimensões quatro vezes superiores ao normal. Isto consegue-se basicamente lendo, através de PEEK, os padrões de pontos da ROM e utilizando esta informação para apresentar cada dígito sob a forma de uma matriz ou quadro *(array)* de 4×4 caracteres gráficos usuais (códigos 128-143). Mas elaborar a versão ampliada de cada dígito é um processo moroso, e o programa tem de ser suficientemente rápido para actualizar toda a imagem em menos de um segundo; portanto, o programa coloca, primeiramente, os caracteres gráficos seleccionados num quadro de cadeias *(string array)*, de onde podem ser rapidamente lidos e impressos à medida que vão sendo necessários, antes de começar a simular um relógio.

Isto significa que, quando se inicia a execução do programa, o *écran* apresentar-se-á vazio durante cerca de 20 segundos. Só depois deste tempo de espera o programa pedirá ao utilizador que

introduza o tempo correcto (horas, minutos e segundos desejados), ou seja, que ele "acerte o relógio".

```
    5 REM RELOGIO
   10 GO TO 1000
  100 LET t=PEEK 23672+256*PEEK 2
3673+65536*PEEK 23674
  110 LET t1=PEEK 23672+256*PEEK
23673+65536*PEEK 23674: IF t1>t
THEN LET t=t1
  120 LET t=INT (t/50)
  130 LET h=INT (t/3600): LET t=t
-3600*h
  140 LET m=INT (t/60): LET s=t-6
0*m
  150 IF h>12 THEN LET h=h-12: GO
 TO 1220
  160 LET h$=STR$ h: LET m$=STR$
m: LET s$=STR$ s
  170 LET p$=h$+":"+m$+":"+s$
  200 FOR a=1 TO 8: LET b=4*a-4
  210 LET c=CODE p$(a)-CODE "0"+2
: IF c<1 OR c>12 THEN LET c=1
  220 PRINT AT 9,b;c$(c,1);AT 10,
b;c$(c,2);AT 11,b;c$(c,3);AT 12,
b;c$(c,4)
  230 NEXT a
  240 GO TO 100
1000 REM Preparacao e indicacao
do tempo
1100 DIM c$(12,4,4): DIM h$(2):
DIM m$(2): DIM s$(2)
1110 FOR a=2 TO 12: LET b=PEEK 2
3606+256*PEEK 23607+8*(CODE "0"+
a-2)
1120 FOR c=1 TO 4: LET d=PEEK (b
+2*c-2): LET e=PEEK (b+2*c-1)
1130 LET f=64
1140 FOR g=1 TO 4
1150 LET d1=INT (d/f): LET d=d-f
*d1
1160 LET e1=INT (e/f): LET e=e-f
*e1
1170 LET c$(a,c,g)=CHR$ (128+d1+
4*e1): LET f=f/4
1180 NEXT g: NEXT c: NEXT a
1200 INPUT "Indique: horas : ";h
;TAB 10;"mins : ";m;TAB 10;"segs
 : ";s
1210 INPUT "Carregue em ENTER"'"
para ligar o relogio "; LINE a$
1220 LET t=50*(s+60*m+3600*h)
1230 LET t1=INT (t/65536): LET t
=t-65536*t1
1240 LET t2=INT (t/256): LET t=t
-256*t2
1250 POKE 23674,t1: POKE 23673,t
2: POKE 23672,t
1260 GO TO 100
```

10 A parte inicial do programa (montagem dos caracteres "ampliados", etc.) foi colocada no final da listagem (a partir da linha 1ØØØ) de modo a aumentar a velocidade de execução do ciclo principal.

100-240 Ciclo principal do programa, executado continuamente de forma a actualizar a hora apresentada.

100-120 Estas linhas obtêm o tempo mais recente, em segundos, inspeccionando através de PEEK a variável FRAMES duas vezes seguidas e tomando o resultado mais elevado, em cada execução do ciclo principal.

130-140 Convertem o tempo obtido em segundos, minutos e horas.

150 Reajusta o relógio após as 12:59:59.

160-170 Definem uma cadeia de 8 caracteres (p$) contendo o tempo em horas:minutos:segundos.

200-230 Estas linhas formam um ciclo para apresentar cada um dos caracteres contidos em p$.

210 Determina qual o elemento do quadro c$ a utilizar. Prepara a impressão de um espaço em branco se o carácter em p$ não for do intervalo Ø-9 ou então:

220 Imprime quatro cadeias de 4 caracteres, do quadro c$, apresentando o dígito resultante no *écran*.

1000 Início da rotina de preparação inicial do programa; ver linha 10.

1100-1180 Definem o quadro c$ e preenchem-no com os símbolos gráficos apropriados. c$ contém um quadro de 4×4 caracteres gráficos para cada um dos 12 símbolos a apresentar no *écran*, sendo o primeiro um espaço em branco e os outros onze correspondendo aos dígitos Ø a 9 e ao sinal de dois pontos. Esta técnica podia ser expandida de forma a lidar com maior gama de caracteres.

1200-1250 Obtêm do utilizador a hora por ele desejada e aferem a variável FRAMES de acordo com ela.

Nota: quem possua um modelo americano do *Spectrum* deve substituir "5Ø" por "6Ø" nas linhas 120 e 1220.

SOMAS RÁPIDAS

Este programa utiliza a variável de sistema FRAMES para medir o tempo consumido a resolver uma série de dez operações de adição e subtracção extremamente simples. O número de respostas correctas e o tempo total gasto na resolução das operações é dado no final.

```
  50 REM "CONTAS"
 100 RANDOMIZE : LET s=0
 110 POKE 23672,0: POKE 23673,0:
 POKE 23674,0
 200 FOR g=1 TO 10: CLS : PRINT
AT 4,10;"Problema ";g;AT 10,10;
 210 IF RND<.5 THEN GO TO 400
 300 LET a=INT (90*RND)+10: LET
b=INT (90*RND)+10
 310 LET c=a+b: PRINT a;" + ";b;
" = ";
 320 GO TO 500
 400 LET a=INT (90*RND)+10: LET
b=INT (90*RND)+5
 410 IF b>=a THEN GO TO 400
 420 LET c=a-b: PRINT a;" - ";b;
" = ";
 500 INPUT LINE a$: IF a$="" THE
N GO TO 500
 510 FOR x=1 TO LEN a$: IF a$(x)
<"0" OR a$(x)>"9" THEN GO TO 500
 520 NEXT x
 530 LET res=VAL a$: PRINT res
 540 IF res=c THEN LET s=s+1: PR
INT AT 14,12;"CORRECTO"
 550 IF res<>c THEN PRINT AT 14,
12;"ERRADO"
 560 PRINT AT 18,8;"Acertou ";s;
" em ";g
 570 FOR t=1 TO 200: NEXT t
 580 NEXT g
 600 LET t=PEEK 23672+256*PEEK 2
3673+65536*PEEK 23674
 610 LET t1=PEEK 23672+256*PEEK
23673+65536*PEEK 23674
```



```
 620 IF t<t1 THEN LET t=t1
 630 PRINT AT 4,0,,AT 14,0,,
 640 PRINT AT 10,5;"Tempo gasto:
";INT (t/50);" segundos"
```

100-110 Colocam a zero a variável FRAMES (inscrevendo zeros nos seus três *bytes*). Como RANDOMIZE utiliza o valor de FRAMES, tem de ser utilizada antes de FRAMES passar a zero.

200-580 Ciclo principal do programa, executado uma vez por cada operação apresentada.

210 Esta linha determina se a operação a apresentar irá ser uma adição ou uma subtracção.

300-320 Apresentam o enunciado da operação de adição.

400-420 Apresentam o enunciado da operação de subtracção.

500-530 Aceitam a resposta do utilizador. Se tivéssemos utilizado na linha 500 apenas INPUT res, o utilizador poderia introduzir o próprio enunciado apresentado, p. ex. "21−7", e o computador aceitá-lo-ia como a resposta correcta. Para evitar isso, esta rotina aceita a resposta na forma de uma cadeia, verifica se a cadeia apenas contém os dígitos de Ø a 9 e, por fim, em caso afirmativo, converte-a num número.

540-560 Actualizam a pontuação e imprimem um comentário dependente da resposta dada pelo utilizador.

570 Esta linha introduz um curto tempo de espera. O comando PAUSE não pode aqui ser utilizado pois a "pausa" por ele provocada terminaria se o utilizador mantivesse qualquer tecla pressionada.

600-620 Obtêm o tempo total decorrido, inspeccionando a variável FRAMES duas vezes consecutivas através de PEEK e tomando o valor mais elevado.

630-640 Limpam parte da imagem e seguidamente imprimem o tempo gasto pelo utilizador.

ESPAÇO DE I/O

Além de lidar com 65 536 células de memória, o microprocessador *Z80* colocado no "coração" do *Spectrum* também tem acesso a

65 536 endereços de I/O *(Input/Output)*, embora na prática — devido à forma como o *hardware* do *Spectrum* funciona — o número de endereços utilizáveis seja muito menor do que esse.

Pode-se escrever nessas células ou ler o seu conteúdo através dos comandos OUT e IN da linguagem BASIC, os quais actuam exactamente como os comandos POKE e PEEK, mas sobre células de I/O em vez de células de memória.

Estas instruções só se tornam verdadeiramente úteis quando exista um dispositivo de I/O adicional (impressora, p. ex.) ligado ao *Spectrum*. Existe contudo uma aplicação extremamente útil do comando IN, que consiste em examinar o teclado de uma forma muito mais flexível do que a conseguida com a utilização de INKEY$.

De facto, INKEY$ sofre de duas limitações; primeira: não consegue detectar se — por exemplo — ambas as teclas "A" e "B" são pressionadas simultaneamente; segunda: não pode detectar, por si só, a operação de uma tecla *shift*. Embora estas particularidades não tenham geralmente importância, existem ocasiões, nomeadamente em programas de jogos onde intervém o movimento, em que elas podem constituir um obstáculo.

A utilização de IN para examinar o teclado, tal como é descrita no capítulo 23 do manual de programação BASIC do *Spectrum* (págs. 117 e 118), liberta o utilizador destas limitações, embora exija a utilização de outra rotina que determine com exactidão qual ou quais teclas foram pressionadas.

Pode ainda surgir um pequeno problema quando se emprega o comando IN para examinar o teclado. Com efeito, enquanto INKEY$ possui uma rotina de "amortecimento" para dar tempo a que uma tecla pressionada "estabilize", IN não tem qualquer rotina do género. Isto significa que, no momento em que uma tecla é pressionada (ou deixa de o ser), a função IN pode detectar uma rápida sequência de operações dessa tecla em vez de uma única operação. Se isto vier a causar problemas num programa, será aconselhável introduzir um pequeno atraso nesse programa (através de um ciclo FOR/NEXT, por exemplo), de modo a dar tempo a que o sinal produzido ao accionar as teclas estabilize.

O resultado esperado de uma inspecção ao teclado, feita através de uma declaração IN, pode ser calculado a partir de:

191

— 1 se a tecla da ponta exterior de uma meia fila for pressionada.
— 2 se a tecla seguinte dessa mesma meia fila for pressionada.
— 4 se a tecla do meio dessa mesma meia fila for pressionada.
— 8 se a tecla seguinte dessa mesma meia fila for pressionada.
— 16 se a tecla do extremo interior dessa meia fila for pressionada.

Por exemplo, IN 32766 daria como resultado:

191 se nenhuma das teclas desde "B" a SPACE tivessem sido pressionadas.
190 se a tecla SPACE tivesse sido pressionada.
189 se a tecla SYMBOL SHIFT tivesse sido pressionada.
167 se ambas as teclas "b" e "N" tivessem sido pressionadas.

Experimente a rotina da pág. 118 do Manual de Basic de Spectrum.

CORRIDA DE REACÇÃO

Este é um jogo simples para dois jogadores, os quais têm de carregar na "sua" tecla ("B" para o jogador da esquerda, SPACE para o outro) o mais depressa possível, logo que apareça uma estrela no écran. Oferecem-se dez tentativas aos jogadores e apresenta-se uma pontuação actualizada em cada tentativa. Se ambos os jogadores pressionarem as suas teclas exactamente ao mesmo tempo, não se atribui qualquer pontuação por essa tentativa. As duas teclas foram escolhidas por estarem na mesma "meia fila" do teclado, de modo a poderem ser examinadas simultaneamente por meio de uma única declaração IN.

```
  5 REM "CORRIDA DE REACCAO"
 10 RANDOMIZE : LET l=0: LET r=
0
100 FOR a=1 TO 10
110 FOR b=100 TO 100+300*RND
120 IF IN 32766<>191 THEN PRINT
```

```
AT 15,6;"TIREM OS DEDOS": BEEP
.2,30: CLS : GO TO 110
 130 NEXT b
 140 PRINT AT 15,15;"*"
 150 LET i=IN 32766: IF i=191 TH
EN GO TO 150
 160 LET s=-24: IF i=175 THEN LE
T l=l+1: LET s=0
 170 IF i=190 THEN LET r=r+1: LE
T s=12
 180 PRINT AT 15,0;l;TAB 15;" ";
TAB 30;r: BEEP 1,s
 190 NEXT a
```

10 Define os valores iniciais

100-190 Ciclo principal, executado uma vez em cada tentativa.

110-130 Introduzem uma espera de duração variável e aleatória, certificando-se que nenhum dos jogadores carrega numa tecla enquanto dura a espera.

140 Imprime a estrela.

150 Espera que uma tecla da meia-fila B-SPACE seja pressionada.

160-170 Calculam o resultado.

180 Apresenta nova pontuação e faz emitir um sinal sonoro apropriado.

Nota: a declaração BEEP será estudada num dos capítulos seguintes; foi, contudo, aqui utilizada porque, além de tornar o jogo mais atractivo, é um meio eficaz de introduzir um tempo de espera onde existia o perigo de a pausa provocada por uma declaração PAUSE ser prematuramente terminada por alguém carregar numa tecla.

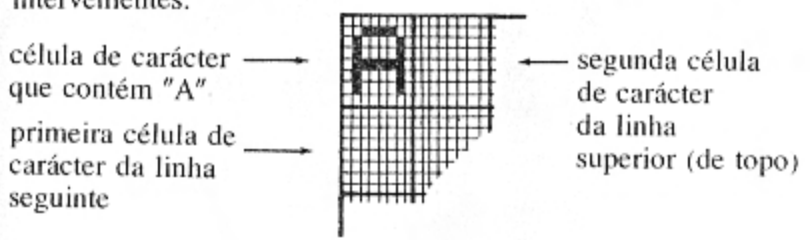
Capítulo 9

Belas-artes

Um dos atractivos do *Spectrum* é a possibilidade de obter gráficos de alta definição. No capítulo 4 mostrámos como se pode desenhar um gráfico tosco utilizando o comando PRINT AT, mas isso só nos permitiu uma definição de 32 pontos horizontais e 22 pontos verticais. Utilizando os comandos de gráficos de alta definição, conseguimos obter 256 pontos horizontais e 176 pontos verticais, o que dá uma imagem aproximadamente com tanto pormenor quanto a que um receptor de TV normal consegue apresentar.

Para compreender como funciona o sistema que gera os gráficos de alta definição é bom saber um pouco acerca da forma como o *Spectrum* produz a imagem na TV e, portanto, iremos estudá-lo antes de nos debruçarmos sobre alguns programas que fazem uso de gráficos de alta definição (ou de alta resolução, segundo algumas nomenclaturas.

A área total do *écran* onde se apresenta a imagem fornecida pelo *Spectrum*, incluindo as últimas linhas da parte inferior, as quais geralmente se reservam para mensagens, comandos de INPUT ou para montagem e correcção de linhas *(editing)*, é dividida em 24×32 posições de caracteres, ou "células" (não confundir com células de memória). Cada célula de carácter encontra-se, por sua vez, dividida numa matriz ou rede de 8×8 elementos chamados *pixels* (*pixel* é uma abreviatura inglesa de *picture element* ou "elemento de imagem"), e qualquer carácter é formado no *écran* de TV tornando cada *pixel* da matriz ou branco ou negro (a cor é uma outra questão, e será tratada noutro capítulo). Assim, se imprimíssemos a letra "A" no canto superior esquerdo do *écran*, poderíamos desenhar um "mapa" dos *pixels* intervenientes:



Se não incluirmos as duas últimas linhas inferiores da imagem e apenas considerarmos as linhas em que podemos imprimir (através de PRINT), temos uma rede de 22×32 células ou posições de caracteres, e

$$\begin{aligned} & 22 \times 8 \times 32 \times 8 \\ = \quad & 176 \times 256 \quad \textit{pixels} \end{aligned}$$

ou seja, uma área com 256 *pixels* de largura e 176 *pixels* de altura.

Se estes números não nos são estranhos, há uma boa razão para isso. Com efeito, eles são, respectivamente, o número de pontos horizontais e verticais que podem ser utilizados pelos comandos PLOT, DRAW e CIRCLE de gráficos de alta definição, embora para utilização nesses comandos os pontos sejam numerados de 0 a 175 e de 0 a 255, em vez de 1 a 176 e de 1 a 256.

De facto, utilizamos estes pequenos *pixels* tanto para formar os gráficos de alta definição como para formar os caracteres normais. Com os comandos de alta definição controlam-se *pixels* individuais: tornando-os negros ou brancos — ou, mais precisamente, atribuindo-lhes *ink* (cor) ou *paper* (fundo), como veremos no capítulo seguinte. Inversamente, o comando PRINT normal afecta todos os 8×8 *pixels* de uma célula de carácter.

PRANCHETA DE DESENHO ELEMENTAR

Este programa permite ao utilizador desenhar figuras no *écran* de TV. Consideramo-lo um programa educativo, pois mostra como funciona a declaração PLOT x,y e familiariza o utilizador com um sistema de coordenadas X-Y, e pode muito simplesmente ser utilizado para distracção. Também exemplifica a utilização do PLOT INVERSE 1;x,y como forma de apagar um ponto, anteriormente impresso através de PLOT.

As figuras são desenhadas utilizando as teclas "5", "6", "7" e "8" para deslocar um pequeno cursor intermitente sobre o *écran*. Para auxiliar o utilizador, no caso de um desenho de precisão, as coordenadas X e Y da posição do cursor em cada momento são continuamente apresentadas no canto inferior esquerdo do *écran*.

Ao deslocar-se, o cursor deixa atrás de si uma linha negra, a não ser que se mantenha a tecla SHIFT pressionada, e nesse caso o cursor desenhará uma linha branca, permitindo apagar quaisquer erros.

Terminada a sua obra-prima, o utilizador pode parar o programa carregando em SHIFT/BREAK. Se dispuser de uma impressora *ZX*, pode seguidamente imprimir uma cópia da figura através do comando COPY, ou então gravá-la em *cassette* com o comando directo

SAVE "nome" SCREEN$

para posterior utilização.

Se, após ter parado o programa através de SHIFT/BREAK, o utilizador verificar que a figura não está completa, não deve introduzir RUN, pois isso iria limpar tudo o que estivesse no *écran*. Em vez disso, deve introduzir a ordem CONTINUE.

Notas:

O programa está estruturado como um ciclo sem fim, o que é uma estrutura geralmente a evitar, mas aceitável neste caso, visto a essência do programa ser a de uma actividade contínua — e seria despropositado fazer com que o programa perguntasse "Quer continuar?" depois de imprimir cada ponto.

A função INKEY$ fornece como resultado um valor de cadeia, mas temos de o converter num código (CODE) numérico de modo a lidar com os códigos 8-11 de movimento do cursor em que não há impressão. Se não fosse por esse motivo, teria sido preferível fazer de k uma variável de cadeia, e compará-la com constantes de cadeia nas linhas 120-180, em declarações do tipo

IF k$="5" - - -

```
  10 LET x=127: LET y=88
 100 FOR c=0 TO 1: PLOT INVERSE
c;x,y
 110 LET k=CODE INKEY$
 120 IF k>=53 AND k<=56 THEN PLO
T x,y
 130 IF k>=8 AND k<=11 THEN PLOT
 INVERSE 1;x,y
 140 PRINT AT 20,0;"x=";x;"  "'"
y=";y;"  "
```

```
 150 IF (k=8 OR k=53) AND x>0 TH
EN LET x=x-1
 160 IF (k=9 OR k=56) AND x<255
THEN LET x=x+1
 170 IF (k=10 OR k=54) AND y>0 T
HEN LET y=y-1
 180 IF (k=11 OR k=55) AND y<175
 THEN LET y=y+1
 190 NEXT c
 200 GO TO 100
```

10 Fixa os valores iniciais das coordenadas x e y.

100-190 Ciclo principal, que faz o cursor acender e apagar alternadamente.

110 Espera que uma tecla seja accionada.

120 Se uma das teclas "5", "6", "7" ou "8" foi pressionada, esta linha marca a negro a posição actual do cursor.

130 Se a tecla SHIFT foi pressionada conjuntamente com uma das teclas "5", "6", "7" ou "8", esta linha marca a branco a presente posição do cursor.

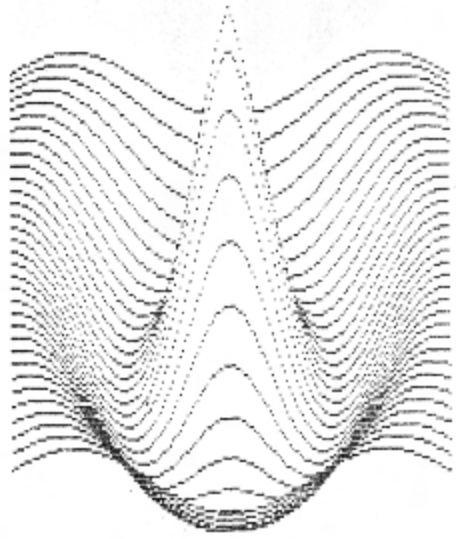
140 Imprime os valores das coordenadas da posição actual do cursor. Repare-se que são impressos espaços em branco a seguir aos números, de modo a assegurar que o número anterior seja apagado do *écran* na totalidade, o que não aconteceria se os espaços em branco não

fossem incluídos e o novo número impresso tivesse menos dígitos que o anterior.

150-180 Estas linhas calculam as novas coordenadas.

200 Regresso ao início do ciclo principal, para outra "cintilação" do cursor.

SOMBRERO



Existe um certo fascínio no uso de um computador para traçar gráficos tridimensionais ("3-D") de funções matemáticas.

O programa SOMBRERO traça o gráfico de uma dessas funções:

$$z = \text{COS } r * \text{EXP}(-r/3)$$

onde r é a distância do centro do plano X-Y ao ponto a ser determinado e representado no gráfico, por exemplo:

$$x^2+y^2$$

Apesar de o programa em si não levar muito tempo a ser introduzido no computador através do teclado, chama-se a atenção para o facto de ele levar cerca de 25 minutos a completar a figura!

```
  50 REM "SOMBRERO"
 100 FOR x=40 TO 215
 110 LET b=999: LET t=0
 120 FOR y=16 TO 144 STEP 4
 130 LET r=50R ((x-127)*(x-127)+
(y-80)*(y-80))/15
 140 LET z=INT (y+90*EXP (-r/3)*
COS r)
 150 IF z<b OR z>t THEN PLOT x,z
 160 IF z<b THEN LET b=z
 170 IF z>t THEN LET t=z
 180 NEXT y
 190 NEXT x
```

100-190 Ciclo que percorre todos os pontos de coordenada x entre 40 e 215.

110 Nesta linha, b e t são utilizados para a eliminação das "linhas ocultas" do gráfico, ou seja, das linhas ou partes de linha que possam ficar "ocultas" por detrás de parte do gráfico da função. São-lhes aqui atribuídos valores fora dos limites dentro dos quais eles se situariam normalmente.

120-180 Ciclo que percorre todos os pontos cuja coordenada y se situe entre 16 e 144, inclusive, e seja um número múltiplo de 4. Por outras palavras: que percorra todos os "quartos" pontos de coordenada y entre 16 e 144.

130-140 Calculam a cota a que se deve situar o ponto a imprimir.

150 Imprime o ponto se este não se encontrar "oculto" atrás de uma parte do gráfico da função que esteja em primeiro plano.

160-170 Actualizam b e t, atribuindo-lhes os valores de cota mais baixo e mais elevado até aqui determinados pelas linhas 130 e 140.

CÁPSULAS

A declaração DRAW x,y fornece-nos uma forma simples de desenhar uma linha recta entre dois pontos quaisquer no *écran*. A linha é traçada a partir do ponto anteriormente determinado até um novo ponto cujas coordenadas são calculadas a partir dos valores

atribuídos às variáveis x e y que se seguem à palavra de comando DRAW.

Contudo, esses dois números não são as coordenadas x e y referentes ao fim da linha, mas, sim, coordenadas relativas. Ou seja: indicam as coordenadas do ponto onde a linha deverá terminar, mas em relação ao ponto onde ela teve início e não em relação ao ponto de coordenadas Ø,Ø do *écran*. Por exemplo, a declaração DRAW 1Ø,1Ø desenhará sempre uma linha recta ascendente com a inclinação de 45 graus que irá terminar num ponto situado 10 *pixels* à direita e 10 *pixels* acima do ponto de partida da linha, onde quer que este se situe (a não ser, é claro, que o ponto indicado por DRAW para fim da linha se viesse a situar fora da área do *écran* já referida, em que podemos "imprimir", e, nesse caso, o *Spectrum* imprimiria a mensagem "B Integer out of range" na última linha do *écran*).

Embora isto pareça um pouco estranho a princípio, é vantajoso na medida em que uma rotina destinada a desenhar determinada figura pode — se apenas utilizar declarações DRAW — ser usada sem alterações para desenhar a mesma figura em qualquer parte do *écran*.

O programa apresentado a seguir faz exactamente isso, reproduzindo 32 vezes no *écran* o mesmo desenho maçador de uma cápsula espacial (contudo, o leitor tem de se treinar e aperfeiçoar a fazer coisas deste tipo!).

```
  50 REM "CAPSULAS"
 100 FOR v=15 TO 135 STEP 40
 110 FOR h=25 TO 235 STEP 30
 120 PLOT INVERSE 1; OVER 1;h,v
 130 RESTORE
 140 READ m,x,y
 150 IF m=Ø THEN DRAW INVERSE 1;
OVER 1;x,y
 160 IF m=1 THEN DRAW x,y
 170 IF m<>9 THEN GO TO 140
 180 NEXT h
 190 NEXT v
1000 DATA Ø,-12,7,1,24,Ø,1,Ø,-14
,1,-24,Ø,1,Ø,14
1010 DATA 1,12,8,1,12,-8
1020 DATA Ø,-7,-14,1,3,-8,1,-16,
Ø,1,3,8,9,Ø,Ø
```

100-110 Juntamente com as linhas 180 e 190, estas linhas determinam as coordenadas centrais de cada uma das 32 figuras.

120 Desloca a posição de traço para o centro da figura seguinte, sem contudo fazer qualquer marca ou traço no *écran*.

140 Faz o computador voltar ao início da lista de dados e começar a lê-los (novamente, no caso de a figura a desenhar não ser a primeira) desde o primeiro, neste caso na linha 1000.

140-170 Desenham um cápsula.

Os elementos DATA (dados) (linha 1000 e seguintes) estão organizados em grupos de três números. O segundo e terceiro número de cada grupo são usados como valores a atribuir às variáveis das declarações DRAW das linhas 150 e 160. O primeiro número de cada grupo especifica o que há a fazer: se esse número é zero, a linha 150 desloca a posição de impressão sem afectar a imagem no *écran;* se esse número é 1, a linha 160 irá traçar uma linha no *écran*.

SERENDIPITY

Serendipity é um vocábulo invulgar, que entrou na língua inglesa em consequência de uma história de Horace Walpole, *Os Três Príncipes de Serendib*.

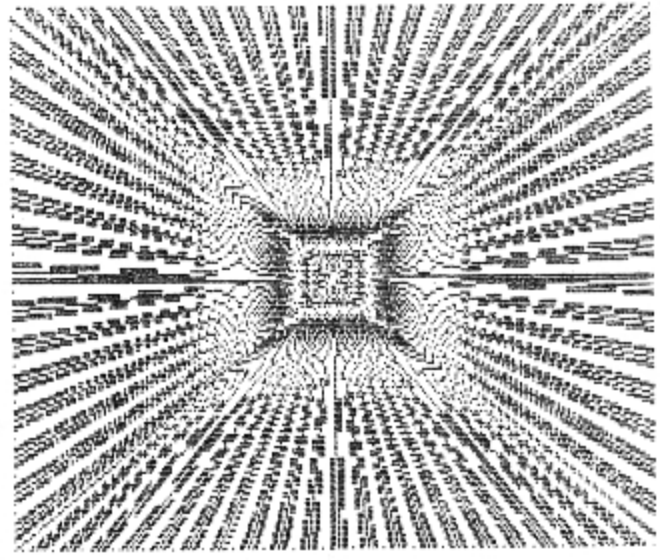
Os três príncipes, em busca de certos objectivos, acabavam sempre por descobrir acidentalmente coisas diferentes mas bem mais proveitosas do que as que procuravam. E daqui resultou o curioso vocábulo, que se aplica à eventualidade de uma descoberta importante no decurso da procura de coisa diferente e de menor importância ou utilidade, ou ainda ao facto de poderem advir vantagens inesperadas de certas limitações.

O programa apresentado a seguir é um desses casos, o que justifica a escolha do nome.

Devido a haver apenas um número finito de *pixels* utilizáveis, os segmentos de recta traçados diagonalmente no *écran* mostram-se ligeiramente denteados. Na prática, isto não representa problema importante, e por vezes até se aproveita com vantagem, como acontece neste programa, que desenha uma sequência aleatória de 100 belas figuras.

O programa tira também vantagens de pequenos defeitos existentes nos receptores de televisão. Em primeiro lugar, os receptores de televisão tendem a esbater o pormenor de precisão, de modo que uma linha composta de *pixels* negros e brancos alternados parecerá ser cinzenta. Em segundo lugar, se o receptor for a cores, embora o programa desenhe apenas em branco e negro, a imagem parecerá estar tingida com manchas irisadas e tremeluzentes. Devido a estes efeitos, a versão aqui impressa é muito menos atraente do que a "representação ao vivo".

```
  5 REM "SERENDIPITY"
 10 RANDOMIZE
100 FOR p=1 TO 100
110 LET s=2+INT (3*RND)
120 LET o=INT (2*RND)
130 FOR x=255 TO 0 STEP -s
140 PLOT x,0: DRAW OVER o;255-2
*x,175
150 NEXT x
160 FOR y=0 TO 175 STEP s
170 PLOT 0,y: DRAW OVER o;255,1
75-2*y
180 NEXT y
190 NEXT p
```



100-190 Ciclo destinado a contar 100 figuras.
110 Esta linha escolhe s, que é o valor do incremento *(step)* para as linhas 130 e 160, sob a forma de um número aleatório entre 2 e 4.
120 Atribui aleatoriamente a "o" o valor ∅ ou 1. A variável o é utilizada nas linhas 140 e 170 para que o programa trace linhas negras (quando o=∅) ou trace linhas brancas, invertendo a imagem já existente no *écran* (o=1).
130-150 Estas linhas desenham uma metade da figura.
160-180 Desenham a outra metade.

ROTOR

Geram-se algumas figuras muito bonitas utilizando a capacidade do computador para repetir uma forma básica muitas vezes, com pequenas alterações.

O programa listado abaixo desenha apenas um grupo de triângulos, um dentro de outro, por sua vez dentro de outro, numa série em que os triângulos vão sendo cada vez mais pequenos. O programa repete isto seis vezes para terminar numa forma hexagonal.

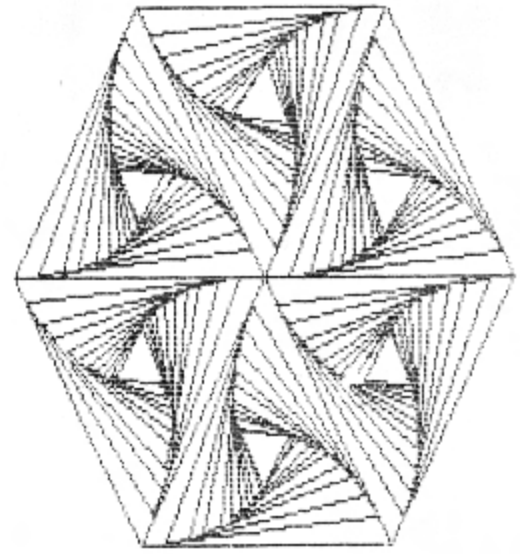
Obtêm-se variações da figura se se excluir um dos lados de cada triângulo, por omissão da linha 150, ou se se alterar a linha 140 para

```
140 DRAW INVERSE 1; OVER 1; - - -
```

a qual irá deslocar a posição de traçagem para o início da linha seguinte sem desenhar absolutamente nada.

(OVER e INVERSE são estudadas com maior minúcia no capítulo seguinte.)

```
 100 FOR a=0 TO 1.7*PI STEP PI/3
 110 LET x=127: LET y=88: LET l=
100
 120 FOR d=a TO a+1 STEP .1
 130 PLOT x,y: DRAW l*COS d,l*SI
N d
 140 DRAW l*SIN (PI/6-d)-l*COS d
,l*COS (PI/6-d)-l*SIN d
 150 DRAW -l*SIN (PI/6-d),-l*COS
 (PI/6-d)
 160 LET x=x+.1*l*COS d: LET y=y
+.1*l*SIN d: LET l=.85*l
 170 NEXT d
 180 NEXT a
```



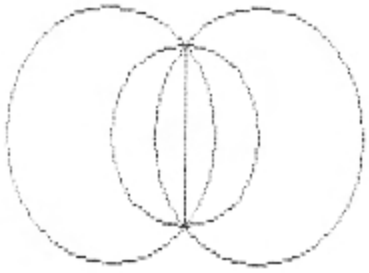
100-180 Ciclo destinado a desenhar 6 grupos de triângulos.
110 Define as condições iniciais para um grupo.
120-170 Ciclo para desenhar os 10 triângulos que formam um grupo.
130-150 Desenham um triângulo.
160 Determina a dimensão e o ponto de início do desenho do triângulo seguinte.

BOLHAS

Acrescentando um terceiro número a um comando DRAW de modo a que fique na forma.

```
DRAW x,y,a
```

faz-se com que ele desenhe parte de um círculo. A rotina listada abaixo dá uma breve demonstração disto:



```
 100 FOR a=-1.5*PI TO 1.5*PI STE
= PI/2
 110 PLOT 127,58
 120 DRAW 0,58,a
 130 NEXT a
```

Em geral, o comando DRAW nesta forma é utilizado como parte de uma rotina destinada a desenhar formas complicadas. mas, por brincadeira, apresentamos um jogo simples baseado nesta forma do citado comando.

O objectivo do jogo é tentar capturar o maior número possível de "ovos espaciais" lançando um campo de forças em volta deles. A "rede" constituída pelo campo de forças tem a forma de uma "bolha", e o jogador pode controlar o tamanho da dita "bolha". Existe contudo um único problema: se o campo de forças tocar qualquer dos "ovos", rompe-se e fica inutilizável.

Podem jogar dois ou mais jogadores, cada um dos quais deve abandonar o jogo quando o seu campo de forças se romper.


```
  50 REM "BOLHAS"
 100 FOR a=1 TO 15: PRINT AT 15*
RND,30*RND;"o": NEXT a
 110 PLOT 80,10: DRAW 0,-10: DRA
W 85,0: DRAW 0,10
 200 INPUT "Tamanho da bolha (1-
9)? ";t
 210 IF t<1 OR t>9 THEN GO TO 20
0
 220 PLOT 80,10: DRAW OVER 1;85,
0;-(4.35+t/10)
 300 PRINT AT 19,10;"Outra vez ?
"
 310 IF INKEY$="s" THEN RUN
 320 IF INKEY$<>"n" THEN GO TO 3
10
```

100-110 Estas linhas imprimem 15 "ovos espaciais" em posições aleatórias e, seguidamente, desenham o "lançador da bolha do campo de forças".

200-220 Obtêm do jogador o tamanho da bolha, verificam a sua validade (se está dentro dos valores previstos) e, seguidamente, desenham a bolha. A função OVER 1 é aqui utilizada porque faz com que o campo de forças apresente uma rotura se tocar em algum "ovo".

300-320 Esperam que o jogador seguinte comece novamente o jogo (ou desista — n).

CARACTERES PESSOAIS

Quem não estiver satisfeito com os caracteres e símbolos gráficos do teclado do *Spectrum* pode, se o desejar, acrescentar mais alguns de sua autoria. Esta possibilidade é útil, não só para imprimir caracteres ou símbolos inexistentes na língua inglesa e utilizados em alguns idiomas, como o português (cedilhas, acentos, etc.), mas também para criar formas especiais. Por exemplo, se quiséssemos escrever um programa de "Invasores do Espaço", poderíamos criar um carácter especial que se assemelhasse a um dos "invasores extraterrestres" e fazer com que o computador o imprimisse (PRINT) no *écran* como qualquer outro carácter. Se, porventura, precisarmos de uma forma (ou

figura) cujas dimensões superem as de um carácter usual, criamos dois ou mais caracteres gráficos especiais, de molde a que, quando impressos conjuntamente, constituam a forma desejada.

Como dissemos anteriormente, cada célula de carácter é constituída por uma matriz ou quadro de 8×8 *pixels*, cada um dos quais com a possibilidade de ser definido em branco ou em negro. Portanto, cada *pixel* pode ser representado por um único dígito binário — um *bit* — que terá o valor Ø se o *pixel* for branco, e o valor 1 se o *pixel* for negro. (Analisaremos o problema da cor no capítulo seguinte.)

Deste modo, o carácter completo seria representado por oito *bytes* de oito *bits*. Por exemplo, os *pixels* necessários para formar a letra "A" poderiam ser representados na seguinte forma:

Byte 1 : 0 0 0 0 0 0 0 0
2 : 0 0 1 1 1 1 0 0
3 : 0 1 0 0 0 0 1 0
4 : 0 1 0 0 0 0 1 0
5 : 0 1 1 1 1 1 1 0
6 : 0 1 0 0 0 0 1 0
7 : 0 1 0 0 0 0 1 0
8 : 0 0 0 0 0 0 0 0

E, de um modo semelhante, definiremos os desenhos de caracteres por nós concebidos.

Cada carácter utilizado pelo *Spectrum* tem de ter um número de código, pois o computador utiliza esses códigos para representar internamente os caracteres nos seus cálculos e processamentos diversos. O Apêndice A do manual de programação BASIC fornecido com o *Spectrum* inclui uma listagem de todos os 256 códigos possíveis.

Ao observar essa lista, verifica-se que alguns desses códigos (0 a 5 e 24 a 31) não são utilizados, que alguns (6 a 23) são códigos ditos de "controle" e que muitos outros (165 a 255) são usados para representar palavras de comando *(keywords)* — mas não nos preocupemos com isto por enquanto. Os que, de momento, nos interessam são os códigos 144 a 164, os quais são denominados *"user graphics A-U"*, que podemos traduzir por "gráficos do utilizador A-U", mas cujo significado mais correcto

será talvez "caracteres gráficos de A a U *definíveis* pelo utilizador". Estes 21 códigos estão, pois, reservados para o utilizador os atribuir aos caracteres gráficos por si escolhidos ou criados.

