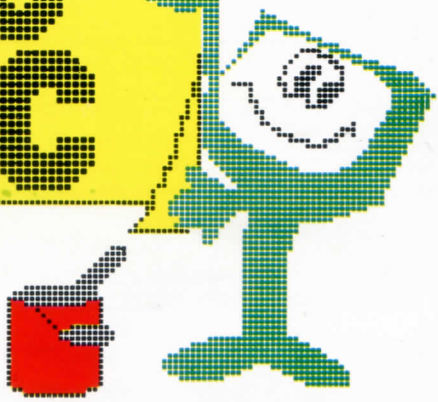
VIDEO BASIC



20 LIÇÕES DE BASIC
PARA APRENDER COM O SPECTRUM

EDIÇÕES LATINAS

JACKSON

- O que é um robot
- Como ensinar um robot
- O futuro da robótica
- Projecto e codificação de um caracter
- READ DATA
- RESTORE
- USR BIN
- Designação de dados
- Videojogo N.º 11

11

Spectrum

16K/48K/PLUS

TIMEX COMPUTER 2048

VIDEO BASIC
Uma publicação de:
EDIÇÕES LATINAS-JACKSON
Director editor:
Manuel A. Lopes
Director editor da JACKSON ESPANHA:
Lorenzo Bertagnolio
Director de produção:
Vicente Robles
Autor:
Softidea
Redacção Software:
Francesco Franceschini
Stefano Cremonesi
Desenho gráfico:
Studio Nuovaidea
Ilustrações:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Edições Latinas, Lda.
Direcção, redacção e administração, números atrasados e assinaturas:
Av. Almirante Reis, 219, 3.º-Esq.
1000 Lisboa
Fotocomposição e impressão:
Antunes & Amílcar, Lda.

Reservados todos os direitos de reprodução e publicação de desenho, fotografia e textos.
© Grupo Editorial Jackson 1985
© Edições Ingelek 1985
© Edições Latinas 1985
ISBN do tomo 1: 84-85831-12-8
ISBN do fascículo: 84-85831-11-X
ISBN da obra completa: 84-85831-10-1
Depósito Legal N.º: 11335/86
Plano geral da obra:
20 fascículos e 20 cassetes, de publicação quinzenal.
Edições Latinas-Jackson garante a publicação de todos os fascículos e cassetes que compõem esta obra e o fornecimento de qualquer número atrasado, até 1 ano, depois de terminada a sua publicação. Está reservado ao editor, o direito de modificar o preço de venda dos fascículos durante a sua saída, se o mercado assim o exigir.

1.ª Quinzena – Abril 1986

EDIÇÕES LATINAS

JACKSON

SUMÁRIO

Introdução

Nesta edição veremos, em primeiro lugar, como funcionam, como aprendem e como trabalham os robots, comentando também as fascinantes perspectivas e as possibilidades que estes dispositivos podem oferecer na futura vida quotidiana.
Veremos, seguidamente, como é possível inserir permanentemente num programa informações úteis ou recorrentes, usando as instruções READ, DATA e RESTORE.
Para terminar, apresentaremos um exemplo sobre a técnica a adoptar para personalizar a saída de visualização, aprendendo a definir e memorizar novos caracteres no nosso computador.

O que é um robot

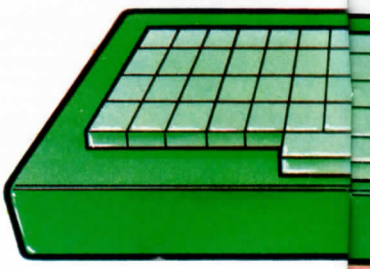
Desde antes da chegada dos computadores electrónicos, que uma das maiores aspirações do homem foi a de conseguir inventar e construir novas máquinas que fossem capazes de aliviá-lo nos trabalhos mais pesados, perigosos ou repetitivos. A história ensina-nos sempre: cada degrau da escada do progresso esteve normalmente caracterizado (e assim continua sendo hoje em dia) por uma grande descoberta ou invento fundamentais (o fogo, o ferro, o vapor ou a electricidade), descobertas que tendo sido no início realizadas para resolver determinados problemas em concreto, ao ser aplicadas e desenvolvidas proporcionaram-nos, e continuam a proporcionar-nos, novos aperfeiçoamentos e progressos por vezes inesperados.

O computador, apesar de não poder ser considerado uma invenção no sentido estrito, não foge a esta regra.

Nascido para "calcular", o computador converteu-se na base de uma infinidade de novas aplicações, que se baseiam e dependem exclusivamente dele. Entre estas encontramos a robótica, isto é, a ciência que se ocupa da automatização do trabalho através de máquinas controladas pelo computador.

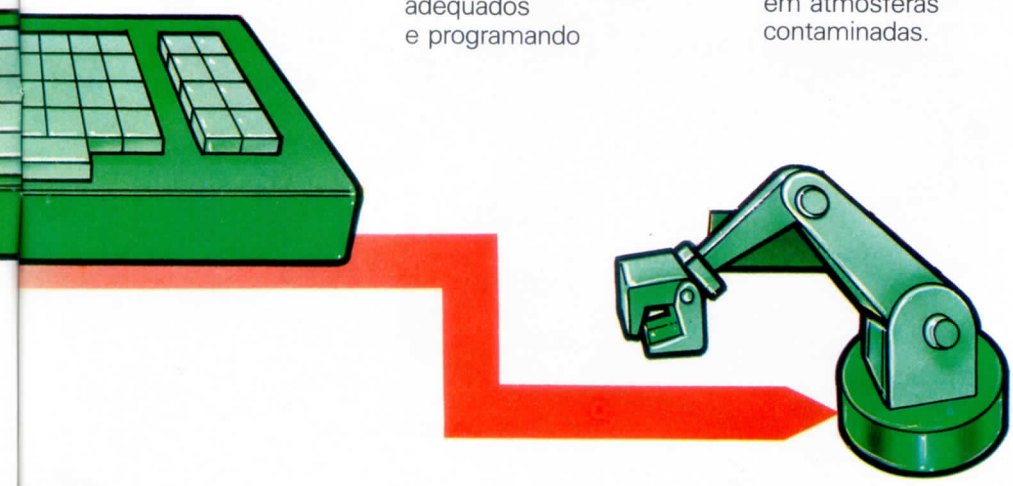
Robot é um termo que suscita na mente de muitas pessoas dúvidas e perplexidades: é a recordação de homenzinhos metálicos e de máquinas falantes, tão queridas pela ficção-científica comercial, que tornam talvez difícil a compreensão exacta do termo; sendo assim tentaremos, em primeiro lugar, explicar o seu significado exacto.

Um robot não é mais do que uma máquina automática que, sob o controle de um computador, desenvolve um trabalho pré-estabelecido: por exemplo, pintar um automóvel, apertar porcas ou soldar chapas.



