CURSO PRÁTICO 63 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

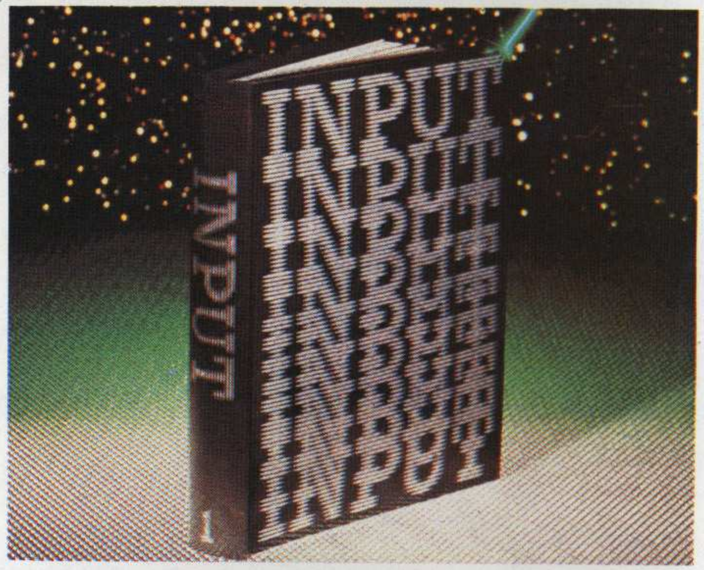
NOVA CULTURAL

Cz$ 50,00

INPUT

Vol. 5 — Nº 63

NESTE NÚMERO



PLANO DA OBRA

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

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o nº (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

Atenção: Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

Obs.: Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO

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

NOVA CULTURAL

Editor
VICTOR CIVITA

REDAÇÃO
Diretor Editorial: Carmo Chagas

Editores Executivos: Antonio José Filho, Berta Sztark Amar

Editor Chefe: Paulo de Almeida
Editora de Texto: Ana Lúcia B. de Lucena
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi
Secretária de Redação/ Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES

Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP
Tradução, adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

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

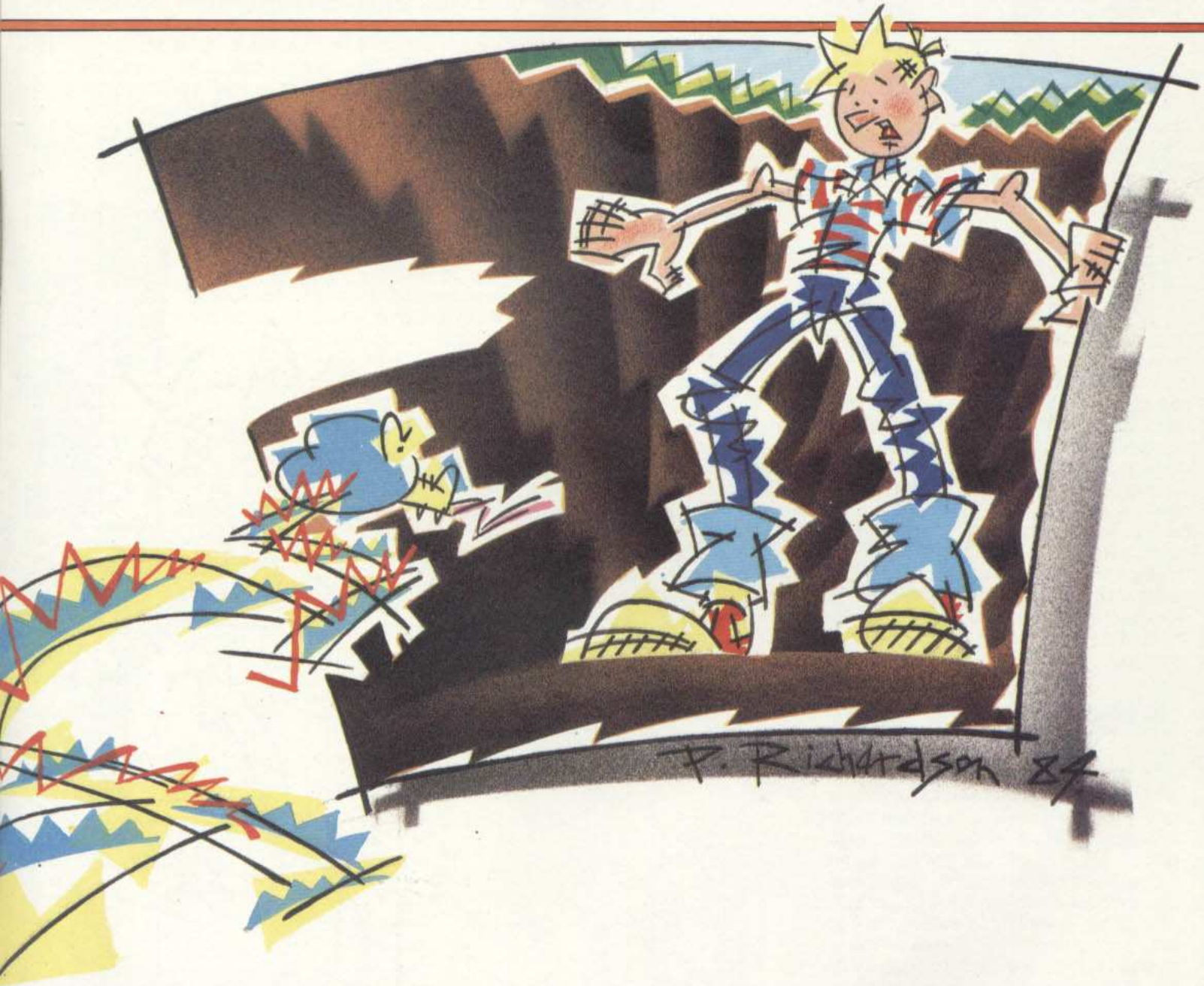
PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atílio Roberto Bonon