Tomando o primeiro destes códigos, veremos como é o carácter correspondente através do comando directo

```
    PRINT CHR$ 144
ou  PRINT "shift/graphics A graphics"
```

Com toda a probabilidade, veremos imprimir a letra maiúscula "A".

Isto sucede porque, ao ligar o *Spectrum*, este atribui automaticamente aos 21 caracteres gráficos a definir pelo utilizador a forma das letras maiúsculas do alfabeto de A a U. Porquê?

Pensando novamente em como um carácter pode ser representado por um padrão de 8 *bytes*, verificamos que existe uma necessidade óbvia de arranjar um sítio para colocar o padrão para os nossos caracteres definidos pelo utilizador. O *Spectrum* reserva 21×8=168 *bytes* no extremo superior da RAM para este efeito, e procura automaticamente nesta área o padrão necessário, sempre que tem de imprimir qualquer dos caracteres gráficos definidos pelo utilizador.

Para facilitar a colocação do padrão na zona da RAM apropriada, o *Spectrum* faz uso especial da função USR. USR "n" dá o endereço do primeiro *byte* da área da RAM na qual está guardado o padrão do carácter gráfico "n" definível pelo utilizador. (Note-se que "n" pode ser introduzido na forma de uma letra minúscula, letra maiúscula ou carácter gráfico.)

E para auxiliar o utilizador a introduzir o padrão, o *Spectrum* possui a função BIN, a qual deve ser seguida de um número binário expresso em zeros e uns, e que dá o número decimal equivalente.

Como experiência, vamos desligar o *Spectrum* por alguns segundos, voltamos a ligá-lo, introduzimos e fazemos executar o programa seguinte:

```
1Ø PRINT CHR$ 144;
2Ø POKE USR "a",BIN 1ØØØØØØ1
3Ø PRINT CHR$ 144
```

Veremos que o segundo A a ser impresso aparecerá com duas "orelhas", e isto porque o que fizemos na linha 20 foi acrescentar um ponto negro em cada extremo da fila superior (primeira a contar de cima) de *pixels* do carácter gráfico "A" definível pelo utilizador (código 144).

Este carácter continuará assim até que introduzamos outro número através de POKE ou desliguemos simplesmente o computador.

Quem esteja familiarizado com o sistema binário de numeração sabe, certamente, que o conjunto de 8 *bits* 10000001 representa o número decimal 129, e quem não esteja familiarizado com este sistema confirmá-lo-á introduzindo o comando directo

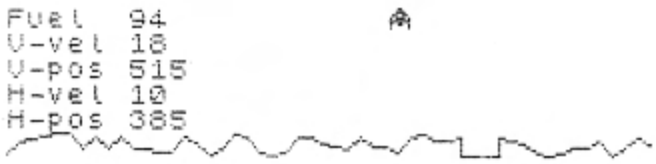
```
PRINT BIN 1ØØØØØØ1
```

e, portanto, a linha 2Ø poderia igualmente ser escrita da forma:

```
2Ø POKE USR "a";129
```

Mas é claro que, para definir um carácter completamente novo, temos de introduzir através de POKE conjuntos (ou padrões de 8 *bits)* em todas as 8 células de memória reservadas para esse determinado carácter a definir pelo utilizador. O programa seguinte mostra uma forma de o fazer.

ALUNAGEM



Nenhum livro de programas de computador está completo sem um programa de alunagem e, como nos permite exemplificar o uso de caracteres gráficos definidos pelo utilizador, apresentamos aqui um desses programas.

O objectivo do jogo é conseguir pousar o veículo de alunagem na superfície da Lua sem o danificar e — de preferência — na área de alunagem plana, especialmente preparada para o efeito, não só por esta ser o único local plano da superfície nas redondezas, mas também para evitar que a base lunar tenha de enviar uma equipa de salvamento à nossa procura.

Quando o jogo se inicia, o nosso veículo encontra-se a 2100 m acima da superfície, deslocando-se horizontalmente a uma velocidade de 30 m por segundo e começa a cair.

O nosso computador de pouso *!?- -X! avariou-se outra vez, portanto teremos de efectuar a manobra manualmente, e, ainda por cima, os retrofoguetes do veículo funcionam apenas de forma binária: ou ligados ou desligados.

Os únicos comandos de que dispomos são as teclas "5", "7" e "8", as quais ligam respectivamente os retrofoguetes lateral direito (atrasa a velocidade horizontal, impelindo o veículo para a esquerda), vertical (retarda a velocidade vertical de queda) e lateral esquerdo (aumenta a velocidade horizontal, impelindo o veículo para a direita). Contudo, estes retrofoguetes apenas podem ser accionados um de cada vez e não em simultâneo. Se queremos ter êxito na manobra, deveremos pousar na superfície lunar com as velocidades horizontal e vertical e a posição horizontal relativa, apresentadas no *écran,* tão próximas de zero quanto possível. Boa sorte para o piloto e não esquecer de controlar permanentemente o nível de combustível do depósito!

Nota: para quem ainda não é "perito" neste tipo de jogo, aconselhamos a aumentar a quantidade de combustível inicialmente disponível, alterando na linha 20 o valor de f para f=15Ø (ou mesmo mais), pois, enquanto a prática não é muita, há o risco (muito provável) de ficar sem combustível antes de tocar a superfície, o que resulta, obviamente, no não funcionamento dos retrofoguetes e na perda total de controle do veículo de alunagem.

```
  5 REM "ALUNAGEM"
 10 RANDOMIZE
 20 LET f=100: LET vv=0: LET vp
=2100: LET hv=30: LET hp=2300
100 POKE USR "a",BIN 00011000
110 POKE USR "a"+1,BIN 00111100
120 POKE USR "a"+2,BIN 01100110
130 POKE USR "a"+3,BIN 10111101
```

```
 140 POKE USR "a"+4,BIN 11111111
 150 POKE USR "a"+5,BIN 10011001
 160 POKE USR "a"+6,BIN 10000001
 170 POKE USR "a"+7,0
 200 LET y=0: PLOT 0,0
 210 FOR a=0 TO 250 STEP 5
 220 IF a=180 THEN DRAW 0,-y: DR
AW 15,0: DRAW 0,y: LET a=195
 230 LET dy=5-INT (11*RND): IF y
+dy<0 OR y+dy>7 THEN GO TO 230
 240 DRAW 5,dy: LET y=y+dy
 250 NEXT a
 300 PRINT AT 16,0;"Fuel  ";f;".
"
 310 PRINT "V-vel ";vv;".."'"V-p
os ";vp;".."
 320 PRINT "H-vel ";hv;".."'"H-p
os ";hp;".."
 330 PRINT OVER 1;AT 21-vp/100,2
3-hp/100;CHR$ 144;CHR$ 8;
 340 IF vp=0 OR (vp<50 AND ABS (
hp-2300)>49) THEN GO TO 500
 350 IF hp=-800 THEN GO TO 600
 360 FOR a=1 TO 50: NEXT a
 400 LET k$=INKEY$: IF f=0 THEN
LET k$=""
 410 IF k$="5" OR k$="7" OR k$="
8" THEN LET f=f-1
 420 LET vp=vp-vv: IF vp<0 THEN
LET vp=0
 430 LET vv=vv+1: IF k$="7" THEN
 LET vv=vv-2
 440 LET hp=hp-hv: IF hp<-800 TH
EN LET hp=-800
 450 IF k$="8" THEN LET hv=hv+1
 460 IF k$="5" THEN LET hv=hv-1
 470 PRINT "."; GO TO 300
 500 PRINT AT 0,0;"Alunou,"'
 510 IF vv>10 OR ABS hv>10 THEN
PRINT "Mas esta'morto !": STOP
 520 IF vv>5 OR ABS hv>5 THEN PR
INT "- 'a justa -"
 530 IF ABS hp>49 THEN PRINT "Ma
s no local errado!"
 540 IF vv<5 AND ABS hv<5 AND AB
S hp<50 THEN PRINT "Felicitacoes
, Buck Rogers !"
 550 STOP
 600 PRINT AT 0,0;"Voce esta'for
a deste mundo !"
```


10- 20 Definem os valores iniciais.

100-170 Alteram o caracter gráfico "A" definível pelo utilizador de modo a assemelhar-se ao veículo de alunagem.

200-250 Desenham a superfície lunar; geralmente de forma aleatória, com excepção de uma área plana para a alunagem desenhada pela linha 220.

300-470 Ciclo principal do programa.

300-320 Actualizam os valores apresentados para o combustível (*fuel*), velocidade, etc.

330 Esta linha imprime o veículo de alunagem numa posição apropriada. A declaração OVER 1 é incluída para que o desenho da superfície lunar não desapareça quando o veículo pousar nela. Repare-se em CHR$ 8: esta função faz recuar de um carácter a posição de impressão depois de imprimir o veículo, de modo a que o próximo carácter a ser impresso se lhe sobreponha.

340 Esta linha faz o programa sair do ciclo principal se o veículo tiver alunado (bem ou mal).

350 Esta linha também faz o programa sair do ciclo principal, se o veículo sair dos limites do *écran*.

360 Ciclo destinado a retardar a acção até um ritmo aceitável — uma declaração PAUSE não se pode aqui utilizar, pois seria anulada ao serem accionadas as teclas de controle.

400 Esta linha verifica se houve pressionamento de alguma tecla e ignora-o se já não houver combustível.

410-460 Calculam as novas velocidades e posições relativas do veículo tendo em conta qualquer accionamento das teclas "5", "7" ou "8".

470 Apaga a imagem anterior do veículo imprimindo sobre ela um espaço em branco, e faz a execução do programa voltar ao início do ciclo principal.

500-550 Imprimem os comentários apropriados depois de se ter verificado a alunagem (mesmo se mal sucedida).

600 Fim do programa, se o veículo de alunagem sair dos limites do *écran*.

"•" representa espaço em branco.

UMA MUDANÇA COMPLETA

O leitor sabia que se podem redefinir todos os caracteres alfanuméricos do *Spectrum* (códigos 32 a 127) ?

Como vimos no Capítulo 8, a variável de sistema CHARS (23606,23607) contém um número que é inferior em 256 ao endereço do início da tabela de padrões de pontos para os caracteres normais que se encontra na ROM.

É atribuída a esta variável o número decimal 15360 sempre que se liga o *Spectrum* ou se introduz o comando NEW.

Se alterarmos este valor, o *Spectrum* irá buscar os padrões de pontos a outro local da memória quando imprimir qualquer carácter. Experimentaremos essa alteração, introduzindo o comando directo

```
POKE 23606,8
```

e verificaremos que, carregando na tecla "A", o computador imprimirá "B", se carregarmos em "B", ele imprimirá "C", e assim por diante. O que fizemos ao introduzir o comando acima foi alterar o valor de CHARS, deslocando-o 8 *bytes* ou um carácter.

Aproveitaremos esta característica se atribuirmos um novo valor a CHARS de modo a que esta variável de sistema aponte uma área da RAM na qual possamos gravar, através de POKE, os nossos próprios padrões de pontos.

Olhando para o mapa de memória do *Spectrum*, verificamos que os primeiros 168 *bytes*, ou *bytes* de "topo", estão geralmente reservados para os caracteres gráficos definíveis pelo utilizador, mas podemos utilizar a RAM logo abaixo deste ponto se deslocarmos o extremo superior da RAM, ou RAMTOP, para baixo, por meio de uma declaração CLEAR apropriada.

Para não termos de começar do nada, a solução mais simples é, em primeiro lugar, copiar os padrões de pontos já existentes na ROM para a nova área escolhida e, seguidamente, alterar aqueles que queremos que fiquem diferentes. O programa listado abaixo mostra como podemos começar a fazer isto:

```
100 CLEAR 32767-768-168-1
110 LET u=32767-768-168
120 FOR a=0 TO 767: POKE (u+a),
PEEK (15616+a): NEXT a
```



```
130 LET v=u-256: POKE 23606,v-2
56*INT (v/256): POKE 23607,INT (
v/256)
140 LET x=v+8*CODE "R"
150 POKE x,0: POKE x+1,62
160 POKE x+2,66: POKE x+3,66
170 POKE x+4,62: POKE x+5,34
180 POKE x+6,66: POKE x+7,0
190 PRINT "URSS"
200 POKE 23606,0: POKE 23607,60
210 PRINT "URSS"
```

100 Esta linha reserva espaço na RAM para 768 *bytes* de padrões de pontos definidores de caracteres.

110-120 Copiam os padrões de pontos existentes na ROM (o que demora cerca de 15 segundos).

130 Actualiza o valor de CHARS.

140-180 Alteram o padrão que define "R".

190 Mostra o que se fez.

200-210 Voltam aos padrões de pontos da ROM originais, de modo a que tudo regresse à ordem primitiva.

Capítulo 10

Som e cor

Uma utilização hábil das capacidades de som e cor do *Spectrum* dá vida aos jogos e permite atrair a atenção do utilizador para informações importantes num programa de aplicação.

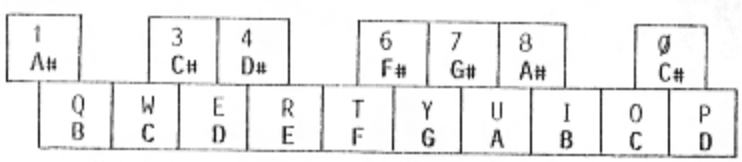
O comando BEEP é bastante simples de utilizar. Consiste apenas na palavra de comando BEEP seguida de dois números: o primeiro define a duração da nota, e o segundo a sua tonalidade (frequência). Experimentemos, por exemplo, as rotinas seguintes:

```
FOR a=Ø TO 3Ø: BEEP .Ø1,a: NEXT a
FOR a=Ø TO 5: FOR b=12 TO 4 STEP -8: BEEP .5,b:
NEXT b: NEXT a
FOR a=1 TO 4Ø: BEEP 1/a,.5: NEXT a
```

Surgirá, contudo, um ligeiro problema se quisermos que o *Spectrum* emita um BEEP enquanto está a executar outra coisa qualquer. Não o conseguiremos. O computador ou emite um BEEP ou executa qualquer outra parte do programa, mas não consegue fazer ambas as coisas ao mesmo tempo. Isto significa que, por exemplo, a acção num jogo em que o movimento seja essencial tenha de ficar "suspensa" enquanto se produzem efeitos sonoros.

UM TECLADO SONORO

Este programa permite tocar "música" utilizando as duas filas superiores do teclado do *Spectrum* como se fossem as teclas brancas e negras de um piano:



É claro que apenas poderemos tocar uma nota de cada vez e, se mantivermos a tecla SHIFT pressionada, todas as notas serão aumentadas de uma oitava.

A duração de cada nota a tocar é fixada em 0.2 segundos pela linha 200; alterando-a para um valor inferior, tal como 0.02, obtêm-se efeitos interessantes, pois o som produzido assemelhar-se-á ao de um clarinete desafinado, mas, em contrapartida, controlar-se-á a duração de cada nota a tocar.

O problema que se nos deparou na elaboração deste programa consistiu em estabelecer uma relação entre o código obtido quando uma tecla é pressionada e o número a colocar na declaração BEEP. O programa tem igualmente de ignorar o accionamento de teclas "não válidas", ou sejam, as teclas não consideradas neste caso como parte do pretenso teclado de piano (mais propriamente de órgão), e que não estão representadas na figura da página anterior.

A solução adoptada foi a de elaborar um quadro *(array)* com elementos suficientes para representar todos os códigos que pudessem ser originados por uma declaração INKEY$, incluindo o Ø. Cada elemento do quadro poderia então conter o número da frequência correspondente ou — no caso de um código não válido — um número especial que o programa reconhecesse facilmente.

```
 50 REM "TECLADO SONORO"
100 DIM p(123): FOR a=1 TO 123:
LET p(a)=-99: NEXT a
110 FOR n=-2 TO 14: READ k: LET
p(k+1)=n: NEXT n
120 FOR n=10 TO 26: READ k: LET
p(k+1)=n: NEXT n
200 LET n=p(1+CODE INKEY$): IF
n>-60 THEN BEEP .2,n
210 GO TO 200
900 DATA 49,113,119,51,101,52,1
14,116,54,121,55,117,56,105,111,
48,112
910 DATA 7,81,87,4,69,5,82,84,1
0,10,89,11,85,9,73,79,12,80
```

100 Define um quadro com todos os elementos inicializados como "não válidos".

110 Atribui os valores das frequências aos elementos do quadro que correspondem ao teclado de oitava "inferior". Note-se

que isto se tornou relativamente fácil devido ao facto de que (subindo na escala) os valores das frequências são os números inteiros de −2 a 14.

120 Atribui os valores das frequências (10 a 26) aos elementos do quadro correspondentes à oitava "superior".

200 Esta linha faz com que uma nota seja tocada se uma tecla "válida" for pressionada.

210 Faz com que a execução do programa volte atrás para que a nota seguinte possa ser tocada (se houver alguma).

900 Códigos das teclas "válidas" para a oitava "inferior".

910 Códigos das teclas "válidas" para a oitava "superior".

SIMÃO

Simão diz: "Copiem o que eu fizer." Nesta versão computadorizada de um velho jogo infantil, o *Spectrum* toca uma sequência aleatória de notas, sequência essa que o jogador deve repetir.

Há cinco níveis de perícia neste jogo: quanto mais alto for o nível, maior é o número de notas utilizado pelo programa e mais rapidamente se ouve tocar a sequência de notas, relativamente ao nível anterior. Cada jogo consiste em dez partidas, cada uma das quais com uma sequência de notas mais longa do que a anterior. No final do jogo dá-se a pontuação.

```
   5 REM "SIMAO"
  10 RANDOMIZE : DIM p(10): DIM
i$(1)
 100 PRINT AT 6,4;"Nivel de peri
cia (1-5) ?"
 110 INPUT s: IF s<1 OR s>5 THEN
 GO TO 110
 120 PRINT AT 6,30;s: LET s=s+3
 130 PRINT AT 10,0;"Este jogo va
i usar ";s;" notas."
 140 PAUSE 20: PRINT '"Elas esta
o numeradas: "
 150 FOR n=1 TO s: PAUSE 20: PRI
NT AT 12,23;n: BEEP 1,n: NEXT n
 200 LET c=0
 210 FOR g=1 TO 10: PAUSE 100: C
LS : PRINT AT 5,10;"Jogada ";g
 220 FOR n=1 TO g
 230 PAUSE 20: LET p(n)=1+INT (s
*RND): BEEP 2/s,p(n)
 240 NEXT n
 300 PRINT AT 10,8;"Repita isto
!"
 310 FOR n=1 TO g
 320 LET k=CODE INKEY$-CODE "0":
 IF k<1 OR k>8 THEN GO TO 320
 330 BEEP .5,k: IF k<>p(n) THEN
GO TO 360
 340 NEXT n: PRINT AT 13,10;"Cor
recto"
 350 GO TO 400
 360 PRINT AT 13,11; FLASH 1;"ER
RADO": BEEP 1,-30
 370 PAUSE 50: PRINT AT 15,8;"Er
a a nota ";p(n): BEEP 1,p(n)
 400 IF n>1 THEN LET c=c+n+g
 410 FOR a=1 TO 200: NEXT a
 420 NEXT g
 500 CLS : PRINT AT 6,6;"A sua p
ontuacao : ";c
 510 IF c>99 THEN PRINT AT 8,10;
"Excelente !"
 520 PRINT AT 14,10;"Outro jogo
?"
 530 INPUT LINE i$: IF i$="n" TH
EN STOP
 540 IF i$<>"s" THEN GO TO 530
 550 PRINT AT 14,8;"O mesmo nive
l ?"
 560 INPUT LINE i$: IF i$="n" TH
EN RUN
 570 IF i$="s" THEN CLS : GO TO
200
 580 GO TO 560
```

10 Inicialização

100-120 Pedem ao jogador o nível de perícia desejado e aceitam o valor escolhido (se for válido).

130-150 Apresentam as notas que vão ser utilizadas.

200 Início do jogo; esta linha inicializa a pontuação em Ø.

210-420 Ciclo principal do programa, executado uma vez em cada jogada.

220-240 Estas linhas fazem o computador tocar uma sequência aleatória de notas, e gravam essa sequência no quadro p ().

300 Pede a resposta ao jogador.

310-340 Ciclo destinado a obter do jogador o número esperado de notas.

320 Verifica a validade das teclas pressionadas, anulando a operação de qualquer tecla não válida.

330 Faz o computador tocar a nota escolhida pelo jogador. Sai do ciclo, saltando para a linha 360, se a escolha for errada.

360-370 Dizem ao jogador que ele se enganou e indicam-lhe a nota que ele devia ter escolhido.

400-420 Actualizam a pontuação. Esperam um momento (por meio de um ciclo FOR-NEXT na linha 410, pois se uma declaração PAUSE fosse aqui utilizada, a "espera" por ela provocada poderia terminar prematuramente pela operação acidental de qualquer tecla). Em seguida, fazem a execução do programa voltar atrás para nova jogada.

500-510 Imprimem a pontuação e um comentário (se for caso disso).

520-580 Convidam o jogador para um novo jogo.

ATRIBUTOS DE IMAGEM

Além de podermos fazer o computador imprimir caracteres e desenhar figuras no *écran*, também conseguimos controlar a cor e a luminosidade dos elementos impressos ou desenhados e até fazer "piscar" (ou cintilar — *to flash*) partes da imagem. Estas funções são conhecidas pelo nome de "atributos" de imagem.

Ao todo, a imagem produzida pelo *Spectrum* possui quatro atributos. Estes atributos são, nomeadamente:

Cor do fundo (*paper colour*). Também chamada "cor do papel", esta é a cor usada para o fundo da imagem, e é fixada em branco quando se liga o *Spectrum* ou quando se faz uso de um comando NEW. As cores disponíveis são referenciadas pelos números Ø a 7 nos programas que as utilizam.

Por comodidade e para diminuir o risco de erro, os nomes das cores estão impressos por cima das teclas com os números Ø a 7.

Cor da tinta (*ink colour*). Esta é a cor em que se imprime um carácter ou desenha uma figura. É fixada em negro quando se liga o computador ou se introduz um NEW. Como acontece com a cor do fundo, as oito cores disponíveis são referenciadas pelos números Ø a 7.

Brilho (*bright*). O *Spectrum* define dois níveis de brilho (*brightness*) ou luminosidade no *écran* de TV:

BRIGHT Ø dá uma luminosidade normal.
BRIGHT 1 produz um maior brilho.

Intermitência na cintilação (*flash*). O *Spectrum* pode chamar a atenção do utilizador para uma determinada parte da imagem fazendo-a piscar (*flash*), e provoca este efeito alternando as cores do fundo e da tinta:

FLASH Ø é o estado normal.
FLASH 1 provoca a alternância das cores.

É importante compreender que os quatro atributos *paper* (fundo), *ink* (tinta), *bright* (brilho) e *flash* (intermitência na cintilação) funcionam a nível de "célula de carácter". Isto significa que se podem controlar os atributos de qualquer das 24×32 posições de caracteres possíveis no *écran*, mas não é possível controlá-los relativamente a *pixels* individuais dentro das células de caracteres. Como vimos anteriormente, cada célula de carácter é constituída por 8×8 *pixels*. Todos esses 8×8 *pixels* de uma célula partilham um conjunto comum de atributos. Isto implica que só temos uma cor de tinta e uma cor de fundo em cada célula de carácter — embora células de carácter diferentes possam ter atributos diferentes.

Os 768 (32×24) conjuntos de atributos para as 32×24 células de caracteres do *écran* de TV são armazenados na área de atributos da RAM. Estes conjuntos ocupam 768 *bytes*, um *byte* por cada célula de carácter. Em cada *byte*, os *bits* Ø a 2 controlam a cor da tinta, os *bits* 3 a 5 a cor do fundo, o *bit* 6 é o controle de brilho e o *bit* 7 o de cintilação ou intermetência. O valor numérico destes *bytes* é obtido através da função ATTR.

ATRIBUTOS PERMANENTES

As palavras de controle PAPER, INK, BRIGHT e FLASH podem ser utilizadas para formar declarações independentes, por exemplo:

```
1Ø PAPER 7
2Ø BRIGHT Ø
```

Sob esta forma, elas actuam quando o *Spectrum* executa a declaração, e mantêm-se "activas" até que sejam alteradas por uma outra declaração.

Experimentemos introduzir:

```
PAPER 3: INK 4: PRINT "ABCD"
```

e verificaremos que o "ABCD" será impresso numa cor verde berrante sobre um fundo vermelho. Se em seguida "dissermos" ao *Spectrum* para imprimir qualquer outra coisa, por exemplo

```
PRINT "OLA'"
```

também isso será impresso nas mesmas cores.

Note-se que tudo o que tenha sido impresso anteriormente no *écran* manterá as mesmas cores, e as duas últimas linhas, reservadas para o *input* do utilizador e para as mensagens do *Spectrum*, continuarão com as suas cores primitivas.

Para imprimir numa cor diferente, vamos executar uma outra declaração, como a seguinte:

```
PAPER Ø: INK 7: PRINT "AMIGOS"
```

BRIGHT e FLASH podem também ser utilizados de forma semelhante:

```
BRIGHT 1: PRINT "BRIGHT"
FLASH 1: PRINT "FLASH"
```

Os atributos definidos desta maneira — por meio de declarações cuja primeira palavra seja PAPER, INK, BRIGHT ou FLASH — são conhecidos como "atributos permanentes".

Apesar de normalmente estes atributos só actuarem quando se imprime ou desenha qualquer coisa, alteraremos de imediato os atributos de todas as primeiras 22 linhas se utilizarmos uma declaração CLS.

Experimetemos, por exemplo, o seguinte:

```
    BRIGHT 1: CLS
ou  PAPER 3: CLS
ou  FLASH 1: CLS
```

A rotina seguinte mostra como — mesmo quando se utilizam os comandos de gráficos de alta definição — os atributos não se podem alterar dentro de uma mesma célula de carácter. Esta rotina desenha duas linhas de cores diferentes que se intersectam. Quando se aproximam uma da outra o suficiente para estarem na mesma célula de carácter, a segunda linha altera a cor da primeira.

```
1Ø INK 2: PLOT Ø,2Ø: DRAW 255,3Ø
2Ø INK 5: PLOT Ø,5Ø: DRAW 255,-3Ø
```

BORDER

Pensamos que esta é uma boa ocasião para examinar a palavra de comando BORDER. A declaração

```
BORDER n
```

onde n pode assumir os valores de Ø a 7, define a cor do *écran* fora da área utilizada para impressão (enquadramento, moldura ou margem). Vai também afectar a cor das últimas linhas do *écran* (as utilizadas para o *input* do utilizador e para as mensagens do *Spectrum*), mas somente quando o *Spectrum* quer em seguida imprimir uma mensagem ou quando é executada uma declaração de INPUT. Para visualizar este efeito, vamos introduzir a seguinte rotina:

```
1Ø BORDER Ø
2Ø PAUSE 5Ø
```

```
30 INPUT "Carregue em ENTER";a$
40 FOR a=1 TO 200: NEXT a
50 BORDER 7
```

Vamos também experimentar fazer executar esta mesma rotina sem a linha 30.

ATRIBUTOS TEMPORÁRIOS

Podemos também incluir as palavras de comando PAPER, INK, BRIGHT e FLASH numa declaração PRINT, PLOT, DRAW ou CIRCLE:

```
PRINT INK 3;"Viva"
```

Neste caso, o efeito é limitado à declaração em causa, e qualquer coisa impressa ou desenhada por meio de uma declaração posterior voltará a ter os atributos permanentes. Para ver isto, vamos introduzir (e fazer com que o computador execute) a rotina seguinte:

```
10 FLASH 0
20 PRINT FLASH 1;"PISCA"
30 PRINT "NAO PISCA"
```

TARTAN DE CAMBRIDGE

Este programa é muito simples e produz uma imagem atraente. É também um programa útil para regular adequadamente os comandos de imagem do receptor de televisão: devemos procurar obter a imagem mais nítida possível, assegurando-nos de que distinguimos os dois níveis de luminosidade (*brightness*) de cada cor.

```
  5 REM "TARTAN DE CAMBRIDGE"
 10 FOR p=1 TO 7
 20 FOR b=0 TO 1
 30 PRINT BRIGHT b; PAPER p;"·"
 40 NEXT b
 50 NEXT p
 60 GO TO 10
```

CORES 8 E 9

É também possível atribuir à INK (cor da tinta) e a PAPER (cor do fundo) os valores 8 e 9. Estes números fazem com que o *Spectrum* calcule a cor a usar quando ocorrem a impressão ou o desenho.

8 obtém uma cor "transparente": faz com que o *Spectrum* não altere a cor de tinta ou de fundo que tenha sido anteriormente utilizada para uma determinada célula de carácter.

9 significa "contraste". O *Spectrum* escolhe automaticamente entre branco e negro, de modo a tornar a imagem o mais nítida possível, dependendo da outra cor (tinta ou fundo) utilizada. Se atribuir o valor 9 a ambas as cores de fundo e tinta, obteremos uma impressão a negro sobre um fundo branco.

BRIGHT e FLASH podem igualmente tomar o valor 8 (transparente); este número aplicado aqui faz também com que o *Spectrum* mantenha o valor anterior do atributo. Isto exemplifica-se pela rotina seguinte:

```
10 PRINT BRIGHT 1;"•"
20 PRINT FLASH 1;"•"
30 PRINT AT 0,0; BRIGHT 8;"AB"
40 PRINT FLASH 8;"CD"
```

"•" = espaço em branco

KLIVEOSCÓPIO

Este programa leva cerca de dez minutos a executar, apresentando durante esse período uma sequência de padrões simétricos muito coloridos, de complexidade gradualmente crescente.

```
  50 REM "KLIVEOSCOPIO"
 100 FOR c=50 TO 1 STEP -1
 110 FOR a=0 TO 11
 120 FOR b=a TO 19
 130 LET d=INT (a*b/c+(a+b)/c+c)
: PAPER d-9*INT (d/9)
 140 PRINT AT a,b;".";AT a,31-b;
".";AT 21-a,b;".";AT 21-a,31-b;"
."
 150 PRINT AT b,a;".";AT b,31-a;
".";AT 21-b,a;".";;AT 21-b,31-a;
"."
 160 NEXT b
 170 NEXT a
 180 NEXT c
```

DESENHAR SOBRE O FUNDO

Uma declaração PRINT define sempre todos os atributos das células dos caracteres que imprime, de acordo com os últimos valores atribuídos, quer sejam atributos permanentes definidos por declarações INK, PAPER, BRIGHT ou FLASH, quer sejam atributos temporários definidos através de palavras de comando apropriadas, incluídas numa declaração PRINT.

Os comandos para alta definição PLOT, DRAW e CIRCLE funcionam, contudo, de forma diferente. Normalmente, estes comandos apenas afectam a cor da tinta das células de carácter neles envolvidas. Os outros atributos só são alterados se forem temporariamente definidos através de palavras de comando PAPER, BRIGHT ou FLASH incluídas na própria declaração PLOT, DRAW ou CIRCLE.

Por exemplo, a rotina

```
10 PAPER 7: INK 0: CLS
20 PAPER 6: CIRCLE 100,100,50
```

desenha um círculo negro sobre um fundo branco; a declaração PAPER 6 na linha 20 não tem qualquer efeito. No entanto, se alterarmos a linha 20 para

```
20 CIRCLE PAPER 6;100,100,50
```


a cor do fundo das células de carácter, pelas quais o círculo passa ao ser desenhado, tornar-se-á amarela.

Isto porporciona-nos uma forma de alterar rapidamente a cor do fundo de uma fila ou de uma coluna de células de carácter, desenhando uma linha através delas por meio de uma declaração DRAW (ou CIRCLE) incluindo uma definição PAPER.

QUADRADOS QUE "RESPIRAM"

Como uma demonstração prática, o programa seguinte desenha uma série de quadrados concêntricos cuja cor varia constantemente. A cor da tinta e a do fundo são mantidas uma igual à outra, mas é de notar que, enquanto a cor da tinta pode ser definida permanentemente (linha 110), a cor do fundo tem de ser redefinida em cada declaração DRAW (linhas 120 e 130).

```
  10 REM QUADRADOS QUE RESPIRAM
  50 BORDER 0: PAPER 0: CLS : LE
T c=1
 100 FOR a=-8 TO 8 STEP 16
 110 FOR b=41-5*a TO 41+5*a STEP
 a: INK c
 120 PLOT 128-b,87-b: DRAW PAPER
 c;b+b,0: DRAW PAPER c,0,b+b
 130 DRAW PAPER c,-b-b,0: DRAW P
APER c,0,-b-b
 140 LET c=c+1: IF c>7 THEN LET
c=1
 150 NEXT b
 160 NEXT a
 170 GO TO 100
```

50 Limpa o *écran*, tornando-o completamente negro. Atribui um valor inicial à cor c.

100-160 Ciclo destinado a alternar a sequência de desenho dos quadrados do interior para o exterior e do exterior para o interior.

110-150 Ciclo destinado a desenhar uma sequência de quadrados concêntricos.

120-130 Desenham um quadrado de cor c.

140 Esta linha altera a cor c.
170 Faz com que a sequência seja repetida indefinidamente.

OVER E INVERSE

As palavras de comando OVER e INVERSE têm semelhanças com as palavras de comando de atributo INK, PAPER, BRIGHT e FLASH, pelo facto de afectarem a impressão ou o desenho e de poderem ser usadas como a primeira palavra de uma declaração, de modo a atribuírem valores permanentes aos atributos, ou serem incluídas em declarações PRINT, DRAW, PLOT ou CIRCLE, de forma a atribuírem valores temporários a esses mesmos atributos.

Contudo, estes valores não são armazenados na área dos atributos da RAM, mas constituem instruções para o *Spectrum* sobre a forma como este deve interpretar um comando PRINT, PLOT, DRAW ou CIRCLE.

As regras básicas são as seguintes:

OVER Ø: Apaga o que quer que estivesse anteriormente na célula de carácter afectada (se se tratar de PRINT) ou no *pixel* afectado (se se tratar de PLOT, DRAW ou CIRCLE).

OVER 1: Se esta palavra for utilizada com uma declaração PLOT, DRAW ou CIRCLE, altera a cor do *pixel* para tinta se ela anteriormente fosse cor de fundo, ou para fundo se ela anteriormente fosse cor de tinta.
Se for utilizada com uma declaração PRINT, todos os *pixels* que tenham cor de fundo no carácter a imprimir mantêm a cor inalterada, enquanto os que tenham cor de tinta nesse carácter mudam para:
— cor de tinta se anteriormente fosse "fundo".
— cor de fundo se anteriormente fosse "tinta".

INVERSE Ø: É o estado normal: tanto a impressão (PRINTing) como o traçado de linhas (PLOTing, etc.) utilizam cor de tinta.

INVERSE 1: Neste estado, a impressão ou o traçado utilizam a cor do fundo. Quando esta ordem é utilizada com PRINT, o fundo da célula onde se imprime o carácter passa para a cor de tinta, ou seja, as cores de fundo e tinta são trocadas, resultando daí um carácter em *inverse video* (com a imagem em "negativo").

Se OVER 1 e INVERSE 1 forem ambas simultaneamente utilizadas com PLOT, DRAW ou CIRCLE, o resultado será não haver qualquer alteração da imagem no *écran*.

Se INVERSE 1 e OVER 1 foram ambas simultaneamente utilizadas com um comando PRINT, obter-se-á como resultado uma combinação do carácter anteriormente existente na célula com o "novo" carácter a imprimir aí, isto devido à função OVER 1, sendo este resultado apresentado em *inverse video*.

Uma declaração PRINT altera sempre as cores de tinta e de fundo das células de carácter afectadas pela impressão para os seus últimos valores, quer estes definam cores permanentes atribuídas às células por declarações PAPER ou INK independentes quer sejam valores temporários estabelecidos por uma função PAPER ou INK contida na declaração PRINT.

Podemos utilizá-la de modo a alterar as cores do fundo e da tinta de qualquer célula de carácter, sem afectar o carácter nela previamente impresso, por meio de uma declaração com o formato

```
PRINT AT r,c; OVER 1; INK i; PAPER p;"•"
```

pois a impressão numa determinada célula de carácter de um espaço em branco com OVER 1 deixa inalterado o carácter nela previamente impresso.

Isto conduz a uma forma rápida de alterar a cor do fundo, ou da tinta de todo o *écran*, sem afectar de outro modo a imagem nele apresentada:

```
DIM z$(22*32): PRINT AT Ø,Ø; OVER 1; INK i; PAPER
p;z$
```

(A primeira declaração da linha serve para criar uma cadeia z$ contendo espaços em número exactamente suficiente para encher o *écran*.) Se quisermos — por exemplo — alterar a cor da tinta,

mas deixar as cores do fundo inalteradas, deveremos atribuir à cor do fundo o valor 8 (transparente).

Analogamente, usa-se a combinação INVERSE 1 OVER 1 com um comando DRAW, PLOT ou CIRCLE para mudar as cores do fundo ou da tinta das células de carácter intervenientes sem afectar a forma do que quer que nelas tivesse sido previamente impresso ou traçado. Mas é preciso notar que, para alterar a cor do fundo, a função PAPER tem de ser incluída no comando DRAW, PLOT ou CIRCLE. Para ter um exemplo desta técnica, vamos experimentar a rotina:

```
1ØØ PAPER 7: INK Ø: INVERSE Ø: OVER Ø: CLS
11Ø FOR a=1 TO 22*32: PRINT "*";: NEXT a
12Ø INK 4: INVERSE 1: OVER 1: PLOT Ø,Ø: DRAW
PAPER 2;255,175
```

Esta rotina, primeiramente, enche o *écran* com estrelas negras sobre um fundo branco e, em seguida, altera as que se situam sobre uma linha diagonal, ficando estas a verde sobre fundo vermelho.

CARACTERES DE CONTROLE DE IMPRESSÃO

Se dermos uma olhadela ao Apêndice A do manual de programação BASIC do *Spectrum*, veremos que alguns dos caracteres com os códigos 6 a 23 correspondem às funções de impressão AT, INVERSE, etc. Por eles serem caracteres, podemos incluí-los numa cadeia, e o *Spectrum* actuará sobre eles quando a cadeia for impressa.

Podemos criar estes caracteres utilizando a função CHR$ X, a qual obtém o único carácter cujo código é X. Assim,

```
    LET A$="1"+CHR$ 6 +"2"
ou  LET A$="1•2": LET A$(2)=CHR$ 6
```

dá-nos uma cadeia cujo segundo carácter é a vírgula de impressão (PRINT *comma*).

```
CLS: PRINT A$
```


imprime "1" no início da primeira linha do *écran*, e "2" no meio, como se tivéssemos feito executar

```
CLS: PRINT "1", "2"
```

CHR$ 8 é útil, pois tem como efeito deslocar a posição de impressão uma célula para a esquerda:

```
CLS: PRINT "AB"+CHR$ 8 +"CD"
```

daria ACD como resultado.

CHR$ 9 a 12 não têm qualquer função, infelizmente.

CHR$ 13 desloca a posição de impressão para o início da linha seguinte.