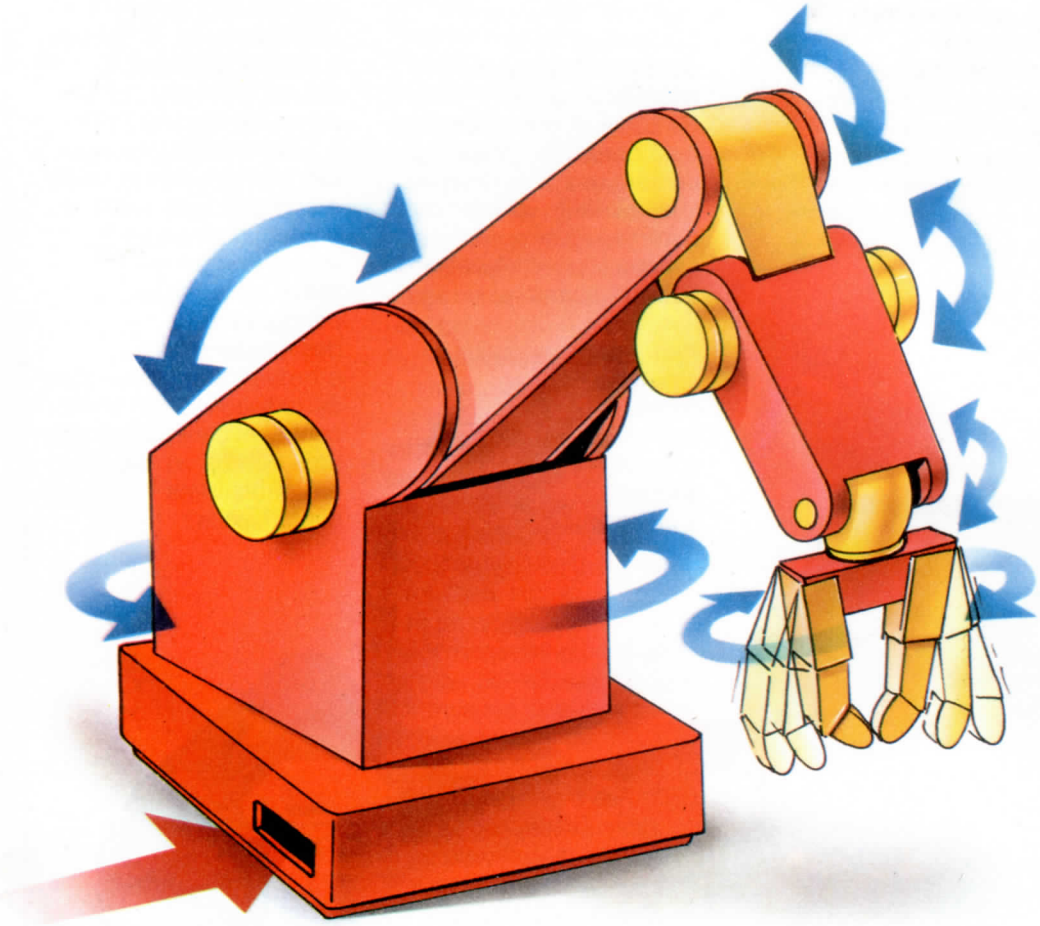
HARDWARE

A grande vantagem de um robot sobre uma máquina automática normal é que ao estar ligada a um computador, pode ser programado – com modificações que habitualmente não são muito difíceis – para desenvolver distintos trabalhos (embora não muito diferentes).
Para além da programabilidade, é preciso acrescentar ainda uma outra vantagem: graças aos progressos da microelectrónica, os robots são dotados de uns dispositivos especiais chamados sensores que – apesar de estar bastante longe dos sentidos humanos – lhes permitem reconhecer quando acontecem determinadas situações especiais ou anómalas.
Vejamos um exemplo.
Suponhamos que temos que unir com uma porca duas peças de metal preparadas adequadamente.
Se usamos um robot para esta operação, devemos contar com o aparecimento de determinadas eventualidades, devidas a um alinhamento errado das peças ou ao tamanho inadequado de qualquer um dos seus componentes.
Equipando a máquina com os sensores adequados e programando devidamente o computador de controle, é possível parar a máquina quando suceda qualquer um destes casos.
Uma máquina normal, na presença destas situações anómalas, continuaria o seu trabalho, originando o início da produção de uma peça defeituosa, ou danificando a própria máquina.
Assim, a introdução do robot acrescenta à linha de montagem um elemento de controle e comprovação, além de produtivo. Aliás, muitos robots podem realizar trabalhos que seriam periogosos ou desagradáveis para o hmem, como medições de radioactividade numa central nuclear, desactivar uma bomba ou trabalhar em atmosferas contaminadas.



Como ensinar um robot?

A maior parte dos robots existentes trabalham em fábricas nas quais tudo está organizado ao seu redor: habitualmente, têm de levantar as peças dos tapetes rolantes, trabalhá-las seguindo uma certa sucessão de operações elementares e, finalmente, depositá-las sobre um outro tapete

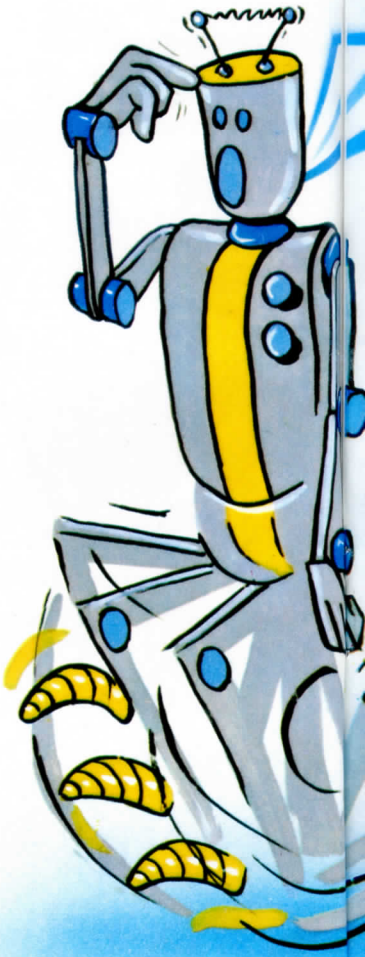


rolante, que se ocupa de enviá-las a outros pontos de elaboração.
De qualquer maneira, os préstimos de um robot dependem de muitos factores e o último não é o económico; portanto existem (e estes são a maioria, dado o seu menor custo e a sua relativa simplicidade) robots construidos para utilizações bastante diversificadas, que são adaptados em cada passo para desenvolver tarefas mais específicas.
Os robots mais comuns são os chamados "de braço": o seu mecanismo está construido tendo como modelo o braço humano. Esta disposição permite – tal como nos braços verdadeiros – uma notável liberdade e capacidade de movimentos, permitindo às "mãos" (que costumam ser pinças, ligadas a motores eléctricos) trabalhar em posições que de outra forma seriam inacessíveis. Actualmente está a ser estudada também a maneira de dotar estas "mãos" de uma espécie de sentido de tacto, permitindo assim uma tracção adequada ao peso e à solidez do objecto a levantar (até agora nenhum robot consegue distinguir uma peça de ferro com a forma de um ovo de um ovo verdadeiro), com resultados facilmente compreensíveis. Portanto, projectar, construir e ensinar um robot não é uma operação fácil: é necessário, em primeiro lugar, analisar, e subdividir a acção que o robot terá de realizar em todos os seus possíveis e imagináveis passos elementares (lembras-te dos algoritmos?), classificando assim todas as suas diferentes necessidades de movimento. Com base nestes movimentos – e em função do esforço necessário para realizar cada uma destas acções – decidir-se-á o número e a posição de braços mecânicos, de juntas (ou seja, das articulações entre os diferentes braços) e dos seus respectivos motores.
Além do mais, para poder realizar o seu trabalho, o robot deverá estar equipado com vários tipos de sensores (fotoeléctricos, electrónicos ou mecânicos), adequados ao género de trabalho requerido. Através de um ou mais interfaces estes sensores terão que enviar informações codificadas ao computador de controlo, que supervisa a direcção e a coordenação dos movimentos, que assim poderá dirigir a execução, a modificação ou a paragem das operações previstas no princípio.
Portanto, o computador de controle terá de ser programado adequadamente para esta espécie de necessidades, tendo devidamente em conta todo os imprevistos e anomalias possíveis.
Isto repete-se para dezenas ou centenas de acções específicas.

Mais uma vez deves reparar na importância que tem – antes de passar à verdadeira fase de programação – a avaliação, até nos mais pequenos detalhes, de todo o conjunto de operações e isto independentemente de que sejam importantes ou insignificantes, pois o microprocessador do robot – desde que seja operativo – terá sempre que controlar a totalidade.
Num robot, especialmente se é de modelo industrial, esta fase de análise está revestida de uma dificuldade especial, dado que normalmente um único microprocessador não é capaz de enfrentar, sozinho, todo o trabalho que o robot costuma poder desenvolver.
Habitualmente, para não dizer sempre, é necessário programar uma série de micro processadores para que realizem todas as operações particulares (como por exemplo, mexer um braço ou agarrar um objecto), coordenando-os todos mediante uma grande central de controlo capaz de seguir em cada momento as fases de elaboração.
Como poderás imaginar, se já é difícil programar um só computador, programar dez ou quinze (e ainda por cima, para que possam trabalhar coordenados entre si) torna-se uma empresa extremamente delicada.