CHR$ 16 (INK) e CHR$ 17 (PAPER) devem ser seguidos de um único carácter, que tenha um código de Ø a 9 (correspondente a uma cor de número Ø a 9). Experimente-se, por exemplo, fazer executar a seguinte rotina:

```
1Ø FOR C=Ø TO 9
2Ø PRINT CHR$ 16 + CHR$ C + "INK = ";C
3Ø PRINT CHR$ 17 + CHR$ C + "PAPER = ";C
4Ø NEXT C
```

Do mesmo modo, os códigos 18 a 21, de controle de FLASH, BRIGHT, INVERSE e OVER, devem ser seguidos, cada um deles, por um único carácter, que tenha código Ø, 1 ou 8 (transparente). Vamos experimentar o seguinte:

```
PRINT "ESCURO" + CHR$ 19 + CHR$ 1 + "CLARO"
```

CHR$ 22 (AT) deve ser seguido de dois *bytes*, que definam os números de linha e de coluna:

```
PRINT CHR$ 22 + CHR$ 7 + CHR$ 8 + "OLA'"
```

é o mesmo que

```
PRINT AT 7,8;"OLA'"
```

CHR$ 23 (TAB) tem igualmente de ser seguido de dois *bytes*, servindo o primeiro para definir o número de coluna; o segundo é ignorado, mas tem, todavia, de ser incluído. Assim,

```
PRINT CHR$ 23 + CHR$ 18 + CHR$ 99 + "ADEUS"
```

é equivalente a

```
PRINT TAB 18; "ADEUS"
```

Note-se que, apesar de em todos os exemplos aqui apresentados se ter utilizado o sinal "+" para ligar partes individuais de uma cadeia numa declaração PRINT, poder-se-ia usar o sinal de ponto e vírgula em seu lugar. Exemplificando:

```
PRINT CHR$ 23;CHR$ 18;CHR$ 99;"ADEUS"
```

Note-se também que todos os atributos (FLASH, PAPER, etc.) controlados desta forma são temporários, e só se mantêm até ao fim da declaração PRINT que os inclui.

Tudo isto pode parecer excessivamente complicado, mas o leitor deve lembrar-se de que assim pode construir uma cadeia que inclua estes códigos e que eles actuarão sempre que a cadeia seja impressa.

COR DIRECTA

É possível controlar a cor apresentada quando o programa é listado e também a cor dos caracteres numa cadeia literal (isto é: uma cadeia definida entre aspas) através do teclado, sem fazer uso dos comandos PAPER, INK, etc.

Se, ao introduzir uma linha de programa, carregarmos simultaneamente nas teclas CAPS SHIFT e SYMBOL SHIFT de modo a que o cursor passe para "E", e, seguidamente, pressionarmos qualquer das teclas "0" a "7", alteramos a cor do fundo do que quer que se introduza daí para a frente.

Se mudarmos o cursor para "E" e mantivermos a tecla CAPS SHIFT pressionada quando carregarmos em qualquer das teclas "0" a "7", alteramos a cor da tinta de tudo o que venha a seguir-se.

E isto não é tudo: se pressionarmos a tecla "9" em vez de qualquer das teclas "0" a "7", os caracteres seguintes serão apresentados sobre fundo brilhante, enquanto, se mantivermos a tecla CAPS SHIFT pressionada quando carregarmos na tecla "9", os caracteres seguintes aparecerão a piscar *(flashing)*. A tecla "8" efectua o processo inverso, fazendo o brilho voltar ao normal ou terminando com o piscar.

Tudo isto, na verdade, faz inserir caracteres de controle na listagem de um programa. Todas as cores, o piscar ou o brilho que definirmos deste modo serão visíveis sempre que o programa for listado, mas quando o programa é executado apenas afectarão a imagem se os caracteres de controle forem inseridos numa cadeia literal.

Apesar de os verdadeiros códigos de controle não serem visíveis numa listagem, notaremos a sua presença se "EDITarmos" uma linha que os contenha. Neste caso, o cursor parecerá hesitar quando passa pelo local da linha onde algum deles se situa.

Uma ideia interessante: atribuindo desta forma à "tinta" a mesma cor que ao fundo, oculta-se parte da listagem de um programa de qualquer olhadela inoportuna, embora essa parte se torne visível se o programa for listado através de uma impressora *ZX*.

Capítulo 11

Movimento

Os jogos mais emocionantes são, sem dúvida, os de grande acção, com objectos a "voar" com rapidez pelo *écran*. Neste capítulo debruçar-nos-emos sobre as formas de programar o *Spectrum* de modo a criar objectos com movimento.

Mas — antes que o leitor se entusiasme muito — deve ser dito desde já que não é possível atingir um nível muito elevado em jogos cujos programas sejam escritos em BASIC, pois a sua execução é demasiado lenta. Portanto, apesar de os programas apresentados neste capítulo definirem todos eles jogos bastante divertidos, muitos apenas oferecem uma versão simplificada do jogo: sacrifica-se a complexidade para se conseguir uma rapidez de movimento razoável.

Quem quiser escrever programas para jogos complexos e de acção muito rápida terá de utilizar linguagem máquina, ou talvez Forth, as quais permitem uma execução dezenas de vezes mais rápida do que a permitida pelo BASIC, mas situam-se fora do âmbito deste livro.

A técnica básica utilizada para criar a impressão de movimento é muito simples: consiste em imprimir *(print)* ou "traçar" *(plot)* repetidamente um objecto no *écran* em posições ligeiramente diferentes umas das outras. Por exemplo:

```
      FOR a=Ø TO 255: PLOT a,1ØØ: NEXT a
ou    FOR a=Ø TO 31: PRINT AT Ø,a;"*": NEXT a
```

Se experimentarmos estas rotinas, verificaremos que, em vez de um único objecto a deslocar-se sobre o *écran*, se obterá uma linha de comprimento crescente. Apesar de isto não ser o que geralmente se deseja, por vezes é útil e constitui a base do jogo LESMAS.

LESMAS

Lee Gentry

É um jogo para dois jogadores, onde cada um tenta apanhar o outro numa rede de baba viscosa!

As duas lesmas rastejam progressivamente através de um labirinto, deixando atrás delas um rasto venenoso. Se uma das lesmas tocar num rasto ou em qualquer das paredes do labirinto, tem morte imediata e o jogo termina.

A finalidade do jogo consiste, portanto, em manobrar a nossa lesma de forma a que a do adversário fique encurralada. As lesmas estão sempre em movimento, e por isso é necessário ter reflexos rápidos para evitar que a nossa lesma tenha morte instantânea.

A lesma 1 começa a rastejar a partir do canto superior esquerdo do *écran* e é controlada pelas quatro teclas "W", "A", "S" e "Z", conforme se queira que ela se desloque respectivamente para cima, para a esquerda, para a direita e para baixo.

A lesma 2 começa a deslocar-se a partir do canto inferior direito do *écran*, em sentido contrário ao da lesma 1, e é controlada pelas teclas "O", "K", "L" e SYMBOL SHIFT.

O programa utiliza as variáveis *a* e *b* para seguir a posição em cada momento da lesma 1 no *écran*, e mais duas variáveis, *c* e *d*, para "recordar" a direcção em que ela se está a deslocar: *c* terá o valor 1 se a lesma estiver a deslocar-se para a direita, −1 se estiver a deslocar-se para a esquerda; *d* terá o valor 1 se a lesma estiver a deslocar-se verticalmente, para cima, no *écran*, e −1 se estiver a deslocar-se para baixo. Como a lesma não pode estar parada, o programa não permite que *c* e *d* tomem ambas o valor zero ao mesmo tempo. Calcular a posição seguinte da lesma é apenas questão de adicionar *c* com *a* e *d* com *b*.

As variáveis *u*, *v*, *w* e *x* que aparecem no programa têm funções idênticas às mencionadas e referem-se à lesma 2.

Empregamos o comando PLOT para marcar no *écran* as últimas posições das lesmas e a função POINT para verificar se elas estão prestes a colidir com qualquer coisa.

Com a função IN examinamos o teclado (ver Capítulo 8); aqui preferimo-la a INKEY$, pois com dois jogadores há uma boa probabilidade de ambos pressionarem uma tecla ao mesmo

tempo, situação essa que a função INKEY$ não pode resolver.

Na versão portuguesa preferimos manter a letra "s" para a variável identificativa da lesma (*slug*, em inglês), por a letra "l" se poder confundir com o algarismo "1".

```
   1 REM "LESMAS"
   5 GO TO 300
  10 REM Lesma 1
  20 LET e=(IN 65022=189)-(IN 65
022=190)
  30 LET f=(IN 64510=189)-(IN 65
278=189)
  40 IF e<>0 OR f<>0 THEN IF NOT
 (e<>0 AND f<>0) THEN LET c=e: L
ET d=f
  50 LET a=a+c: LET b=b+d
  60 IF POINT (a,b) THEN LET s=2
  GO TO 200
  70 PLOT a,b
 100 REM Lesma 2
 110 LET y=(IN 49150=189)-(IN 49
150=187)
 120 LET z=(IN 57342=189)-(IN 32
766=189)
 130 IF y<>0 OR z<>0 THEN IF NOT
 (y<>0 AND z<>0) THEN LET w=y: L
ET x=z
 140 LET u=u+w: LET v=v+x
 150 IF POINT (u,v) THEN LET s=1
: GO TO 200
 160 PLOT u,v
 170 GO TO 20
 200 REM Fim
 210 PAUSE 1: PAUSE 100: CLS
 220 PRINT AT 10,8;"A lesma ";s;
" ganhou"
 230 PRINT AT 12,4;"Tente outra
vez, lesma ";(NOT (s-1))+1
 240 STOP
 300 LET a=20: LET b=155: LET c=
1: LET d=0
 310 LET u=235: LET v=25: LET w=
-1: LET x=0
 320 CLS
 330 LET m=40
 340 FOR n=1 TO 6
 350 FOR p=1 TO 7
 360 IF n<=5 AND p<=6 THEN PLOT
m*p-28,m*n-32: DRAW 30,0
 370 IF n<=4 THEN PLOT m*p-33,m*
n-27: DRAW 0,30
 380 NEXT p
 390 NEXT n
 400 PLOT 0,0: DRAW 255,0: DRAW
0,175: DRAW -255,0: DRAW 0,-175
 410 GO TO 10
```

20-170 Ciclo principal do programa, executado para deslocar cada lesma de um *pixel*.

20- 40 Estas linhas verificam se alguma das teclas do jogador da esquerda ("W","A","S","Z") está a ser pressionada e, em caso afirmativo, atribui os valores adequados às variáveis *c* e *d*.

50 Calcula a nova posição da lesma 1.

60 Termina o programa se a lesma colide com qualquer coisa.

70 Traça no *écran* a nova posição da lesma 1.

100-160 Rotina semelhante à anterior, mas para movimentar a lesma 2.

200-240 Rotina de fim de programa.

300-410 Rotina de inicialização, que desenha o labirinto, define as posições e as direcções iniciais do deslocamento das lesmas. Colocámo-la no final da listagem para que o ciclo principal (linhas 20-170) ficasse perto do início e assim fosse executado o mais rapidamente possível.

PONTOS MÓVEIS

Para a maioria dos jogos, pretende-se um objecto móvel em vez de uma linha de comprimento crescente. Para obter o efeito desejado, a imagem anterior tem de ser apagada logo que a nova for apresentada no *écran:*

```
1Ø FOR a=1 TO 255
2Ø PLOT a,Ø: PLOT INVERSE 1;a-1,Ø
3Ø NEXT a
4Ø PLOT INVERSE 1;255,Ø: GO TO 1Ø
```

(A linha 4Ø apaga o último ponto no limite do *écran* antes de repetir o movimento.)

Executar esta rotina mostra como se emprega PLOT para dar uma impressão de movimento regular, mas neste caso a imagem é muito pequena e não se desloca muito depressa.

Experimentemos utilizar PRINT AT em vez de PLOT:

```
1Ø FOR a=1 TO 31
2Ø PRINT AT Ø,a;"*"; AT Ø,a-1;"●"
3Ø NEXT a
4Ø PRINT AT Ø,31;"●": GO TO 1Ø
```

Isto já se aproxima muito mais do desejado — na verdade teremos mesmo de diminuir a velocidade, acrescentando a linha

```
25 PAUSE 3
```

para que possamos ver a estrela convenientemente. Na prática, o programa tem de fazer outras coisas além de mover simplesmente um objecto, com o inconveniente de retardar a acção. De facto, um dos maiores problemas a enfrentar quando se escreve um programa de jogos em BASIC onde intervenham formas e gráficos móveis consiste em manter suficientemente alta a velocidade de execução do ciclo principal que define a imagem.

Em ambas as rotinas anteriores fizemos com que o "novo" objecto fosse impresso no *écran* antes de a imagem anterior ser apagada, pois por este processo se obtém geralmente um movimento mais regular, especialmente se houver uma demora significativa entre os dois acontecimentos.

QUEDA DE ELEFANTES

Uma das vantagens da utilização de PRINT AT para apresentar a imagem de objectos em movimento é a possibilidade de fazer uso de caracteres gráficos definidos pelo utilizador.

O jogo seguinte, simples mas cativante, tem origem numa velha história que diz que os elefantes descem das árvores

sentando-se numa folha e esperando que o Outono chegue... O que a história não menciona é que um elefante que caia tem de ser apanhado num prato, pois de outro modo esborrachar-se-á ao chocar com o solo!

O jogo principia com 32 elefantes a flutuar no céu. Começam então a cair um a um, e ao jogador cabe a função de deslocar o "pires" ou "disco voador" (utilizando as teclas "5" e "8" para o deslocar), de modo a interceptar-lhe a queda. O programa proporciona também um acompanhamento "musical".

No final do jogo, informa-se o jogador de quantos elefantes salvou. A quem se queixe de que o pires não se desloca com a rapidez suficiente para apanhar todos os elefantes, deve responder-se com firmeza: "É a vida!"

```
  50 REM "ELEFANTES"
 100 DATA 0,0,0,0,0,51,126,255
 110 DATA 0,48,126,95,127,127,17
9,51
 120 DATA 0,0,0,0,0,0,195,126
 130 FOR a=0 TO 23: READ b: POKE
USR "a"+a,b: NEXT a
 140 RANDOMIZE
 150 DIM h(32): LET m=16: LET s=
0
 160 BORDER 1: PAPER 1: CLS
 170 PRINT AT 21,0; PAPER 4;,,
 180 FOR a=31 TO 0 STEP -1: PRIN
T INK 7;AT 0,a;"B": PAUSE 10: NE
XT a
 200 FOR t=1 TO 32
 210 LET x=INT (32*RND): IF h(x+
1)<>0 THEN GO TO 210
 220 LET h(x+1)=1
 300 FOR y=1 TO 19
 310 LET n=m
 320 IF INKEY$="5" AND m>0 THEN
LET n=n-1
 330 IF INKEY$="8" AND m<31 THEN
 LET n=n+1
 340 PRINT AT 19,m;".";AT 19,n;
INK 3;"C";AT y,x; INK 7;"B";AT y
-1,x;"."
 350 LET m=n: BEEP .005,25-y
 360 NEXT y
 400 IF x<>m THEN PRINT AT 19,x;
".";AT 20,x; INK 2;"A": BEEP .2,
-50: GO TO 430
 410 BEEP .05,10: BEEP .05,15: B
```

```
EEP .1,22: LET s=s+1
 420 PRINT AT 19,x;" "; INK 7;AT
 20,x;"B"
 430 NEXT t
 500 PRINT AT 5,7; INK 9;"SALVOU
 ";s;" ELEFANTES"
```

Note-se que as letras maiúsculas A, B e C na listagem devem ser introduzidas como caracteres gráficos A, B e C. Depois de o programa ter sido executado uma vez, eles aparecerão na listagem nas suas formas correctas.

100-130 Estas linhas definem os três caracteres gráficos definíveis pelo utilizador: "A" representa os "horrendos restos mortais" de um elefante "esborrachado" (na versão original, o autor faz um trocadilho com *grizzly* — acinzentado, pardo — e *grisly* — horrendo, macabro —, pois estas palavras têm pronúncia idêntica); "B" representa um elefante vivo e "C" representa o pires. Os oito números binários necessários para definir cada carácter foram "traduzidos" em oito números decimais de forma a facilitar a sua introdução pelo utilizador (linhas 100-120).

150 Define um quadro *(array)* h() de 32 elementos, com todos os elementos inicializados em zero. Isto assinala quando cada um dos 32 elefantes cai durante o jogo. A posição inicial do prato é fixada em 16, e a pontuação em 0.

160-170 Estabelecem as cores do fundo e desenham o "solo".

180 Imprime uma linha de 32 elefantes.

200-430 Ciclo principal do programa, executado uma vez por cada elefante.

210-220 Escolhem um elefante que ainda não tenha caído; em seguida assinalam-no como em queda (atribuem à variável h(x+1) o valor 1, de modo a que o elefante x em causa não seja escolhido novamente).

300-360 Ciclo interior, executado para fazer cair o elefante escolhido.

310-330 Calculam a nova posição (n) do prato a partir da posição anterior (m) e da utilização da tecla "5" ou "8".

340 Apaga as imagens anteriores do elefante em queda e do prato e imprime novas imagens nas últimas posições calculadas para o elefante e para o prato.

350 Actualiza a posição anterior do prato e faz emitir um ruído.

360 Faz a execução voltar atrás para que o elefante desça mais um pouco.

400-420 Por esta altura, o elefante já desceu até ao nível do prato; estas linhas fazem emitir um ruído adequado e imprimem a forma apropriada, dependente do facto de o jogador ter ou não conseguido apanhar o elefante no prato (x=m ou x<>m, respectivamente).

430 Apronta a queda do próximo elefante.

500 Apresenta a pontuação final. À cor da tinta é atribuído o número 9 (contraste) para evitar ter de se pensar numa cor apropriada!

BOLA DEMOLIDORA

Quem está a mover objectos no *écran* quererá provavelmente saber se eles colidiram com outro objecto ou se estão prestes a fazê-lo. Uma forma de o conseguir seria manter um registo da posição de cada objecto no *écran*. Ou talvez se pudesse elaborar um quadro que funcionasse como um "mapa" do *écran*, de forma a que o valor de cada elemento do quadro indicasse o que estava a ser apresentado em cada momento nessa parte do *écran*.

Outra solução é elaborar o programa de forma a imprimir elementos diferentes com cores diferentes (ou com atributos BRIGHT ou FLASH diferentes). A função ATTR indica então que se encontra em qualquer parte do *écran*.

ATTR utiliza um par de variáveis x e y, tal como a função AT numa declaração PRINT — com a diferença de que ATTR necessita que elas sejam colocadas entre parênteses (que linguagem complicada!). Esta função fornece como resultado um número inteiro entre 0 e 255, o qual é a soma de:

128 se houver intermitência *(flashing)*
\+ 64 se houver brilho *(bright)*
\+ 8 vezes o número da cor do fundo (0-7)
\+ o número da cor da tinta (0-7)

Ora isto é uma trapalhada tremenda. Para separar os diversos atributos individuais temos de utilizar rotinas como a seguinte:

```
LET a= ATTR (y,x)
Flash=INT (a/128)
Brilho=INT ((a-128*INT (a/128))/64)
Fundo=INT ((a-64*INT (a/64))/8)
Tinta=a-8*INT (a/8)
```

Este jogo constitui um bom exemplo da utilização de ATTR para determinar o que se encontra no *écran*, e onde. O jogo principia com um muro construído com várias camadas de tijolos, umas atrás das outras, e uma bola em movimento, a qual o jogador tenta atingir com uma "raqueta", de forma a que ela vá bater no muro. Quando a bola atinge o muro, chocando com um tijolo, o tijolo atingido desaparece e é atribuída uma pontuação — tanto maior quanto maior for a profundidade da camada a que o tijolo pertença.

Nesta versão do jogo, o jogador principia com dez bolas, e perde uma sempre que não consiga atingi-la com a raqueta. A raqueta é comandada com as teclas "K" (para cima) e "M" (para baixo), e a bola ressaltará perpendicularmente à superfície da raqueta ou, se bater na parte lateral desta, retrocederá na direcção em que veio. Se o jogador fizer um buraco que atravesse o muro de um lado ao outro, as bolas podem passar através do buraco e fazer ricochete contra a "parede de fundo" do "campo" (situada por detrás do muro), dando melhores hipóteses ao jogador de atingir um tijolo da última camada do muro, que é a que oferece maior pontuação

Atribuindo cores diferentes às várias camadas de tijolos, podemos utilizar a função ATTR para determinar não só se a bola atingiu um tijolo, mas também de qual camada. Por outro lado, se o receptor de TV for a cores, este processo torna o jogo visualmente mais atraente.

Uma armadilha em que o principiante pode cair ao utilizar a função ATTR resulta do facto de todas as células de carácter do *écran* terem cores de tinta associadas a elas, mesmo que nada apareça nessa célula. Assim:

```
PAPER 7: CLS : PRINT AT 3,3; INK 4;"●"
```


dará origem a um *écran* vazio, mas se fôssemos utilizar ATTR para averiguar qual a cor de "tinta" da célula 3,3 verificaríamos que era a cor 4.

Por este motivo, este programa foi escrito de forma a utilizar cores de tinta e fundo permanentes e de valor 0, só as alterando temporariamente — por meio de funções INK ou PAPER incluídas numa declaração PRINT — quando é absolutamente necessário fazer aparecer qualquer coisa no *écran*. Deste modo,

```
PRINT AT y,x;"●"
```

não só irá apagar qualquer imagem precedente como podemos ter a certeza de que irá também alterar a cor da tinta, atribuindo-lhe o número zero.

```
  50 REM "BOLA DEMOLIDORA"
 100 PAPER 0: INK 0: BORDER 7: I
NVERSE 0: OVER 0: FLASH 0: CLS
 110 DATA 0,126,126,126,126,126,
126,0
 120 DATA 0,60,126,126,126,126,6
0,0
 130 FOR a=USR "a" TO USR "a"+15
: READ b: POKE a,b: NEXT a
 140 RANDOMIZE : LET t=10: LET s
=0: LET c=10
 150 PRINT INK 6;"PONTOS : 0"
 160 FOR a=1 TO 10: PRINT INK 6;
AT 0,21+a;CHR$ 145: NEXT a
 170 FOR a=1 TO 5: FOR b=1 TO 21
 180 PRINT PAPER 7; INK 6-a;AT b
,a;CHR$ 144
 190 NEXT b: NEXT a
 200 FOR t=1 TO 10: PRINT AT 0,2
1+t;"•"
 300 LET x=6: LET y=1+INT (21*RN
D)
 310 LET h=1: LET v=1: IF RND>.5
 THEN LET v=-1
 400 PRINT INK 6;AT y,x;CHR$ 145
 410 IF INKEY$="k" AND c>1 THEN
PRINT AT c,31;"•": LET c=c-1
 420 IF INKEY$="m" AND c<21 THEN
 PRINT AT c,31;"•": LET c=c+1
 430 PRINT INK 7;AT c,31;CHR$ 13
8: IF x>30 THEN GO TO 600
 440 IF x=30 THEN LET d=c-y: IF
d=0 OR d=v THEN LET h=-1: BEEP .
```

```
02,12: IF d=v THEN LET v=-v
 450 IF x<1 THEN LET h=1: BEEP .
02,0
 460 IF y<2 THEN LET v=1: BEEP .
02,0
 470 IF y>20 THEN LET v=-1: BEEP
 .02,0
 480 PRINT AT y,x;"•": LET x=x+h
: LET y=y+v: LET a=ATTR (y,x)
 490 LET a=a-8*INT (a/8): IF a=0
 OR a=7 THEN GO TO 400
 500 PRINT INK 7;AT y,x;"*": BEE
P .05,12+3*a
 510 LET s=s+a: PRINT INK 6;AT 0
,8;s; INK 0;AT y,x;"•"
 520 GO TO 300
 600 BEEP .1,-20: PRINT AT y,x;"
•"
 610 NEXT t
 700 PRINT INK 6;AT 0,10; FLASH
1;"JOGO TERMINADO"
```

100 Atribui a todos os atributos de imagem os seus valores nominais.

110-130 Definem os caracteres gráficos definíveis pelo utilizador "A" e "B" de modo a que se pareçam com um tijolo e com uma bola, respectivamente. Por comodidade, estes são referidos na listagem do programa como CHR$ 144 e CHR$ 145.

140 Atribui valores iniciais à pontuação e à posição da raqueta.

150-190 Imprimem os cabeçalhos e as camadas de tijolos.

200-610 Ciclo principal do programa, executado uma vez por cada conjunto de dez bolas.

300-310 Lançam uma bola a partir da posição x,y com uma velocidade horizontal h=1 e uma velocidade vertical v igual a + 1 ou a − 1.

400-490 Ciclo interior, executado de modo a mover a bola no *écran* até que ela saia pelo lado direito ou choque contra um tijolo. Permite também que o jogador mova a raqueta.

400 Imprime a bola na posição corrente x,y.

410-420 Apagam a imagem precedente da raqueta e calculam a sua nova posição se alguma das teclas "K" ou "M" tiver sido pressionada.

430 Imprime a raqueta na sua última posição. Faz também com que a sequência de execução saia do ciclo se a bola se encontrar do lado direito do *écran*.

440 Algoritmo (do autor) destinado a preparar a bola de modo a que ela ressalte correctamente ao colidir com a superfície frontal ou com os lados da raqueta, se estiver em posição para o fazer.

450-470 Preparam o ricochete da bola se esta colidir contra o lado superior, inferior ou lateral esquerdo da área de jogo.

480 Apaga a imagem anterior da bola. Calcula a sua nova posição e examina o que se encontra nessa célula de carácter.

490 Se a cor da tinta da célula correspondente à nova posição da bola for 0 (negro — o que significa que a célula está vazia) ou 7 (branco — raqueta), o ciclo volta à linha 400 de modo a deslocar novamente a bola.

500-520 A bola colidiu com um tijolo e, portanto, estas linhas fazem uma estrela piscar nessa posição, o computador emitir um ruído, incrementar a pontuação e a sequência de execução voltar à linha 300 de forma a lançar novamente a bola.

600-610 A bola saiu da área de jogo. Se restarem ainda bolas, a sequência de execução volta à linha 200 para tomar mais uma bola e continuar o jogo.

700-710 O jogador acaba de perder a última bola e, portanto, o jogo termina.

PÁSSAROS

Podemos também criar a impressão de movimento fazendo "rolar" *(scroll)* repetidamente para cima toda a imagem do *écran*, como exemplificamos neste programa.

Imagine o leitor que é um pássaro, esvoaçando alegremente ao longo do topo do *écran* (utilizando a tecla "5" para se deslocar para a esquerda e a tecla "8" para ir para a direita). Mas, como qualquer pássaro autêntico, terá de comer; se quer continuar a voar tem de continuar a comer.

A "comida" é constituída pelos "insectos" verdes que flutuam ao seu encontro (na realidade são tão pequenos que aparecem apenas como pontos no écran), e deve tentar apanhar tantos quantos puder. O número que aparece na parte inferior do *écran* indica a reserva de alimento que lhe resta; se não apanhar quaisquer insectos, a reserva vai diminuindo até que — quando o número chega a zero — não consegue voar mais.

O pior é que há uma série de indivíduos a dispararem setas contra si, setas essas a que tem de escapar, pois de contrário é morto.

No programa da BOLA DEMOLIDORA, utilizámos a função ATTR para determinar o que é que estava num determinado local do *écran* (se é que estava alguma coisa). Temos de fazer uma coisa análoga neste programa, de forma a determinar se um insecto ou uma flecha atingiu a última posição do pássaro mas, para variar, utilizaremos a função SCREEN$ em vez de ATTR.

SCREEN$ (x,y) fornece como resultado o carácter que se encontre na coluna x, linha y do *écran*, desde que este seja um dos caracteres que se situem entre SPACE e o símbolo de direitos autorais, ou de *copyright*, © (códigos 32 a 127 — ver Apêndice A do manual de programação BASIC do *Spectrum*). Se na célula em questão se encontrar qualquer outro padrão, a função dará como resultado uma cadeia nula (de comprimento zero) — muito embora esta função possa ser "persuadida" a reconhecer caracteres gráficos definidos pelo utilizador, tal como é explicado no Apêndice I deste livro. Note-se que, como a função SCREEN$ não examina a área de atributos da RAM, mas apenas a área de imagem (ficheiro de imagem), não lhe é possível distinguir a versão normal da versão em *inverse video* de um carácter.

```
  50 REM "PASSAROS"
 100 RANDOMIZE : PAPER 7: INK 0:
 BORDER 7: FLASH 0: BRIGHT 0: OV
ER 0: INVERSE 0
 110 DATA 0,0,24,126,189,36,0,0,
0,0,153,126,60,36,0,0
 120 FOR a=0 TO 15: READ b: POKE
 a+USR CHR$ 144,b: NEXT a
 130 LET s=50: LET x=15
 200 FOR f=144 TO 145: LET s=s-1
: PRINT AT 0,x; INK 1;CHR$ f
 210 IF INKEY$="5" AND x>0 THEN
LET x=x-1
```



```
 220 IF INKEY$="8" AND x<31 THEN
 LET x=x+1
 230 PRINT INK 2;AT 19,31*RND;"↑
";AT 19,31*RND;"↑"
 240 IF RND<.3 THEN PRINT INK 4;
AT 19,31*RND;"."
 250 POKE 23692,0: PRINT ',,,s:
IF s<=0 THEN GO TO 300
 260 LET p$=SCREEN$ (0,x): IF p$
="↑" THEN GO TO 300
 270 IF p$="." THEN LET s=s+20:
BEEP .05,30
 280 NEXT f
 290 GO TO 200
 300 FOR a=0 TO 19: BEEP .05,20-
a
 310 PRINT INK 1;AT a,x;"•";AT a
+1,x;CHR$ 145
 320 NEXT a
 330 BEEP .1,-40
```

100-130 Estabelecem as condições iniciais, definem dois caracteres gráficos definíveis pelo utilizador e os valores iniciais da pontuação (s) e da coordenada horizontal da posição do pássaro (x).

200-280 Ciclo destinado a fazer o pássaro bater as asas uma vez, sendo este efeito conseguido por alternância dos dois caracteres gráficos.

200 Imprime o pássaro na sua última posição calculada.

210-220 Calculam a posição seguinte a ocupar pelo pássaro se a tecla 5 ou a tecla 8 tiver sido pressionada.

230 Imprime duas novas setas algures na parte inferior do *écran*.

240 Imprime, de vez em quando, um novo insecto na parte inferior do écran.

250 Imprime o último número indicativo da reserva de alimento e faz com que a imagem role *(scroll)* uma linha para cima.

260-270 Verificam o que se encontra na última posição ocupada pelo pássaro — se lá se encontrar mais alguma coisa além deste — e tomam a acção apropriada.

290 Faz com que a sequência de execução volte à linha 200 para novo batimento de asas do pássaro.

300-330 O pássaro está morto, portanto "deixam-no" cair.

DESPISTE

Podemos fazer com que uma linha pareça mover-se, apagando repetidamente a imagem anterior e desenhando sempre uma nova linha numa posição ligeiramente diferente. É um processo bastante lento, mas, contudo, suficientemente rápido para um jogo simples como este.

O *écran* apresenta a vista que se obtém olhando sobre o *capot* de um carro de competição: uma estrada estendendo-se até ao horizonte.

Basta manter o carro na estrada, utilizando as teclas 5 e 8 para o guiar. Mas, além de serpentear de um lado para o outro, a estrada torna-se cada vez mais estreita!

```
  50 REM "DESPISTE"
 100 DATA 0,224,224,224,224,224,
224,225
 101 DATA 0,0,0,60,126,255,255,2
55
 102 DATA 0,7,7,7,7,7,7,255
 110 FOR a=0 TO 23: READ b: POKE
 a+USR CHR$ 144,b: NEXT a
 120 LET c$=CHR$ 144+CHR$ 145+CH
R$ 146
 130 BORDER 7: PAPER 4: INK 0: C
LS
 140 LET m=0: LET d=123: LET r=d
: LET d1=d: LET r1=r: LET s=0: L
ET w=40
 150 FOR a=1 TO 10: PRINT PAPER
1;: NEXT a
 200 LET m=m+(INKEY$="5")-(INKEY
$="8"): LET d=d+m+4*5IN s
 210 IF d<0 THEN LET d=0
 220 IF d>255 THEN LET d=255
 230 LET r=127+.5*(d-127): IF r<
25 THEN LET r=25
 240 IF r>225 THEN LET r=225
 250 PLOT INVERSE 1;r1-w,0: DRAW
 INVERSE 1;d1-r1+w,136: DRAW INV
ERSE 1;w+r1-d1,-136
 260 LET w=w-.1: PRINT AT 21,14;
c$
 270 PLOT r-w,0: DRAW d-r+w,136:
 DRAW w+r-d,-136
 280 LET d1=d: LET r1=r: LET s=s
+.1
 290 IF w-ABS (123.5-r)>12 THEN
GO TO 200
```



```
 300 PRINT FLASH 1; INK 3;AT 2,3
;" DESPISTOU-SE "
 310 PRINT AT 5,2;"decorridos ";
s;" quilometros"
```

100-150 Definem as condições iniciais, incluindo a cadeia de três caracteres c$ que representa a imagem da parte da frente do carro. Repare-se na linha 150, a qual "enche" a parte de cima do *écran* com "céu" azul.

200-290 Ciclo principal do programa.

200 Acerta a variável se a tecla "5" ou a tecla "8" for pressionada. Repare-se que a primeira declaração desta linha é equivalente às duas linhas seguintes:
IF INKEY$="5" THEN LET m=m+1
IF INKEY$="8" THEN LET m=m−1

210-220 A variável d define a posição do ponto no horizonte onde a estrada desaparece. Estas linhas evitam que lhe seja atribuído um valor que o situe fora dos limites do *écran*.

230-240 Calculam o valor da variável r, que representa a posição do extremo mais próximo da estrada, e evitam que ele se aproxime muito dos limites do *écran*.

250 Apaga a imagem anterior da estrada.

260-270 Tornam a estrada um pouco mais estreita e, seguidamente, apresentam a imagem do carro e da estrada na sua nova posição no *écran*.

280-290 Incrementam a distância percorrida (s) e fazem a sequência de execução voltar à linha 200 se o carro ainda não se tiver despistado.

300-310 Rotina de finalização.

Capítulo 12

GO SUB, DEF FN e grande estilo

Encontramos muitas vezes a necessidade de incluir um mesmo (ou semelhante) conjunto de instruções em diversos sítios de um programa. A linguagem BASIC permite escrever esse conjunto de instruções apenas uma vez — na forma de uma sub-rotina —, que "chamaremos" tantas vezes quantas as necessárias.

Uma sub-rotina de BASIC assemelha-se a qualquer outra parte de um programa escrito em BASIC, com a excepção de terminar com uma declaração RETURN. É muitas vezes conveniente considerar que um programa com sub-rotinas é constituído por uma rotina "principal" suportada pelas sub-rotinas.

Para utilizar (ou chamar) uma sub-rotina, necessitamos de incluir uma declarção

```
GO SUB n
```

no corpo principal do programa, sendo n o número da primeira linha da sub-rotina. GO SUB é semelhante a GO TO, excepto no facto de fazer com que o *Spectrum* "recorde" o local do programa em que situa a declaração GO SUB que acabou de ser executada. Assim, quando chega à declaração RETURN no fim da sub-rotina, o *Spectrum* volta à declaração que se situa imediatamente a seguir a GO SUB. A rotina seguinte constitui um exemplo simples:

```
10 PRINT 10
20 GO SUB 100
30 PRINT 30
40 STOP
100 PRINT 100
110 RETURN
```


e imprime:

```
10
100
30
```

As linhas 10 a 40 constituem neste caso o programa principal, e as linhas 100 e 110 a sub-rotina. A declaração STOP (linha 40) é essencial, pois, se não fosse incluída, o *Spectrum* voltaria a executar as linhas 100 e 110 quando tivesse acabado a execução da linha 30. Quando chegasse à linha 110 pararia, emitindo a mensagem de erro

```
RETURN without GO SUB
```

e isto porque desta vez o *Spectrum* não saberia a que linha voltar. Uma sub-rotina deve ser executada apenas por meio de uma declaração GO SUB.

Para verificar que uma sub-rotina pode ser chamada tantas vezes quantas se quiser e que um GO SUB pode ser inserido em qualquer parte de uma linha que inclua várias declarações, experimente-se introduzir a rotina:

```
10 PRINT 1: GO SUB 100: PRINT 2: GO SUB 100
20 GO SUB 100: PRINT 3
30 STOP
100 PRINT "sub-r": RETURN
```

a qual imprimirá:

```
1
sub-r
2
sub-r
sub-r
3
```

Note-se que, apesar de ser usual colocar as sub-rotinas no fim de um programa, elas podem ser inseridas em qualquer posição, desde que nos certifiquemos de que elas não possam ser

executadas a não ser por meio de uma declaração GO SUB.

Uma sub-rotina pode chamar outra, e assim

```
1Ø PRINT 1: GO SUB 1ØØ
2Ø PRINT 2: GO SUB 2ØØ
3Ø STOP
1ØØ PRINT 1ØØ: GO SUB 2ØØ: RETURN
2ØØ PRINT 2ØØ: RETURN
```

imprime:

```
1
1ØØ
2ØØ
2
2ØØ
```

Como se mostra no Capítulo 24 do manual do *Spectrum*, parte da RAM está reservada para aquilo que se chama "GOSUB *stack*" ou "pilha GOSUB". É esta a área utilizada para guardar os números de linha e de declaração da declaração GO SUB, de modo a que o *Spectrum* "saiba" para onde voltar quando encontrar um RETURN. Esta informação é armazenada como uma lista: sempre que uma declaração GO SUB é executada, o *Spectrum* adiciona os números de linha e de declaração do GO SUB ao fim da lista.

Seguidamente, ao encontrar a declaração RETURN, o *Spectrum* retira o par de números mais recente do fim da lista. Desta forma, o programa volta sempre à declaração imediatamente a seguir ao GO SUB que tenha sido executado mais recentemente.

Por esta razão, nunca se deve sair de uma sub-rotina por meio de uma declaração GO TO; embora possa por vezes parecer resultar, é um processo arriscado, pois deixa informação indesejada armazenada na área da pilha GO SUB.

Vimos como uma sub-rotina pode chamar outra, e não existe limite para o grau de encaixe (*nesting*), excepto quanto à disponibilidade de área suficiente na RAM para conter todos os endereços de retorno. O BASIC do *Spectrum* também permite que uma sub-rotina se chame a si própria (este processo denomina-se "recursão"), desde que nos certifiquemos que o processo

acabará por parar e não entrará num ciclo sem fim!

Por exemplo, o programa seguinte calcula o valor do factorial de n (n!)

```
(n!=n*(n-1)*(n-2) - - *1)

1Ø INPUT "n ? ";n
2Ø LET x=n: LET y=1: GO SUB 1ØØ
3Ø PRINT "Factorial ";n;" = ";y
4Ø STOP
1ØØ IF x>Ø THEN LET y=y*x: LET x=x-1: GO
SUB 1ØØ
11Ø RETURN
```

DEF FN

Verificamos frequentemente que uma mesma — ou semelhante — expressão ou cálculo aparece várias vezes num programa, e seria bastante cómodo se apenas tivéssemos de escrevê-la uma única vez.