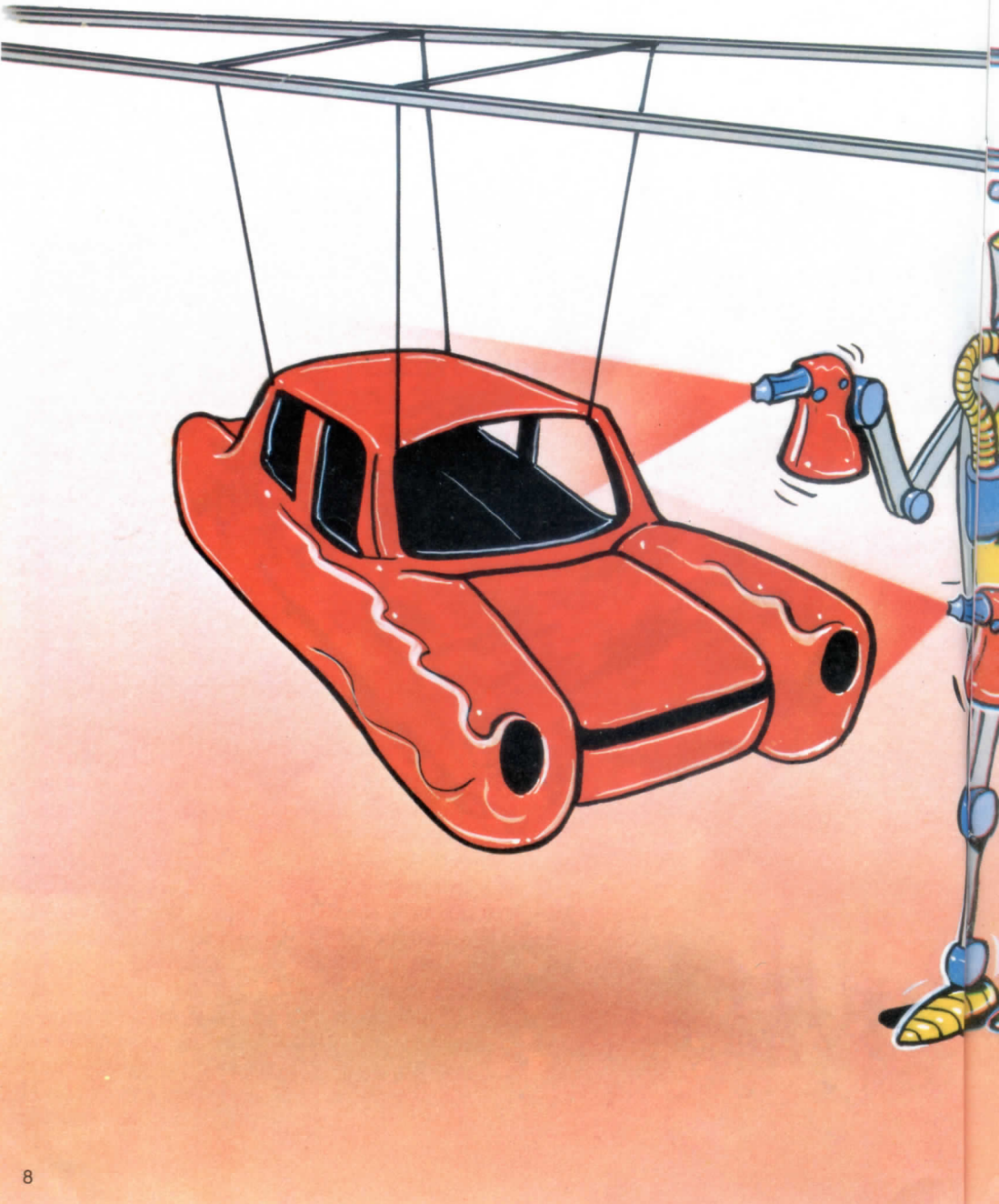
Futuros desenvolvimentos

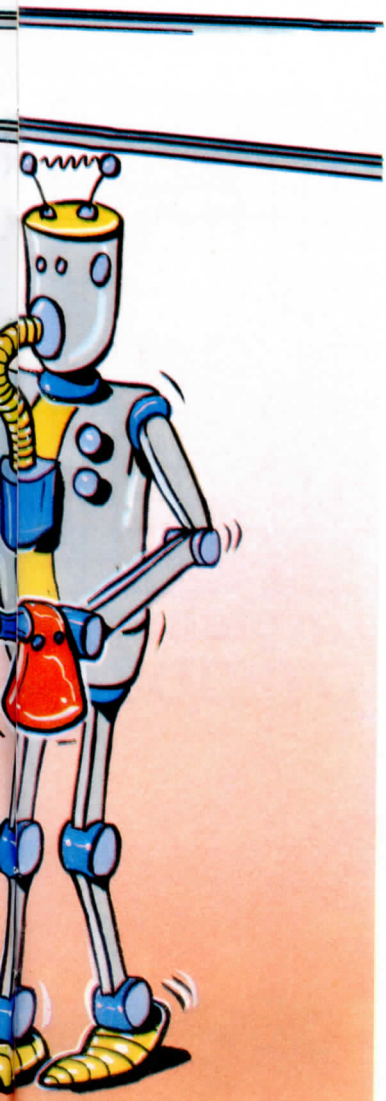
A robótica é uma ciência muito jovem, pode dizer-se que apenas nos seus princípios. No futuro poderá, portanto, haver muitos desenvolvimentos e melhorias, graças também aos numerosos projectos de investigação que actualmente estão em curso em todo o mundo.
Neste momento, um dos caminhos mais investigados (e na realidade não só pela robótica) é o da chamada "inteligência artificial".
Em poucas palavras, a inteligência artificial está a tentar dotar os computadores de uma espécie de inteligência autónoma e automática, capaz de aprender e recordar os feitos passados para os aplicar aos futuros (exactamente como acontece com as crianças que após queimarem-se uma vez já não tentam tocar o fogo).
Até agora os resultados não foram muito animadores, mas dá a impressão que dentro



HARDWARE

HARDWARE

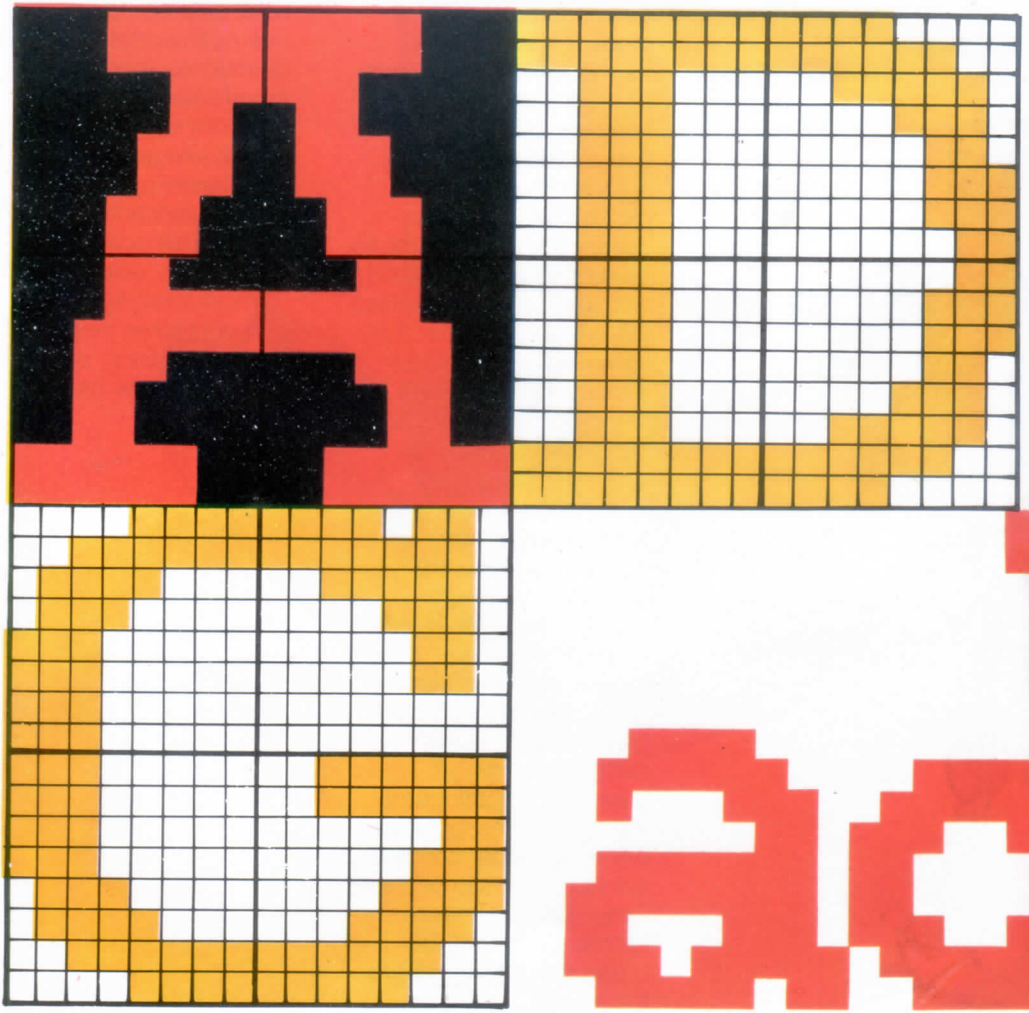




de poucos anos se poderá começar a ver as coisas mais claras.
De qualquer modo, para além dos temores que possa inspirar a ideia de máquinas inteligentes, um robot capaz de aprender do ambiente que o rodeia, permitiria resolver muitos dos problemas ligados ao desenho e à programação do próprio robot, evitando a longa e complicada fase de estudo e análise, que, como já vimos contitui a maior dificuldade na construção e instalação de todos os dispositivos automáticos.
Embora os desenvolvimentos da robótica pareçam prometer no futuro próximo algumas coisas interessantes, no momento actual ainda ficam muitos problemas: os robots não sabem mexer-se como desejaríamos (ninguém até agora conseguiu fazer um robot andar como um homem), são bastante caros (especialmente os robots industriais) e não é possível adaptá-los a usos universais.
De qualquer maneira, existem já no mercado robots de muitos tipos, até para usos pessoais.

Mas o objecto destes últimos, mais do que outra coisa, costuma fazer referência ao ensinamento dos rudimentos da robótica e só raramente permitem aprender com facilidade, e com um gasto relativamente baixo, o funcionamento e a programação dos robots, sendo normalmente ligável aos computadores pessoais mais difundidos, para poder controlar os seus movimentos.

Definição de caracteres



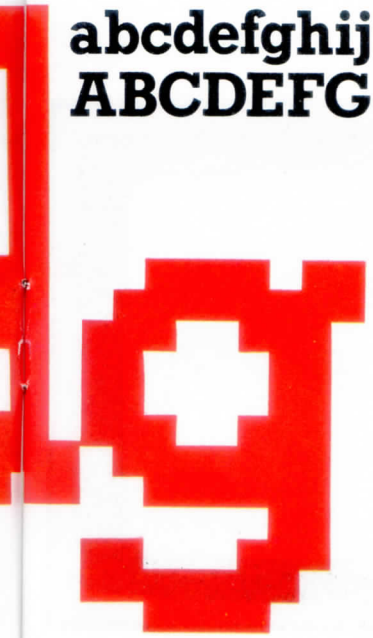
Um caracter está encerrado numa matriz de 8 por 8 pontos. Quando desejes representá-lo em detalhe, será preciso operar sobre uma matriz de pontos maiores
No exemplo podes observar como para reproduzir detalhadamente um alfabeto especial se recorreu a uma matriz de 16 por 16 (4 caracteres).

LINGUAGEM

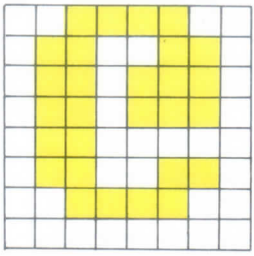
Acontece-nos por vezes que, por gosto ou por necessidade, desejamos modificar um ou mais caracteres entre aqueles normalmente disponíveis no teu Spectrum. Para fazer isto é preciso realizar uma simples operação, mais conhecida como definição de caracteres, que permite criar um novo conjunto de símbolos que se fazem corresponder com cada uma das teclas existentes no teclado. Assim, se por exemplo desejássemos dispor no teclado das letras do alfabeto grego, bastaria que "ensinássemos" ao computador a forma de cada um dos novos caracteres, de maneira que a tecla "A" correspondesse à letra "alfa", a letra "B" ao "beta" e assim sucessivamente.

Tu já sabes o que faz o Spectrum para ser capaz de visualizar todos os caracteres que normalmente podes ver no ecran e já sabes também que a forma dos caracteres (constituída por séries de 8 bytes para cada caracter) encontram-se cuidadosamente guardadas na memória de leitura (ROM), para que não se percam cada vez que se desliga o computador. Portanto, quando carregas numa tecla, esta corresponde-se com uma direcção da ROM que contém a sequência de informações que o computador precisa conhecer para que o caracter possa aparecer no ecran. Dado que a memória ROM não pode ser alterada de nenhuma maneira, o que se tem que fazer primeiro, se se deseja definir novos caracteres, é transferir todas as imagens dos antigos caracteres para uma zona acessível também à escritura (isto é a memória RAM). Agora já está tudo preparado para aceitar as eventuais modificações que desejamos fazer aos diferentes caracteres. No entanto, para definir um novo caracter é preciso em primeiro lugar examinar o que se

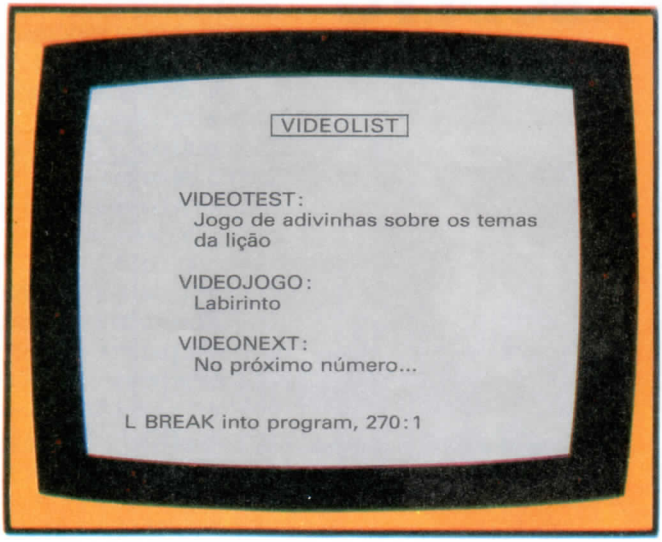
abcdefghij
ABCDEFG



deve fazer para criar um caracter. Cada caracter é composto por uma combinação de 64 pontos (obtida empregando os 8 bits de cada byte) dispostos em 8 linhas e 8 colunas. Eis aqui um exemplo, representado por uma sequência deste tipo:

BINÁRIO	DEC.	
0 0 1 1 1 1 0 0	60	
0 1 1 0 0 1 1 0	102	
0 1 1 0 1 1 1 0	110	
0 1 1 0 0 0 0 0	96	
0 1 1 0 0 1 1 0	102	
0 0 1 1 1 1 0 0	60	
0 0 0 0 0 0 0 0	0	

onde cada 1 ou 0 representam respectivamente um ponto (pixel) aceso ou apagado. Para um computador, aceso ou apagado significam valor 1 ou 0: então será suficiente introduzir nas localizações dedicadas à definição desse caracter os oito números binários, lidos linha a linha, que definem as combinações de pixels.
No nosso exemplo, tomando a primeira linha (00111100) obteremos um determinado número binário, que convertido a décimal (60), nos proporciona um dos oito

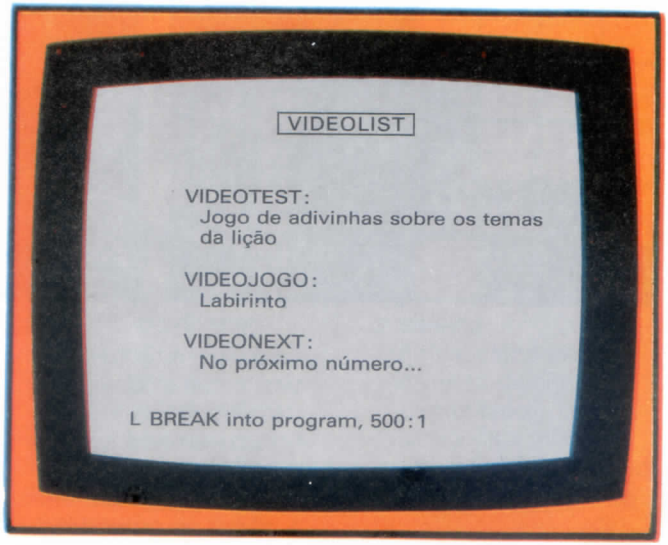


valores a inserir nas localizações da memória que especificarão a descrição e a estrutura desse caracter. Para um melhor entendimento, esse valor está escrito à direita da mesma linha. A mesma operação com as outras sete linhas proporciona-nos os restantes valores, necessários para definir a totalidade do caracter. Os oito números que derivam de C serão: 60, 102, 110, 110, 96, 102, 60 e 0. Agora já se podem inserir na memória. Quando o computador tenha que visualizar esse caracter, tomará os oito números da memória e passá-los-á ao ecran. Isto é tudo.

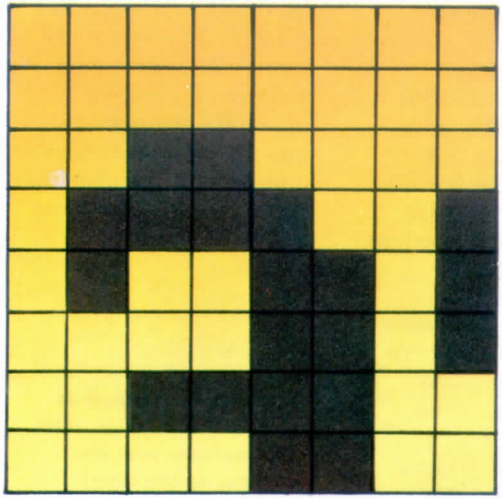
Assim, o primeiro passo que é preciso realizar para a definição de um caracter é o de desenhar uma quadrícula de 64 quadrados, na qual se definem os pixels a acender ou a apagar.

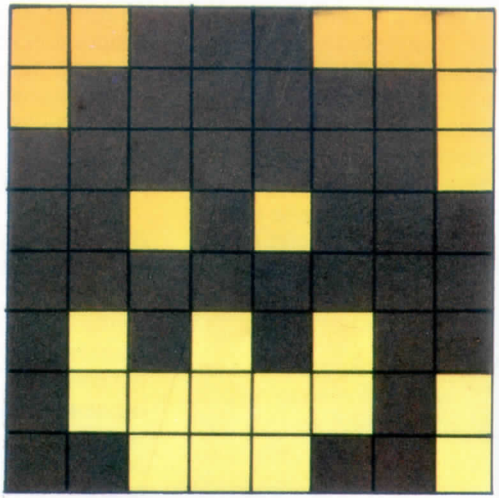
Na criação de qualquer caracter é aconselhável tentar não usar nunca um 1 ou um 0 isolados: utilizando um televisor que não seja de excelente qualidade, é provável que o ponto correspondente não seja visualizado.

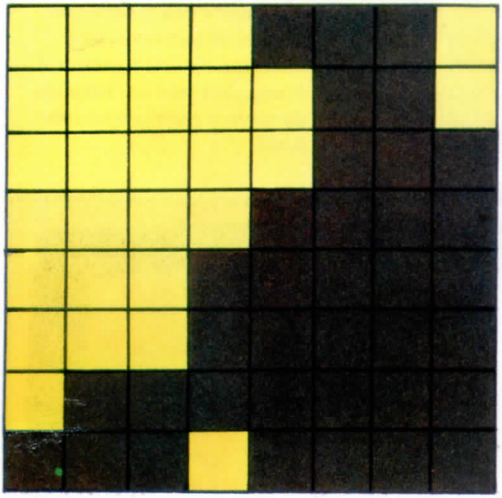
Estes ecrans mostram as mesmas informações, mas representadas com um conjunto de caracteres diferente.
A primeira usa o conjunto normal (o existente na memória do teu Spectrum).
A segunda usa um definido de novo e memorizado na RAM.

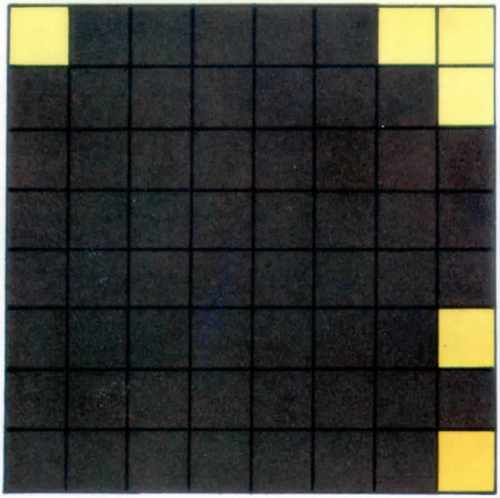


LINGUAGEM

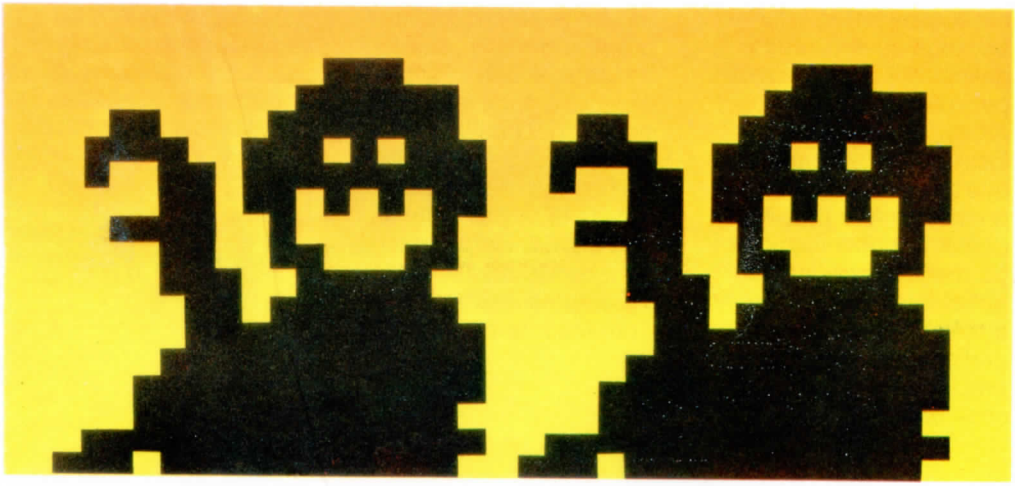








Feito isto uma primeira vez, o caminho a seguir para introduzir outros caracteres será sempre o mesmo: mudarão unicamente os números a inserir no ordenador (cada caracter está claramente definido por uma única e particular sequência de pontos luminosos) e as direcções e localizações nas quais será preciso introduzir



estes números. No final, o conjunto de caracteres terá sido definido e poderá ser empregado à vontade em qualquer programa.