Poderíamos — evidentemente — fazer uso de uma sub-rotina. Contudo, as sub-rotinas em BASIC têm a desvantagem de apenas funcionarem com os nomes das variáveis incluídos na própria sub-rotina. Por exemplo, poderíamos escrever uma sub-rotina destinada a calcular o valor médio de dois números:

```
1ØØ REM SUB-ROTINA PARA A MEDIA DE X E Y
11Ø LET media=(x+y)/2
12Ø RETURN
```

mas isto apenas daria a média dos valores das variáveis x e y. Se no nosso programa quisermos calcular a média das variáveis p e q e também de c e d utilizando a sub-rotina teríamos de redefinir os valores das variáveis em cada utilização:

```
2Ø LET x=p: LET y=q: GO SUB 1ØØ
55 LET x=c: LET y=d: GO SUB 1ØØ
```

As funções pré-programadas do *Spectrum* permitem, contudo, especificar qual a variável a tomar, e assim podemos escrever SQR (a) onde pretendermos a raiz quadrada de a, e SQR (z) onde pertendermos a raiz quadrada de z. Ao definirmos a nossa própria função, podemos especificar de forma semelhante qual ou quais variáveis devem ser usadas.

Para definir uma função, devemos incluir na listagem do programa uma declaração com o formato

```
DEF FN a (argumentos) = expressão
```

onde a é o nome que pretendemos dar à nova função; deve, contudo, ser constituído por uma única letra (ou por uma letra seguida de um $ se se pretende que a função dê uma cadeia como resultado). Como é habitual no caso de nomes, o *Spectrum* não faz distinção entre letras minúsculas e letras maiúsculas.

Os "argumentos" entre parênteses são nomes fictícios atribuídos aos valores sobre os quais se vai actuar. Mas, também neste caso, eles devem ser constituídos por uma única letra ou por uma letra seguida de um $.

Assim, podemos definir uma função (à qual podemos decidir chamar FN a) destinada a calcular a média de dois números pela declaração

```
DEF FN a(x,y)=(x+y)/2
```

e depois utilizá-la, recorrendo à "expressão" FN a(e,f), onde estas duas últimas letras representam os dois números dos quais pretendemos obter a média.

O *Spectrum*, quando se lhe depara a função FN a(e,f), inicia uma pesquisa pela listagem do programa, começando na primeira linha, até encontrar uma declaração DEF FN a correspondente à função. Quando isto acontece, executa a expressão à direita do sinal de igualdade, com as variáveis "fictícias" (x e y neste caso) substituídas pelos valores reais (e e f neste caso).

Para ver como isto funciona na prática, vamos fazer o *Spectrum* executar o seguinte:

```
10 DEF FN m(x,y,z)=(x+y+z)/3
20 PRINT FN m(1,2,3)
```



```
30 LET p=5: LET q=6
40 PRINT FN m(p,7,q)
```

o que irá imprimir:

```
2
6
```

Note-se que se existir mais do que um argumento entre os parênteses, esses argumentos devem ser separados por vírgulas. Se não existirem argumentos, os parênteses continuam a ser necessários, como no exemplo:

```
10 DEF FN r()=PEEK 23730+256* PEEK 23731
20 PRINT FN r()
```

o qual imprime o valor da variável de sistema RAMTOP.

A expressão que se situa imediatamente a seguir ao sinal de igualdade numa declaração DEF FN pode ser qualquer expressão válida em BASIC, e incluir quaisquer outras funções definíveis pelo utilizador. Pode igualmente incluir variáveis que não sejam os argumentos fictícios situados entre os parênteses. Por exemplo:

```
10 DEF FN a(x)=x+y
20 LET y=1
30 PRINT FN a(2)
```

imprimirá 3.

Se uma variável com o mesmo nome de uma das variáveis fictícias for utilizada em qualquer outra parte do programa, será ignorada quando a função é calculada, não tendo por isso qualquer influência no resultado. Por exemplo, se a linha 20 da rotina apresentada acima fosse:

```
20 LET x=99: LET y=1
```

a rotina ainda imprimiria 3.

GRANDE ESTILO

Apesar de o BASIC ser uma boa linguagem para o principiante e para escrever programas curtos, apresenta algumas desvantagens quando se está a elaborar um programa grande.

O primeiro problema, não muito importante, diz respeito à velocidade com que o programa é executado. Devido ao facto de o BASIC do *Spectrum* ser uma linguagem "interpretável" — onde o computador traduz ou interpreta cada passo à medida que avança na execução do programa — não é tão rápido como a linguagem máquina ou qualquer uma das linguagens compiláveis, tal como o FORTRAN, o FORTH ou o Pascal. (Numa linguagem compilável, todo o programa é traduzido para linguagem máquina — ou qualquer coisa muito próximo dela — antes de ser executado.)

Além disso, sempre que se modifica o "rumo" da execução do programa devido a uma declaração GO TO, GO SUB ou NEXT, o *Spectrum* tem de pesquisar o programa linha a linha, começando pela primeira, até encontrar aquela para onde deve dirigir o rumo da execução. Isto significa que os ciclos que ocorram perto do fim de uma longa listagem de programa levarão mais tempo a ser executados do que se tivessem sido colocados perto do início. Ocorre um atraso semelhante quando se utiliza uma função definivel pelo utilizador, pois o *Spectrum* tem, nesse caso, de procurar a declaração DEF FN correspondente.

Contudo, o problema principal que se apresenta ao elaborar programas grandes em BASIC não se situa no computador, mas sim relativamente ao programador que os elabora. Quanto maior for um programa, mais difícil se torna verificar se está todo correcto, se não existem erros.

Considere-se, por exemplo, a estrutura de um programa, ou seja, a forma como ele avança de uma acção para a seguinte. Se pensarmos que cada declaração IF - THEN GO TO num programa dá origem a duas "vias" alternativas, chegaremos à conclusão de que um programa que contenha apenas dez declarações IF - THEN GO TO pode ter mais de mil "vias" possíveis. Como ter a certeza de que todas elas estão correctas? Dissemos no capítulo "Ciclos e decisões" que desenhar linhas para indicar onde o curso de um programa se desvia da sequência de execução normal ajudará a compreender como o programa funciona. Se o resultado se assemelhar a um prato de *spaghetti* é provável que

o nosso raciocínio esteja tão emaranhado como o *spaghetti*. Evite-se, pois, o emprego de GO TO sempre que se puder.

Para mais fácil compreensão, dividamos um programa longo em blocos ou módulos bem definidos, idealmente apenas com um único ponto de entrada e um único ponto de saída para cada bloco. As linguagens mais aperfeiçoadas, tal como o *Pascal*, forçam-nos a fazê-lo, mas em BASIC temos de ser nós a procurar fazê-lo. Podemos — por exemplo — empregar um bloco separado de números de linha para cada secção do programa; os programas deste livro estão escritos desse modo.

Outra técnica consiste em escrever um programa "principal" muito curto, o qual "chama" várias sub-rotinas para executarem as principais funções do programa. Contudo, estamos aqui a utilizar o mecanismo de sub-rotina como um auxiliar para uma melhor compreensão, e não para nos permitir chamar a mesma sub-rotina várias vezes. Por exemplo, a secção principal de um programa de jogos poderia apresentar-se assim:

```
100 LET pontuacao=0
110 GO SUB 1000: REM elabora o tabuleiro
120 GO SUB 2000: REM executa o jogo
130 GO SUB 3000: REM apresenta as pontuacoes
140 INPUT "Outro jogo (s/n) ? ";i$
150 IF i$="s" THEN GO TO 110
160 IF i$<>"n" THEN GO TO 140
170 STOP
```

As sub-rotinas 1000, 2000 e 3000, por sua vez, poderiam chamar outras sub-rotinas, de modo a fragmentar ainda mais o programa em segmentos de dimensões mais "manejáveis".

A listagem de um programa tornar-se-á de compreensão mais fácil se:

— utilizarmos declarações REM para rotular as principais secções. Linhas que consistam apenas no número de linha e na palavra de comando REM também se utilizam para fragmentar a massa compacta da listagem de um programa longo;
— fizermos as linhas corresponderem a acções completas, se possível, e assim

```
100 FOR a=1 TO 10: LET y(a)=a: NEXT a
110 FOR a=16 TO 24: LET y(a)=33: NEXT a
```

é mais fácil de interpretar do que, por exemplo:

```
100 FOR a=1 TO 10: LET y(a)=a: NEXT a: FOR a=16
TO 24
110 LET y(a)=33: NEXT a
```

— fizermos com que FOR e NEXT sejam, cada uma delas, a primeira declaração de uma linha, excepto no caso em que um ciclo completo esteja contido numa mesma linha;
— atribuirmos às variáveis nomes significativos (abreviaturas, etc.).

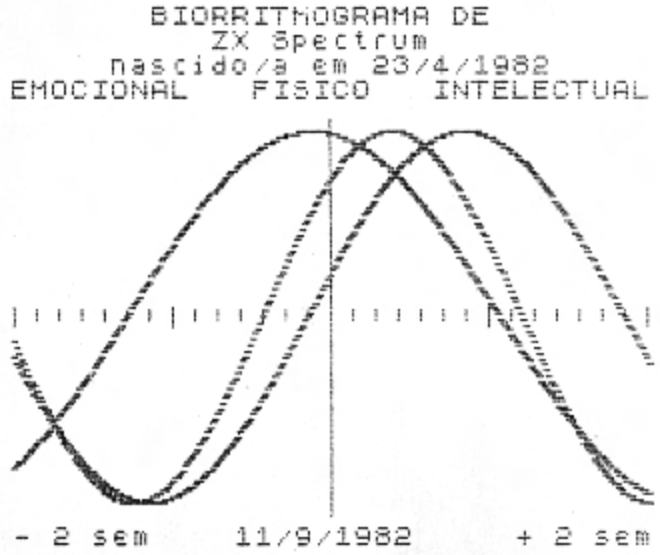
Um outro problema que ocorre devido à impossibilidade de se estar ao mesmo tempo totalmente atento a todos os pormenores de um programa grande, consiste na duplicação involuntária e indesejada de nomes de variáveis. Também, neste caso, as linguagens como o *Pascal* são superiores ao BASIC, pois permitem utilizar variáveis "locais", apenas válidas numa parte do programa — de forma muito semelhante ao que acontece com as variáveis "fictícias" de uma declaração DEF FN, as quais apenas são válidas para essa declaração. No BASIC do *Spectrum*, todas as outras variáveis são "globais" e, portanto, apenas podemos elaborar um memorando pormenorizado do uso dado a cada nome de variável.

O resto deste capítulo contém uma série de programas razoavelmente grandes, destinados não só ao interesse e recreação do leitor, mas também a exemplificar os assuntos tratados anteriormente.

BIORRITMOS

Este programa exemplifica a utilização de sub-rotinas quando uma operação semelhante é necessária em mais do que um lugar num programa.

O programa baseia-se na teoria de que os nossos estados físico, emocional e intelectual seguem ciclos regulares de 23, 28



e 33 dias, respectivamente, a partir da data do nascimento. Os "altos e baixos" destas três curvas são calculados e traçados por este programa para um período de quatro semanas, centrado em qualquer data que o utilizador especifique. Diz-se que os dias em que uma curva intersecta a linha média (zero) são "críticos". Parece que, pelo estudo cuidadoso dos gráficos, se encontram razões que justificam mais ou menos o que nos sucede a nível de saúde, e talvez seja esta a melhor razão para a utilização desta técnica de "previsão".

Se utilizarmos a cor na impressão ou nos gráficos traçados pelo *Spectrum*, como acontece neste programa, verificaremos que é necessário passar algum tempo a experimentar várias combinações de cores até obter a imagem mais atraente. Por exemplo, geralmente é preferível empregar um fundo negro e, na maior parte dos casos, as cores púrpura *(magenta)*, verde *(green)*, azul-celeste *(cyan)* e amarela *(yellow)* dão melhores resultados do que o azul *(blue)* ou vermelho *(red)*. Por outro lado, linhas diagonais delgadas (digamos, com um *pixel* de largura) causam efeitos curiosos, nem sempre desejados; neste programa, as curvas são traçadas com 3 *pixels* de largura para evitar tal problema.

À medida que executarmos o programa, verificaremos que, sempre que uma linha desenhada se aproxima de uma outra linha já desenhada e de cor diferente, parte desta outra linha (onde se dá a intersecção das duas linhas) altera a cor de modo a ficar da mesma cor da linha que a cruza. Isto deve-se à limitação do *Spectrum* de apenas poder apresentar duas cores ("fundo" e "tinta") em qualquer célula de carácter.

O programa utiliza duas sub-rotinas, com início nas linhas 1000 e 2000, para executarem funções que são repetidas com ligeiras variações.

```
  50 REM "BIORRITMOS"
 100 PAPER 7: INK 0: BORDER 7: I
NVERSE 0: OVER 0: FLASH 0: CLS
 110 PRINT AT 5,10; INK 3;"BIORR
ITMOS": INPUT "Qual e'o seu nome
 ? "; LINE n$
 120 PRINT AT 8,0;"Ola', ";n$;AT
 10,0;"Qual e'a sua data de nasc
imento?"
 130 GO SUB 1000: PRINT AT 12,19
;d$: LET b$=d$: LET z=x
 140 PRINT AT 14,0;"Data em caus
a ?": GO SUB 1000
 150 PRINT AT 14,17;"••";d$: LET
 d=x-z
 160 INK 3: PRINT AT 16,0;"Nessa
 data"
 170 PRINT "tera'";d;" dias de i
dade,"
 180 PRINT "tera'comido ";3*d;"
refeicoes"
 190 PRINT "e tera'dormido cerca
de";8*d;" horas."
 200 INPUT "Carregue em ENTER pa
ra obter";"o seu Biorritmograma
"; LINE i$
 210 PAPER 0: INK 6: BORDER 0: C
LS
 220 PRINT TAB 7;"BIORRITMOGRAMA
DE"
 230 PRINT TAB 15-LEN n$/2;n$'TA
B 9-LEN b$/2;"nascido/a em ";b$
 240 FOR a=1 TO 255 STEP 9: PLOT
 a,73: DRAW 0,3: NEXT a
 250 FOR a=1 TO 255 STEP 63: PLO
T a,71: DRAW 0,6: NEXT a
 260 PLOT 127,10: DRAW 0,128
 270 PRINT AT 21,0;"- 2 sem";TAB
 25;"+ 2 sem"
 280 PRINT AT 21,15-LEN d$/2;d$
 300 INK 3: PRINT AT 3,0;"EMOCIO
NAL": LET c=28: GO SUB 2000
 310 INK 4: PRINT AT 3,12;"FISIC
O": LET c=23: GO SUB 2000
 320 INK 5: PRINT AT 3,21;"INTEL
ECTUAL": LET c=33: GO SUB 2000
 330 PAPER 7: INK 0: BORDER 7
 340 STOP
1000 DATA 0,31,28,31,30,31,30,31
,31,30,31,30,31
1010 INPUT "Ano ? ";y: LET x=365
*y+INT (y/4)-INT (y/100)
1020 INPUT "Mes (1-12) ? ";m: IF
 m<1 OR m>12 THEN GO TO 1020
1030 RESTORE : FOR a=1 TO m: REA
D b: LET x=x+b: NEXT a
1040 LET l=y=4*INT (y/4) AND y<>
100*INT (y/100)
1050 IF l AND m>2 THEN LET x=x+1
1060 READ b: IF l AND m=2 THEN L
ET b=29
1070 INPUT ("Dia (1-";b;") ? ");
d: IF d<1 OR d>b THEN GO TO 1070
1080 LET x=x+d: LET d$=STR$ d+"/
"+STR$ m+"/"+STR$ y
1090 RETURN
2000 FOR a=0 TO 253: LET b=d-14+
a/9
2010 PLOT a,74+60*SIN (2*PI*b/c)
: DRAW 2,0
2020 NEXT a
2030 RETURN
```

100-150 Obtêm a data de nascimento e a data em causa pretendida, utilizando a sub-rotina que tem início na linha 1000. Calculam o número de dias (d) que separam as duas datas e elaboram duas cadeias (b$ e d$) para as conter.

160-190 Imprimem alguns factos de interesse.

200 Aguarda que o utilizador faça a "digestão" dos factos.

210-280 Desenham marcas de calibragem (escalas, etc.) para o gráfico e imprimem os cabeçalhos. Repare-se na técnica usada nas linhas 230 e 280 para imprimir uma cadeia centrada sobre uma linha vertical, sendo a cadeia de comprimento variável.

300-320 Traçam os três gráficos, utilizando a sub-rotina 2000, em três cores diferentes.

330-340 Fazem as cores do fundo, da tinta e da margem voltarem aos seus valores iniciais, como forma de cortesia para o utilizador seguinte. Em seguida fazem o programa parar.

1000 É uma sub-rotina bastante útil, que pede que o utilizador lhe forneça uma data sob a forma de ano, mês e dia, verifica a validade do dia e do mês (ou seja, verifica a coerência dos valores dados pelo utilizador para o dia e para o mês). Seguidamente, forma uma cadeia normalizada (d$) contendo a data sob a forma dd/mm/yyyy (onde y é um dos algarismos que compõem o número do ano — do inglês *year*) e também um "número de dias" (x), o qual serve para calcular o número de dias decorridos entre duas datas, tal como é necessário neste programa, ou — por exemplo — num sistema de facturação com a função de procurar facturas com mais de 30 dias.
A linha 1040 mostra-se interessante: atribui à variável l o valor "verdadeiro" se se tratar de um ano bissexto; caso contrário, atribui-lhe o valor "falso". Isto é depois utilizado pelas linhas 1050 e 1060 para acertar o número de dias, de acordo com o valor de l.

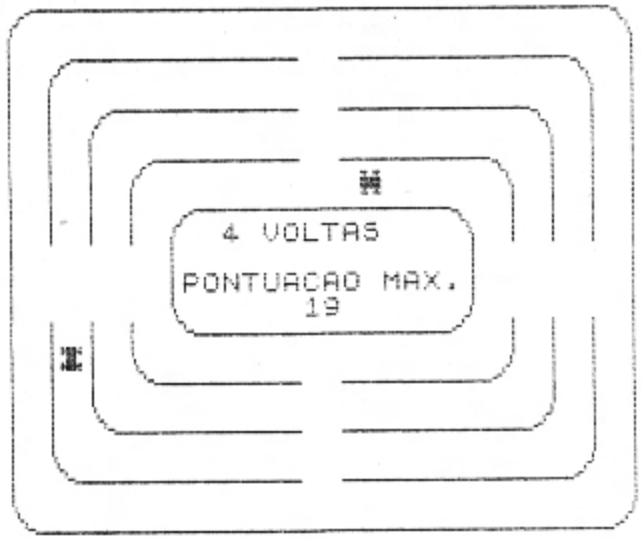
2000 Sub-rotina destinada a calcular as curvas.

COLISÃO

O jogador e o Rei da Velocidade do *Spectrum* estão ambos a conduzir num circuito de corridas com quatro faixas de rodagem. Mas não se trata de uma corrida! Os carros circulam em sentidos opostos e o adversário tenta sistematicamente chocar de frente com o jogador! Quantas voltas ao circuito conseguirá este aguentar antes do fim inevitável? Isto não faz lembrar a Estrada Marginal à hora de ponta?

Ao começar o jogo, o programa pede a indicação de um valor de "Perícia", o qual é um número entre 1 e 9. Quanto maior for esse número mais suicida se tornará o adversário.

Quando o carro do jogador estiver em movimento, a única possibilidade de que este dispõe é mudar de faixa de rodagem nas aberturas das linhas divisórias. Carrega-se na tecla "I" para passar para uma faixa interior (em relação àquela em que se encontra o carro) e na tecla "O" para passar para uma exterior.




```
  50 REM "COLISAO"
 100 GO SUB 9000: REM Preparacao
 para a primeira corrida
 200 IF yy=21-yt THEN LET nyx=yx
+1: IF yx=30-yt THEN LET nyy=yy-
1: LET y$=CHR$ 145
 210 IF yy=yt THEN LET nyx=nyx-1
: IF yx=yt+1 THEN LET nyy=yy+1:
LET y$=CHR$ 145
 220 IF yx=yt THEN LET nyy=yy+1:
 IF yy=20-yt THEN LET nyx=yx+1:
LET y$=CHR$ 144
 230 IF yx=31-yt THEN LET nyy=yy
-1: IF yy=yt+1 THEN LET nyx=yx-1
: LET y$=CHR$ 144
 240 IF nyx=15+(yy>12) OR nyy=11
 THEN LET yf=2*((yt<7 AND INKEY$
="i")-(yt>1 AND INKEY$="o"))
 250 IF yf<>0 THEN LET nyx=nyx+(
(nyx=yt)-(nyx=31-yt))*SGN yf: LE
T nyy=nyy+((nyy=yt)-(nyy=21-yt))
```

```
*SGN yf: LET yt=yt+SGN yf: LET y
f=yf-SGN yf
 250 IF nyy=cy AND nyx=cx THEN G
O TO 400
 300 IF cy=21-ct THEN LET ncx=cx
-1: IF cx=ct+1 THEN LET ncy=ncy-
1: LET c$=CHR$ 145
 310 IF cy=ct THEN LET ncx=cx+1:
 IF cx=30-ct THEN LET ncy=cy+1:
LET c$=CHR$ 145
 320 IF cx=ct THEN LET ncy=cy-1:
 IF cy=ct+1 THEN LET ncx=cx+1: L
ET c$=CHR$ 144
 330 IF cx=31-ct THEN LET ncy=cy
+1: IF cy=20-ct THEN LET ncx=cx-
1: LET c$=CHR$ 144
 340 IF (ncx=15+(ncy>8)) OR ncy=
11 THEN IF RND<s/10 THEN LET cf=
2*SGN (yt-ct)
 350 IF cf<>0 THEN LET ncx=ncx+(
(ncx=ct)-(ncx=31-ct))*SGN cf: LE
T ncy=ncy+((ncy=ct)-(ncy=21-ct))
*SGN cf: LET ct=ct+SGN cf: LET c
f=cf-SGN cf
 400 PRINT AT yy,yx;".";AT cy,cx
;"•";AT nyy,nyx; INK 1;y$;AT ncy
,ncx; INK 2;c$
 410 LET yx=nyx: LET yy=nyy: LET
 cx=ncx: LET cy=ncy
 420 IF yx=16 AND yy<8 THEN LET
vol=vol+1: PRINT AT 9,11;vol;" V
OLTA";("S" AND vol>1)
 430 IF yx<>cx OR yy<>cy THEN GO
 TO 200
 500 FOR a=1 TO 6: FOR b=144 TO
145: BEEP .03,-40: PRINT INK a;A
T yy,yx;CHR$ b: NEXT b: NEXT a
 510 PRINT AT yy,yx;CHR$ 146
 520 IF vol>po THEN LET po=vol:
PRINT AT 11,9;"PONTUACAO MAX.";A
T 12,15;po
 530 INPUT "Pressione ENTER p/ n
ova corrida"; LINE i$
 540 PRINT AT yy,yx;"•": GO SUB
9200: GO TO 200
9000 REM Desenho da pista
9010 INK 0: PAPER 7: FLASH 0: BR
IGHT 0: OVER 0: INVERSE 0: BORDE
R 7: CLS
9020 FOR a=32 TO 160 STEP 32: PL
OT a/2,a/2-13
9030 DRAW 256-a,0: DRAW 12,12,PI
/2
```



```
9040 DRAW 0,176-a: DRAW -12,12,P
I/2
9050 DRAW a-256,0: DRAW -12,-12,
PI/2
9060 DRAW 0,a-176: DRAW 12,-12,P
I/2
9070 NEXT a
9080 FOR a=2 TO 6: PRINT AT a,15
;"••";AT a+13,15;"••": NEXT a
9090 FOR a=10 TO 12: PRINT AT a,
2;"•••••";AT a,25;"•••••": NEXT
a
9100 REM Carros
9110 DATA 231,66,255,255,255,255
,66,231
9120 DATA 189,255,189,60,60,189,
255,189
9130 DATA 36,90,189,126,126,189,
90,36
9140 RESTORE 9100
9150 FOR a=0 TO 23: READ b: POKE
 USR "a"+a,b: NEXT a
9160 LET po=0
9200 REM Valores iniciais
9210 LET yx=15: LET yy=1: LET yt
=1: LET nyx=yx: LET nyy=yy: LET
y$=CHR$ 144: LET yf=0
9220 LET cx=16: LET cy=7: LET ct
=7: LET ncx=cx: LET ncy=cy: LET
c$=CHR$ 144: LET cf=0
9230 LET vol=0: PRINT AT 9,10;"•
••••••••••"
9240 INPUT "Pericia (1-9) ? ";s
9999 RETURN
```

100 Esta linha chama a sub-rotina para desenhar o circuito, definir os caracteres definíveis pelo utilizador e os valores iniciais de algumas variáveis.

200-430 Ciclo principal do programa, destinado a fazer ambos os carros avançar um pouco.

200-230 Calculam a próxima posição do carro do jogador.

240 Examina as teclas "I" e "O" quando o carro se encontra em pontos apropriados do circuito, verificando se alguma delas foi pressionada e acertando a *flag* (indicador) de movimento "yf" de acordo com o resultado da verificação.

250 Esta linha desloca o carro do jogador meia distância entre duas faixas de rodagem se yf<>zero.

260 Termina o jogo se o carro colidiu com o do *Spectrum*.
300-330 Calculam a próxima posição do carro do *Spectrum*.
340 Fixa um valor para o indicador de movimento "cf" se o *Spectrum* decide mudar de faixa.
350 Desloca o carro do *Spectrum* meia distância entre uma faixa e outra se cf<>zero.
400-410 Apagam as imagens anteriores de ambos os carros e actualizam coordenadas.
420 Actualiza o contador de voltas, se for caso disso.
430 Faz a sequência de execução do programa voltar à linha 200 se o carro do *Spectrum* ainda não chocou com o outro.
500-540 Rotina de finalização: actualiza a pontuação máxima se a pontuação do jogador tiver melhorado neste último jogo e, seguidamente, convida a "vítima" para outra corrida.
9000 Rotina principal de preparação para a corrida.
9200 Ponto de chamada da sub-rotina para preparar as posições iniciais para outra corrida.

yx,yy Coordenadas x e y do carro do jogador.
yt Representa o número da faixa de rodagem em que vai o jogador; a faixa 1 é a faixa exterior e as quatro faixas são numeradas 1, 3, 5 e 7.
cx,cy,ct São as coordenadas e o número da faixa do carro do *Spectrum*.
nyx,nyy Valores de xx e yy seguintes *(next)*.
ncx,ncy Valores de cx e cy seguintes.
yf,cf Indicadores aos quais são atribuídos os valores 2 ou −2 quando um carro está prestes a mudar de faixa.

3 − D

Este programa lida com desenhos "estruturais" a 3 dimensões, permitindo ao utilizador vê-los de qualquer ângulo, alterar as suas dimensões e até incluir perspectiva.

Cada desenho é constituído a partir de segmentos de rectas. As coordenadas x, y e z de cada extremo das 400 linhas permitidas (como máximo) são agrupadas num quadro que se pode gravar (e ler) separadamente em *cassette*. Isto permite ter outro programa para gerar ou utilizar estes dados.

O programa em si está dividido num certo número de "módulos", os quais são seleccionados de uma secção de "menu principal". O programa também faz uso de sub-rotinas e de funções definíveis pelo utilizador para executar tarefas usuais.

Quando o programa é executado, a primeira coisa que ele apresenta é o menu principal:

Carregue na tecla:

A — Para apresentar as coordenadas no *écran (display data)*
I — Para introduzir coordenadas
L — Para introduzir *(load)* coordenadas gravadas em *cassette*
P — Para imprimir as coordenadas
T — Para terminar
G — Para gravar as coordenadas em *cassette*
V — Para ver o desenho

A opção "Apresentar as coordenadas no *écran*" pede a indicação da linha a partir da qual se deseja se apresentem as coordenadas; as linhas usadas para elaborar o desenho são numeradas de 1 até 400 (máximo com 16 K). Dada a indicação, o computador apresentará as coordenadas x, y e z de cada extremo de 17 linhas consecutivas, e perguntará se se deseja ver mais algumas ou voltar ao menu principal.

A opção "Introduzir coordenadas" permite acrescentar novos dados (coordenadas) ou alterar dados existentes. Para uma linha do desenho deve introduzir-se:

— o número da linha (número inteiro de 1 a 400);
— as coordenadas x, y e z de um dos extremos da linha;
— as coordenadas x, y e z do outro extremo da linha.

Cada um dos valores das coordenadas deve ser um número inteiro compreendido entre −100 e 100.

Os sete números são introduzidos como uma única linha, devendo esses valores ser separados por espaços (em branco). Se algum dos valores não é válido (incoerente ou fora dos parâmetros admitidos), o programa ignora a linha toda e espera que o utilizador introduza outra versão. Se, pelo contrário, todos os valores forem aceitáveis, serão apresentados no *écran* e, após cerca de um segundo de espera, o cursor reaparecerá na parte de baixo do *écran*, pronto para o utilizador introduzir outra linha.

Pode-se continuar a introduzir linhas deste modo, ou voltar ao menu principal introduzindo simplesmente uma linha "vazia" (carregando apenas em ENTER). Não é necessário introduzir as linhas por uma ordem determinada e quaisquer valores previamente introduzidos podem ser alterados introduzindo uma nova linha com o mesmo número.

A opção "Introduzir coordenadas gravadas em *cassette*" permite introduzir *(load)* um ficheiro de dados (coordenadas e números de linha) previamente gravado em *cassette*, e apaga quaisquer dados já existentes na memória do computador. Uma vez carregado em memória o ficheiro de dados, pode-se verificá-lo; se a verificação falhar, o programa parará; para que ele arranque de novo, introduz-se GO TO 1000.

A opção "Imprimir as coordenadas" fornece, por meio da impressora *ZX*, uma cópia permanente das coordenadas contidas num determinado bloco de linhas previamente seleccionado. Para isso, o programa pede a indicação dos números da primeira e última linha do bloco que se deseja ver impresso. Devem introduzir-se os dois números pedidos numa única linha, separados por um espaço em branco.

A opção "Terminar" pára o programa. Se se quiser fazê-lo "arrancar" de novo, introduz-se GO TO 1000 para preservar os dados ou RUN para os apagar.

A opção "Gravar as coordenadas em *cassette*" permite gravar as coordenadas sob a forma de um ficheiro de dados identificado por um nome. Como em todos os ficheiros do *Spectrum*, o nome

terá de ser pelo menos um e o seu comprimento não deve exceder 10 caracteres. Gravadas as coordenadas, o programa insistirá para que se volte com a fita a trás e se passe a gravação, para verificar se ela se efectuou em condições. Se a verificação falhar, faz-se o programa arrancar novamente por meio de um GO TO 1000, pois de contrário perdem-se todos os dados contidos em memória.

Note-se que, para gravar o próprio programa em vez das coordenadas, deve-se primeiramente fazê-lo parar (utilizando a opção T ou carregando em CAPS/SHIFT e BREAK, se estiver a efectuar-se algum cálculo, ou, então, pressionando CAPS/SHIFT e "6" se o programa estiver a aguardar um *input*. Seguidamente, introduz-se CLEAR antes de gravar o programa, pois isto apaga os quadros de coordenadas e outras variáveis, reduzindo assim consideravelmente o tempo gasto a gravar, e, subsequentemente, a recarregar *(load)* o programa em memória.

Por fim, a opção V permite "ver o desenho" e, se a seleccionarmos, pedir-nos-á para introduzir:

— o primeiro e o último números de linha do bloco de linhas que desejamos que o computador desenhe;
— um factor de "escala", o qual deverá ser um número inteiro compreendido entre 1 e 999 e que determina as dimensões do desenho. É conveniente começar com um factor de escala igual a 50;
— a amplitude da rotação horizontal, expressa em graus. Deverá ser um número inteiro de 0 a 360;
— a amplitude da rotação vertical, também expressa em graus e sob a forma de um número inteiro entre 0 e 360;
— um valor de "profundidade", que deverá ser um número inteiro de 0 a 9. Este valor determina a proporção de "perspectiva" utilizada: 0 não dá nenhuma, e 3 ou 4 dão geralmente o melhor efeito.

Estes valores devem ser todos introduzidos numa única linha, separados por espaços em branco. Se algum dos valores não for válido, o programa ignora a linha toda e aguarda que o utilizador a introduza de novo.

Introduzidos estes valores, o programa traça um desenho baseado nos dados referentes ao bloco de linhas especificado. Repare-se que isto permite armazenar dados para mais de um

desenho num único ficheiro ou desenhar apenas uma parte de um trabalho mais vasto.

Se o desenho resultante não couber no *écran*, o programa parará; deverá então utilizar-se um GO TO 1000 para o fazer arrancar de novo.

Depois de traçado o desenho, o programa perguntará se desejamos uma cópia permanente feita pela impressora *ZX*.

```
  50 REM "3-D"
 100 INPUT "Numero maximo de lin
has (<=400)?";ml
 110 DIM p$(ml,6): DIM z$(6)
 120 LET t$="linha X1  y1  z1  x
2  y2  z2"
1000 REM Menu principal
1010 CLS : PRINT AT 5,7;"Carregu
e na tecla:"''
1020 PRINT "A - p/ apresentar as
 coordenadas        no ecran (dis
play data)"
1030 PRINT "I - p/ introduzir as
 coordenadas"
1040 PRINT "L - p/ introduzir (l
oad) coorde-        nadas gravada
s em casset-        te"
1050 PRINT "P - p/ imprimir as c
oordenadas"
1060 PRINT "T - p/ terminar"
1070 PRINT "G - p/ gravar (save)
 as coorde-        nadas em cass
ette"
1080 PRINT "U - p/ ver o desenho
"
1100 LET k$=FN a$(INKEY$)
1110 IF k$="A" THEN GO TO 2000
1120 IF k$="I" THEN GO TO 3000
1130 IF k$="L" THEN GO TO 4000
1140 IF k$="P" THEN GO TO 5000
1150 IF k$="G" THEN GO TO 6000
1160 IF k$="U" THEN GO TO 7000
1170 IF k$<>"T" THEN GO TO 1100
1200 CLS : PRINT AT 5,7; FLASH 1
;" O PROGRAMA PAROU "
1210 PRINT ''"    Para continuar
,introduza:"
1220 PRINT AT 10,4;"RUN (apaga a
s coordenadas)"''
1230 PRINT "ou          GO TO 100
0"''TAB 4;"(nao apaga as coorden
adas)"
```



```
1240 STOP
2000 REM Apresentar as coordenad
as
2010 CLS : PRINT AT 10,4;"A part
ir da linha ?"
2020 GO SUB 8100: LET l=i: IF l<
1 OR l>ml THEN GO TO 2020
2100 CLS : PRINT t$
2110 FOR a=l TO l+16: IF a>ml TH
EN GO TO 2200
2120 PRINT 'a;: IF p$(a)=z$ THEN
 GO TO 2140
2130 FOR b=1 TO 6: PRINT TAB 1+4
*b;CODE p$(a,b)-133;: NEXT b
2140 NEXT a
2200 PRINT ''"Mais (S/N) ?"
2210 LET k$=FN a$(INKEY$)
2220 IF k$<>"n" AND k$<>"N" AND
k$<>"s" AND k$<>"S" THEN GO TO 2
210
2230 IF k$="n" OR k$="N" OR a>ml
 THEN GO TO 1000
2240 LET l=a: GO TO 2100
3000 REM Introduzir dados
3010 CLS : PRINT "Introduza o nu
mero de linha, se-guido pelas co
ordenadas x, y e zdo primeiro po
nto,e depois pelascoordenadas x,
 y e z do segundo ponto. Todos e
stes elementos de-vem ser separa
dos por espacos."
3020 PRINT ''"Ou carregue em ENTE
R para voltar"''"          ao menu p
rincipal"''t$
3100 INPUT LINE i$: IF i$="" THE
N GO TO 1000
3110 GO SUB 8000: LET l=i: IF l<
1 OR l>ml THEN GO TO 3100
3120 FOR a=1 TO 6
3130 GO SUB 8000: IF ABS i>100 T
HEN GO TO 3100
3140 LET p$(l,a)=CHR$ (133+i)
3150 NEXT a
3160 POKE 23692,0: PRINT 'l;
3170 FOR a=1 TO 6: PRINT TAB 2+4
*a;CODE p$(l,a)-133;: NEXT a
3180 GO TO 3100
4000 REM Introduzir coordenadas
       gravadas em cassette
4010 CLS : PRINT AT 5,0;"Nome do
 ficheiro ?";CHR$ 6;
4020 INPUT LINE i$: PRINT i$
4030 LOAD i$ DATA p$()
```

```
4040 INPUT "Quer verificar (S/N)
 ?"; LINE a$
4050 IF a$="N" OR a$="n" THEN GO
 TO 1000
4060 IF a$<>"S" AND a$<>"s" THEN
 GO TO 4040
4070 PRINT '"Volte a passar a g
ravacao para"'TAB 11;"verificar
."
4080 VERIFY i$ DATA p$()
4090 GO TO 1000
5000 REM Imprimir as coordenadas
5010 GO SUB 8200
5020 LPRINT 't$
5030 FOR a=l1 TO l2: LPRINT 'a;:
 IF p$(a)=z$ THEN GO TO 5050
5040 FOR b=1 TO 6: LPRINT TAB 2+
4*b;CODE p$(a,b)-133;: NEXT b
5050 NEXT a
5060 LPRINT '''
5080 GO TO 1000
6000 REM Gravar as coordenadas
         em cassete
6010 CLS : PRINT AT 5,0;"Nome do
 ficheiro ?";CHR$ 8;
6020 INPUT LINE i$: IF i$="" THE
N GO TO 6020
6030 IF LEN i$>10 THEN LET i$=i$
( TO 10)
6040 PRINT i$
6050 SAVE i$ DATA p$()
6060 PRINT '"Volte a passar a gr
avacao para"'TAB 11;"verificar"
6070 PRINT '"Se a verificacao fa
lhar, o pro- grama ira' parar"''
"Use GO TO 1000 para continuar;
 nao utilize RUN !"
6080 VERIFY i$ DATA p$()
6090 GO TO 1000
7000 REM Ver o desenho
7010 CLS : PRINT AT 5,0;"Introdu
za :"''
7020 PRINT TAB 4;"Primeiro numer
o de linha",TAB 4;"Ultimo numero
 de linha"
7030 PRINT TAB 4;"Factor de esca
la (1-999)"
7040 PRINT TAB 4;"Rotacao horizo
ntal (0-360)"
7050 PRINT TAB 4;"Rotacao vertic
al (0-360)"
7060 PRINT TAB 4;"Profundidade (
0-9)"
7070 PRINT '"Separados por espac
os"
```



```
7100 GO SUB 8100: LET l1=i: IF l
1<1 OR l1>ml THEN GO TO 7100
7110 GO SUB 8000: LET l2=i: IF l
2<l1 OR l2>ml THEN GO TO 7100
7120 GO SUB 8000: LET s=i/100: I
F s<.01 OR s>9.9 THEN GO TO 7100
7130 GO SUB 8000: LET hr=i: IF h
r<0 OR hr>360 THEN GO TO 7100
7140 GO SUB 8000: LET vr=i: IF v
r<0 OR vr>360 THEN GO TO 7100
7150 GO SUB 8000: LET d=i: IF d<
0 OR d>9 THEN GO TO 7100
7200 CLS
7210 LET ch=COS (hr*PI/180): LET
 sh=SIN (hr*PI/180)
7220 LET cv=COS (vr*PI/180): LET
 sv=SIN (vr*PI/180)
7300 FOR l=l1 TO l2: IF p$(l)=z$
 THEN GO TO 7400
7310 LET x1=FN b(1): LET y1=FN b
(2): LET z1=FN b(3)
7320 LET x2=FN b(4): LET y2=FN b
(5): LET z2=FN b(6)
7330 LET xx1=x1*ch-y1*sh: LET xx
2=x2*ch-y2*sh
7340 LET yy1=y1*ch+x1*sh: LET yy
2=y2*ch+x2*sh
7350 LET yy1=yy1*cv-z1*sv: LET y
y2=yy2*cv-z2*sv
7360 LET zz1=z1*cv+yy1*sv: LET z
z2=z2*cv+yy2*sv
7370 LET p=1+d*zz1/1000: LET xx1
=xx1*p: LET yy1=yy1*p
7380 LET p=1+d*zz2/1000: LET xx2
=xx2*p: LET yy2=yy2*p
7390 PLOT xx1+127,yy1+87: DRAW x
x2-xx1,yy2-yy1
7400 NEXT l
7410 INPUT "Quer imprimir (S/N)
? "; LINE i$
7420 IF i$="s" OR i$="S" THEN CO
PY : GO TO 1000
7430 IF i$="n" OR i$="N" THEN GO
 TO 1000
7440 GO TO 7410
8000 REM Atribuir o primeiro nu-
      mero de i$ a i, reduzir i$
8010 LET sgn=1
8020 LET i=0: IF i$="" OR i$=" "
 THEN RETURN
8030 IF i$(1)<>"-" AND (i$(1)<"0
" OR i$(1)>"9") THEN LET i$=i$(2
 TO ): GO TO 8020
```

```
8040 IF i$(1)="-" THEN LET sgn=-
1: LET i$=i$( 2 TO ): GO TO 8020
8050 FOR c=1 TO LEN i$: LET c$=i
$(c)
8060 IF c$<"0" OR c$>"9" THEN GO
 TO 8080
8070 NEXT c
8080 LET i=sgn*VAL i$( TO c-1):
LET i$=i$(c TO )
8090 RETURN
8100 REM Input de i$,chamar 8000
8110 INPUT LINE i$: GO TO 8000
8200 REM Obter a primeira e a
     ultima linha l1 & l2
8210 CLS : PRINT AT 10,0;"Introd
uza os numeros da primeirae da u
ltima linha, separados porum esp
aco em branco:"
8220 INPUT LINE i$: GO SUB 8000:
 LET l1=i
8230 IF l1<1 OR l1>ml THEN GO TO
 8200
8240 GO SUB 8000: LET l2=i
8250 IF l2<1 OR l2>ml THEN GO TO
 8200
8260 RETURN
9000 DEF FN a$(a$)=CHR$ (CODE a$
-(32 AND CODE a$>96))
9100 DEF FN b(z)=(CODE p$(l,z)-1
33)*s
```

1000-1170 Apresentam o menu principal e obtêm a escolha do utilizador, saltando seguidamente para o módulo seleccionado por este.