Para recuperar o antigo conjunto de caracteres basta desligar o computador e voltar a ligá-lo após alguns segundos: esta operação apagará todas as modificações efectuadas na memória RAM, reestabelecendo as direcções do ROM que correspondem às teclas habituais.

USR-BIN

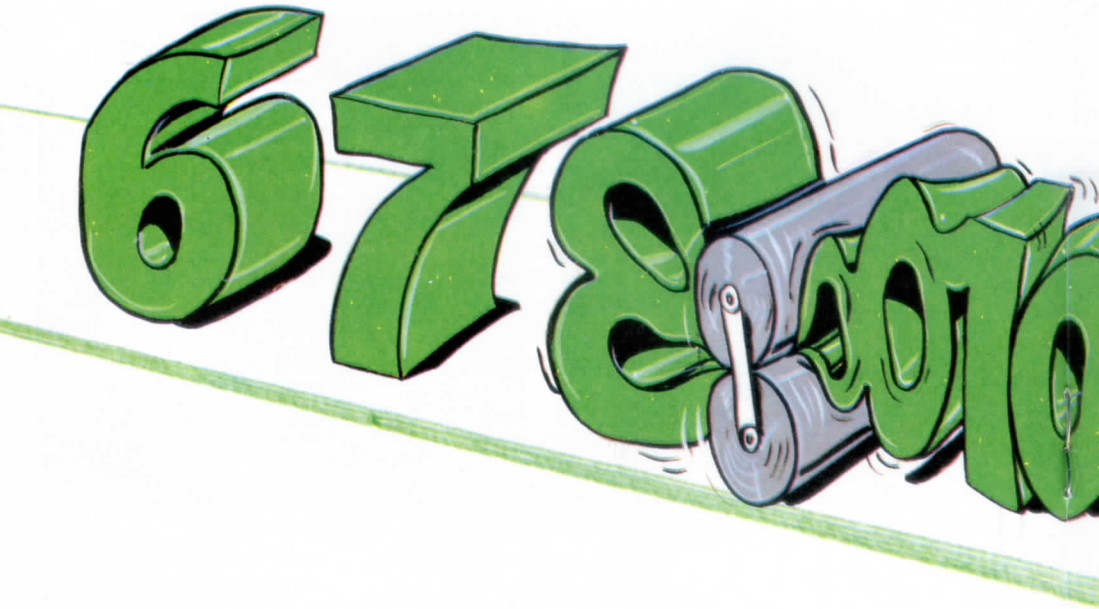
O teu Spectrum pode funcionar de diversos modos, entre os quais o modo gráfico.
O que significa gráfico?
É muito simples: quando estás no modo gráfico (cursor G), premindo os números de 1 a 8, aparecem os símbolos gráficos representados sobre as respectivas teclas, enquanto as letras do alfabeto permanecem inalteradas.
Mas o assunto não é tão fácil como poderia parecer. A cópia do alfabeto que se obtem premindo as teclas no modo gráfico, não é lida pelo Spectrum na memória ROM (como acontece quando se encontra no modo L), mas numa zona da memória RAM, onde é situada automaticamente no momento de ligar o computador.
Isto significa que, se se quiser, é possível modificar conforme as próprias necessidades

esses caracteres, trocando-os por outros definidos e personalizados pelo usuário. Assim, alterando as localizações de memória que contêm, por exemplo, o caracter A, é possível definir um novo caracter, de maneira que, de ali em diante, ao premir a tecla A no modo gráfico lhe corresponda precisamente esse caracter.

Já vimos anteriormente o que é preciso fazer para construir e definir sobre o papel o caracter desejado. Uma vez encontrada a sequência de 0 e 1 correspondente a cada uma das 8 linhas que identificam o caracter, será necessário introduzi-las na memória. Tomando de novo o exemplo do caracter , tinhamos obtido:

```
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 0 1 1 1 0
0 1 1 0 1 1 1 0
0 1 1 0 0 0 0 0
0 1 1 0 0 1 1 0
0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0
```

Transformando estes números binários em décimais obtêm-se

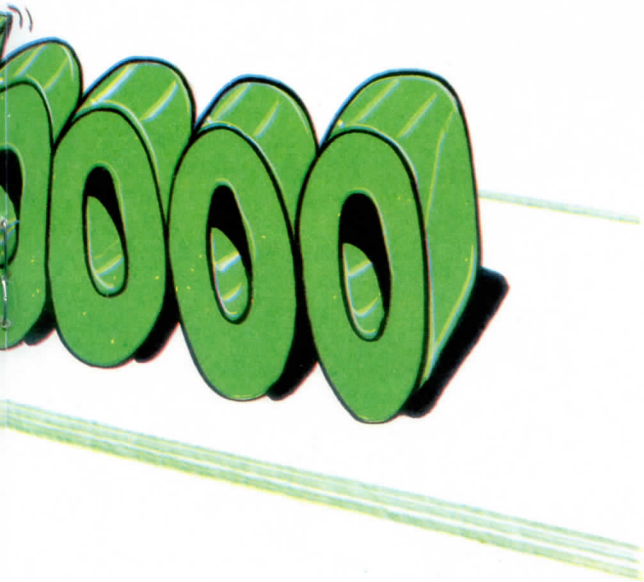


os 8 números: 60, 102, 110, 110, 96, 102, 60, 0.
Decidimos que a tecla que terá de corresponder a este novo caracter terá que ser a B, de maneira que quando a premimos no modo gráfico, se visualize C . Mas agora surge um novo problema: como encontrar as direcções da memória RAM correspondentes a B, nas quais escrever estes 8 números?
A resposta chama-se USR.
USR é uma função que converte um argumento do tipo cadeia composto por um único caracter na direcção ocupada pelo primeiro dos 8 bytes do caracter gráfico do argumento. Portanto:

```
USR "B"
```

devolver-nos-á a direcção na qual memorizar o primeiro dos 8 números, USR "B" + 1 o segundo e assim sucessivamente.
Proporcionando a USR mais de um argumento obter-se-á a mensagem de erro: INVALID ARGUMENT.
O programa que carrega na memória o caracter gráfico que definimos será:

```
10 FOR A=0 TO 7
20 READ B
30 POKE USR "B"+A, B
40 NEXT A
50 DATA 60, 102, 110, 110
60 DATA 96, 102, 60, 0
```



Uma vez executado o programa, a imagem do caracter C terá sido inserida no ordenador e poderá ser chamada premindo no modo G (gráfico) o caracter B.
Se o desejares também é possível saltar a fase de conversão dos números binários em décimais, empregando uma segunda função: BIN.
BIN usa-se para introduzir no Spectrum um número na sua forma binária em vez da décimal.
Portanto:

```
POKE 300, BIN 11111111
```

é totalmente equivalente a:

```
POKE 300, 255
```

e, com efeito

```
PRINT BIN, 11111111
```

imprime como resultado 255 décimal
Assim, o nosso programa também se poderia ter escrito desta maneira:

```
10 FOR A=0 TO 7
20 READ B
30 POKE USR "B"+A, B
40 NEXT A
50 DATA BIN 00111100
60 DATA BIN 01100110
70 DATA BIN 01101110
80 DATA BIN 01101110
90 DATA BIN 01100000
100 DATA BIN 01100110
110 DATA BIN 00111100
120 DATA BIN 00000000
```

Na prática, a função BIN permite-nos introduzir nas linhas DATA a matriz de pontos de um caracter, exactamente igual a como o tenhamos desenhado.

Sintaxe da função USR

Uma cadeia de um único caracter compreendido entre A e U

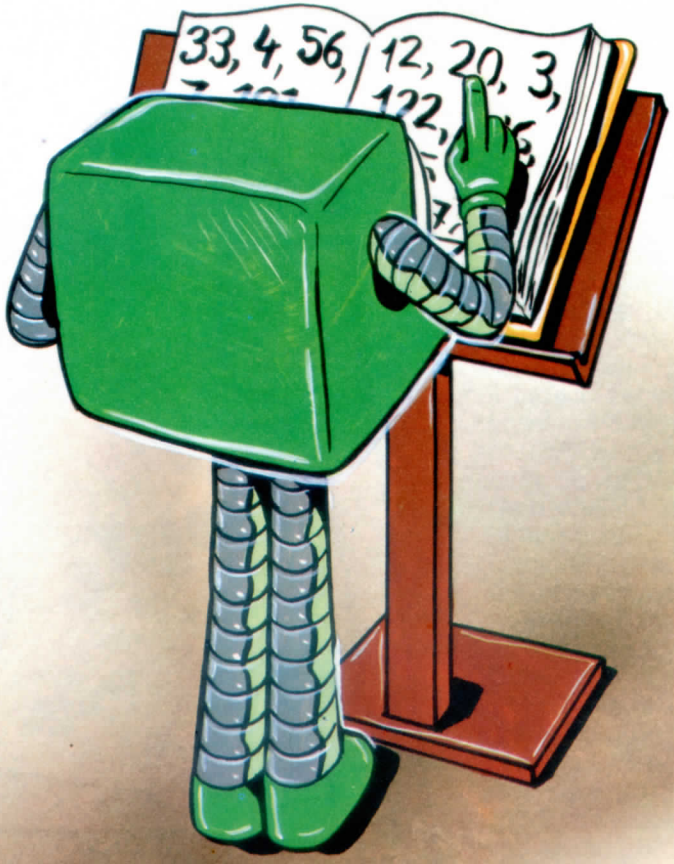
Sintaxe da função BIN

BIN número binário com um máximo de 16 bits.

READ/DATA

As instruções READ e DATA permitem ler dados de um programa, evitando assim ter que premi-los no teclado.
A introdução dos dados a elaborar realiza-se, além do mais, sem a momentânea paragem do programa, ao contrário do que acontece com a instrução INPUT.
Provavelmente perguntas-te a ti próprio para que poderá servir semelhante estrutura, visto que até agora, te tens arranjado lindamente sem ela.
Um exemplo esclarecer-te-á rapidamente.
Acontece com muita frequência que um programa requer como acção preliminar a execução da chamada inicialização das variantes, ou seja, a inserção de alguns valores específicos dentro de determinadas variantes (por exemplo, os nomes e a duração de cada mês do ano).
Para realizar esta operação temos à nossa disposição três alternativas possíveis:
– empregar uma longa série de LET no princípio do programa;
– pedir em cada instrução,

usando INPUT, os valores a destinar a cada variante: – usar READ E DATA; Afastemos rapidamente a primeira solução: iria requerer demasiado trabalho ao premir as teclas do programa e ocuparia demasiada memória (lembra-te que cada instrução conservada no computador ocupa uma determinada quantidade de memória). A segunda solução já é



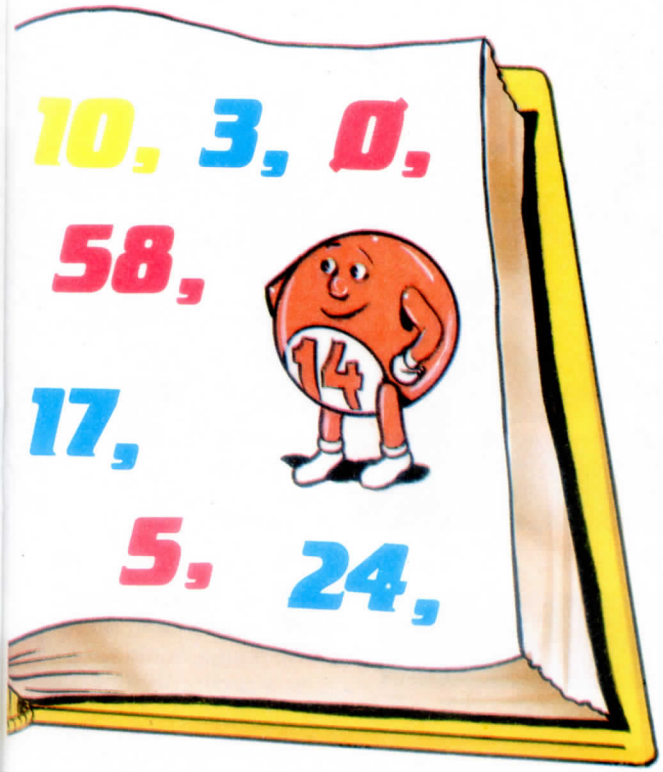
mais aceitável. No caso dos meses poderia escrever-se:

```
10 DIM M$ (12, 10):DIM G (12)
20 FOR I=1 TO 12
30 INPUT M$ (I), G (I)
40 NEXT I
```

e com estas poucas instruções teriamos podido arranjar-nos. Mas o problema apenas ficou parcialmente resolvido: com cada RUN teriamos que encarregar-nos de premir nas teclas JANEIRO, 31; FEVEREIRO, 28; etc., indicando toda uma série de valores, que, no fim de contas, não sofrem nenhuma modificação de uma a outra execução, e que, portanto, seria cómodo poder conservar permanentemente no programa. Isto significa que cada vez que

empreguemos este programa teremos que atribuir pelo teclado a totalidade das 24 informações que nos são requeridas. É um trabalho longo, aborrecido e inútil. Adoptando READ e DATA, teriamos em troca:

```
10 DIM M$ (12, 10):DIM G (12)
20 FOR I=1 TO 12
30 READ M$ (I):READ G (I)
40 NEXT I
50 DATA JANEIRO, 31, FEVEREIRO, 28, MARÇO 31, ABRIL, 30
60 DATA MAIO, 31, JUNHO, 30, JULHO, 31, AGOSTO, 31
70 DATA SETEMBRO, 30, OUTUBRO, 31, NOVEMBRO, 30,
   DEZEMBRO, 31
```



Na prática estas instruções indicam ao computador que faça o mesmo que com as linhas vistas anteriormente, com a única diferença que cada variante das matrizes M$ () e D () devem ser lidas (READ) já não desde o teclado – como antes acontecia devido às INPUT – mas desde as linhas de DATA. Os dados são inseridos no computador uma única vez e, o que é ainda mais importante, quem usar o programa não se vê na obrigação de conhecer e teclar estes dados (imagina os nomes das equipas do campeonato de futebol ou outra série de dados, possívelmente menos conhecidos que os nomes dos meses do ano).

Desta maneita é possível manter, conservando-os no programa, valores que de outra forma se perderiam sempre que se desliga o computador e com cada nova execução do mesmo programa.
As instruções DATA contêm os diferentes valores que desejamos designar; estas permitem reagrupar todos os dados num único ponto do programa, normalmente no princípio ou no fim, onde são mais facilmente legíveis. Cada fim do tipo cadeia deve, como é norma, estar entre aspas.
A instrução READ permite ler os dados que foram memorizados nas DATA.
É óbvio que é preciso ter o máximo cuidado para que o tipo de variante da instrução READ corresponda com o contido na instrução DATA, como acontece com os INPUT enviados pelo teclado: nunca deve acontecer que uma variante numérica possa conter um alfanumérico, ou que uma variante do tipo cadeia se encontre num valor numérico. Este tipo de erro está sempre à espreita!
As linhas DATA do listrado anterior são três unicamente por motivos de legibilidade: teriamos podido pôr um número diferente e sem que fossem necessariamente consecutivas.
Se existem várias linhas DATA, em diferentes lugares do programa, estas leêm-se consecutivamente, como se se tratasse de uma única linha.

Exemplos

```
10 READ A:READ B
20 DATA 3, 5
```

Às variantes A e B atribuem-se respectivamente os valores 3 e 5.

```
10 READ A
20 READ B: READ C
30 DATA 4. 7
```

Desta vez o programa falha: à variante C não corresponde nenhum valor e provocaremos a mensagem OUT OF DATA

```
10 READ A
20 READ A$
30 READ C
40 DATA 3, "4", 5
```

A, A$ e C, tomam respectivamente os valores 3, "4" e 5. Repara nas aspas no 4: a segunda variante é do tipo cadeia. O 4 será inserido na memória como caracter e não como número.

```
10 READ C$, A, D$
20 DATA "PATETA", "PLUTO"
```

Nesta ocasião os erros são dois: a variante A, sendo de tipo numérico, não pode tomar o valor PLUTO (aparecerá a mensagem de erro: NONSENSE IN BASIC) e a variante D$ carece do seu correspondente DATA.

```
10 FOR I=1 TO 6
20 READ K$
30 PRINT K$
40 NEXT I
50 DATA "FRANCISCO", "MARIO"
60 DATA "MATEUS", "CARLOS"
70 DATA "ZÉ", "SANTIAGO"
```

A variante K$ irá tomando sucessivamente os 6 valores especificados nas instruções DATA. A instrução PRINT K$, situada no ciclo FOR, provocará a impressão de FRANCISCO, MÁRIO,... SANTIAGO. Observa como a variável antes K$ é do mesmo tipo (cadeia) que os valores que deverá tomar como consequência das READ.

```
 5 CLS
10 PRINT "SISTEMA SOLAR": PRINT
15 FOR K=1 TO 9
20 PRINT
25 READ P$, D
30 PRINT "O PLANETA"; P$
35 PRINT "TEM UM DIÂMETRO DE"; D; "KM"
40 NEXT K
45 DATA "MERCÚRIO", 4880, "VENUS", 12096
50 DATA "TERRA", 12740, "MARTE", 6780
55 DATA "JÚPITER", 141560, "SATURNO", 120800
60 DATA "URANO", 51000, "NEPTUNO", 49300
65 DATA "PLUTÃO", 7000
```

Isto, mais do que um exemplo é uma demonstração da utilidade de READ... DATA. Com poucas instruções é possível inserir permanentemente informações que nunca irão mudar de uma execução a outra, permitindo assim uma considerável economia de tempo e de espaço.