1200-1240 Fazem a sequência de execução sair do módulo, apresentam um aviso e seguidamente param o programa.

2000-2240 Módulo que apresenta as coordenadas no *écran*.

2010-2030 Obtêm o primeiro número de linha, verificam-no, limpam o *écran* e imprimem o cabeçalho t$.

2110-2140 Ciclo executado 17 vezes para apresentar 17 linhas.

2120-2130 Apresentam o número de linha e, seguidamente, os 6 números que constituem as coordenadas (se existirem) dos extremos dessa linha.

2200-2240 Perguntam ao utilizador se deseja ver mais linhas ou voltar ao menu principal.

3000-3180 Módulo para introdução de coordenadas.

3010-3020 Apresentam instruções e a linha de cabeçalho t$.

3100-3180 Ciclo executado uma vez por cada linha.

3100 Obtém do utilizador a cadeia da linha de *input* e volta ao menu principal se a cadeia está vazia.

3110 Toma da cadeia o número de linha e verifica-o.

3120-3150 Ciclo destinado a tomar as 6 coordenadas da cadeia, verificar a sua validade e depois colocá-las no quadro *(array)* p.

3160-3180 Apresentam o número de linha e 6 coordenadas que tenham sido introduzidas na linha com esse número (permitindo o "rolamento" da imagem se tal for necessário), voltando atrás seguidamente para a linha de *input* seguinte.

4000-4090 Módulo para carregar na memória dados gravados em cassette *(load)*. Obtém do utilizador o nome do ficheiro, apresenta-o no écran (o CHR$ 8 na linha 4010 faz a posição de impressão recuar de modo a que, quando o nome do ficheiro é impresso no écran, ela sobreponha o símbolo *"?"*). Seguidamente, carrega em memória o quadro *(array)* p$(), verifica-o se o utilizador o desejar e volta ao menu principal.

5000-5080 Módulo para impressão das coordenadas.

5010-5020 Obtém o primeiro e o último número de linha (L1 & L2) do bloco a ser impresso. Imprimem a linha de cabeçalho t$.

5030-5050 Ciclo executado para imprimir cada uma das linhas L1 a L2 e "saltar" *(skip)* a impressão de coordenadas se uma linha não contiver quaisquer dados.

5040 Ciclo destinado a imprimir 6 coordenadas de uma linha.

5060-5080 Imprimem algumas linhas em branco e voltam ao menu.

6000-6090 Módulo para gravar *(save)* as coordenadas em *cassette*.

6010-6040 Obtêm o nome do ficheiro; não aceitam uma cadeia vazia e truncam tudo o que exceder um máximo de 10 caracteres. Feito isto, apresentam o nome no *écran*.

6050-6090 Passam o quadro p$( ) para *cassette* e fazem com que o utilizador volte atrás e passe a gravação, para que esta seja verificada, antes de regressarem ao menu principal.

7000-7440 Módulo para ver o desenho.

7010-7150 Obtêm do utilizador o primeiro e o último números de linha, factores de escala, rotações e profundidade, e verificam-nos.

7200-7220 Limpam o *écran* e calculam alguns valores a utilizar posteriormente.

7300-7400 Ciclo executado uma vez por cada linha a ser desenhada.

7310-7320 Extraem de p$( ) as seis coordenadas para uma linha.

7330-7340 Calculam novas coordenadas, tomando em consideração a rotação horizontal.

7350-7360 Calculam novas coordenadas, tomando em consideração a rotação vertical.

7370-7380 Fazem alguns acertos de forma a obter um efeito de profundidade.

7390 Desenha a linha no *écran*.

7410-7440 Imprimem uma cópia permanente *(hard copy)* do desenho por meio da impressora *ZX* e fazem o programa voltar ao menu principal.

8000-8090 Sub-rotina que procura um número inteiro na cadeia i$, atribui o valor desse número a i e, seguidamente, retira o número de i$. É atribuído a i o valor Ø se não for encontrado qualquer número em i$. Chamando repetidamente esta sub-rotina, extraem-se números sucessivos de i$.

8100 Sub-rotina que obtém do utilizador uma cadeia de *input* j$ e que dela extrai o primeiro número por meio da sub-rotina 8000.

8200 Sub-rotina que pede ao utilizador para introduzir uma linha contendo dois números de linha e que, seguidamente, extrai esses números (*l1* e *l2*) e verifica a sua validade.

9000 Nesta linha, uma função definida pelo utilizador dá a "versão" maiúscula de um carácter que poderia ser maiúsculo ou minúsculo.

9100 Função definida pelo utilizador destinada a extrair o valor da coordenada z (z=1 a 6) da linha l do quadro de coordenadas.

Os dados (coordenadas e números de linha) são armazenados no quadro de caracteres p$(400,6), sendo utilizado um carácter para armazenar cada coordenada, de acordo com a fórmula:

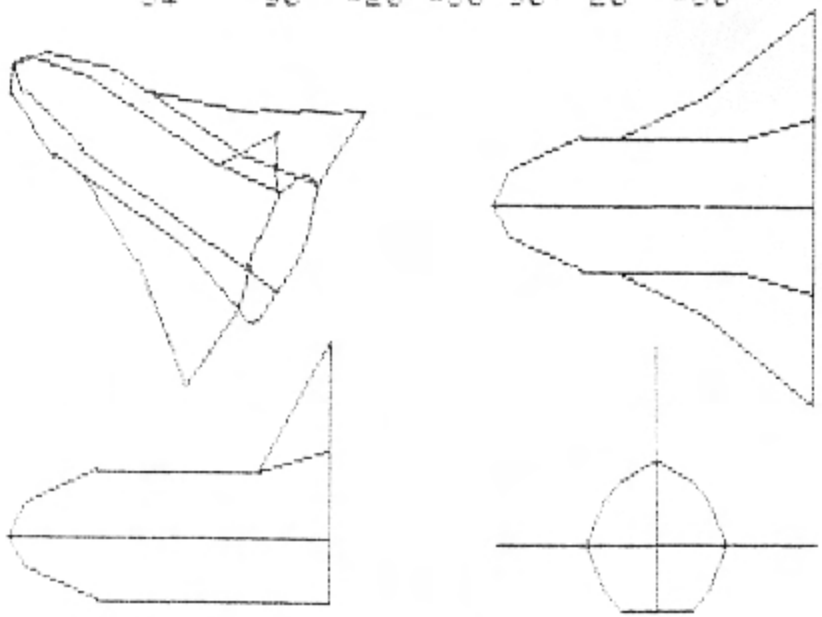
b$(L,A) = CHR$ (Valor da coord.+133)

Isto permite ao programa conter valores inteiros de −100 a 100 sem nenhum valor se assemelhar ao carácter "espaço em branco" (código 32), o qual é utilizado para representar "ausência de dados".

As dimensões máximas do quadro foram escolhidas de modo a que este caiba perfeitamente na memória de um *Spectrum* de 16 K; contudo, quem dispuser de um de 48 K e quiser mesmo utilizar mais de 400 linhas, pode atribuir à variável ml (*max line* — número máximo de linha) qualquer valor até 5800!

Valores para o *Space shuttle* dados pela impressora *ZX*

Linha	x1	y1	z1	x2	y2	z2
1	-80	0	0	-60	0	15
2	-80	0	0	-60	0	-15
3	-80	0	0	-60	-15	0
4	-80	0	0	-60	15	0
5	-80	15	0	-40	30	0
6	-80	-15	0	-40	-30	0
7	-80	0	-15	-40	0	-30
8	-80	0	15	-40	0	30
9	-40	30	0	50	30	0
10	-40	-30	0	50	-30	0



LABIRINTO

Este programa elabora um labirinto aleatório e deixa, seguidamente, o utilizador "entrar" nele e tentar encontrar a saída.

O labirinto é construído a partir de uma grelha quadriculada e, quando o jogo começa, o utilizador selecciona as suas dimensões, variáveis ente 3×3 e 30×20.

Uma vez dentro do labirinto, podemos carregar nas teclas

"F" — para se deslocar um quadrado para a frente (desde que não haja uma parede imediatamente em frente!);
"E" ou "D" — para se voltar para a esquerda ou para a direita;
"R" — para rodar sobre si mesmo e encarar a direcção de onde veio (para voltar atrás, por exemplo).

Repare-se que apenas "F" permite realmente a deslocação para outra parte do labirinto; "E", "D" e "A" apenas fazem rodar sobre si próprio, no mesmo lugar.

Além de apresentar uma vista da parte do labirinto que se encontra directamente à nossa frente, o *écran* mostram também uma "bússola", para sabermos para que direcção estamos voltados, e ainda o número de "passos" dados até ao momento.

Se estivermos completamente perdidos, carregamos na tecla H, obtendo um mapa de todo o labirinto, com uma setazinha a indicar o local onde nos encontramos e o sentido para onde estamos voltados, mas a utilização desta opção custa uma penalidade de 50 passos.

A saída encontra-se indicada por uma grande cruz diagonal.

```
 500 REM "LABIRINTO"
1000 REM Inicializar e obter as
       dimensoes do labirinto
1010 RANDOMIZE : INK 0: PAPER 7:
 FLASH 0: BRIGHT 0: OVER 0: INVE
RSE 0: BORDER 7: CLS
1020 GO SUB 9100: REM Caracteres
     definidos pelo utilizador
1030 INPUT "DIMENSOES DO LAB. :
X (3-30) ";mx;TAB 20;"Y (3-20) "
;my
1040 LET mx=INT mx: IF mx<3 OR m
x>30 THEN GO TO 1030
1050 LET my=INT my: IF my<3 OR m
y>20 THEN GO TO 1030
1060 LET passo=0
2000 REM Construir o labirinto
2010 FOR a=1 TO mx+1: PLOT 8*a-1
,167: DRAW 0,-8*my: NEXT a
```

```
2020 FOR a=1 TO my+1: PLOT 7,175
-8*a: DRAW 8*mx,0: NEXT a
2030 LET x=1: LET y=INT (1+my/4+
RND*my/2): PRINT AT y,0;">"
2040 PLOT 8,8*y-1: DRAW INVERSE
1;0,7
2050 PRINT AT y,x; PAPER 6; OVER
 1;" "
2060 LET dx=1: LET dy=0
2070 LET yourx=x: LET youry=y: L
ET d=0
2100 IF RND>.1 AND dy=0 THEN LET
 dy=INT (2*RND)*2-1: LET dx=0
2110 IF RND>.8 OR y+dy>my OR y+d
y<1 THEN LET dx=1: LET dy=0
2120 GO SUB 9000
2130 IF x<=mx THEN GO TO 2100
2140 PRINT AT y,x;">": LET x=x-1
2200 LET ny=y+dy: LET nx=x+dx
2210 IF RND>.4 AND ny<=my AND ny
>=1 AND nx<=mx AND nx>=1 THEN IF
 ATTR (ny,nx)=56 THEN GO TO 2290
2220 LET u=0: DIM u(4,2)
2230 IF x<mx THEN IF ATTR (y,x+1
)=56 THEN LET u=u+1: LET u=u+1:
LET u(u,1)=1
2240 IF x>1 THEN IF ATTR (y,x-1)
=56 THEN LET u=u+1: LET u(u,1)=-
1
2250 IF y<my THEN IF ATTR (y+1,x
)=56 THEN LET u=u+1: LET u(u,2)=
1
2260 IF y>1 THEN IF ATTR (y-1,x)
=56 THEN LET u=u+1: LET u(u,2)=-
1
2270 IF u=0 THEN GO TO 2300
2280 LET u=INT (1+u*RND): LET dx
=u(u,1): LET dy=u(u,2)
2290 GO SUB 9000: GO TO 2200
2300 PRINT AT y,x; PAPER 5; OVER
 1;" "
2310 LET cx=8*x+3: LET cy=171-8*
y
2320 IF POINT (cx+4,cy)=0 THEN I
F ATTR (y,x+1)=48 THEN LET x=x+1
: GO TO 2220
2330 IF POINT (cx-4,cy)=0 THEN I
F ATTR (y,x-1)=48 THEN LET x=x-1
: GO TO 2220
2340 IF POINT (cx,cy-4)=0 THEN I
F ATTR (y+1,x)=48 THEN LET y=y+1
: GO TO 2220
```



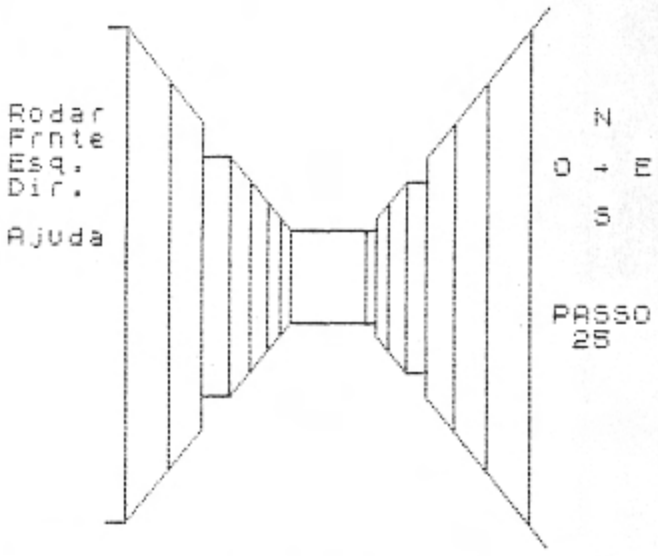
```
2350 IF POINT (cx,cy+4)=0 THEN I
F ATTR (y-1,x)=48 THEN LET y=y-1
: GO TO 2220
2400 FOR z=1 TO mx*my/10
2410 LET x=2+INT ((mx-2)*RND): L
ET y=2+INT ((my-2)*RND)
2420 LET dx=2*INT (2*RND)-1: LET
 dy=0
2430 IF RND>.5 THEN LET dy=2*INT
 (2*RND)-1: LET dx=0
2440 GO SUB 9000
2450 NEXT z
2460 PLOT 7,167: DRAW 8*mx,0: PL
OT 7,175-8*(my+1): DRAW 8*mx,0
3000 REM Copiar o labirinto para
       m$()
3010 INPUT "Carregue em ENTER pa
ra comecar "; LINE i$
3020 DIM m$(my+2,mx+2)
3030 POKE 23606,PEEK 23675: POKE
 23607,PEEK 23676-1
3040 FOR y=0 TO my+1: FOR x=0 TO
 mx+1
3050 LET m$(y+1,x+1)=SCREEN$ (y,
x): PRINT AT y,x;" "
3060 NEXT x: NEXT y
3070 POKE 23606,0: POKE 23607,60
4000 REM Deslocar-se pelo labiri
nto
4010 CLS : LET p1=87: LET p=80:
LET x=yourx: LET y=youry
4020 PRINT AT 4,29;"N";AT 6,27;"
O ";CHR$ (149+d);" E";AT 8,29;"S
"
4030 LET passo=passo+1: PRINT AT
 12,27;"PASSO";TAB 28;passo
4040 PRINT AT 4,0;"Rodar"'"Frnte
"'"Esq."'"Dir."'"Ajuda"
4100 FOR z=1 TO 20
4110 PLOT 127-p,87-p: DRAW 0,2*p
: PLOT 127+p,87-p: DRAW 0,2*p
4120 LET c=CODE m$(y+1,x+1): LET
 w=CODE m$(y+1,x): LET s=CODE m$
(y+2,x+1)
4130 IF d=0 THEN LET wr=(s=33)+(
s=35): LET wl=(c=33)+(c=35): LET
 wf=(c=33)+(c=34)
4140 IF d=1 THEN LET wr=(w=33)+(
w=34): LET wl=(c=33)+(c=34): LET
 wf=(s=33)+(s=35)
4150 IF d=2 THEN LET wr=(c=33)+(
c=35): LET wl=(s=33)+(s=35): LET
 wf=(w=33)+(w=34)
4160 IF d=3 THEN LET wr=(c=33)+(
```

```
c=34): LET wl=(w=33)+(w=34): LET
 wf=(c=33)+(c=35)
4170 IF z=1 THEN LET fw=wf
4200 LET dp=p1-p: LET dwr=dp AND
 wr: LET dwl=dp AND wl
4210 LET er=NOT wr AND ((d=3 AND
 x=mx) OR (d=1 AND x=1))
4220 LET el=NOT wl AND ((d=3 AND
 x=1) OR (d=1 AND x=mx))
4230 IF NOT er THEN PLOT 127+p,8
7-p: DRAW dp,-dwr: PLOT 127+p,87
+p: DRAW dp,dwr
4240 IF NOT el THEN PLOT 127-p,8
7-p: DRAW -dp,-dwl: PLOT 127-p,8
7+p: DRAW -dp,dwl
4250 IF er THEN PLOT 127+p,87+p:
 DRAW dp,-(2*p+dp): PLOT 127+p,8
7-p: DRAW dp,2*p+dp
4260 IF el THEN PLOT 127-p,87+p:
 DRAW -dp,-(2*p+dp): PLOT 127-p,
87-p: DRAW -dp,2*p+dp
4270 IF wf THEN PLOT 127-p,87-p:
 DRAW 2*p,0: PLOT 127-p,87+p: DR
AW 2*p,0: GO TO 5000
4300 IF (d=0 AND x=mx) OR (d=2 A
ND x=1) THEN LET pd=.9*p: PLOT 1
27-pd,87-pd: DRAW 2*pd,2*pd: PLO
T 127-pd,87+pd: DRAW 2*pd,-2*pd:
 GO TO 5000
4310 LET x=x+(d=0)-(d=2): LET y=
y+(d=1)-(d=3): LET p1=p: LET p=I
NT (.8*p)
4320 NEXT z
5000 REM Obter o passo do joga-
         dor
5010 LET k$=INKEY$: IF k$>"Z" TH
EN LET k$=CHR$ (CODE k$-32)
5020 IF k$<>"A" AND k$<>"E" AND
k$<>"D" AND k$<>"R" AND k$<>"F"
THEN GO TO 5000
5030 IF k$="A" THEN GO TO 6000
5040 IF k$="F" THEN GO TO 5100
5050 IF k$="E" THEN LET d=d-1: I
F d<0 THEN LET d=3
5060 IF k$="D" THEN LET d=d+1
5070 IF k$="R" THEN LET d=d+2
5080 IF d>3 THEN LET d=d-4
5090 GO TO 4000
5100 IF fw THEN GO TO 5000
5110 IF yourx=1 AND d=2 THEN PRI
NT AT 10,12; PAPER 5; FLASH 1;"E
NTRADA";AT 11,12;"FECHADA": GO T
O 5000
```



```
5120 IF yourx=mx AND d=0 THEN GO
 TO 7000
5130 LET yourx=yourx+(d=0)-(d=2)
: LET youry=youry+(d=1)-(d=3)
5140 GO TO 4000
6000 REM Apresentar a planta do
labirinto c/a posicao do jogador
6010 CLS : POKE 23606,PEEK 23675
: POKE 23607,PEEK 23676-1
6020 LET xp=INT ((30-mx)/2): LET
 yp=INT ((20-my)/2)
6030 FOR y=1 TO my+2: PRINT AT y
p+y-1,xp;m$(y): NEXT y
6040 POKE 23606,0: POKE 23607,60
6050 LET passo=passo+50: PRINT O
VER 1;AT 0,5;"PENALIDADE: 50 PAS
SOS"
6060 PRINT AT yp+youry,xp+yourx;
 OVER 1;CHR$ (149+d)
6070 INPUT TAB 2;"Com ENTER volt
a ao labirinto "; LINE i$
6080 GO TO 4000
7000 REM Finalizar
7010 PAPER 5: CLS
7020 PRINT AT 8,9;"SAIU AO FIM D
E";AT 10,14;passo;AT 12,13;"PASS
OS"
7030 DATA 0,5,9,5,9,12,9,12,15,1
5
7040 FOR t=1 TO 3: RESTORE 7000
7050 FOR a=1 TO 10: READ b: BEEP
 .05,b: NEXT a
7060 NEXT t: PAPER 7
7070 STOP
9000 REM Apagar parede em x+dx,
         y+dy
9010 IF dy=0 THEN PLOT 8*(x+(dx=
1))-1,167-8*y: DRAW INVERSE 1;0,
7
9020 IF dx=0 THEN PLOT 8*x-1,175
-8*(y+(dy=1)): DRAW INVERSE 1;7,
0
9030 LET x=x+dx: LET y=y+dy
9040 PRINT AT y,x; PAPER 6; OVER
 1;" "
9060 RETURN
9100 REM Definir caracteres def.
         pelo utilizador
9110 DATA 0,0,0,0,0,0,0,0
9120 DATA 255,1,1,1,1,1,1,1
9130 DATA 1,1,1,1,1,1,1,1
9140 DATA 255,0,0,0,0,0,0,0
```

```
9150 DATA 1,0,0,0,0,0,0,0
9160 DATA 0,0,0,8,124,8,0,0
9170 DATA 0,0,0,16,16,16,56,16,0
9180 DATA 0,0,0,32,124,32,0,0
9190 DATA 0,0,16,56,16,16,16,0
9200 RESTORE 9100
9210 FOR a=0 TO 71: READ b: POKE
  USR "a"+a,b: NEXT a
9220 RETURN
```



A estrutura global do programa é muito simples, e o seu comprimento deve-se à necessidade de prover diferentes possibilidades quando se constrói o labirinto ou se desenham as vistas interiores. Utilizámos duas sub-rotinas: a primeira é chamada de vários pontos do programa enquanto o labirinto está a ser criado; a segunda sub-rotina gera os gráficos definíveis pelo utilizador e foi elaborada na forma de sub-rotina apenas pela conveniência de a manter fora do programa principal.

1000-1060 Inicializam e, seguidamente, obtêm do jogador as dimensões do labirinto e verificam a sua validade.
2000-2450 Constroem o labirinto.
2010-2020 Desenham uma rede de quadrados completa.

2030-2070 Seleccionam um ponto de entrada no lado esquerdo da grelha.
2100-2130 Abrem um caminho aleatório apagando paredes até ser atingido o lado direito da grelha. O caminho é marcado a amarelo para que o programa o possa reencontrar posteriormente — além de tornar a imagem mais atraente.
2140 Marca a saída.
2200-2350 O programa volta a seguir o caminho amarelo, em sentido inverso juntando-lhe passagens laterais onde quer que lhe seja possível, até se atingir a entrada.
2200-2210 Durante 60% do tempo o programa continua na mesma direcção, se o quadrado seguinte não tiver sido «visitado» (ATTR=56, branco).
2220-2280 O programa selecciona um quadrado adjacente que ainda não tenha sido visitado, ou salta para 2300 caso não exista algum.
2290 Esta linha elimina a parede e marca o quadrado a amarelo.
2300 O caminho não pode ir mais além nesta direcção e, portanto, o quadrado é marcado a azul-celeste.
2310-2350 Descobrem qual dos 4 quadrados circundantes constituía o passo anterior do caminho (ATTR=48, amarelo) e voltam então a esse quadrado regressando depois à linha 2230.
2410-2450 Complicam o labirinto abatendo aleatoriamente cerca de 5% das paredes.
3000-3070 Copiam o labirinto terminado, do *écran* para m$(); os primeiros 5 caracteres definidos pelo utilizador (ver linhas 9100 -) são todos os necessários para desenhar um labirinto.
4000-4040 Desenham a bússola, listam comandos e imprimem o número de passos.
4100-4320 Desenham a vista actual. O ciclo da linha 4100 à linha 4320 limita a profundidade da vista a 20 quadrados mas, na prática, a linha 4270 ou 4300 terminará o desenho quando for alcançada uma parede fronteira (ou a saída).
5000-5140 Obtêm o comando do utilizador, verificam-no

quanto à sua validade, ajustam seguidamente as coordenadas (yourx,youry) ou a direcção (d) e fazem a sequência de execução do programa voltar à linha 4000 para a vista seguinte, ou à linha 6000 se a tecla "A" tiver sido pressionada.

6000-6080 Mostram o mapa do labirinto imprimindo m$(), com a variável de sistema CHARS convenientemente alterada.

7000-7070 Rotina de saída triunfante.

DRAGÃO

Lee Gentry

O DRAGÃO é um jogo de astúcia e raciocínio rápido, jogado no topo de um planalto rectangular. Deslocamo-nos sobre o planalto utilizando as teclas "5", "6", "7" e "8", mas deve ter-se cuidado para não ultrapassar a borda e cair!

O planalto está coberto de bosques e pântanos, os quais atrasarão o andamento de quem os quiser atravessar. Para tornar as coisas mais complicadas, os pântanos são invisíveis! Um número intermitente algures sobre o planalto representa um tesouro que devemos apanhar. Quando tivermos recolhido um tesouro, logo aparecerá outro

Mas, ao mesmo tempo que nós, há no planalto um dragão enraivecido e esfomeado! Tal como nós, ele é entravado pelos pântanos, e ainda mais pelos bosques, mas, no seu desespero para os atravessar, esmaga-os ao passar através deles, reduzindo assim a percentagem de "protecção". Sendo um dragão tão maldoso, ele esconderá qualquer tesouro que encontre e, assim, o leitor terá de o descobrir apenas por memória e sorte!

Por quanto tempo conseguirá o leitor sobreviver ao dragão?

```
  2 REM "DRAGAO"
  5 CLS : RESTORE : LET h=0
 10 FOR a=0 TO 7
 20 READ b: POKE USR CHR$ 116+a
,b
 30 READ b: POKE USR CHR$ 100+a
,b
 40 READ b: POKE USR CHR$ 109+a
,b
 50 NEXT a
 60 FOR a=20 TO 5 STEP -1: BEEP
.1,a: NEXT a
 70 FOR a=5 TO 20: BEEP .1,a: N
EXT a
 80 GO SUB 800
100 REM A sua deslocacao
110 LET b=b+(b<=0): BEEP .01,10
120 LET p(w,z)=e: LET i=w: LET
j=z
130 IF a>=0 THEN LET w=w+(INKEY
$="6")-(INKEY$="7"): LET z=z+(IN
KEY$="8")-(INKEY$="5")
140 IF p(w,z)=4 OR p(w,z)=9 THE
N LET r=p(w,z): GO TO 600
150 IF w=x AND z=y THEN GO SUB
500: GO TO 170
160 IF a>=0 AND p(w,z)<>0 THEN
LET a=p(w,z)-2: BEEP .1,-10
170 PRINT AT i,j-2;(" " AND e<=
0); INK 4;(CHR$ 153 AND e=1); IN
K 6;(CHR$ 79 AND e=2)
180 LET e=p(w,z): LET p(w,z)=5
190 PRINT AT w,z-2;CHR$ 156
300 REM A deslocacao do Dragao
310 LET a=a+(a<=0): BEEP .01,-2
0
320 LET c=0: LET d=0
330 IF ABS (w-u)>=ABS (z-v) THE
N LET c=SGN (w-u)
340 IF c=0 THEN LET d=SGN (z-v)
350 LET p(u,v)=(e AND e<>1)
360 PRINT AT u,v-2;(" " AND f<>
2); INK 6;(CHR$ 79 AND f=2)
370 IF b>=0 THEN LET u=u+c: LET
v=v+d
380 PRINT AT u,v-2;CHR$ 147
390 IF p(u,v)=5 THEN LET r=5: G
O TO 600
400 IF b>=0 AND p(u,v)>0 THEN L
ET b=b-4: BEEP .1,-38
410 LET f=p(u,v): LET p(u,v)=4
420 GO TO 100
500 REM Pontuacao
510 BEEP 1.5,45
520 LET s=s-k: PRINT AT 0,0;"Po
nt. Max. = ";h;TAB 16;"A sua pon
t. = ";s
```

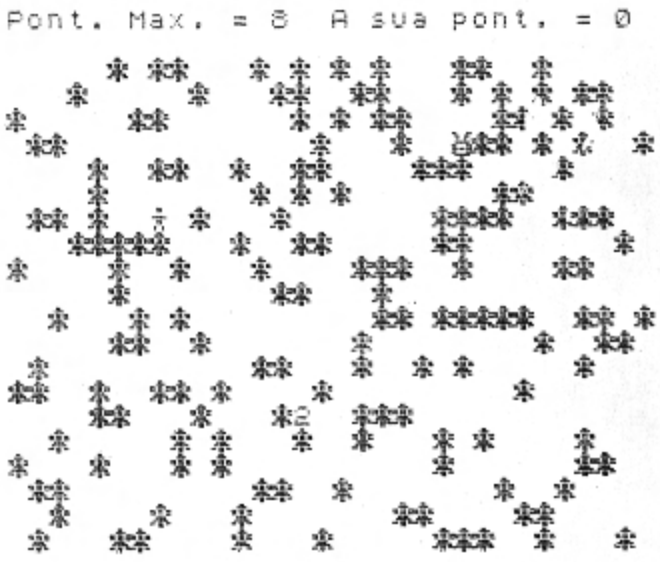
```
 530 IF p(x,y+1)<7 AND l=1 THEN
LET p(x,y+1)=1: PRINT AT x,y-1;
INK 4;CHR$ 163
 540 LET p(x,y)=l: PRINT AT x,y-
2; INK 4;(CHR$ 163 AND l=1)
 550 LET x=INT (RND*20+2): LET y
=INT (RND*32+2)
 560 IF ABS (x-u)<6 AND ABS (y-v
)<6 THEN GO TO 550
 570 LET k=-INT (RND*9+1): LET l
=p(x,y)
 580 LET p(x,y)=k: PRINT AT x,y-
2; FLASH 1; INK 6; PAPER 2;-k
 590 RETURN
 600 REM Fim
 610 IF r=9 THEN FOR a=1 TO 15:
BEEP .3,-a-30: NEXT a
 620 IF r<>9 THEN BEEP 5,-38
 630 CLS
 640 IF r>7 THEN PRINT AT 12,6;"
Voce caiu do penhasco": GO TO 67
0
 650 IF r=4 THEN PRINT AT 12,0;"
IDIOTA-Foi esbarrar com o Dragao
"
 660 PRINT AT 14,10;"Ele comeu-o
!"
 670 PRINT AT 16,8;"Voce esta mo
rto !"
 680 PRINT AT 8,10;"Pont. Max. "
;h;TAB 10;"V. pontuou ";s
 690 IF s>h THEN PRINT AT 4,0; I
NK 1; PAPER 6;"***********CAMPE
AO!***********"
 700 IF s>h THEN LET h=s
 710 PRINT AT 20,5;"Outra tentat
iva ? (s/n)"
 720 FOR a=25 TO 5 STEP -1: BEEP
 .1,a: NEXT a
 730 IF INKEY$="s" THEN GO TO 60
 740 IF INKEY$<>"n" THEN GO TO 7
30
 750 BORDER 7: BRIGHT 0: CLS : S
TOP
 800 REM Preparacao
 810 DIM l$(32): DIM p(22,34)
 820 LET s=0: LET a=0: LET b=0
 830 LET c=0: LET d=0: LET e=0:
LET f=0
 840 BORDER 2: INPUT ""
 850 FOR a=1 TO 30
 860 LET x=INT (RND*20+2): LET y
=INT (RND*32+2)
 870 LET p(x,y)=2: PRINT AT x,y-
2; INK 6;CHR$ 79
 880 NEXT a
 890 LET u=INT (RND*20+2): LET v
=INT (RND*32+2)
 900 LET w=INT (RND*20+2): LET z
=INT (RND*32+2)
 910 IF ABS (w-u)<6 AND ABS (z-v
)<6 THEN GO TO 900
 920 PAUSE 200: CLS : GO SUB 550
 930 PRINT AT 0,0;"Pont. Max. =
";h;TAB 16;"A sua pont. = ";s; P
APER 2;l$
 940 PRINT AT w,z-2;CHR$ 156: LE

T p(w,z)=5
 950 PRINT AT u,v-2;CHR$ 147: LE
T p(u,v)=4
 960 FOR a=1 TO 175
 970 LET p=INT (RND*20+2): LET q
=INT (RND*32+2)
 980 IF p(p,q)<>0 THEN GO TO 970
 990 LET p(p,q)=1: PRINT AT p,q-
2; INK 4;CHR$ 163
1000 IF a<35 THEN LET p(1,a)=9:
LET p(22,a)=9
1010 IF a<23 THEN LET p(a,1)=9:
LET p(a,34)=9
1020 NEXT a
1030 FOR a=1 TO 10: BEEP .5,20+(
10*(-1 AND a=2*INT (a/2))): NEXT
 a
1040 RETURN
9000 DATA 24,195,24,60,102,24,21
9,36,0,219,126,126,60,195,24,126
,219,24,219,195,36,24,126,36
```

5- 80 Definem os caracteres gráficos definíveis pelo utilizador e tocam a "melodia de abertura". Seguidamente, utilizam a sub-rotina 800 para desenhar o planalto e instalar uma imagem dele no quadro *(array)* b().

100-420 Ciclo principal do programa, executado repetidamente para movimentar quer a nós quer ao dragão. A nossa posição é dada por w e z, a do dragão por u e v, e a do tesouro por x e y.

500-590 Sub-rotina chamada sempre que o leitor alcance o tesouro.

600-750 Vários tipos de finalização, seguida de um convite para jogar novamente.



Capítulo 13

Aplicações

Parece que a maior parte do *software* escrito para o *Spectrum* se destina a jogos de vários tipos. Isto não é necessariamente um mal, pois escrever programas de jogos é uma forma agradável de aprender a programar, e as regras e a estrutura simples da maior parte dos jogos significam que os programas não deveriam ser muito complicados. E, no fim de contas, os jogos são divertidos!

Mas não nos esqueçamos de que o *Spectrum* é, em muitos aspectos, mais poderoso do que a primeira geração de computadores comerciais, e pode efectuar muitas tarefas úteis referentes a gestão de negócios ou de engenharia.

E não está limitado às aplicações tradicionais de cálculo e de contabilidade, pois quando apetrechado com um *port* de I/O adequado torna-se num controlador ideal para uso laboratorial em todas as ciências.

Neste capítulo damos alguns exemplos de tais programas "úteis"; mas encontraremos aplicação para eles na forma em que se apresentam, e também podemos adaptá-los a necessidades específicas efectuando as modificações necessárias, apesar de não terem sido apresentadas quaisquer explicações relativas às listagens de tais programas, pois o objectivo deste capítulo é indicar utilizações possíveis e não propriamente ensinar técnicas de programação.

Em geral, o *Spectrum* adequa-se melhor a aplicações que necessitem apenas de dar entrada ou saída (*input* ou *output*) a um número de dados relativamente pequeno, e que não utilizem ficheiros com mais de 30 000 caracteres, aproximadamente. Fora disso, os únicos limites à utilização do *Spectrum* são a imaginação de cada qual e o número de horas de um dia!

MÉDIAS

Lee Gentry

```
DADOS INTRODUZIDOS :
1 : 1                2 : 2
3 : 4                4 : 8
```

```
5  :  16              6  :  32
7  :  64              8  :  80
9  :  90              10 :  95
11 :  95              12 :  90
13 :  80              14 :  64
15 :  32              16 :  16
17 :  8               18 :  4
19 :  2               20 :  1
Media aritmetica  = 39.2
Media geometrica  = 16.431174
Media harmonica   = 4.9541434
Raiz media quadratica= 53.838648
```

Este programa, destinado aos amadores de estatística, permite introduzir quantos dados numéricos desejarmos e calcula seguidamente as médias aritmética, geométrica e harmónica, e também a raiz média quadrática.