Sintaxe da instrução DATA

DATA constante [, constante] [, ...]

Sintaxe da instrução READ

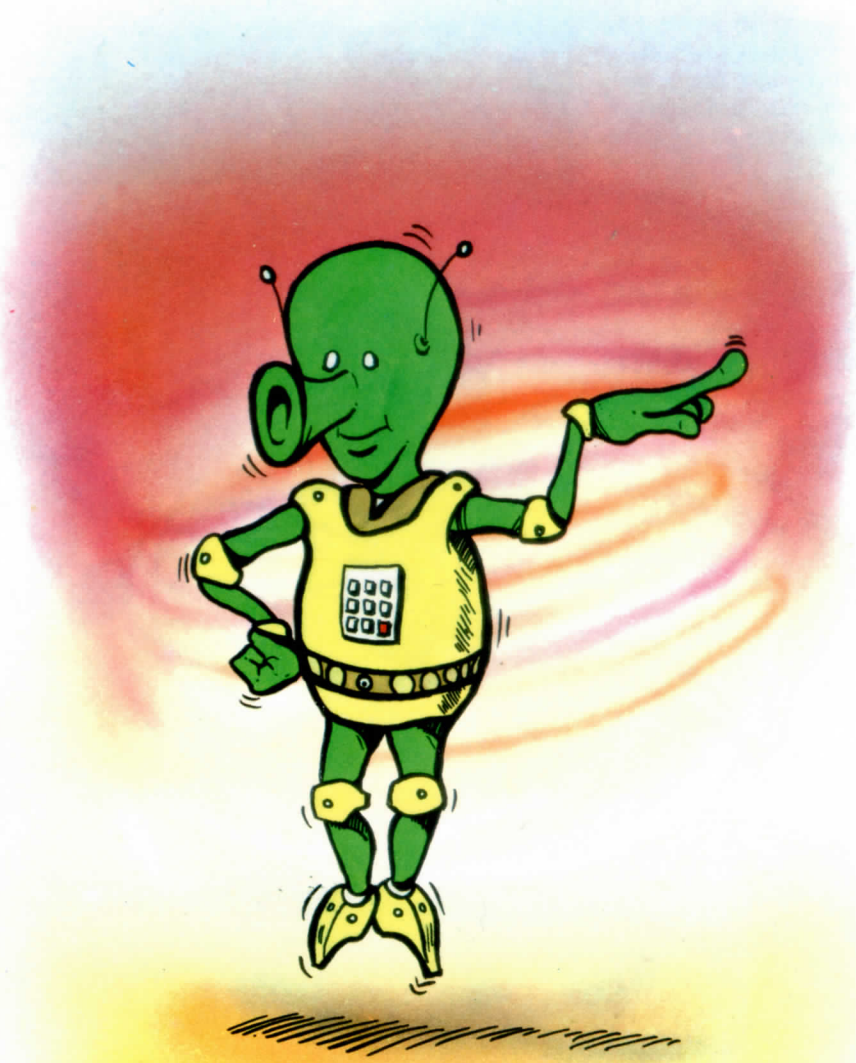
READ variável [, variável] [, ...]

RESTORE

Para manter sob controle a ordem em que são tomados os valores das DATA, podemos imaginar que no interior da memória existe uma espécie de índice (chamado ponteiro) que vai marcando de cada vez qual deve ser o valor sucessivo que pode ser lido com uma instrução READ.

Cada vez que o computador lê um elemento qualquer desde as linhas DATA, esse ponteiro é deslocado para que indique o elemento sucessivo, de forma que o intérprete BASIC sempre saiba até onde chegou a leitura das DATA. Portanto, a cada READ corresponderá um avanço automático do

ponteiro das DATA. No entanto, pode às vezes ser muito útil ter a possibilidade, no decurso dum programa, de aceder às mesmas informações mais de uma vez. A instrução RESTORE serve precisamente para devolver o ponteiro à primeira linha DATA



do programa, fazendo com que as constantes fiquem de novo à disposição, como se nunca tivessem sido lidas.

Quando se encontra a instrução RESTORE – e independentemente do número de valores que já tenham sido lidos – faz-se com que a instrução READ seguinte volte atrás para ler o primeiro elemento da lista das DATA, como se fosse a primeira vez. Assim o seguinte programa:

```
10 FOR I=1 TO 1000
20 READ A
25 PRINT A
30 RESTORE
40 NEXT I
50 DATA 27, 10, 60
```

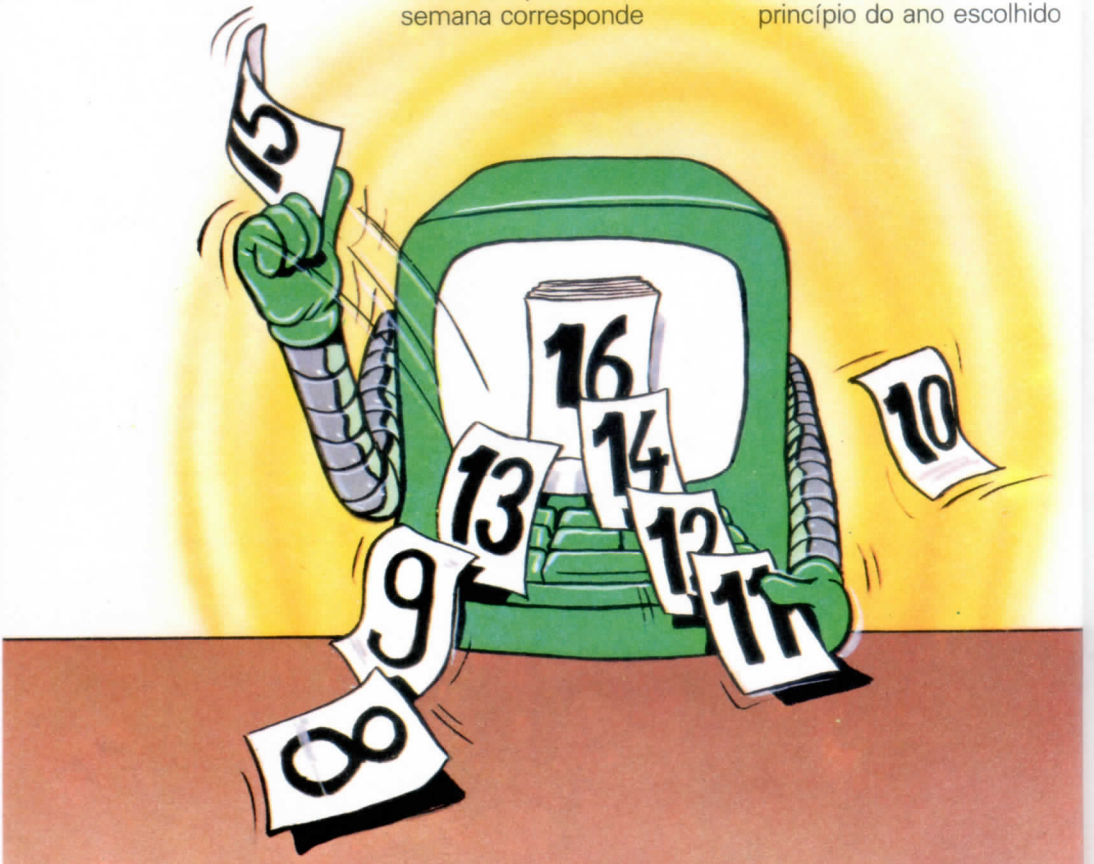
terá como único resultado que se imprima 1000 vezes o valor 27, dado que o RESTORE situado na linha 30 impede o ponteiro dos DATA de avançar até ao término seguinte.
Agora experimenta tirar a linha 30 e pôr o programa em marcha: se tinhas dúvidas sobre o funcionamento de RESTORE, desaparecerão imediatamente.
Se se desejar RESTORE pode ser seguido por um número de linha. Neste caso o ponteiro, em vez de ser deslocado ao primeiro elemento da primeira DATA, é deslocado ao primeiro elemento da DATA correspondente ao número especificado.

Sintaxe da instrução

RESTORE (número de linhas)

Calendário

O primeiro exemplo da nossa lição é uma útil aplicação sobre o uso das intruções READ/DATA. Como resultado da execução deste programa obtém-se a impressão de um calendário de qualquer ano a partir de 1984. Vejamos como. Em primeiro lugar é preciso conhecer que dia da semana corresponde ao primeiro dia do ano escolhido. Isto pode conseguir-se de uma forma muito simples, tendo bem presente uma data em concreto: por exemplo, o dia 1 de Janeiro de 1984 era um Domingo.
A partir daqui basta calcular os dias que passaram desde o princípio do ano escolhido



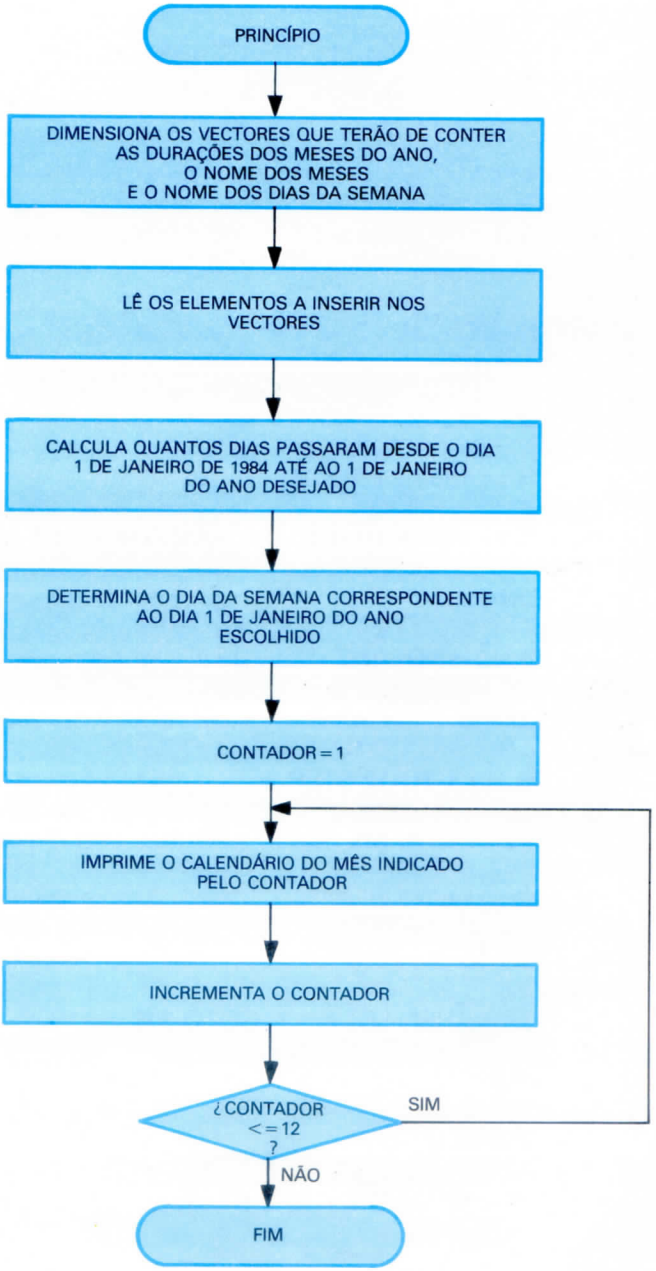
e o princípio de 1984 (e tendo em conta, como é natural, os anos bissextos) para saber automaticamente em que dia da semana calha o primeiro dia de Janeiro do ano escolhido. Tomemos por exemplo, 1986. Entre o 1 de Janeiro de 1984 e o 1 de Janeiro de 1986 há 731 dias (365 + 366, visto que 1984 era bissexto). Agora basta dividir 731 por 7.

731 : 7 = 104 e restam 3

para saber que o 1 de Janeiro de 1986 será uma Quarta-feira. O três do resto diz-nos precisamente que o princípio do 86 será três dias depois que o princípio do 84: portanto (e com uma fórmula talvez não muito correcta).

DOMINGO + 3 = QUARTA-FEIRA

A partir de agora o caminho é bastante fácil: bastará imprimir, um após outro, os meses do ano, desde que a paginação seja correcta, para obter o calendário.

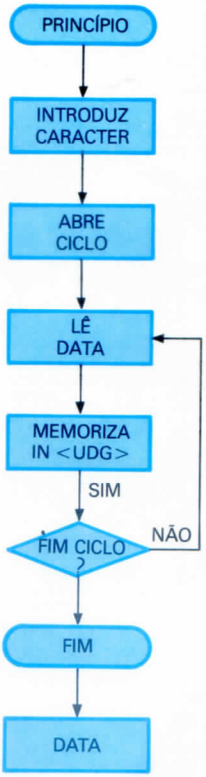


```
 10 RESTORE
 20 DIM M (12): DIM MS (12, 10): DIM DS (7, 1)
 30 FOR I = 1 TO 7: READ DS (I): NEXT I
 40 FOR I = 1 TO 12: READ M (I), MS (I): NEXT I
 50 REM
 60 REM CALCULA O PRIMEIRO DIA DO ANO
 70 INPUT "ANO"; AN
 80 INPUT "1 – ECRAN/2 – IMPRESSORA"; LINE Z$
 90 IF CODE Z$ < 49 OR CODE Z$ > 50 OR Z$ = "" THEN GOTO 80
100 IF Z$ = "2" THEN OPEN # 2, "P"
110 IF Z$ = "1" THEN OPEN # 2, "S"
120 LET NG = 0
130 IF AN < 1985 THEN GOTO 70
140 CLS
150 REM ENCONTRA OS DIAS
160 FOR I = 1984 + 1 TO AN
170 LET ND = ND + 365
180 IF (I/4 = INT (I/4)) AND (I <> AN) THEN LET ND = ND + 1:
    REM ANO BISSEXTO
190 NEXT I
200 REM ENCONTRA DIA DA SEMANA
210 LET ND = ND – (INT (ND/7)) * 7
220 LET J = ND + 1
230 REM
240 IF AN/4 = INT (AN/4) THEN LET M (2) = 29
250 FOR H = 1 TO 12 STEP 2
260 FOR I = H TO H + 1
270 PRINT TAB 4; M$ (I); "  "; AN
280 PRINT: PRINT "  ";
290 FOR K = 1 TO 7
300 PRINT G$ (K); "  ";
310 NEXT K
320 PRINT
330 IF J = 1 THEN LET K = 1: GOTO 370
340 FOR K = 1 TO J – 1
350 PRINT "  ";
360 NEXT K
370 LET J = K
380 FOR K = 1 TO M (I)
390 PRINT "0" AND K < 10"; K"  ";
400 LET J = J + 1
```

```
410 IF J=8 THEN LET J=1:PRINT
420 NEXT K
430 PRINT '''
440 NEXT I
450 IF Z$="1" THEN PRINT # 0; AT 1, 8; "PRIME UMA TECLA":
    PAUSE 0:CLS
460 NEXT H
470 CLOSE # 2:STOP
480 DATA "SEGUNDA-FEIRA", "TERÇA-FEIRA", "QUARTA-FEIRA",
    "QUINTA-FEIRA", "SEXTA-FEIRA", "SÁBADO", "DOMINGO"
490 DATA 31, "JANEIRO", 28, "FEVEREIRO", 31, "MARÇO", 30, "ABRIL",
    31, "MAIO", 30, "JUNHO"
500 DATA 31, "JULHO", 31, "AGOSTO", 30, "SETEMBRO", 31,
    "OUTUBRO", 30, "NOVEMBRO", 31, "DEZEMBRO"
```

Definição de um caracter

O programa seguinte permite-te definir um caracter gráfico e memorizá-lo numa das letras (do A ao U) reservadas para os U.D.G. Para desenhar um determinado caracter, põe em prática as técnicas explicadas anteriormente e depois insere na linha 60 da lista os valores decimais correspondentes.




```
10 INPUT CS
20 FOR I=0 TO 7
30 READ N
40 POKE USR CS+I, N
50 NEXT I
60 DATA 60, 66, 129, 129, 129, 129, 66, 60
```

EXERCÍCIOS

¿ Que palavras (constantes da cadeia) serão impressas?

```
 10 READ A: READ B: RESTORE A*B: READ A$
 20 PRINT A$
 30 STOP
 90 DATA 10, 13
100 DATA "VIDEO"
110 DATA "BASIC"
120 DATA "SOFTIDEIA"
130 DATA "JACKSON"
140 DATA "SINCLAIR"
150 DATA "SPECTRUM"
160 DATA "16 K"
170 DATA "48 K"
180 DATA "PLUS"
190 DATA "QUANTUM LEAP"
```

Muitos discursos de homens políticos parecem ter uma estrutura baseada neste programa; se trocássemos as linhas DATA por palavras como "conjuntura", "plataforma", etc, obteríamos um claro exemplo disso.

```
 10 RESTORE
 20 FOR I=1 TO 2
 40 LET A=INT (RND*10)
 50 FOR K=1 TO A
 60 READ Z$: NEXT K
 70 IF I=1 THEN READ A$
 80 IF I=2 THEN READ B$
 90 RESTORE 150: NEXT I
100 PRINT A$; " "; B$
110 PRINT # 0; AT 1, 0; "MAIS OUTRA? (S/N)": LET Z$=INKEY$: IF Z$=" "
    THE GOTO 110
120 IF Z$="S" THEN RUN
130 STOP
140 DATA "A GIRAFA", "O HOMEM", A "VACA", "O BURRO ", "O CAVALO",
    "O ELEFANTE", "O CÃO", "O GATO", "A GALINHA", "O COELHO"
150 DATA "RI COMO UMA HIENA", "EXPLODE", "DÁ SALTINHOS",
    "RODOPIA", "SALTA", "FOGE", "RELINCHA", "BARRINHA",
    "MUGE", SORRI"
```

BOLETIM DE ASSINATURA

Envie a **EDIÇÕES LATINAS – VIDEOBASIC**
Av. Almirante Reis, 219, 3.º-Esq. • 1000 LISBOA

Desejo subscrever o curso completo de VIDEOBASIC por 7.500$00.

Junto envio cheque n.º_____ Banco_____

Nome

Morada

Cód. Postal Cidade

(Em vez de cortar o cupão tire uma fotocópia)

Timex Computer 2068 Personal Color

CARACTERÍSTICAS

- Escrita automática de instruções de comando
- Verificação imediata de erros
- Memória de 72 K expansível através de Cartridge com 56 K (Rom)
- Utiliza TIMEX COMMAND Cartridges
- Linguagem de programação BASIC
- Apresentação de Textos e Gráficos
- Capacidades em tratamentos de Texto, Cor e Som
- Ficha de expansão para periféricos
- 42 Teclas tipo Máquina de Escrever

Sensacional!...
...dois computadores num só!

Importante:
...a cartridge emuladora permite-lhe utilizar todo o software existente para o ZX Spectrum.®

TIMEX COMPUTER 2068

® ZX Spectrum é uma marca registada de Sinclair Research, Lda.