Os dados devem ser introduzidos um de cada vez. Para terminar a introdução, vamos introduzir "s".

```
  10 REM "MEDIAS"
  20 LET a$="": LET n=0: PRINT "
INTRODUZA OS DADOS :"'"(Introdu
za 's' para terminar a"'"introdu
ção dos dados e iniciar"'"o calc
ulo das medias)"
  30 INPUT "DADO ";(n+1)'i$: IF
i$<>"s" THEN LET n=n+1: LET a$=a
$+i$+CHR$ 124: GO TO 30
  40 CLS : PRINT "DADOS INTRODUZ
IDOS :": DIM x(n)
  50 FOR a=1 TO n
  60 LET b=0
  70 LET b=b+1: IF a$(b)<>CHR$ 1
24 THEN GO TO 70
  80 LET x(a)=VAL a$(1 TO b-1):
LET a$=a$(b+1 TO ): PRINT a;" :
";x(a),
  90 NEXT a
 100 REM Media aritmetica
 110 LET b=0
 120 FOR a=1 TO n: LET b=b+x(a):
 NEXT a
 130 PRINT '"Media aritmetica =
";b/n
 200 REM Media geometrica
 210 LET b=1
 220 FOR a=1 TO n: IF x(a)>0 THE
```



```
N LET b=b*x(a): NEXT a
 230 IF a<=n THEN PRINT "A media
 geometrica nao e'valida": GO TO
 300
 240 PRINT "Media geometrica = "
;b↑(1/n)
 300 REM Media harmonica
 310 LET b=0
 320 FOR a=1 TO n: IF x(a)<>0 TH
EN LET b=b+1/x(a): NEXT a
 330 IF a<=n THEN PRINT "A media
 harmonica nao e' valida": GO TO
 400
 340 PRINT "Media harmonica  = "
;n/b
 400 REM Raiz media quadratica
 410 LET b=0
 420 FOR a=1 TO n: LET b=b+x(a)↑
2: NEXT a
 430 PRINT "Raiz media quadratic
a= ";SQR (b/n)
 440 INPUT "Pressione ENTER p/ r
ecomecar";i$: RUN
```

À medida que cada um dos dados é introduzido (linha 30), vai sendo acrescentado ao fim da cadeia a$, ficando os dados separados uns dos outros por CHR$124. Isto é repetido até que "s" seja introduzido.

G. P. G. P.

Este "General Purpose Graph Plotter" (traçador de gráficos para fins genéricos correntes) representa um auxiliar dinâmico durante apresentações de negócios, ou na preparação de relatórios impressos.

Admite até 9 ficheiros de dados, cada um dos quais com valores para os 12 meses de um ano. A informação (dados) contida em dois ficheiros quaisquer pode ser representada (traçada — *plotted*) em qualquer altura sob a forma de um gráfico de barras ou de um gráfico de linha. O dimensionamento do eixo vertical é automático, de modo a dar um factor de escala máximo que é uma potência de dez vezes 1, 2 ou 5.

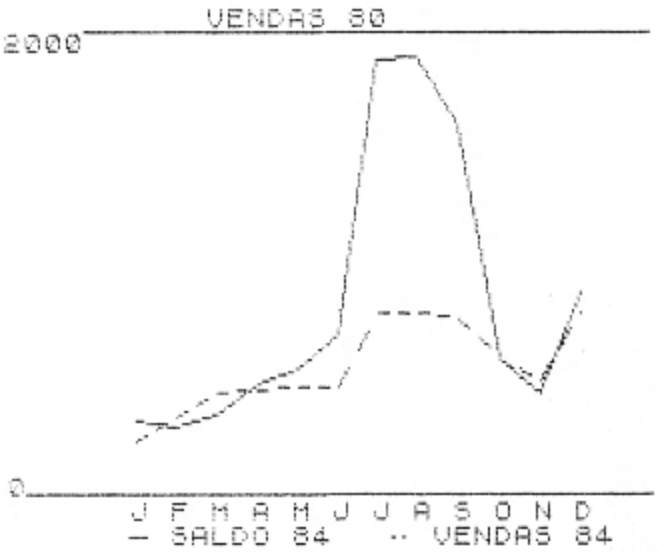
Diferentes grupos de ficheiros podem ser gravados — com o programa — em *cassette* sob nomes diferentes.

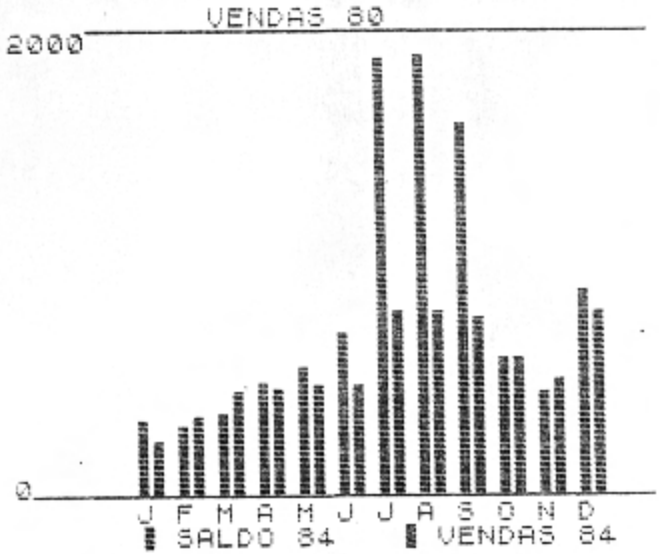
Quando não está a apresentar um gráfico, o programa apresenta ao utilizador um menu de actividades possíveis:

— traçar um novo gráfico de um ou de dois ficheiros e seguidamente copiá-lo para a impressora, se o desejarmos;
— reexaminar a informação (dados) contida em qualquer dos 9 ficheiros, e alterá-la se necessário;
— introduzir ou mudar o título do conjunto de 9 ficheiros;
— passar o programa e os dados para *cassette*, usando o título como nome.

O programa está projectado de forma a ser executado automaticamente quando carregado em memória a partir de *cassette*. Se parar por qualquer razão, fazemo-lo arrancar de novo com GO TO 1000, pois RUN apagará todos os dados (é claro que se terá de empregar RUN ao executá-lo pela primeira vez, para definir os quadros de dados).

Na prática, a única maneira de o fazer parar é carregar em CAPS/SHIFT e BREAK enquanto ele está a efectuar cálculos ou CAPS/SHIFT e "6" enquanto ele está à espera de um *input*.






```
  50 REM "GPGP"
 100 DIM d(9,13): DIM m$(9,10):
DIM t$(10): DIM x$(13)
 110 LET l$="J F M A M J J A S O
 N D"
 120 LET m$="JanFevMarAbrMaiJunJ
ulAgoSetOutNovDez"
 130 GO SUB 9100
1000 REM PARTE PRINCIPAL
1010 PAPER 7: INK 0: CLS : PRINT
 TAB 10;t$
1020 FOR a=1 TO 9: PRINT AT a+1,
9;a;TAB 12;n$(a): NEXT a
1100 PRINT AT 15,4;"Introduza :"
1110 PRINT AT 17,4;"D - p/ desen
har (plot)";TAB 4;"R - p/ reexam
inar dados"
1120 PRINT TAB 4;"G - p/ gravar
(save)";TAB 4;"T - p/ introduzir
 titulo"
1130 INPUT LINE i$: LET i$=FN a$
(i$)
1140 FOR a=15 TO 21: PRINT AT a,
0,,: NEXT a
1150 IF i$="R" THEN GO TO 2000
1160 IF i$="D" THEN GO TO 3000
1170 IF i$="G" THEN GO TO 4000
```

```
1180 IF i$="T" THEN GO TO 5000
1190 GO TO 1100
2000 INPUT "Ficheiro (1-9) ? ";
LINE i$
2010 GO SUB 9000: IF i<1 OR i>9
THEN GO TO 2000
2020 LET f=i: PRINT AT f+1,9; OV
ER 1; FLASH 1;"•"
2100 FOR m=1 TO 12
2110 PRINT AT FN c(),FN d();m$(m
*3-2);"•";d(f,m)
2120 NEXT m
2200 INPUT "Alterar (s/n) ? "; L
INE i$
2210 IF i$="n" OR i$="N" THEN GO
 TO 1000
2220 IF i$<>"s" AND i$<>"S" THEN
 GO TO 2200
2230 INPUT "Nome do ficheiro ? "
; LINE n$(f): PRINT AT f+1,12;n$
(f)
2300 FOR m=1 TO 12
2310 INPUT "Dados para ";(m$(m*3
-2 TO m*3));" ? "; LINE i$
2320 GO SUB 9000: IF e<>0 THEN G
O TO 2310
2330 LET d(f,m)=i: LET x$=STR$ i
2340 PRINT AT FN c(),2+FN d();x$
2350 NEXT m
2360 INPUT "Pressione ENTER p/vo
ltar ao menu"; LINE i$
2400 LET a=0
2410 FOR m=1 TO 12: IF d(f,m)>a
THEN LET a=d(f,m)
2420 NEXT m
2430 FOR b=-30 TO 30: IF 10↑b>=a
 THEN GO TO 2450
2440 NEXT b
2450 LET m=10↑b: IF m/2>=a THEN
LET m=m/2
2460 IF .2*10↑b>=a THEN LET m=.2
*10↑b
2470 LET d(f,13)=m
2480 GO TO 1000
3000 REM Desenho dos graficos
3010 INPUT "Linhas ou Barras (L/
B) ? "; LINE p$: LET p$=FN a$(p$
)
3020 IF p$<>"L" AND p$<>"B" THEN
 GO TO 3010
3030 INPUT "Primeiro ficheiro (1
-9) ? "; LINE i$
3040 IF LEN i$<>1 OR i$<"0" OR i
```



```
$>"9" THEN GO TO 3030
3050 LET f1=CODE i$-CODE "0"
3060 PRINT AT 21,6; INK 2;CHR$ (
133+(11 AND p$="L")); INK 0;"•";
n$(f1)
3100 INPUT "Segundo ficheiro (1-
9) ? "; LINE i$
3110 IF i$="" THEN LET f2=0: GO
TO 3200
3120 IF LEN i$<>1 OR i$<"0" OR i
$>"9" THEN GO TO 3100
3130 LET f2=CODE i$-CODE "0"
3140 PRINT AT 21,19; INK 1;CHR$
(133+(12 AND p$="L")); INK 0;"•"
;n$(f2)
3200 FOR a=0 TO 20: PRINT AT a,0
,,: NEXT a
3210 LET hi=d(f1,13)
3220 IF f2>0 THEN IF d(f2,13)>hi
 THEN LET hi=d(f2,13)
3300 PLOT 0,17: DRAW 255,0: PLOT
 0,167: DRAW 255,0
3310 PRINT AT 0,10;t$;AT 1,0;hi;
AT 19,0;0;AT 20,6;t$
3320 IF p$="B" THEN GO TO 3500
3400 LET s=FN b(d(f1,1)): PLOT 5
1,s
3410 FOR a=2 TO 12: LET y=FN b(d
(f1,a)): DRAW 16,y-s: LET s=y: N
EXT a
3420 IF f2=0 THEN GO TO 3900
3430 LET s=FN b(d(f2,1)): PLOT 5
1,s
3440 FOR a=2 TO 12: LET y=FN b(d
(f2,a))
3450 DRAW 4,(y-s)/4
3460 PLOT 27+16*a,s+(y-s)/2: DRA
W 4,(y-s)/4
3470 LET s=y: PLOT 35+16*a,s
3480 NEXT a
3490 GO TO 3900
3500 FOR a=1 TO 12
3510 FOR s=34+a*16 TO 37+a*16
3520 INK 2: PLOT s,17: DRAW 0,FN
 b(d(f1,a))-17
3530 IF f2>0 THEN INK 1: PLOT s+
6,17: DRAW 0,FN b(d(f2,a))-17
3540 NEXT s
3550 NEXT a
3560 INK 0
3900 INPUT "Quer imprimir (S) ?
"; LINE i$
3910 IF i$="s" OR i$="S" THEN CO
PY
```

```
3920 GO TO 1000
4000 REM Gravar em cassette
4010 SAVE t$ LINE 4020
4020 GO SUB 9100: GO TO 1000
5000 REM Input do titulo
5010 INPUT "Titulo ? "; LINE t$
5020 GO TO 1000
8000 REM Funcoes
8010 DEF FN a$(a$)=CHR$ (CODE a$
-(32 AND CODE a$>96))
8020 DEF FN b(v)=INT (v*150/hi+1
7)
8030 DEF FN c()=13+m-6*INT ((m-1
)/6)
8040 DEF FN d()=16*INT ((m-1)/6)
9000 REM Converter i$ em i
9010 LET i=0: LET e=1: IF i$=""
THEN RETURN
9020 LET dp=0
9030 FOR c=1 TO LEN i$: LET c$=i
$(c)
9040 IF c$="." THEN LET dp=dp+1:
 GO TO 9060
9050 IF c$<>"+" AND (c$<"0" OR c
$>"9") THEN RETURN
9060 NEXT c: IF dp>1 THEN RETURN

9070 LET i=VAL i$: LET e=0: RETU
RN
9100 REM Definir caracteres--e--
9110 DATA 0,0,0,0,255,0,0,0,0,0,
0,0,204,0,0,0
9120 RESTORE : FOR a=0 TO 15: RE
AD b: POKE USR "a"+a,b: NEXT a
9130 RETURN
```

Agenda

Lee Gentry

Não temos agora desculpa para o esquecimento "daqueles" aniversários ou de encontros importantes!

Este programa leva o *Spectrum* a actuar como uma agenda de secretária. Introduzimos mensagens ou notas de qualquer comprimento sob qualquer data, e elas são arquivadas segundo a ordem das datas respectivas. Inclusivamente, podemos arquivar

mais do que uma mensagem na mesma data. Observamos o que se encontra na agenda, pesquisando entre duas datas quaisquer e, tendo encontrado o que procurávamos, podemos voltar ao menu principal. Além disso, se a informação se desactualizar, podemos apagá-la e libertar espaço de memória para novas anotações. E, naturalmente, podemos transferir a agenda para *cassette* em qualquer altura.

A operação do programa é auto-explicativa mas, na eventualidade de ele vir a parar por qualquer razão, utiliza-se GO TO 13Ø para o fazer arrancar de novo e não RUN, pois isto apagaria todas as anotações.

```
   5 REM "AGENDA"
  10 REM Preparacao
  20 DEF FN f(x)=INT (x-100*INT
(x/100))
  30 DEF FN f$(x)=("0" AND FN f(
x)<10)+STR$ FN f(x)
  40 DEF FN g$(x$,x)=x$(x+4 TO x
+5)+"/"+x$(x+2 TO x+3)+"/"+x$(x
TO x+1)
  50 LET a$="": LET l$=""
  60 LET c=0
  70 DIM b$(5)
 100 REM Menu
 110 PRINT AT 20,0;"Pressione um
a tecla p/ continuar";
 120 PAUSE 0
 130 CLS
 140 PRINT AT 2,13;"AGENDA"
 150 PRINT AT 5,5;"(1)...Introdu
zir mensagem"
 160 PRINT AT 7,5;"(2)...Apagar
dia"
 170 PRINT AT 9,5;"(3)...Mostrar
"
 180 PRINT AT 11,5;"(4)...Gravar
 'Agenda'"
 185 PRINT AT 13,5;"(5)...Termin
ar s/ gravar"
 190 PRINT AT 21,0;"Introduza a
opcao desejada"
 200 INPUT n: LET n=INT n: IF n<
1 OR n>5 THEN GO TO 200
 210 CLS
 220 IF n=1 THEN GO TO 300
 230 GO TO 300*n+300
 300 REM Input
 310 PRINT AT 0,8;"Escrever apon
tamento"
```

```
 320 LET c=c+1
 330 LET m$="": LET a=1
 340 INPUT "Dia  ";d'"Mes  ";m'"
Ano  ";y
 350 LET d$=FN f$(y)+FN f$(m)+FN
 f$(d): LET e$=FN g$(d$,1)
 360 CLS : PRINT AT 0,0;e$;TAB 1
2;"Apontamento"
 370 INPUT "Paragrafo ";(a),i$:
PRINT AT 2,2;i$
 380 LET l=LEN i$-32*INT (LEN i$
/32)
 390 FOR z=l TO 29: LET i$=i$+"
": NEXT z
 400 LET m$=m$+"••"+i$
 410 PRINT AT 21,0;"Mais algum ?
 (s/n)"
 420 IF INKEY$="s" THEN LET a=a+
1: GO TO 350
 430 IF INKEY$<>"n" THEN GO TO 4
10
 440 LET b$=STR$ (LEN a$+1): LET
 l$=l$+b$
 450 PRINT AT 2,0;m$: LET a$=a$+
m$
 460 LET b$=STR$ (LEN m$): LET l
$=l$+b$+d$
 470 FOR z=1 TO 16*c-16 STEP 16
 480 LET p=VAL l$(z TO z+4)
 490 LET r=(l$(z+10 TO z+15)=d$)
+(2 AND l$(z+10 TO z+15)>d$)
 500 IF r=1 THEN LET c=c-1: GO T
O 660
 510 IF r=2 THEN GO TO 540
 520 NEXT z
 530 GO TO 100
 540 LET v=VAL l$(16*c-15 TO 16*
c-11): LET p=VAL l$(z TO z+4)
 550 LET a$=a$( TO p-1)+a$(v TO
)+a$(p TO v-1)
 560 LET h$=l$(16*c-10 TO 16*c)
 570 FOR y=16*c-31 TO z STEP -16
 580 LET l$(y+21 TO y+31)=l$(y+5
 TO y+15)
 590 NEXT y
 600 LET l$(z+5 TO z+15)=h$
 610 FOR y=z TO c*16-16 STEP 16
 620 LET b$=STR$ (VAL l$(y TO y+
4)+VAL l$(y+5 TO y+9))
 630 LET l$(y+16 TO y+20)=b$
 640 NEXT y
 650 GO TO 100
```



```
 660 FOR y=z+16 TO 16*c STEP 16
 670 LET l$(y TO y+4)=STR$ (VAL
l$(y TO y+4)+LEN m$)
 680 NEXT y
 690 LET v=VAL l$(z TO z+4)+VAL
l$(z+5 TO z+9)
 700 LET a$=a$( TO v-1)+m$+a$(v
TO LEN a$-LEN m$)
 710 LET l$(z+5 TO z+9)=STR$ (VA
L l$(z+5 TO z+9)+LEN m$)
 720 LET l$=l$( TO LEN l$-16)
 730 GO TO 100
 900 REM Apagar
 910 PRINT AT 0,6;"Apagar aponta
mento"
 920 INPUT "Apagar qual ?"'"Dia
";d'"Mes ";m'"Ano ";y
 930 LET d$=FN f$(y)+FN f$(m)+FN
 f$(d)
 940 LET e$=FN g$(d$,1)
 950 FOR z=1 TO 16*c STEP 16
 960 LET p=VAL l$(z TO z+4)
 970 IF l$(z+10 TO z+15)=d$ THEN
 GO TO 1010
 980 IF l$(z+10 TO z+15)<d$ THEN
 NEXT z
 990 PRINT AT 5,4;"Nao existe ap
ontamento em"'';TAB 12;e$
1000 GO TO 100
1010 LET a$=a$( TO p-1)+a$(VAL l
$(z+5 TO z+9)+p TO )
1020 LET v=VAL l$(z TO z+4)
1030 FOR y=z TO 16*c-20 STEP 16
1040 LET l$(y+5 TO y+15)=l$(y+21
 TO y+31)
1050 LET v=VAL l$(y TO y+4)+VAL
l$(y+5 TO y+9)
1060 LET l$(y+16 TO y+20)=STR$ v
1070 NEXT y
1080 LET l$=l$( TO 16*c-16)
1090 PRINT AT 5,4;"O apontamento
 de ";e$''TAB 11;"foi apagado"
1100 LET c=c-1
1110 GO TO 100
1200 REM Imprimir
1210 PRINT AT 0,7;"Mostrar apont
amentos"
1220 INPUT "Desde quando ?"'"Dia
 ";d'"Mes ";m'"Ano ";y
1230 LET d$=FN f$(y)+FN f$(m)+FN
 f$(d)
1240 INPUT "Ate' quando ?"'"Dia
```

```
";d'"Mes ";m'"Ano ";y
1250 LET e$=FN f$(y)+FN f$(m)+FN
 f$(d)
1260 IF e$<d$ THEN LET i$=e$: LE
T e$=d$: LET d$=i$
1270 LET f=0
1280 FOR z=1 TO 16*c STEP 16
1290 LET p=VAL l$(z TO z+4)
1300 IF d$<=l$(z+10 TO z+15) THE
N GO TO 1340
1310 NEXT z
1320 LET f=1
1330 GO TO 1440
1340 IF e$<l$(z+10 TO z+15) THEN
 LET f=z: GO TO 1310
1350 CLS
1360 PRINT AT 0,0;FN g$(l$,z+10)
1370 PRINT AT 2,0;a$(p TO p-1+VA
L l$(z+5 TO z+9))
1380 PRINT AT 20,0;"Carregue em
 's' p/ continuar a"
1390 PRINT AT 21,0;"apresentacao
, e 'n' p/ terminar"
1400 IF INKEY$="n" THEN GO TO 13
0
1410 IF INKEY$<>"s" THEN GO TO 1
400
1420 NEXT z
1430 IF f<>1 THEN FOR z=1 TO 50:
 NEXT z: GO TO 100
1440 LET d$=FN g$(d$,1)
1450 LET e$=FN g$(e$,1)
1460 PRINT AT 5,4;"Nao ha'aponta
mentos entre ";TAB 12;d$;" e";TA
B 12;e$
1470 GO TO 100
1500 REM Gravar
1510 SAVE "AGENDA" LINE 1520
1520 PRINT AT 5,0;"Rebobine a fi
ta - para VERIFICAR";AT 7,0;"Dep
ois, pressione qualquer tecla"
1530 PAUSE 0
1540 VERIFY "AGENDA"
1550 GO TO 100
1800 CLS : PRINT INK 2; FLASH 1;
AT 7,8;"PROGRAMA TERMINADO"
1810 PRINT AT 10,5;"Para recomec
ar introduza:";AT 13,4;"RUN - Ap
aga os apontamentos.";AT 15,4;"G
O TO 130 - Nao os apaga.": STOP
```


SÓCIOS

Este programa deverá facilitar a vida a um secretário de uma associação com muito trabalho. Guarda dados sobre um máximo de 365 sócios (num *Spectrum* de 48 K), que podem ser gravados em *cassette*, e permite alterar ou fazer imprimir a informação sob várias formas.

A informação arquivada relativa a cada sócio consiste nomeadamente em:

— número de associado. São números atribuídos sequencialmente, à medida que se acrescentam novos sócios ao ficheiro, partindo de um número inicial introduzido quando o ficheiro foi criado;
— nome, número de telefone e endereço, até um total de 94 caracteres;
— duas "chaves" de 3 caracteres, capazes de representar informação tal como a classe de associado a que o sócio pertence, e o mês no qual paga as quotas.

Se a associação é muito grande, ultrapassa-se a dificuldade gravando uma *cassette* separada com os dados de cada grupo de 365 sócios. Consegue-se fazer executar este programa num *Spectrum* de 16 K mas, nesse caso, o número de inscrições terá de ficar limitado a 42, alterando a primeira declaração da linha 100 para LET m=42.

Se o programa eventualmente parar por qualquer razão, não se deve usar RUN para o fazer arrancar de novo, pois isso destruiria todos os registos. Em sua substituição, utilize-se GO TO 1ØØØ.

O *écran* terá geralmente o seguinte aspecto:

```
Número   :
Nome     :
Telef.   :
Ender.   :
         :
         :
         :
         :
Chave 1  :
Chave 2  :
Número + - M   N   P   G   ■
```

O quadrado negro é um cursor intermitente, que aparece por todo o programa para sinalizar o *input* seguinte necessário. Neste caso, ele é apresentado na linha principal do menu, e o utilizador pode optar entre introduzir:

número : introduzindo um número válido de associado, obter-se-á a apresentação no *écran* dos dados existentes sobre esse sócio.

\+ : apresenta os dados referentes ao sócio de número imediatamente superior ao do apresentado no momento.

– : apresenta os dados referentes ao sócio anterior;

M : permite modificar os dados sobre o sócio que esteja a ser apresentado no momento;

N : para acrescentar um novo sócio ao ficheiro;

P : para imprimir *(print out)* dados sobre os sócios na impressora *ZX;*

G : para gravar em *cassette* o programa com os dados.

Se introduzirmos M ou N, a linha do menu principal, apresentada na parte inferior do *écran,* desaparece, e o cursor intermitente desloca-se para a posição de impressão imediatamente a seguir a "Nome :". Introduz-se então o nome do sócio. Feito isto, o cursor desloca-se para baixo, para a introdução do número de telefone do sócio em causa. Continua-se a preencher as linhas no *écran* até que, após introduzir a informação a incluir em "Chave 2 :", o menu principal apareça de novo.

Se se introduzir uma quantidade demasiada de informação em qualquer dos elementos apresentados no menu principal, o programa ignora o excesso, e apenas apresenta no *écran* os caracteres para os quais tenha espaço disponível.

Se se escolher a opção P (*print* — imprimir), a imagem no *écran* muda para:

Chave 1 :
Chave 2 :
Tudo :

e o programa aguarda que se introduzam dados diante destas três

linhas antes de começar a imprimir. Para as linhas "Chave 1 :" e "Chave 2 :", há a escolher entre não introduzir nada (carregando simplesmente na tecla ENTER) e introduzir uma chave com um máximo de 3 caracteres. Neste último caso, apenas serão impressos os registos dos sócios que tenham uma chave correspondente. Isto permite — por exemplo — fazer imprimir uma lista dos sócios cujas quotas devam ser pagas em "AGO".

A terceira linha (Tudo :) requer uma resposta de "S" ou "N". Introduzindo "S", obtém-se a impressão de todos os dados referentes a cada sócio e, deste modo, pode-se utilizar a entrada para indicar endereços a colar em sobrescritos, p. ex.

Introduzir N provocará a impressão apenas do número de associado, do seu nome e das duas chaves.

Por fim, com a opção G (gravar) grava-se em *cassette* o programa com a última versão dos dados. O programa pedirá o nome do ficheiro a utilizar e, após a gravação, dará a oportunidade de verificar a informação gravada. O programa é gravado juntamente com os dados, e de forma a que seja automaticamente executado quando for carregado em memória; também neste caso haverá a possibilidade de verificar a informação que tenha sido carregada em memória.

```
  50 REM "SOCIOS"
 100 LET m=365: DIM d$(m,100)
 110 DIM z$(25): DIM x$(3): DIM
k$(1)
 120 INK 0: PAPER 7: FLASH 0: BR
IGHT 0: OVER 0: INVERSE 0: BORDE
R 7: CLS
 200 PRINT AT 8,4;"NOVO FICHEIRO
 DE DADOS"
 210 PRINT AT 15,0;"Numero do pr
imeiro socio ? "; FLASH 1;"."
 220 GO SUB 9000: IF i=0 THEN BE
EP .1,20: GO TO 220
 230 LET prim=i: LET segt=i: LET
 corr=0
1000 REM Imprimir menu principal
1010 CLS
1020 PRINT AT 3,0;"Numero:";AT 5
,0;"Nome  :"
1030 PRINT AT 7,0;"Telef.:";AT 9
,0;"Ender.:"
1040 PRINT AT 10,6;":";AT 11,6;"
:";AT 12,6;":";AT 13,6;":"
```

```
1050 PRINT AT 15,0;"Chave1:";AT
16,0;"Chave2:"
2000 REM Escolher opcao do menu
          principal
2010 PRINT AT 21,0;"Numero  +  -
  M  N  P  G  T  "; FLASH 1;"."
2020 GO SUB 9000: PRINT AT 21,0,
;: IF i>0 THEN GO TO 3000
2030 IF CODE i$>90 THEN LET i$=C
HR$ (CODE i$-32)
2040 IF i$="+" THEN GO TO 3100
2050 IF i$="-" THEN GO TO 3200
2060 IF i$="M" THEN GO TO 4000
2070 IF i$="N" THEN GO TO 4100
2080 IF i$="P" THEN GO TO 5000
2090 IF i$="G" THEN GO TO 6000
2095 IF i$="T" THEN GO TO 6161
2100 GO TO 2000
3000 REM Recuperar reg. numerado
3010 IF i<prim OR i>=segt THEN G
O TO 2000
3020 LET corr=i: GO TO 3300
3100 REM Recuperar reg. seguinte
3110 IF corr>=segt-1 THEN GO TO
2000
3120 LET corr=corr+1: GO TO 3300
3200 REM Recuperar reg. anterior
3210 IF corr<=prim THEN GO TO 20
00
3220 LET corr=corr-1
3300 REM Recuperar reg. corrente
3310 FOR a=3 TO 16: PRINT AT a,7
;z$: NEXT a
3320 PRINT AT 3,8;corr: LET pos=
7
3330 FOR l=5 TO 13: IF l=6 OR l=
8 THEN LET l=l+1
3340 PRINT AT l,8;
3350 IF pos>100 THEN GO TO 3500
3360 LET q=CODE d$(corr+1-prim,p
os): LET pos=pos+1
3370 IF q<128 THEN PRINT CHR$ q;
: GO TO 3350
3380 PRINT CHR$ (q-128);
3390 NEXT l
3500 PRINT AT 15,8;d$(corr+1-pri
m, TO 3)
3510 PRINT AT 16,8;d$(corr+1-pri
m,4 TO 6)
3520 GO TO 2000
4000 REM Modificar reg. corrente
4010 IF corr<prim OR corr>=segt
THEN GO TO 2000
```



```
4020 GO TO 4200
4100 REM Adicionar novo registo
4110 IF segt>=prim+m THEN PRINT
AT 19,0; FLASH 1;" FICHEIRO ": B
EEP 1,10: GO TO 2000
4120 LET corr=segt: LET segt=seg
t+1
4130 FOR a=3 TO 16: PRINT AT a,7
;z$: NEXT a
4200 REM Introduzir dados
4210 PRINT AT 3,8;corr: LET pos=
7
4220 FOR l=5 TO 13: IF l=6 OR l=
8 THEN LET l=l+1
4230 PRINT AT l,7; FLASH 1;"."
4240 LET j$=z$: IF pos>100 THEN
GO TO 4300
4250 INPUT LINE i$: IF LEN i$>24
 THEN LET i$=i$( TO 24)
4260 IF i$="" THEN LET i$="."
4270 LET k=LEN i$: IF pos+k>100
THEN LET i$=i$( TO 101-pos)
4280 LET k=LEN i$: LET j$(2 TO k
+1)=i$: LET i$(k)=CHR$ (CODE i$(
k)+128)
4290 LET d$(corr+1-prim,pos TO p
os+k-1)=i$: LET pos=pos+k
4300 PRINT AT l,7;j$
4310 NEXT l
4320 PRINT AT 15,7; FLASH 1;"."
4330 INPUT LINE i$: LET d$(corr+
1-prim, TO 3)=i$
4340 PRINT AT 15,7;".";d$(corr+1
-prim, TO 3)
4350 PRINT AT 16,7; FLASH 1;"."
4360 INPUT LINE i$: LET d$(corr+
1-prim,4 TO 6)=i$
4370 PRINT AT 16,7;".";d$(corr+1
-prim,4 TO 6)
4380 GO TO 2000
5000 REM Imprimir
5010 CLS : PRINT AT 5,0;"Chave1:
";"";"Chave2:";"";"Tudo..:"
5020 PRINT AT 5,7; FLASH 1;"."
5030 INPUT LINE i$: LET x$=i$: I
F i$<>"" THEN LET i$=x$
5040 PRINT AT 5,7;".";i$;AT 7,7;
 FLASH 1;"."
5050 INPUT LINE j$: LET x$=j$: I
F j$<>"" THEN LET j$=x$
5060 PRINT AT 7,7;".";j$;AT 9,7;
 FLASH 1;"."
5070 INPUT LINE k$: IF CODE k$>9
```

```
0 THEN LET k$=CHR$ (CODE k$-32)
5080 IF k$<>"S" AND k$<>"N" THEN
 GO TO 5070
5090 PRINT AT 9,7;"•";k$
5100 LPRINT : LPRINT
5110 FOR c=prim TO segt-1: LET l
$=d$(c-prim+1)
5120 IF (i$<>"" AND i$<>l$( TO 3
)) OR (j$<>"" AND j$<>l$(4 TO 6)
) THEN GO TO 5230
5130 IF k$="S" THEN LPRINT
5140 LPRINT c;l$( TO 6);"•";
5150 LET p=7
5160 FOR l=1 TO 7: IF k$="N" AND
 l>1 THEN GO TO 5220
5170 IF p>100 THEN GO TO 5210
5180 LET q=CODE l$(p): LET p=p+1
5190 IF q<128 THEN LPRINT CHR$ q
;: GO TO 5180
5200 LPRINT CHR$ (q-128);
5210 LPRINT
5220 NEXT l
5230 NEXT c
5240 LPRINT
5250 GO TO 1000
6000 REM Gravar dados e programa
6010 INPUT "Nome do ficheiro ?"'
 LINE n$: IF n$="" THEN GO TO 60
10
6020 IF LEN n$>10 THEN LET n$=n$
( TO 10)
6030 LET i$="": SAVE n$ LINE 610
0
6040 CLS : PRINT AT 6,0;"PODE AG
ORA VERIFICAR OS DADOS"'"TRANSFE
RIDOS PARA CASSETTE :"
6050 PRINT '"SE A VERIFICACAO NA
O RESULTAR, O PROGRAMA PARARA' C
OM ERR 9"
6060 PRINT '"DEVE ENTAO FAZE-LO
ARRANCAR DE";TAB 4;"NOVO COM 'GO
 TO 1000'"
6100 INPUT "Quer verificar (s/n)
 ? "; LINE i$
6110 IF i$="n" OR i$="N" THEN GO
 TO 1000
6120 IF i$<>"s" AND i$<>"S" THEN
 GO TO 6100
6140 PRINT AT 20,0;"VOLTE A PASS
AR A GRAVACAO PARA";TAB 10;"VERI
FICAR"
6150 LET i$="": VERIFY n$
6160 GO TO 1000
```



```
6161 CLS : PRINT INK 2; FLASH 1;
AT 7,8;"PROGRAMA TERMINADO"
6162 PRINT AT 10,5;"Para recomec
ar introduza :";AT 13,6;"RUN - A
paga o ficheiro";AT 15,6;"GO TO
1000 - Nao apaga": STOP
9000 LET i=0: INPUT LINE i$
9010 FOR a=1 TO LEN i$
9020 IF (i$(a)<"0" OR i$(a)>"9")
 AND i$(a)<>"•" THEN GO TO 9040
9030 NEXT a
9040 IF a>1 THEN LET i=VAL i$( T
O a-1)
9050 RETURN
```

Num programa que — como este — armazena dados de comprimento variável, como nomes e endereços, existe sempre um conflito entre utilizar registos de comprimento fixo, de acesso rápido, mas que desperdiçam espaço de memória na medida em que têm de ser definidos de modo a admitirem o comprimento dos dados mais longos prováveis, ou fazer os registos com um comprimento exactamente suficiente para conter os dados reais. Este programa adopta uma solução de compromisso, utilizando um registo com o comprimento fixo de 100 *bytes* para cada associado mas permitindo simultaneamente a inclusão de elementos de comprimento variável (tais como o nome) no registo.

m	número máximo de registos (365 para 48 K, 42 para 16 K)
d$(m,100)	quadro destinado a conter o ficheiro de dados. O registo de cada associado ocupa 100 *bytes*, dos quais os 6 primeiros contêm as duas chaves de 3 *bytes*. O nome, o número de telefone e um número máximo de 5 linhas de endereço são armazenados nos restantes 94 *bytes*, adicionando-se o número 128 ao código do último carácter em cada elemento, de forma a actuar como um separador de fim de linha;
prim	primeiro número de sócio do ficheiro;
corr	número de sócio que se pede correntemente, num dado momento;
segt	número do sócio a acrescentar a seguir no ficheiro.

SISTEMAS DE EQUAÇÕES

Lee Gentry

Este programa resolve qualquer sistema de equações lineares (N equações a N incógnitas), desde que o sistema seja compatível e determinado, portanto que tenha solução e que as equações sejam linearmente independentes entre si, o que significa que a solução é única (essa solução é um conjunto de N valores, o qual é solução comum a todas as equações do sistema).

```
  10 REM "SISTEMA DE EQUACOES"
  20 LET m=0: LET l=0:: LET j=1
  30 INPUT "Qual o numero de var
iaveis ? ";n
  40 DIM x(n): DIM a(n,n+1): DIM
 b(n,n+1)
  50 DEF FN f(u,v)=a(u,v)/a(u,u)
  60 DEF FN g(u,v,w)=a(v,w)-a(u,
w)*a(v,u)
  70 DEF FN h(u,v)=x(u)-a(u,v)*x
(v)
 100 REM Input
 110 FOR c=1 TO n
 120 CLS : PRINT AT 0,0;"Equacao
 ";c;TAB 18;n;" Equacoes";AT 20,
0;"Introduza os coeficientes"
 130 FOR d=1 TO n: INPUT "X";(d)
;" : ";i: LET a(c,d)=i: LET b(c,
d)=i: PRINT AT d+2,1;"X";d;" : "
;i: NEXT d
 140 INPUT "Termo independente :
 ";i: PRINT AT d+2,1;"TI = ";i:
LET a(c,d)=i: LET b(c,d)=i
 150 INPUT "Correccao (s/n) ?";T
AB 0;c$: IF c$(1)<>"s" AND c$(1)
<>"n" THEN GO TO 150: IF c$(1)="
s" THEN GO TO 130
 160 NEXT c
 170 CLS
 200 REM Pivot e Calculo
 210 FOR c=1 TO n
 215 LET p=a(c,c)
 220 FOR d=c+1 TO n: LET p=a(c,c
): IF ABS p>=ABS a(d,c) THEN GO
TO 240
 230 FOR e=1 TO n+1: LET s=a(c,e
): LET a(c,e)=a(d,e): LET a(d,e)
=s: NEXT e
 240 NEXT d
```



```
 250 FOR d=n+1 TO c STEP -1: IF
ABS p<1E-5 THEN GO TO 600
 260 LET a(c,d)=FN f(c,d)
 270 NEXT d
 280 FOR d=c+1 TO n
 290 FOR e=n+1 TO c STEP -1: LET
 a(d,e)=FN g(c,d,e): NEXT e
 300 NEXT d
 310 NEXT c
 400 REM Calculo
 410 LET x(n)=a(n,n+1)
 420 FOR c=n-1 TO 1 STEP -1: LET
 x(c)=a(c,n+1)
 430 FOR d=n TO c+1 STEP -1: LET
 x(c)=FN h(c,d): NEXT d
 440 NEXT c
 500 REM Output
 510 FOR c=1 TO n
 520 FOR d=1 TO n-1: PRINT b(c,d
);"*X";d;" + ";: NEXT d
 530 PRINT b(c,n);"*X";d;" = ";b
(c,d+1);TAB 0;
 540 NEXT c
 550 IF l=1 THEN RETURN
 560 FOR c=1 TO n: PRINT AT 20-n
+c,1;"X";c;" =";TAB 5+(1 AND x(c
)>=0);x(c): NEXT c
 570 GO TO 710
 600 REM Dependencia ou incompa-
          tibilidade
 610 FOR k=c TO n
 620 FOR h=c TO n: LET a(k,h)=a(
k,h)=a(k,h+1): NEXT h
 630 LET a(k,n)=0
 640 NEXT k
 650 IF m<>c THEN LET m=c: LET j
=m: GO TO 220
 660 LET j=j+1: IF j<n-c+1 THEN
GO TO 220
 670 LET l=1: GO SUB 500
 680 FOR k=m TO n: IF ABS a(k,n+
1)>1E-5 THEN PRINT AT 21,0;"O SI
STEMA E' INCOMPATIVEL": GO TO 710
 690 NEXT k
 700 PRINT AT 21,0;"EXISTE DEPEN
DENCIA LINEAR"
 710 STOP
```

p	Elemento principal ou *pivot*.
a(x,y)	Matriz de coeficientes: coeficiente modificado da equação x, variável y.

b(x,y)	Coeficiente original de x(y) na equação x.
x(c)	Variável x(c): parte da solução.
FN f(u,v)	Função auxiliar à divisão pelo *pivot* (só para a equação *pivot*).
FN g(u,v,w)	Função auxiliar à eliminação de variáveis.
FN h(u,v)	Função auxiliar para calcular variáveis.

```
1*X1 + 2*X2 + 3*X3 + 4*X4 = 10        X1 = 1
4*X1 + 3*X2 + 2*X3 + 1*X4 = 10        X2 = 1
0*X1 + 0*X2 + 1*X3 + 1*X4 = 2         X3 = 1
1*X1 + 1*X2 + 1*X3 + 0*X4 = 3         X4 = 1
```

BANCO

Lee Gentry

```
                Relatorio          17/12/84
                CONTA BANCARIA


    (1)         Sr G.Gentry
6/11/84         Cheque           20000$00

    (2)         VERBO
6/12/84         Dinheiro          1200$00-

    (3)         VERBO
6/12/84         Dinheiro          1200$00-

    (4)         TIMEDATA
17/12/84        Dinheiro           800$00-

    (5)         Sra G.Gentry
17/12/84        Dinheiro          1000$00-

          Saldo total            15600$00
          -------------------------------
```

Podemos agora organizar as nossas finanças eficientemente!

Este programa contém um ficheiro com um máximo de 99 registos de lançamentos, cada um dos quais compreendendo a

descrição, data, tipo de transacção e a quantia envolvida. Está organizado à volta de um menu principal, que permite ao utilizador optar por:

— obter um relatório (extracto) actualizado de qualquer registo ou de todos eles. Este relatório pode ser apresentado no *écran* ou listado na impressora;
— fazer um lançamento de crédito (receita) ou débito (pagamento);
— ver o saldo corrente;
— efectuar um lançamento de correcção;
— gravar em *cassette* o programa com os últimos dados. O programa é gravado de tal forma que se auto-executará automaticamente quando carregado em memória. O nome utilizado para referenciar a gravação é o "título" introduzido aquando da primeira execução do programa.

Quando o ficheiro de dados ficar cheio, será automaticamente renovado, apagando os primeiros 89 registos e, seguidamente, deslocando para baixo os restantes 10, de modo a formarem o início de um novo ficheiro. O utilizador é avisado deste facto quando sobrarem apenas alguns espaços vazios.

As quantias devem ser introduzidas com ponto decimal (com função de vírgula) no lugar do cifrão e pelo menos com uma casa decimal não inteira, correspondente à dezena de centavos. O zero dos centavos é inserido automaticamente.

Caso o programa eventualmente pare por qualquer razão, faz-se arrancar de novo com GO TO 1Ø0 em vez de RUN, pois isto apagaria o ficheiro de dados.

O programa utiliza a função PRINT não documentada para dirigir o *output*, quer para o *écran* (PRINT #2), quer para a impressora (#3;).

```
  5 REM "BANCO"
 10 DIM c$(100,9): DIM d$(100,8
): DIM n$(100,12)
 20 DIM m(100): DIM b$(32)
 30 LET n=0: LET i=1: LET t=0:
LET o=2: LET j=1: LET i$=""
 40 DEF FN f$(x)=STR$ (x-100*IN
T (x/100))
 50 CLS : INPUT "Titulo",a$
```

```
  60 GO TO 130
 100 REM Menu
 110 PRINT AT 21,0;"Pressione um
a tecla p/ continuar"
 120 PAUSE 0
 130 CLS : PRINT TAB 9;a$''
 140 PRINT AT 3,5;"(1)...Relator
io"
 150 PRINT AT 5,5;"(2)...Credito
(Receita)"
 160 PRINT AT 7,5;"(3)...Debito(
Pagamento)"
 170 PRINT AT 9,5;"(4)...Saldo"
 180 PRINT AT 11,5;"(5)...Corrig
ir"
 190 PRINT AT 13,5;"(6)...Gravar
 ";a$;"'"
 195 PRINT AT 15,5;"(7)...Termin
ar"
 200 IF i>96 THEN PRINT AT 18,3;
"Restam ";100-i;" registos antes
 da"'' FLASH 1; INK 2;"Renovacao
 automatica quando o ";TAB 6;"fi
cheiro ficar cheio      "
 210 INPUT TAB 4;"Seleccione a o
peracao ";k: IF k<1 OR k>7 THEN
GO TO 210
 215 IF k=7 THEN GO TO 1410
 220 PRINT AT 18,0;b$''b$'b$
 230 GO SUB (k*200+100)
 240 GO TO 100
 300 REM Relatorio
 310 PRINT AT 3,4;"*"
 320 INPUT "Ecran (1) ou Impress
ora (2) ? ";o: IF o<>1 AND o<>2
THEN GO TO 320
 330 INPUT "Registos de ";e;" a
";f: IF f>=i THEN LET f=i-1
 340 IF f<1 THEN LET f=1
 350 IF e<1 THEN LET e=1
 360 IF e>f THEN GO TO 330
 370 CLS : LET o=o+1: PRINT #o;T
AB 9;"Relatorio";TAB 24;d$(i)''T
AB 9;a$''
 380 FOR g=e+1 TO f+1
 390 PRINT #o;"  (";g-1;")";TAB
 9;n$(g);TAB 0;d$(g);TAB 9;c$(g)
;
 400 LET po=m(g)-m(g-1): LET x=2
9: GO SUB 1010
 410 IF po<0 THEN PRINT #o;"-";
 420 NEXT g
```



```
 430 IF f<i-1 THEN PRINT #o''"Re
gistos ";f+1;" a ";i-1;" n/ apre
sentados"
 440 LET x=29: LET g=i: LET po=m
(g)
 450 PRINT #o;''TAB 5;"Saldo tot
al";
 460 GO SUB 1000
 470 PRINT #o;TAB 5;"-----------
-------------"''
 480 LET o=2
 490 RETURN
 500 REM Credito
 510 LET s=1
 520 GO TO 740
 600 REM Renovacao
 610 FOR c=2 TO 11
 620 LET c$(c)=c$(c+89): LET m(c
)=m(c+89)
 630 LET n$(c)=n$(c+89): LET d$(
c)=d$(c+89)
 640 NEXT c
 650 FOR c=12 TO 100
 660 LET c$(c)=b$: LET m(c)=0: L
ET n$(c)=b$: LET d$(c)=b$
 670 NEXT c
 680 LET i=12
 690 RETURN
 700 REM Debito
 710 LET s=-1
 720 GO TO 740
 730 REM Credito e Debito
 740 PRINT AT k*2+1,4;"*";AT 15,
0;
 750 LET i=i+1: IF i=101 THEN GO
 SUB 600
 760 INPUT "Data : dia ";d;" mes
 ";m;" ano ";y
 770 LET d$(i)=FN f$(d)+"/"+FN f
$(m)+"/"+FN f$(y): CLS : PRINT A
T 16,2;d$(i);
 780 INPUT "Quantia ";a
 790 LET a=ABS a
 800 LET m(i)=.01*INT (100*(a*s+
m(i-1)+1E-3))
 810 LET x=20: LET po=m(i)-m(i-1
): LET g=i
 820 GO SUB 1010
 830 PRINT TAB 22;(" Credito" AN
D s>0);(" Debito" AND s<0)
 840 INPUT "Sacado";(+"r" AND s<
0),n$(i): PRINT 'TAB 9;n$(i)
```

```
 850 INPUT "Transaccao ";c$(i):
PRINT 'TAB 9;c$(i): IF n=5 THEN
RETURN
 860 PRINT '"Correcto (s/n) ? "
 870 IF INKEY$<>"s" AND INKEY$<>
"n" THEN GO TO 870
 880 IF INKEY$="n" THEN LET q=i-
1: PRINT AT 21,0;b$: GO SUB 1120
 890 RETURN
 900 REM Saldo
 910 LET o=2: LET g=i: LET po=m(
g): LET x=21
 920 CLS : PRINT AT 10,8;"Saldo
em ";d$(i);AT 13,0;
 930 GO SUB 1000
 940 IF m(i)<0 THEN PRINT AT 16,
9; INK 2; FLASH 1;"SEM COBERTURA
"
 950 RETURN
1000 IF m(g)<0 THEN PRINT #o;TAB
 x-LEN STR$ INT ABS po-3;"-";
1010 LET p$=STR$ INT (100*(ABS p
o-INT (.001+ABS po))+.1)
1020 IF LEN p$<2 THEN LET p$=p$+
"0"
1030 PRINT #o;TAB x-LEN STR$ INT
 (.001+ABS po)-2;INT (.001+ABS p
o);"$";p$;
1040 RETURN
1100 REM Correccao
1110 PRINT AT 11,4;"*": INPUT "Q
ual o registo ? ";q: IF q<=0 OR
q>=i THEN GO TO 1110
1120 INPUT "a-apagar m-modificar
 ";q$
1130 IF q$<>"a" AND q$<>"m" THEN
 GO TO 1120
1140 LET n=5: LET q=INT (q+.8):
IF q$="a" THEN GO TO 1230
1150 INPUT "Credito (1), Debito
(2) ";k: IF k<>1 AND k<>2 THEN G
O TO 1150
1160 LET ii=i: LET k=k+1: LET i=
q: LET p=m(q+1)
1170 GO SUB k*200+100
1180 FOR d=q+2 TO ii+(1 AND ii<=
99)
1190 LET w=m(d)+m(q+1)-p: LET m(
d)=.01*SGN w*INT (ABS w*100+.1)
1200 NEXT d
1210 LET n=0: LET i=ii
1220 RETURN
1230 LET p=m(q+1)
1240 FOR d=q+1 TO i
1250 LET m(d)=.01*INT ((m(d+1)-p
+m(q))*100+.1)
1260 LET n$(d)=n$(d+1): LET c$(d
)=c$(d+1): LET d$(d)=d$(d+1)
1270 NEXT d
1280 LET n=0: LET m(i)=0: LET i=
i-1
1290 RETURN
1300 REM Gravar
1310 PRINT AT 13,4;"*": LET z$=(
a$ AND LEN a$<=10): IF LEN a$>10
 THEN LET z$=a$( TO 10)
1320 SAVE z$ LINE 1400
1330 PRINT AT 17,0;a$'"Gravado c
om o 'titulo'"'z$
1340 PRINT AT 20,0;"Ligue o grav
ador para VERIFICAR"'"Depois pre
ssione qualquer tecla"
1350 PAUSE 0
1360 VERIFY z$
1370 RETURN
1400 CLS : GO SUB 1340: GO TO 10
0
1410 CLS : PRINT INK 2; FLASH 1;
AT 7,8;"PROGRAMA TERMINADO"
1420 PRINT AT 10,5;"Para recomec
ar introduza :";AT 13,6;"RUN - A
paga o ficheiro";AT 15,6;"GOTO 1
00 - Nao o apaga": STOP
```

Capítulo 14

Programas utilitários, artimanhas e rotinas úteis

NOTIFICAÇÃO PERMANENTE

Se quisermos tornar a primeira linha do nosso programa numa linha REM com o nosso nome ou alguma forma de notificação de direitos autorais *(copyright)*, devemos executar, depois de a ter introduzido:

```
POKE 1+PEEK 23635+256*PEEK 23636,Ø
```

Isto altera o número de linha para Ø, e depois não é possível alterá-lo ou apagá-lo do modo habitual.

No entanto é evidente que quem conhecer este artifício poderá alterar o número de linha para qualquer coisa mais tratável, introduzindo por meio de POKE (número a seguir à vírgula, na expressão acima) um outro valor: 1, por exemplo.

Uma forma mais confusa de proteger a nota e que permite inclusivamente proteger mais do que uma linha consiste em proceder da seguinte forma:

— em primeiro lugar, antes de introduzir qualquer outra coisa, introduza a linha ou as linhas a proteger;
— seguidamente, executa-se:

```
POKE PEEK 23635+256*PEEK 23636,4Ø
```

Na verdade, pode introduzir-se com POKE qualquer valor entre 40 e 63. Isto altera o número da primeira linha para uma letra ou qualquer outro símbolo seguido de três números;
— por fim, introduz-se o resto do programa.

As linhas presentes quando se utilizou o POKE ficarão ainda visíveis, mas situar-se-ão agora no fim do programa, onde não podem ser apagadas ou alteradas do modo normal. Mas, depois, não existe qualquer forma fácil de descobrir o endereço de

memória com acesso por POKE, de modo a mudar o número da linha para um que seja "conveniente", pois, ao introduzir o resto do programa, fez-se deslocar para cima, na memória, a linha afectada por POKE.

Pode usar-se ainda de maior subtileza, seguindo o mesmo processo mas, desta vez, introduzindo por meio de POKE um valor superior a 64. Neste caso as linhas desaparecem e, apesar de continuarem armazenadas na área de programa da RAM, não podem ser listadas.

TESTE À RAM

Já alguma vez teve a impressão de que a RAM o estava a trair? Se assim foi, este programa talvez o ajude, pois efectua um controle razoavelmente minucioso da totalidade da RAM do sistema, com excepção da parte utilizada pelo próprio programa e das áreas de atributos e de imagem. Este programa introduz, por meio de POKE, uma sequência de números inteiros aleatórios nas várias células de memória e, seguidamente, examina o seu conteúdo através de PEEK para ver se eles continuam lá. Tendo pesquisado uma vez toda a área de memória a testar, o programa imprimirá "OK". Quaisquer erros que encontre provocarão a impressão de "ERRO EM", com o endereço da célula que falhou no teste. A sequência de teste é repetida até que o utilizador faça parar o programa ou que o *écran* fique cheio.

Ao ser executado, o programa pede o endereço inicial da área a testar (o qual deve ser igual ou maior que 24500) e, seguidamente, o endereço final, o qual deve ser inferior a 65536 (menor do que 32768 se se tratar de um *Spectrum* de 16 K). O programa testa então a memória entre os dois endereços.

Este programa foi escrito para ser utilizado sem o *Microdrive* da *Sinclair;* se algum estiver ligado ao computador, o ponto de partida (24500) terá de ser incrementado, tendo em conta a área de mapa do *Microdrive* na RAM.

Além de testar o *hardware* do *Spectrum*, este programa também serve para ver se o *Spectrum* está a ser afectado por qualquer interferência eléctrica. Neste caso, será aconselhável

regulá-lo de modo a testar apenas uma pequena área da RAM, 200 *bytes* por exemplo, para que a resposta apareça dentro de um espaço de tempo aceitavelmente curto.

```
  50 REM "TESTE A RAM"
 100 CLEAR 24500: LET s=24500: R
ANDOMIZE : LET r=INT (1000*RND)
 110 PRINT "Desde o endereco"
 120 INPUT a: IF a<s OR a>65535
THEN GO TO 120
 130 PRINT a;TAB 7;"ao endereco
";
 140 INPUT b: IF b<=a OR b>65535
 THEN GO TO 140
 150 PRINT b'
 200 RANDOMIZE r
 210 FOR c=a TO b: POKE c,INT (2
56*RND): NEXT c
 220 RANDOMIZE r: LET e=0
 230 FOR c=a TO b: IF PEEK c<>IN
T (256*RND) THEN PRINT '"Erro em
      ";c: LET e=e+1
 240 NEXT c
 250 IF e>0 THEN PRINT 'e;" erro
s neste teste"'
 260 IF e=0 THEN PRINT " OK ";
 270 LET r=INT (1000*RND)
 280 GO TO 200
```

100 Desloca a RAMTOP para baixo, de modo a que a RAM acima de 24500 fique livre para ser testada e, seguidamente, selecciona um ponto de partida aleatório r.

110-150 Obtém do utilizador os endereços inicial e final.

200 Determina o ponto de partida da sequência de números aleatórios.

210 Preenche as células da RAM a testar com números inteiros aleatórios.

220 Reajusta o ponto de partida da sequência de números aleatórios.

230-240 Verificam se todas as células a testar ainda contêm os valores originais.

250-260 Imprimem o resultado de um teste da área.

270-280 Escolhem um novo ponto de partida para a sequência de números aleatórios seguinte e depois voltam atrás para executar outro teste.

Note-se que o programa é bastante lento: leva cerca de um minuto para testar cada grupo de 1 K *bytes* da RAM.

TESTE À ROM

Este programa, que leva cerca de três minutos a executar, examina o conteúdo da ROM do *Spectrum* adicionando o conteúdo de cada *byte*, de modo a obter um valor total geral.

Nos dois computadores do autor, o valor encontrado foi 1926175. Se se obtiver qualquer outro valor, existem duas possibilidades:

1) há uma avaria qualquer na ROM;
2) a Sinclair introduziu uma ROM actualizada.

Se se obtiver de facto uma resposta diferente, é preferível confirmá-la com qualquer pessoa que disponha de um *Spectrum;* se o valor obtido com esse outro *Spectrum* for igual ao obtido com o primeiro, agradecemos que nos comuniquem esse facto, para que possamos incluir esse valor em futuras edições deste livro.

Este programa deve ser executado com o *Microdrive* e quaisquer outros periféricos retirados.

```
1ØØ LET a=Ø
11Ø FOR b=Ø TO 16383: LET a=a+PEEK b: NEXT b
12Ø PRINT a
```

CATÁLOGO

Talvez o leitor ainda não se tenha apercebido disso, mas a forma

de descobrir o que foi gravado numa *cassette* é executar o comando directo

CLS : VERIFY "?"

(Presume-se que não exista na *cassette* um ficheiro de nome "?")

Carregue na tecla BREAK quando estiver satisfeito; pode então carregar em COPY se quiser um registo permanente.

Experimentando isto com a *cassette Horizon* da Sinclair, obtém-se:

```
Program: ladob
Bytes: timex
Bytes:
Bytes: timex
Program: parede
Bytes: parede-g
Bytes: c
Program: ordenacao
Bytes: mcode
Bytes: char
Program: evolucao
Bytes: mcode
Bytes: bits
Program: vida
Bytes: p
Bytes: p
Program: desenho
Bytes: c
Program: montecarlo
Bytes: p
Program: caracter
Bytes:
Program: ondas
Bytes: m
```

RENUMERAÇÃO

Este programa "renumera" algumas ou a totalidade das linhas de qualquer programa escrito em BASIC com a excepção dele próprio.

Devemos passar o programa para *cassete* com um nome adequado, tal como RENUMERAR e, querendo utilizá-lo,

carregamos primeiro em memória o programa a ser renumerado, executando em seguida

MERGE "Renumerar"

Quando o programa renumerador tiver sido carregado em memória, executa-se

GOTO 9990

O programa pedirá então os primitivos números de linha inicial e final, da parte do programa que queremos renumerar; se quisermos renumerar todo o programa introduzimos os números 0 e 9989. O programa pede também o número da nova primeira linha e o novo valor do passo, ou seja, a diferença entre números de linha consecutivos (da nova versão, neste caso). Note-se que o programa a ser renumerado não deve incluir quaisquer linhas cujos números sejam superiores a 9989.

O programa não altera os números que se seguem às palavras de comando GO TO, GO SUB ou RESTORE; para fazer isso, resultaria num programa bastante complexo, que levaria um tempo demasiadamente longo a ser executado em BASIC.

```
9990 REM "RENUMERAR"
9991 INPUT "Nums primi.: inicial
 ";rs;TAB 13;"final ••• ";re'"Novo
s nums : inicial•";rn;TAB 13;"pa
sso ••• ";ri
9992 LET rp=PEEK 23635+256*PEEK
23636
9993 LET rv=PEEK 23627+256*PEEK
23628
9994 LET rl=256*PEEK rp+PEEK (rp
+1)
9995 IF rp>=rv OR rl>re THEN STO
P
9996 IF rl>=rs THEN POKE rp,INT
(rn/256): POKE rp+1,rn-256*INT (
rn/256): LET rn=rn+ri
9997 LET rp=rp+PEEK (rp+2)+256*P
EEK (rp+3)+4: GO TO 9994
```

9991 Obtém do utilizador os limites dos números de linha.

9992-9993 Atribuem "rp" ao endereço na RAM do início da área

do programa, e "rv" ao endereço do fim da área do programa.

9994-9997 Ciclo principal, executado uma vez para cada linha de programa.

9994 "rl" número de linha primitivo.

9995 Termina a execução, se tiver sido atingido o fim da área de programa ou se o número de linha primitivo for maior do que o limite estabelecido.

9996 Altera o número de linha, se o número primitivo estiver dentro do limite estabelecido.

9997 Desloca o indicador "rp" para o endereço do início da linha seguinte do programa e volta depois à linha 9994.

BASE PARA BASE

Lee Gentry

Este útil programa converte um número escrito numa base no seu equivalente noutra base. Pode lidar com qualquer base até à base 36.

Dígitos com um valor superior a 9 são representados pelas letras do alfabeto (maiúsculas ou minúsculas, o programa não as distingue) e, deste modo, os doze dígitos de um sistema de base 12 seriam:

Ø 1 2 3 4 5 6 7 8 9 A B

As linhas 20-30 definem o símbolo gráfico definível pelo utilizador CHR$ 155 de modo a torná-lo no símbolo de equivalência

```
Da base : 2      Para a base: 10

11001001     =    201
10 REM "CONVERSAO ENTRE BASES
        DIFERENTES"
20 FOR a=0 TO 7: READ d: POKE
USR CHR$ 155+a,d: NEXT a
  30 DATA 0,126,0,126,0,126,0,0
 100 INPUT "De que base ?   ";a
,"Para que base ? ";b
 110 IF a=0 AND b=0 THEN STOP
 120 IF a<2 OR b<2 THEN GO TO 10
0
 130 PRINT "Da base : ";a,"Para
a base: ";b
 140 INPUT "Qual o numero a conv
erter ?" LINE a$
 200 REM Da base a p/ a base 10
 210 LET s=0
 220 FOR c=LEN a$ TO 1 STEP -1
 230 LET d=CODE a$(c)-48-(7 AND
a$(c)>="A")-(32 AND a$(c)>="a")
 240 LET s=s+d*a↑(LEN a$-c)
 250 NEXT c
 300 REM Da base 10 p/ a base b
 310 LET b$="": PRINT
 320 LET r=INT (s-b*INT (s/b)+.5
): LET s=INT (s/b)
 330 LET b$=CHR$ (r+48+(7 AND r>
9))+b$
 340 IF s<>0 THEN GO TO 320
 350 PRINT a$;TAB 12;CHR$ 155;TA
B 16;b$;TAB 9
 370 GO TO 100
```

ZÍRCULOS

A forma da declaração DRAW, que traça parte de um círculo, fá-lo calculando as coordenadas de várias centenas de pontos sobre o arco e, seguidamente, ligando os pontos por segmentos de recta.

Geralmente não se dá por isso, mas, atribuindo-se ao terceiro número a seguir à palavra de comando DRAW um valor excessivamente grande, o resultado pode ser espectacular.

Para o experimentar, utilizaremos a rotina:

```
1Ø INPUT a: PRINT a
2Ø PLOT 127,1Ø: DRAW Ø,15Ø,a*PI
```

ou mudaremos a linha 20 para:

```
2Ø PLOT 127,1Ø: DRAW OVER 1;Ø,15Ø,a*PI
```

Alguns valores interessantes a experimentar: 119, 251, 253, 261, 383, 3333, 99999 e 99999999.

Nota: Não introduzir números pares.

−65536 não é −65536

Uma razão pela qual a Sinclair pode algum dia vir a actualizar a ROM é a existência de um *bug* (parasita, defeito) na versão actual. De facto, ela confunde, por vezes, os números −65536 e −1E−38. Por exemplo:

```
PRINT −65534−2      dá      −1E−38
```

tal como acontece com qualquer combinação de números inteiros negativos cuja soma algébrica seja igual a −65536. Isto tem outras consequências; por exemplo:

```
PRINT INT −65536      dá      −1
```

e o leitor deve estar interessado em saber o que faz a seguinte rotina:

```
FOR a=−6553Ø TO −6554Ø STEP−1: PRINT a:
NEXT a
```

SCREEN$ DISPARATADO

Outro *bug* da ROM está ligado à função SCREEN$.

Esta função dá, como resultado, o carácter que se encontra na intersecção da linha y com a coluna x do *écran* (mesmo que seja um espaço), tal como deveria, mas pode lançar a desordem numa expressão de que faça parte. Por exemplo, a rotina:


```
1Ø PRINT AT Ø,Ø,"a"
2Ø PRINT SCREEN$ (Ø,Ø)
3Ø PRINT "b"+ SCREEN$ (Ø,Ø)
4Ø PRINT CODE SCREEN$ (Ø,Ø)
5Ø PRINT 2+ CODE SCREEN$ (Ø,Ø)
6Ø IF CODE "a"=CODE SCREEN$ (Ø,Ø) THEN PRINT
"OK"
```

deveria imprimir	mas, na realidade, dá
a	a
a	a
ba	aa
97	97
99	97
OK	

O problema só parece ocorrer se o *Spectrum* tiver de executar outra parte da expressão antes de chegar a SCREEN$ e, assim, se a linha 50 da rotina anterior tivesse sido escrita da forma:

```
PRINT CODE SCREEN$ (Ø,Ø)+2
```

esta teria funcionado do modo esperado.

Mas, para jogar pelo seguro, provavelmente é melhor atribuir o valor-argumento de SCREEN$ a uma variável antes de o processar, como por exemplo:

```
LET x$=SCREEN$( )
```

STR$ ESTRANHA

Existe também um problema associado à função STR$, mas só se manifesta sob certas condições.

Em termos gerais, a expressão:

```
a$ + STR$ b
```

"esquecer-se-á" de a$ se b estiver compreendido entre −Ø.999999 e + Ø.999999 (excepto se for zero). Verifica-se isto, introduzindo:

```
PRINT "A"+STR$ (Ø.1)+"B"
```

expressão esta que deveria imprimir

```
AØ.1B
```

mas que, na realidade, dá

```
Ø.1B
```

Por isso, é melhor evitar a utilização de STR$ depois de "+" numa expressão de cadeia, a não ser que haja a certeza de que o argumento da função nunca tomará um valor fraccionário inferior a um.

Apêndice 1

Locais para PEEK e POKE

Para a maior parte dos programas em BASIC, não temos verdadeiramente necessidade de conhecer o que o processador *Z80* do *Spectrum* está a fazer. Podemos, por exemplo, usar uma variável sem nos preocuparmos com a forma como ela é armazenada em memória, e onde. E podemos utilizar uma declaração PRINT com toda a facilidade sem precisarmos de saber quais as células do ficheiro de imagem e da área dos atributos da RAM que serão afectadas. Todos estes pormenores maçadores e confusos podem ser deixados a cargo do *Z80* e do programa interpretador de BASIC do *Spectrum*, residente na ROM.

Mas, ocasionalmente, seria bom saber com mais minúcia o que se está a passar e talvez mesmo intervir mais directamente do que é possível com os comandos normais de BASIC. PEEK e POKE providenciam a chave para isso, permitindo-nos examinar e até alterar o conteúdo de células de memórias individuais.

Na prática, contudo, não podemos ir tão longe, pois a quase totalidade dos pormenores de operação do *Spectrum* está fechada nas rotinas inalteráveis da ROM. Existem, contudo, alguns artifícios de que nos poderemos servir, alguns "furos" que um programador engenhoso pode utilizar, e os quais são listados neste apêndice.

Lidaremos, em geral, com o conteúdo de *bytes* de memória individuais, os quais podem ser representados pelos números decimais Ø a 255 e ter acesso por uma única declaração PEEK ou POKE. Por vezes, contudo, acontece debruçarmo-nos sobre um par de células que são utilizadas para armazenar um número binário de 16 *bits* (2 *bytes*). O *Spectrum* coloca o *byte* menos significativo em primeiro lugar, de modo a ler o conteúdo combinado das células n e n+1:

```
LET v=PEEK n+256*PEEK (n+1)
```

e atribuir-lhes um novo valor:

```
POKE n,v −256*INT (v/256): POKE n+1, INT (v/256)
```

VARIÁVEIS DE SISTEMA

23552 - 23559 8 bytes KSTATE

Utilizada pelas rotina da ROM do *Spectrum* quando o teclado é "lido"; uma rotina como:

```
1Ø FOR a=Ø TO 7: LET b=a+23552
2Ø PRINT AT a,Ø;b;"  ";PEEK b,
3Ø NEXT a
4Ø GO TO 1Ø
```

pode ser usada para mostrar o que acontece.

Os 8 *bytes* são divididos em dois grupos de 4, de forma a tomar em consideração o "derrube" ou "cadência" de operação de duas teclas, onde uma segunda tecla é pressionada antes da primeira ser largada *(two-key rollover)*. As células 23556 a 23559 constituem a metade principal:

23556 conterá o número 255 se nenhuma tecla for accionada, ou se o outro grupo de quatro células contiver informação acerca do accionamento corrente de qualquer tecla. Enquanto uma tecla isolada for pressionada, a célula 23556 conterá o código para essa tecla — mas sem tomar em consideração a operação das teclas CAPS SHIFT ou SYMBOL SHIFT.

A célula 23559 memoriza o código da **última tecla cujo accionamento tenha afectado este grupo de quatro** *bytes*. Desta vez, o código é acertado, se necessário, tendo em conta as teclas CAPS SHIFT e SYMBOL SHIFT.

A célula 23558 é usada como temporizador, efectuando uma contagem decrescente, a partir do valor existente em REPDEL (variável de sistema com o endereço 23561), até chegar a zero e, seguidamente, a partir do valor existente em REPPER (variável de sistema com o endereço 23562), para dar a função de auto-repetição (da operação de uma tecla).

A célula 23557 é também um temporizador, utilizado para a rotina de "amortecimento" *(de-bouncing routine)*.

23560 1 byte LASTK

Guarda o código da última tecla accionada. Introduza:

```
1Ø PAUSE Ø: PRINT PEEK 23560: GO TO 1Ø
```

Tal como INKEY$, esta variável de sistema não distingue palavras de comando em modo *Extended (Extended Mode keywords* — palavras de comando impressas a verde sobre as teclas ou a vermelho por baixo delas), ou em "modo" *Gráfico (Graphics Mode)*, ou em maiúsculas, fixadas pela função CAPS LOCK, dos outros modos do teclado. Dá os seguintes valores:

4	ao	carregar	em	CAPS	SHIFT	e	TRUE VIDEO
5	"	"	"	"	"	"	IN VIDEO
6	"	"	"	"	"	"	CAPS LOCK
7	"	"	"	"	"	"	EDIT
8	"	"	"	"	"	"	< (tecla "5")
9	"	"	"	"	"	"	(tecla "8")
10	"	"	"	"	"	"	(tecla "6")
11	"	"	"	"	"	"	(tecla "7")
12	"	"	"	"	"	"	DELETE
13	"	"	"	"	"	"	ENTER
14	"	"	"	"	"	"	SYMBOL SHIFT
15	"	"	"	"	"	"	GRAPHICS

Ao contrário de INKEY$, esta variável de sistema não examina, na realidade, o teclado, mas apenas diz ao utilizador qual a última tecla accionada.

23561 1 byte REPDEL

Tempo durante o qual uma tecla deve permanecer pressionada antes de repetir a função a ela associada, tempo este medido em múltiplos de 1/50 de segundo (1/60 de segundo nos E.U.A.). A esta variável atribui-se o valor de 35 quando se liga o computador ou quando se faz um NEW. Este valor pode ser alterado por POKE e, assim, se introduzirmos, por meio de POKE, o valor "1" na célula desta variável, obtemos o tempo mais curto possível, enquanto que o valor de "Ø" dá uma espera de cerca de 5 segundos.

23562 1 byte REPPER

Tempo de espera, em 1/50 (1/60) de segundo, entre repetições sucessivas da função de uma tecla que se mantém pressionada. Esta variável é inicializada em "5" quando se liga o computador ou quando se faz um NEW. Como no caso anterior, atribuindo-se-lhe o valor "1" por meio de POKE obtém-se um tempo de espera mais curto; mas Ø ou 255 dá um tempo de tal forma longo que inutiliza praticamente a função de auto-repetição.

```
POKE 23511,1: POKE 23562,1
```

torna o teclado quase inutilizável!

23563,4 2 bytes DEFADD

Guarda o endereço na RAM do parêntese esquerdo de uma declaração DEF FN acrescido de 1 quando essa função está a ser usada; caso contrário, possui o valor zero. Por exemplo:

```
1Ø DEF FN a(x)=PEEK 23563+256*PEEK 23564
2Ø PRINT FN a(Ø) ,CHR$ PEEK FN a(Ø)
3Ø PRINT PEEK 23563+256*PEEK 23564
```

irá imprimir qualquer coisa como:

```
23762     x
Ø
```

23606,7 2 bytes CHARS

Indicador dos padrões de pontos definidores dos caracteres 32 (espaço) a 127 (©). O endereço inicial do padrão de pontos de um determinado carácter é dado por:

```
PRINT PEEK 236Ø6+256*PEEK 236Ø7
```

O endereço na CHARS é normalmente inicializado em 15360 por NEW ou ao ligar o computador, sendo o endereço do primeiro padrão de pontos na ROM igual a 15360+32*8=15616.

Podem colocar-se na RAM os padrões de pontos de um conjunto de caracteres completamente novo, e fazer POKE 23606/7 para os utilizar. A função SCREEN$ só reconhecerá normalmente caracteres situados entre SPACE e c, ou seja, os que pertencem à tabela de padrões de pontos da ROM (note-se que os caracteres gráficos 128 a 143 não aparecem na tabela de padrões de pontos da ROM).

Se se quiser que ela identifique caracteres gráficos definidos pelo utilizador pode redefinir-se a variável CHARS:

```
POKE 236Ø6,PEEK 23675: POKE 236Ø7,PEEK 23676-1
LET c=CODE SCREEN$ (y,x)+112
POKE  236Ø6,Ø: POKE 236Ø7 ,6Ø
```

rotina esta que irá igualar c ao código do carácter gráfico definível pelo utilizador que se situe em x,y, no *écran*.

23608 1 byte RASP

Define a duração, em 1/50 de segundo (1/60 nos E. U. A.), do sinal de aviso sonoro. A variável é inicializada em 64 quando se liga o computador e não é afectada por NEW.

23609 1 byte PIP

Define a duração do *click* que se ouve quando pressionamos uma tecla. Inicializada em Ø quando se liga o computador, esta variável também não é afectada por NEW.

O leitor pode preferir o *click* mais audível que é obtido atribuindo, por meio de POKE, um valor de cerca de 5 a esta variável: qualquer valor superior a cerca de 20 produzirá um som do género *beep*.

23610 1 byte ERR N

Código das mensagens diminuído de 1. Tem geralmente o valor −1, o qual é lido por um PEEK como 255. Introduzir qualquer outro valor por meio de POKE dá origem a que o código de erro e a mensagem adequada sejam impressos quando o programa parar.

23618,9 2 bytes NEWPPC

Um GO TO ou um GO SUB atribui a esta variável o número da

linha (sob forma binária de 2 *bytes*) para onde o programa deve saltar.

23620 1 byte NSPPC
Contém o número de declaração na linha para onde o programa deve saltar. Tem, normalmente, o valor 255. Se qualquer número entre Ø e 127 for introduzido por POKE na célula desta variável, haverá um salto forçado para a declaração com esse mesmo número que faça parte da linha cujo número se encontra em NEWPPC. Por exemplo:

```
1Ø POKE 23618,1ØØ: POKE 23619,Ø
2Ø POKE 2362Ø,2: PRINT "2Ø"
3Ø PRINT "3Ø"
1ØØ PRINT "1ØØ.1": PRINT "1ØØ.2": PRINT "1ØØ.3"
```

imprime

```
1ØØ.2
1ØØ.3
```

Esta deve ser a única forma de saltar para uma declaração situada no meio de uma linha!

23621,2 2 bytes PPC
Contém o número de linha da declaração que está a ser executada (sob forma binária de 2 *bytes*). Fazer um POKE a esta variável afectará qualquer mensagem de erro que possa ser causada por essa linha.

Uma declaração GO SUB guarda o valor de PPC como sendo o número da linha para a qual o programa deve voltar quando encontra o RETURN.

Assim,

```
1Ø POKE 23621,1Ø: GOSUB 1ØØØ
```

provoca o retorno à linha 10, em vez do retorno à linha imediatamente a seguir à linha 100.

23623 1 byte SUBPPC
Contém o número, na linha, de uma declaração que esteja a ser

executada. Tal como com PPC, o valor desta variável é utilizado pela rotina de mensagens de erro e é igualmente guardado para o retorno de qualquer GO SUB.

23624 1 byte BORDCR
Contém o número definidor da cor da margem (0-7) multiplicador por 8, além dos atributos normalmente utilizados para a metade inferior do *écran*. Fazer um POKE nesta variável não produz qualquer efeito directo, a não ser que o POKE seja seguido de um CLS, o que define os atributos relativos à metade inferior do *écran*, ou por uma declaração INPUT, a qual primeiramente define os atributos da metade inferior do *écran* e, seguidamente, quando a tecla ENTER tiver sido accionada, altera a margem.

23627,8 2 bytes VARS
Contém o endereço do início da área de variáveis da RAM. Como esta se segue imediatamente ao fim do programa, pode-se descobrir quantos *bytes* o programa está a ocupar, introduzindo:

```
PRINT PEEK 23627+256*PEEK 23628-PEEK
23635-256* PEEK 23636
```

23629,23630 2 bytes DEST
Contém o endereço na RAM da variável de destino (a que se encontra à esquerda do sinal =) numa declaração LET. Assim,

```
LET a=PEEK 23629+256*PEEK 2363Ø
```

iguala "a" ao endereço da célula onde esta mesma variável se encontra armazenada. Se a variável "a" ainda não tiver sido utilizada anteriormente, DEST aponta para o carácter "a" na linha de programa.

Assim, pode utilizar-se DEST para descobrir a célula de memória que esteja a ser usada para conter uma determinada variável numérica.

23635,6 2 bytes PROG
Contém o endereço do início do programa escrito em BASIC; veja VARS (23627). Não executar POKE.

23653,4 2 bytes STKEND

Contém o endereço do início da área de memória (RAM) livre.

A RAM entre este endereço e o endereço dado em RAMTOP (23730/1) é utilizada apenas pelas pilhas *(stacks)* da máquina e de GO SUB, e, portanto, consegue-se uma boa ideia do número de *bytes* ainda livres da RAM fazendo:

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

Não executar POKE.

23658 1 byte FLAGS 2

Esta variável é utilizada para diversos *flags* de sistema, mas contém geralmente o valor 0, a não ser que se tenha fixado CAPS LOCK, pois, nesse caso, a variável conterá o valor 8. Atribuindo-lhe, por POKE, o valor 8, fixa-se CAPS LOCK.

23659 1 byte DF.SZ

Número de linhas (incluindo uma em branco) na parte inferior do *écran*. Inicializa-se em 2 quando se liga o computador e sempre que se executa um comando directo ou um INPUT.

Se um número n (maior do que 2 e menor do que 25) for introduzido por POKE nesta célula, o utilizador fica impossibilitado de imprimir (PRINT) nas últimas n linhas do *écran*, e essas linhas serão apagadas pela execução de um INPUT ou por uma mensagem de erro.

Introduzindo, com POKE, o valor 1, permite-se ao utilizador imprimir em 23 linhas:

```
10 POKE 23659,1
20 FOR a=0 TO 22: PRINT AT a,0;a: NEXT a
30 PAUSE 100
40 POKE 23659,2
```

Repare-se que é necessário voltar a atribuir, através de POKE, o valor 2 a esta variável antes de o programa parar ou executar um INPUT ou um CLS.

Atribuindo-lhe o valor 0, o utilizador fica com a possibilidade de imprimir (PRINT) em todas as 24 linhas do *écran*, embora

não possa usar AT para imprimir na vigésima quarta linha. Para o conseguir, terá de fazer qualquer coisa como:

```
PRINT AT 22,31;TAB 10; "linha 24"
```

O utilizador deve voltar a atribuir (através de POKE) a esta variável o valor 2 inicial antes de qualquer declaração INPUT ou CLS ou do fim do programa, pois, de contrário, o programa irá "estoirar" *(crash)* completamente. Isto pode-se utilizar como base para impedir que qualquer pessoa copie ou liste um determinado programa.

23670,1 2 bytes SEED

Esta variável de sistema é a "semente" *(seed)* para o gerador de números aleatórios. É inicializada em zero quando se liga o computador ou quando se executa um NEW. É igualada ao argumento de RANDOMIZE, se esse argumento não for zero, ou ao conteúdo da variável de sistema FRAMES, por meio de RANDOMIZE ou RANDOMIZE 0.

23672,3,4 3 bytes FRAMES

Esta variável conta o número de imagens de TV *(frames)* e é incrementada a intervalos de 20 ms (ou seja, em cada 1/50 de segundo — 1/60 nos E. U. A.).

O capítulo 18 do manual da Sinclair fornece um conjunto de funções destinadas a obter o valor total dos três *bytes* de FRAMES mas, infelizmente, a versão apresentada na primeira edição do livro está errada, pois a linha 30 deveria ter a redacção seguinte:

```
30 DEF FN t()=FN m(FN u(), FN u())
```

Se quisermos atribuir a FRAMES um determinado valor (geralmente zero), deveremos introduzir, através de POKE, o *byte* menos significativo em primeiro lugar. Por exemplo:

```
POKE 23672,0: POKE 23673,0: POKE 23674,0
```

23675,6 2 bytes UDG

Contém o endereço do primeiro padrão de pontos dos gráficos

definíveis pelo utilizador, ou seja, o relativo a CHR$ 144. É inicializada, ao ligar o computador, no valor de P-RAMT (variável de sistema 23732/3) subtraído de 167. Não é afectada por NEW.

Necessitando desesperadamente de espaço de memória, e não precisando de todos os 21 caracteres gráficos definíveis pelo utilizador, pode atribuir-se a UDG um valor mais alto (por meio de POKE, claro), lembrando contudo de fixar RAMTOP num valor inferior em uma unidade ao atribuído a UDG, com uma declaração CLEAR. Por exemplo, para eliminar completamente o espaço dos gráficos definíveis pelo utilizador, executa-se o seguinte:

```
POKE 23675,255
POKE 23676,255 (ou 127 para um Spectrum de 16 K)
CLEAR 65534 (ou 32766 para um Spectrum de 16 K)
```

Vejam-se também as notas relativas à variável de sistema CHARS (23606/7)

23677 1 byte X-COORD
23678 1 byte Y-COORD

Estas células contêm as coordenadas x e y do último ponto traçado. São utilizadas como ponto de partida para os comandos DRAW e CIRCLE e, portanto, a execução de um POKE em qualquer destas células afectaria a operação DRAW ou CIRCLE seguinte.

23679 1 byte P. POSN

Contém o número 33 diminuído do número da coluna na qual aparecerá o próximo carácter a imprimir pela impressora (LPRINTed). Por exemplo, se a célula contiver o número 33, o próximo carácter será impresso na coluna Ø; se ela contiver o valor 2, o carácter seguinte será impresso no fim da linha. O valor 1 significa que o trigésimo segundo carácter na linha em questão já foi impresso, mas a posição de impressão ainda não avançou para a linha seguinte. Executar um POKE nesta célula não tem, por si só, qualquer efeito.

23680 1 byte PR. CC

Diz o manual do *Spectrum* que esta variável representa o *byte* menos significativo do endereço da posição seguinte no *buffer* da impressora ZX. É zero quando o *buffer* está vazio, e é incrementado em um por cada carácter impresso (LPRINTed) numa linha.

Atribuir, através de POKE, um novo valor a esta variável, afecta a posição do próximo carácter a imprimir (LPRINTed), mas deverá também atribuir-se um valor condizente a P POSN (23679), se não o fim da linha não será identificado correctamente.

23688 1 byte S. POSN

Contém 33 menos o número de coluna da posição de impressão no *écran* e assemelha-se a P POSN (23679). Executar um POKE nesta célula também não tem, por si só, qualquer efeito.

23689 1 byte

Contém o número 24, diminuído do número de linha da posição de impressão sobre o *écran*. Executar um POKE nesta célula afecta a posição do elemento impresso após a declaração PRINT seguinte ou o símbolo de mudança de linha (')!

Juntamente com a variável de sistema 23688, pode usar-se esta célula para determinar a posição em que, no *écran*, se encontra a posição de impressão corrente.

23692 1 byte SCRCT

Inicializada para conter o valor 1 por NEW, RUN e quando se liga o computador. Contém o valor 1 adicionado ao número de linhas a serem roladas *(scrolled)* antes da imagem parar com o inquisitivo "scroll?".

É-lhe então atribuído o valor 22 se qualquer tecla for pressionada, exceptuando N e BREAK.

Introduzir com POKE um valor adequado nesta célula permite que a imagem no *écran* role sem que o computador peça autorização. Para obter o número máximo destes "rolamentos automáticos", utiliza-se:

```
POKE 23692,0
```

23693 1 byte ATTR P

Contém os atributos permanentes definidos por declarações COLOUR, FLASH, etc. É inicializada em 56 por NEW e quando se liga o computador. Define-se de imediato a totalidade dos atributos permanentes, introduzindo com POKE um valor adequado nesta célula; ou encontrar-se-á uma aplicação para PEEK 23693, para, por exemplo, descobrir quais são os valores actuais dos atributos permanentes.

23694 1 byte MASK P

Empregada para conter os atributos "transparentes" permanentes; qualquer *bit* que seja um "1" faz o *bit* de atributo correspondente igualar o que já se encontra no *écran* naquela posição, em vez do que se encontra em ATTR P. Esta variável é inicializada em zero por NEW e quando se liga o computador.

23730,1 2 bytes RAMTOP

Contém o endereço do último *byte* da área de sistema BASIC da RAM, o qual é inferior em uma unidade ao valor do endereço do primeiro *byte* da área UDG, dos caracteres gráficos definíveis pelo utilizador.

O conteúdo desta célula é inicializado em 65367 (para um *Spectrum* com 48 K de RAM) ou em 32599 (16 K) quando se liga o computador. Não é afectado por NEW.

Executar um POKE nesta célula não tem qualquer efeito real; RAMTOP só deve ser alterada por meio de uma declaração CLEAR n.

23732,3 2 bytes P-RAMT

Contém o endereço do último *byte* de memória RAM do sistema; inicializada, quando se liga o computador, para conter 65535 (RAM de 48 K) ou 32767 (16 K). Introduzir, por meio de POKE, um outro valor nesta célula não produz qualquer efeito.

Esta variável é utilizada a maior parte das vezes para dizer ao programa se está a ser executado num *Spectrum* de 48 K ou se o está a ser num de 16 K:

```
IF PEEK 23733=255 THEN PRINT "48 K"
```


FICHEIRO DE IMAGEM

16384-22527

Não há verdadeiramente grande vantagem em executar operações de PEEK e POKE na área do ficheiro de imagem da RAM, pois os comandos PLOT, PRINT, POINT e SCREEN$ fazem quase sempre o trabalho pretendido com muito maior facilidade. Mesmo assim, oferecemos a explicação somente para aqueles que estão determinados a fazer a experiência.

Cada posição de carácter no *écran* corresponde a oito *bytes* da RAM, estabelecendo cada *byte* uma fila de oito pontos. O endereço, na RAM, do primeiro *byte* (de topo) de uma posição de carácter é dado por:

```
16348+32*(y+56*INT(y/8))+x
```

onde x e y são os mesmos que seriam utilizados numa declaração PRINT AT y,x.

Os outros sete *bytes* para a posição de carácter estão espalhados na RAM com intervalos de 256 *bytes* entre eles; ou seja, o endereço do *byte* correspondente à segunda fila de pontos num carácter é superior em 256 ao da fila de topo, e o endereço do último *byte* é superior em 7×256 ao primeiro.

Como demonstração, experimente-se:

```
  100 FOR y=0 TO 23: IF y<=21 THE
N PRINT AT y,15;y
  110 LET p=16384+32*(y+56*INT (y
/8))
  120 FOR x=0 TO 31 STEP 8
  130 FOR r=0 TO 7: POKE p+x+r+25
6*r,BIN 10101010: NEXT r
  140 NEXT x: NEXT y
  150 PAUSE 0
```

Isto mostra uma forma (embora bastante desajeitada) de "traçar" *(plot)* nas últimas linhas do *écran*.

ÁREA DE ATRIBUTOS

22528-23295

Esta área da RAM contém os *bytes* dos atributos para cada uma

das 24×32 posições de carácter do *écran*, sendo o endereço em memória do *byte* correspondente ao carácter situado na coluna y, linha x dado por:

22528+x+32*y

É, geralmente, mais fácil alterar o conteúdo da célula por meio de uma declaração

PRINT OVER 1;AT y,x; "•"

que inclua uma função PAPER, INK, BRIGHT ou FLASH temporária, e examinar o seu conteúdo com

ATTR (y,x)

Mas, apenas para mostrar que a execução de POKE nesta área funciona de facto, experimente-se a seguinte rotina, a qual altera a cor do fundo de todas as 24 linhas:

```
 100 FOR a=0 TO 21: PRINT AT a,1
5;a: NEXT a
 110 FOR p=0 TO 56 STEP 8
 120 FOR x=0 TO 31: FOR y=0 TO 2
3
 130 POKE 22528+x+32*y,p
 140 NEXT y: NEXT x: NEXT p
```

BUFFER DA IMPRESSORA

23296-23551

Esta área da RAM guarda uma imagem de pontos de uma fila de 32 caracteres destinados à impressora *ZX*, e é posta a zero (apagada) em todas as células quando se liga o computador, por um comando NEW e após a linha ter sido enviada para a impressora.

Os primeiros 32 *bytes* contêm a fila superior de pontos para todos os 32 caracteres; os 32 *bytes* seguintes contêm a segunda

fila de pontos, e assim por diante. Podemos demonstrar isto com a rotina:

```
 100 LPRINT "A"
 110 FOR a=23296 TO 23551
 120 IF PEEK a<>0 THEN PRINT a;T
AB 8;a-23296,PEEK a
 130 NEXT a
 140 LPRINT
```

a qual imprime:

```
23328   32   60  00111100
23360   64   66  01000010
23392   96   66  01000010
23424  128  126  01111110
23456  160   66  01000010
23488  192   66  01000010
```

(A versão binária à direita foi acrescentada posteriormente.)

ÁREA DE PROGRAMA

Esta é a área da RAM que começa no endereço contido na variável de sistema PROG (23635/6) e termina no endereço imediatamente anterior ao que está contido na variável de sistema VARS (23627/8). É, evidentemente, utilizada para armazenar o programa em BASIC do utilizador.

Podemos ver como o programa é realmente armazenado na RAM, por meio de uma rotina como a seguinte, a qual imprime três colunas:

— a primeira contém o endereço das células da RAM a serem examinadas;
— a segunda apresenta o conteúdo dessas células, expresso sob a forma de um número inteiro decimal;
— finalmente, a terceira apresenta o conteúdo das células expresso, sempre que possível, como um dos caracteres ou palavras de comando do *Spectrum*.

```
    5 REM abcd
  100 LET a=PEEK 23635+256*PEEK 2
3636
  110 PRINT a;"•";PEEK a;
  120 IF PEEK a>=32 THEN PRINT TA
B 10;CHR$ PEEK a;
  130 LET a=a+1: GO TO 110
```

A linha 5 pode conter qualquer tipo de declaração que o leitor deseje. Neste caso, escolhemos uma simples declaração REM e, com o auxílio do Capítulo 24 do manual da Sinclair, podemos interpretar o resultado da execução da rotina:

23755,6	Número de linha 5.
23757,8	6 *bytes* de comprimento.
23759	Palavra de comando REM.
23760-3	Caracteres a seguir a REM.
23764	Código de fim de linha.
23765,6	Número de linha 100.

```
23755  0
23756  5
23757  6
23758  0
23759  234  REM
23760  97   a
23761  98   b
23762  99   c
23763  100  d
23764  13
23765  0
23766  100  d
23767  39   '
23768  0
23769  241  LET
23770  97   a
23771  61   =
23772  190  PEEK
23773  50   2
23774  51   3
23775  54   6
```

Note-se que o *Spectrum* guarda os números de linha em memória sob forma binária de dois *bytes* mas, contrariamente à prática usual, coloca o *byte* mais significativo em primeiro lugar.

PADRÕES DE PONTOS PARA CARACTERES DA ROM

Estes padrões, guardados na ROM entre os endereços 15616 a 16383, são analisados em pormenor na Capítulo 5.

CANAIS

A área de informação de canal da RAM de um *Spectrum* que não esteja equipado com um *Microdrive* compreende quatro registos de cinco *bytes,* que ocupam as células 23734 a 23753.

Os dois primeiros *bytes* de cada registo contêm o endereço da rotina da ROM utilizada para imprimir dados; os dois *bytes* seguintes contêm quer o endereço da rotina da ROM usada para obter INPUT do teclado quer o endereço de uma rotina que se destina a dar a mensagem de erro "INVALID I/O DEV". O quinto *byte* de cada registo contém o código *ASCII (American Standard Code for Information Interchange)* dos caracteres K, S, R e P respectivamente.

As palavras PRINT, LPRINT e INPUT podem ser seguidas de um símbolo de cardinal (#), por exemplo: PRINT#n onde "n" é designado por "número de canal".

Os canais números Ø e 1 utilizam o primeiro registo da área de informação de canal da RAM para imprimir na metade inferior do *écran* e receber o *input* do teclado.

O canal número 2 utiliza o segundo registo e está relacionado com a impressão na metade superior do *écran*.

O canal número 3 utiliza o quarto registo e dá o *output* para a impressora *ZX*.

INPUT#2 ou INPUT#3 dá origem à mensagem de erro "INVALID I/O DEV".

assim, PRINT#3 pode ser usado em vez de LPRINT, e PRINT#Ø é uma forma de imprimir na parte inferior do *écran* normalmente reservada ao INPUT e mensagens de erro.

Apêndice 2

Velocidade

Se quisermos que o programa seja executado o mais rapidamente possível, é útil saber quanto tempo o *Spectrum* leva a executar cada tipo de operação. Descobrimo-lo utilizando uma rotina como a listada a seguir, a qual dá o tempo em ms (milissegundos: milésimos de segundo) gasto a executar quaisquer linhas, numeradas de 1000 a 1999, que sejam acrescentadas.

```
  10 POKE 23672,0: POKE 23673,0:
POKE 23674,0
  20 FOR i=1 TO 100: GO SUB 1000
  NEXT i
  30 LET t=PEEK 23672+256*PEEK 2
3673+65536*PEEK 23674
  40 PRINT AT 0,0;(t-39)/5
  50 STOP
2000 RETURN
```

Esta rotina, se a fizermos executar tal como se apresenta listada, imprimirá Ø como resultado; acrescentando uma linha como

```
1ØØØ LET a=.5
```

e, fazendo reexecutar a rotina, ela dirá que o *Spectrum* leva 1.8 ms para executar a linha acrescentada.

Se experimentarmos diversas versões da linha 1ØØØ, verificaremos que o *Spectrum* leva entre 2 e 4 ms para somar ou subtrair dois números, e 4 a 5 ms para dividir ou multiplicar, dependendo dos números empregados. O operador de potenciação (↑), contudo, leva o tempo incrível de 110 ms para calcular qualquer coisa como .5↑.5.

Os operadores lógicos (=,<>,<=,>=) consomem cerca de 3 a 4 ms, como se verifica utilizando uma linha como:

```
1ØØØ LET a=27<33
```


Expressões de cadeia simples, tais como:

```
LET a$="TIMEX"
```

ou

```
LET a$="I"+"BM"
```

consomem entre 3 ms e 6 ms, dependendo do comprimento das cadeias.

Tempos de execução típicos de várias funções:

SIN .5	44 ms	A$N .5	183 ms	TAN .5	89 ms
ATN .5	62 ms	COS .5	45 ms	ACS .5	185 ms
LN .5	71 ms	EXP .5	543 ms	SQR .5	111 ms
SGN .5	3 ms	INT .5	3 ms	ABS .5	3 ms
BIN 1Ø1Ø1Ø1Ø			3 ms	PI	3 ms
ATTR (Ø,Ø)	5 ms				
POINT (Ø,Ø)	5 ms	PEEK 1ØØØ	3 ms	IN 1ØØØ	3 ms
VAL ".5"	13 ms	STR$ .5	22 ms	CHR$ 32	6 ms
INKEY$	4 ms	CODE "a"	3 ms		
VAL "1234.56789"		41 ms			
SCREEN$ (Ø,Ø)		7 a 11 ms, dependendo do carácter			

E os tempos de execução de declarações de impressão e desenho típicas são as seguintes:

PRINT	2 ms	PRINT .5	17 ms
PRINT "A"	3 ms	PRINT "ABCDEFGHIJKL"	
	8 ms		
PRINT "A";	3 ms	PRINT "A",	10 ms
PRINT TAB Ø;"A"	4 ms	PRINT TAB 3Ø;"A"	19 ms
PRINT AT Ø,Ø;"A"	6 ms	PLOT 1ØØ,1ØØ	3 ms
PLOT 1ØØ,1ØØ: DRAW Ø,Ø		6 ms	
PLOT 1ØØ,1ØØ: DRAW 1ØØ,5Ø		25 ms	
PLOT 1ØØ,1ØØ: DRAW 1ØØ,5Ø,1		600 ms	
CIRCLE 1ØØ,1ØØ,5Ø		800 ms	

Comandos temporários INK, PAPER, FLASH, BRIGHT, OVER ou INVERSE incrementam o tempo de execução em cerca de 2 ms.

Quando o *Spectrum* executa uma declaração GO TO, GO SUB, RETURN, RESTORE ou NEXT, ou uma função FN definível pelo utilizador, tem de encontrar a linha do programa para onde deve saltar, pesquisando para isso toda a listagem do programa, desde o início.

O *Spectrum* leva cerca de 0.3 ms a passar de uma linha a outra e, portanto, um ciclo FOR-NEXT que começasse — por exemplo — na centésima linha de um programa, não poderia ser executado mais do que 30 vezes em cada segundo; deste modo, pode valer a pena colocar o ciclo mais utilizado logo no início da listagem.

Além deste tempo de "pesquisa", as declarações GO TO, GO SUB, RETURN e RESTORE levam, cada uma delas, cerca de 1 ms a 3 ms a ser executadas, e a declaração NEXT cerca de 4 ms.

O *Spectrum* tem igualmente de andar à procura de variáveis quando estas são utilizadas. Elas são armazenadas na área de variáveis da RAM, na ordem pela qual são encontradas quando se executa o programa. O tempo de ter acesso a uma variável é aumentado em cerca de 0.05 ms para cada variável que já tenha sido previamente encontrada (os quadros — *arrays* — contam como uma variável).

Apêndice 3

Outros BASICS

Pode-se encontrar uma enorme quantidade de programas escritos em BASIC em outros livros e em revistas de informática. Infelizmente, existem muitas versões de BASIC. Apesar de parecerem semelhantes, existem diferenças suficientes entre estas versões para que, muito provavelmente, um programa escrito para outro computador não funcione no *Spectrum* sem algumas modificações.

Para ajudar a descobrir as alterações necessárias, os parágrafos seguintes listam as principais áreas de diferença entre os dialectos mais comuns. Mas deve-se reparar que a amplitude e a natureza das alterações necessárias dependem da estrutura do programa a adaptar, da forma como este processa a informação e — muito principalmente — das características da imagem em *écran* do computador para o qual o programa foi escrito. Por isso, é necessário descobrir exactamente como funciona o programa, antes de se tentar fazê-lo executar no *Spectrum;* isso poupa tempo a médio e a longo prazo.

LINHAS DE DECLARAÇÕES MÚLTIPLAS

A maior parte das versões de BASIC permite inserir mais do que uma declaração numa linha, como sucede com o *Spectrum*. Contudo, algumas utilizam o símbolo "\" para separar as declarações, e a da Acorn Atom usa ";".

VARIÁVEIS

A maioria dos BASICs modernos utiliza aritmética e variáveis de vírgula flutuante, como o *Spectrum*, mas alguns também permitem ao utilizador especificar variáveis inteiras, geralmente acrescentando um % ao nome da variável. Assim,

A seria uma variável de vírgula flutuante

mas

A% seria uma variável inteira.

É "inteira" uma variável que só pode assumir valores inteiros (números inteiros). Estas variáveis são geralmente utilizadas por ocuparem menor espaço de memória que as variáveis de vírgula flutuante ou por as operações aritméticas que as utilizam serem, em geral, de execução mais rápida. Nestes casos, não há qualquer problema grave em utilizar as variáveis de vírgula flutuante do *Spectrum* no lugar das variáveis inteiras. Mas, por vezes, o programa baseia-se nas propriedades da aritmética de números inteiros, particularmente a divisão inteira, onde o resultado é arredondado ou truncado, se necessário, de forma a dar um número inteiro. Suspeitando-se de que é este o caso, empregue-se a função INT para "integrar" o resultado de qualquer divisão. Por exemplo:

```
1Ø LET C%=Y/1ØØ
```

tornar-se-ia

```
1Ø LET C=INT (Y/1ØØ)
```

Alguns BASICs, tal como o *Apple II Integer Basic* e o do antigo *ZX80* de 4 K, não têm a capacidade de operar com vírgula flutuante: todas as suas variáveis são inteiras, apesar de não utilizarem o % no final de cada nome de variável. O BASIC da Atom ainda provoca maior confusão, pois, nessa versão, variáveis da forma "A" são consideradas variáveis *inteiras*, enquanto que os nomes de variáveis de vírgula flutuante começam por %, p. ex.: %A. Também nestes casos se deve "desconfiar" das divisões.

Note-se que a maioria dos BASICs que permitem utilizar tanto letras maiúsculas como minúsculas fazem a distinção entre umas e outras, de forma que A seria uma variável diferente de a; contudo, isto não acontece com o *Spectrum*.

QUADROS (ARRAYS)

Não deve haver qualquer dificuldade relacionada com quadros numéricos, pois o *Spectrum* é, pelos menos, o equivalente de outros computadores nesta área. Mas a maior parte dos BASICs empregam quadros cujos índices começam em zero e não em um. Deste modo, DIM A(3) define geralmente um quadro de quatro elementos A(Ø) a A(3), enquanto o *Spectrum* interpretaria isso como a definição de um quadro de três elementos A(1) a A(3).

Algumas versões de BASIC permitem empregar nomes de variáveis indexadas (para quadros ou *arrays*) com vários caracteres. Como tal não acontece no caso do BASIC do *Spectrum*, esses nomes deverão ser reduzidos a nomes de uma única letra, se os quisermos empregar neste computador.

As faculdades relativas aos quadros de cadeias do *Spectrum* são também semelhantes às da maioria dos outros BASICs (exceptuando o facto de — também neste caso — na maior parte dos outros BASICs os índices começarem em Ø em vez de começarem em 1). Existem, contudo, diferenças na forma como eles lidam com o comprimento das cadeias num quadro. A declaração DIM, na maioria dos BASICs, define — além do número de cadeias de um quadro — o comprimento *máximo* de cada cadeia. Cadeias que sejam transferidas para o quadro mantêm o seu comprimento original, a não ser que sejam demasiadamente longas, gerando neste caso, normalmente, uma mensagem de erro. O *Spectrum*, por outro lado, acerta o comprimento de uma cadeia de modo a que ela se ajuste perfeitamente ao comprimentos DIMensionado para o quadro. Isto, por vezes, dá origem a problemas quando se comparam duas cadeias, sendo uma delas um elemento de um quadro de cadeias.

É natural encontrar uma declaração DIM parecida com a seguinte:

```
DIM A$(2,2)[1Ø]
```

Esta declaração define um quadro de 2×2 (ou 3×3) cadeias, tendo cada elemento um comprimento máximo de 10 caracteres. É equivalente à declaração do *Spectrum*

```
DIM A$(2,2,1Ø)
```

ou

```
DIM A$(3,3,1Ø)
```

Alguns BASICs exigem que se DIMensione cada variável de cadeia — mesmo aquelas que não façam parte do quadro — para que lhe seja reservado espaço de memória. Outras versões de BASIC apenas necessitam de declarações DIM para variáveis de cadeia que possam vir a ter um comprimento superior a — por exemplo — 10 caracteres. Em ambos os casos, encontram-se linhas de programa como a seguinte:

```
DIM B$(15)
```

linhas essas que são desnecessárias na versão do programa para o *Spectrum*.

Alguns BASICs permitem que mais do que um quadro seja DIMensionado por uma única declaração DIM, mas isso não é exequível no *Spectrum*. Assim,

```
1Ø DIM A(2,2),B(5Ø)
```

teria de ser alterado para:

```
1Ø DIM A(2,2): DIM B(5Ø)
```

FUNÇÕES ARITMÉTICAS

As funções aritméticas utilizadas pelo *Spectrum* são muito semelhantes às disponíveis em outros computadores, com a excepção de que algumas versões de BASIC utilizam o símbolo '^' ou '**' em vez do "↑" do *Spectrum*.

Alguns computadores também possuem as funções:

LOG: logarítmo de base 10;
LOG (X) é equivalente a LN (X) / LN (1Ø)

DIV: dá a parte inteira do resultado de uma divisão:

```
A DIV B
```


deve ser convertido para INT (A/B)

MOD: dá o resto de uma divisão inteira;

p. ex.: 21 MOD 5

dá como resultado 1. De um modo geral,

A MOD B

deve ser convertido em A-B*INT(A/B)

ASC, CODE

ASC (X$) permite obter o valor decimal do código do primeiro carácter em X$, e é equivalente a CODE.

Quando se efectua a conversão de um programa de um *ZX80* ou de um *ZX81*, deve notar-se que os códigos de carácter desses computadores são diferentes dos utilizados pelo *Spectrum*. Além disso, enquanto o *Spectrum* utiliza os códigos de carácter *ASCII standard* para letras, números e a maior parte dos símbolos usuais, os códigos de caracteres gráficos e de controle diferem entre os computadores.

END

Substituir por STOP, no *Spectrum*.

FOR-TO-STEP - - NEXT

Alguns BASICs permitem utilizar um nome de variável com mais de uma letra para a variável de controle de um ciclo FOR-NEXT. O BASIC do *Spectrum* não o admite.

Levanta-se por vezes um problema subtil devido ao facto de muitas versões de BASIC executarem sempre as declarações entre o FOR e o NEXT pelo menos uma vez, mesmo no caso de o valor inicial da variável de controle ser superior ao limite. Deste modo,

```
1Ø FOR I=1 TO Ø
2Ø PRINT "LINHA 2Ø"
3Ø NEXT I
```

fará com que "LINHA 2Ø" seja impresso por algumas versões de BASIC, mas não pela do *Spectrum*.

Alguns BASICs permitem omitir o nome da variável a seguir ao NEXT, presumindo tratar-se da variável empregada na declaração FOR mais recentemente utilizada. Contudo, isto não é permitido no BASIC do *Spectrum*.

GET, GET$, INKEY$

Em algumas versões de BASIC, uma declaração do tipo

```
GET A$ ou GET A ou LET A$=GET$
ou LET A=GET
```

fará o computador esperar que o utilizador pressione uma tecla e, feito isso, devolve como resultado uma cadeia de um único carácter (ou o código numérico) representando a tecla accionada.

A função INKEY$ do *Spectrum* é semelhante, mas não espera que uma tecla seja pressionada. Para que isso aconteça, é necessária uma pequena rotina:

```
1Ø LET A$=INKEY$: IF A$="" THEN GO TO 1Ø
```

ou

```
1Ø LET A=CODE INKEY$: IF A=Ø THEN GO TO 1Ø
```

Alguns computadores possuem a função INKEY$(X). Esta função espera que uma tecla seja pressionada, mas só durante o tempo x (x pode ser décimos ou centésimos de segundo, dependendo do computador). Pode ser substituída pela linha do *Spectrum:*

```
1Ø PAUSE X: LET A$=INKEY$
```

devido ao facto de o pressionamento de qualquer tecla terminar uma PAUSE (pausa).

GO TO, GO SUB

Alguns BASICs não permitem um GO TO ou GO SUB "computado", tal como o GO TO A*1ØØ do *Spectrum*. O leitor tem, portanto, a possibilidade de efectuar pequenas simplificações num programa tirando partido desta vantagem do *Spectrum*.

Um pequeno número de versões de BASIC permitem incluir um *label* ou "rótulo" no início de uma linha, imediatamente antes ou logo depois do número de linha. Este rótulo parece-se geralmente com um nome de variável, e pode ser seguido de um sinal de ponto e vírgula. O rótulo pode então ser usado depois de um GO TO ou GO SUB em vez do número da linha para a qual o programa deve saltar. Por exemplo:

```
1Ø GOTO A
----
1ØØ A;LET C=D
```

o que pode ser traduzido para:

```
1Ø GOTO 1ØØ
----
1ØØ LET C=D
```

Alguns BASICs têm formas "ON - - GOTO" e "ON - - GO--SUB". Estas fazem o programa (ou melhor: a sequência de execução do programa) saltar para uma de um certo número de linhas, de acordo com o valor de uma variável. Por exemplo:

```
ON I GOTO 1ØØ, 1Ø5, 13Ø
```

fará o programa saltar para a linha 1ØØ se I=1, para a linha 105 se I=2 ou para a linha 13Ø se I=3.

Dependendo do caso particular em questão, estas formas podem ser substituídas por um GO TO (ou GO SUB) computado ou por uma sequência de linhas IF - THEN - GO TO.

Ou então o leitor pode ser verdadeiramente engenhoso e utilizar o operador AND, de modo a que a versão para o *Spectrum* da linha acima seria:

```
GO TO (1ØØ AND I=1)+(1Ø5 AND I=2)+(13Ø AND
I=3)
```

IF - THEN - ELSE

Muitos BASICs não exigem que se inclua o THEN, de forma que se poderia ter

```
    IF A=B GOTO 1Ø
ou  IF A$="SIM" PRINT "NAO"
```

Note-se que as várias versões de BASIC diferem na interpretação que fazem acerca de um valor ser "verdadeiro" ou "falso". Por exemplo:

```
IF A THEN PRINT "VERDADEIRO"
```

imprime "VERDADEIRO" num *Spectrum*, se A tomar qualquer valor à excepção do zero, mas poderá não o fazer com outros BASICs, os quais poderiam — por exemplo — tomar valores negativos como "Falsos" e zero como "Verdadeiro".

Por vezes encontra-se ELSE. Constitui uma extensão extremamente útil à combinação IF-THEN, pois providencia o curso de acção alternativo ao que se segue ao THEN. Por exemplo:

```
IF A>B THEN PRINT "GRANDE A" ELSE PRINT "PEQUENO A"
```

o que deveria ser escrito em BASIC do *Spectrum* como:

```
1Ø IF A>B THEN PRINT "GRANDE A": GO TO 12
11 PRINT "PEQUENO A"
12 - - -
```

ou mesmo:

```
1Ø PRINT ("GRANDE A" AND A>B)+("PEQUENO A" AND A<=B)
```


INSTR(A$,B$)

Esta é uma função útil que verifica se B$ se encontra em A$. Se for esse o caso, a função indica em que carácter de A$ começa B$. Por exemplo:

```
INSTR("SINCLAIR", "IN")
```

irá dar "2" como resultado. Se B$ não for encontrada em A$, a função devolverá o valor "Ø" como resultado.

Para executar esta função no *Spectrum*, é necessária uma rotina especial:

```
1Ø FOR A=Ø TO LEN A$-LEN B$
2Ø IF B$=A$(A+1 TO A+LEN B$) THEN LET A=A+1: GO TO 4Ø
3Ø NEXT A: LET A=Ø
4Ø - - - -
```

a qual é equivalente a:

```
LET A=INSTR(A$,B$)
```

LEFT$, MID$, RIGHT$

Estas funções são encontradas em muitas versões de BASIC, e operam sobre uma cadeia, dando como resultado uma parte dessa cadeia.

LEFT$(X$,Y) permite obter os primeiros Y caracteres de X$; a sua equivalente para o *Spectrum* é X$(TO Y).

RIGHT$(X$,Y) permite obter os últimos Y caracteres da direita de X$; pode ser traduzida para X$(LEN X$-Y+1 TO).

MID$(X$,Y,Z) permite obter Z caracteres de X$, começando no Y-ésimo carácter. É equivalente a X$(Y TO Y+Z-1).

LET

A palavra LET pode ser omitida em muitas versões de BASIC; por exemplo:

```
1Ø A=1ØØ
```

cuja versão para o *Spectrum* deveria ser:

```
1Ø LET A=1ØØ
```

MAT

Algumas versões aperfeiçoadas de BASIC dispõem de comandos que permitem operar directamente sobre matrizes (quadros — *arrays*). Por exemplo:

```
1Ø DIM A(X,Y)
2Ø DIM B(X,Y)
  .
  .
1ØØ MAT A=B
```

deverá tornar todos os elementos do quadro A() iguais aos elementos correspondentes do quadro B().

Seria necessária uma rotina especial para fazer o mesmo no *Spectrum:*

```
1ØØ FOR p=1 TO X: FOR q=1 TO Y
1Ø1 LET A(p,q)=B(p,q)
1Ø2 NEXT q: NEXT p
```

PEEK, POKE, USR, CALL, LINK

LINK e CALL assemelham-se a USR no facto de chamarem uma rotina em linguagem máquina, mas, contudo, não devolvem um valor como resultado ao programa principal em BASIC. O efeito desses comandos depende inteiramente da máquina que esteja a ser utilizada. Para converter um programa que contenha qualquer destas funções num outro que possa ser executado num *Spectrum*, o leitor deve primeiramente determinar com exactidão o que se pretendia que o comando fizesse, e só então escrever o código do *Spectrum* destinado a executar a mesma função.

PLOT, DRAW

Cada versão de BASIC que possui uma capacidade de traçagem *(plotting)* ou de desenho parece utilizar uma forma diferente das declarações PLOT e DRAW. Tem que se determinar o que se pretendia no original e, seguidamente, escrever uma rotina do *Spectrum* para fazer o mesmo.

PRINT

É muitas vezes necessário modificar declarações PRINT para se poder fazer uso da organização de *écran* do *Spectrum*. Também se encontram funções CHR$ utilizadas em declarações PRINT para darem códigos de controle ou símbolos gráficos especiais; infelizmente, estas diferem de computador para computador.

PRINT USING é um comando muito poderoso, disponível em algumas versões de BASIC. Este comando dá ao programador o controle do "formato" do *output* a imprimir, de acordo com um padrão incluído algures no programa. Este padrão pode geralmente especificar o espaçamento, a justificação (ajustamento para perfeita igualdade nas linhas de uma página) à direita e à esquerda dos elementos impressos, e até o número de algarismos a imprimir depois do ponto decimal (vírgula).

PROCedimentos

O BASIC do *Spectrum* permite-nos definir uma nova função com uma única linha como, por exemplo:

```
DEF FN a(k,y)=INT ((x+y)/2)
```

e alguns BASICs permitem uma definição de função com várias linhas, como a seguinte:

```
DEF FN a(x,y)
LET s=Ø
FOR k=x TO y
LET s=s+k
NEXT s
= s
```

(a qual deveria dar a soma dos números de x a y).

Mas existe uma outra combinação, bastante semelhante, utilizada em algumas formas avançadas de BASIC, conhecida como um "procedimento" (*procedure*). É um cruzamento entre uma função e uma sub-rotina, pois pode utilizar variáveis "fictícias" como uma função, e é escrita como uma sub-rotina.

O procedimento é definido por uma secção de programa que começa com a palavra DEFPROC seguida de um nome, e termina com a palavra ENDPROC.

Um exemplo seria:

```
DEF PROC printotl (x,y,z)
PRINT "Total = "; x+y+z
ENDPROC
```

Este procedimento poderia ser depois utilizado no programa sempre que necessário, bastando para isso dar o nome do procedimento, seguido pelos valores reais a serem processados. Assim, as declarações:

```
printotl (5,6,2Ø)
```

e

```
printotl (A,B,C)
```

imprimiriam 31 e a soma de A, B e C, respectivamente.

RND

Alguns BASICs exigem que RND seja seguido por um número colocado entre parênteses; p. ex.:

```
LET A=RND(1)
```

Este número pode geralmente ser ignorado, excepto no caso de o programa ter sido escrito para um BASIC que lide apenas com números inteiros (tal como o BASIC do *ZX80* 4 K). Nestes

casos, RND(n) dá geralmente como resultado um número inteiro entre 1 e n, ou entre Ø e n−1.

LIGAÇÃO DE CADEIAS

Alguns BASICs utilizam o símbolo & ou ! para unir duas cadeias. Por exemplo:

```
LET A$=B$ & C$
```

ou

```
LET A$=B$! C$
```

as quais, no *Spectrum*, têm como equivalente:

```
LET A$=B$+C$
```

REPEAT/DO--UNTIL

Esta combinação faz com que as declarações existentes entre as palavras de comando REPEAT e UNTIL sejam repetidas (*repeated*) até que (*until*) a expressão condicional associada com o UNTIL seja verdadeira. Por exemplo:

```
1Ø REPEAT
2Ø INPUT "Nome";n$
3Ø UNTIL n$="Pai Natal"
```

A forma mais simples de converter esta combinação para o *Spectrum* consiste em mudar o REPEAT para uma REM e o UNTIL para um IF - THEN GO TO. Por exemplo:

```
1Ø REM
2Ø INPUT "Nome" n$
3Ø IF n$<>"Pai Natal" THEN GO TO 1Ø
```

Note-se que

REPEAT - - - - UNTIL 0

ou REPEAT - - - - UNTIL FALSE

se encontra muitas vezes, e é um ciclo sem fim; o leitor deve simplesmente substituir a declaração UNTIL por um GO TO incondicional.

Alguns BASICs usam a palavra DO em vez de REPEAT.

VAL

A função VAL do *Spectrum* é aborrecida devido ao facto de fazer parar o programa se não tiver nenhuma expressão numérica perfeitamente válida contida no seu argumento de cadeia. Outros BASICs apenas devolveriam, neste caso, o valor zero como resultado.

?

É utilizada por muitos BASICs como uma abreviatura de PRINT, e de uma forma altamente exótica pela Acorn Atom e pelo micro *BBC* para significar ou PEEK ou POKE, dependendo do contexto.

ERRATA: A figura da pág. 116 pertence à pág. 119.