Preparador de Texto/Coordenador: Eliel Silveira Cunha
Preparadores de Texto: Alzira Moreira Braz, Ana Maria Dilguerian, Levon Yacubian, Luciano Tasca, Maria Teresa Galluzzi, Maria Teresa Martins Lopes, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria de Fátima Cardoso, Nair Lucia de Britto
Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, nº 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

AVALANCHE: AS COBRAS VIVEM!

- NÍVEL DO JOGO
- O ATRASO DAS COBRAS
- MOVIMENTO DAS LÍNGUAS
- A COBRA ESTÁ SUBMERSA?
- SINCRONISMO



As cobras ainda não constituem ameaça: podem ser vistas dentro dos buracos, mas estão sempre quietas. Agora vamos dar-lhes movimento, complicando um pouco mais a vida do pobre Willie.

Nossa aventura está bem movimentada: Willie anda e salta, as pedras rolam, os pássaros voam. As cobras, porém, permanecem totalmente imóveis em seus buracos. Está na hora de lhes dar um pouco de vida.

S

A rotina a seguir atiça as cobras, fazendo-as sair de sua inatividade.

```
10 REM org 59823
20 REM snk ld a,(57344)
30 REM cp 2
40 REM jr nc,sko
50 REM ret
60 REM sko ld ix,57350
70 REM ld hl,425
80 REM call skm
90 REM ld hl,369
100 REM call skm
110 REM ld hl,282
120 REM call skm
130 REM ret
```

```
140 REM skm push hl
150 REM ld de,(57354)
160 REM sbc hl,de
170 REM pop hl
180 REM jr c,sns
190 REM ret
200 REM sns ld a,(ix+0)
210 REM inc a
220 REM res 4,a
230 REM ld (ix+0),a
240 REM inc ix
250 REM cp 7
260 REM jr nc,sco
270 REM ret
280 REM sco ld bc,57224
290 REM ld d,43
300 REM cp 15
310 REM jr nz,ste
320 REM ld bc,15616
330 REM ld d,45
340 REM ste ld a,d
350 REM call 58217
360 REM ret
```

A primeira tarefa da rotina consiste em verificar se as cobras devem entrar em ação. Para isso, o valor do nível do jogo, armazenado em 57344, é carregado no acumulador e comparado com 2. Essa comparação é feita por meio de uma subtração cujo resultado não armazenamos. Se compararmos 2 com um número menor, teremos um resto. A baliza *carry* será então ajustada com 1 e o processador retornará, porque as cobras não são necessárias.

Se a baliza carry não foi ajustada, a instrução **jr nc,sko** pula a instrução **ret**. Você estará no terceiro ou quarto nível do jogo (valor 2 ou 3) e as cobras devem ser atiçadas.

AS COBRAS AMEAÇAM

A posição de memória 57350 carrega o primeiro dos atrasos das cobras, que indicam quando elas devem mexer a língua. IX é carregado com esse endereço, e HL, com 425, a posição da primeira língua de cobra. A rotina **skm**, que executa o movimento, é chamada.

A posição da língua da próxima cobra é carregada em HL, e **skm** volta a ser chamada. Em seguida, a posição da língua da terceira cobra é carregada em HL e **skm** é chamada mais uma vez.

Quando todas as línguas estiverem se mexendo, a rotina retorna.

SE A MARÉ SOBE...

Quando a rotina **skm** é chamada, as posições das línguas são colocadas na pilha para armazenamento temporário. A posição do mar em 57354 é colocada no par DE. Neste ponto do programa, precisamos verificar a altura da maré. Se a cobra estiver submersa, a parte da rotina que movimenta sua língua pode ser pulada.

A seguir, a posição do mar no par DE é subtraída da posição da língua em HL. Se não houver resto, a posição do mar se encontra acima da posição da língua da cobra na tela. Nesse caso, a instrução **jr c,sns** não tem efeito e o processador retorna. Mas, se houver resto, ou seja, se a posição do mar estiver abaixo da posição da língua na tela, a instrução **jr c,sns** faz com que o processador salte a instrução **ret** e continue a rotina.

ESTÁ NA HORA DE MEXER?

O atraso da cobra é carregado da posição apontada por IX para o acumulador, onde é incrementado. O bit 4 é apagado para evitar que o atraso se torne maior que 15. O resultado dessa operação é armazenado de volta na posição indicada por IX. Esse apontador é então incrementado para indicar o atraso da cobra seguinte. Seu valor é comparado com o conteúdo do acumulador pela instrução **cp 7**. Se o atraso dessa cobra for maior ou igual a 7, **jr nc,sno** manda o processador para a rotina que imprime a língua. Se o valor for menor que 7, a baliza carry é ajustada com 1, a instrução não tem efeito e o processador retorna.

Cada vez que essa rotina é chamada, um atraso diferente da cobra é encontrado nos endereços 57350, 57351 e 57352, pois o apontador IX é incrementado entre cada chamada que se faz à rotina **skm**. A língua da cobra fica para fora oito ciclos, e não aparece nos oito ciclos seguintes.

IMPRESSÃO

Chegamos, finalmente, à rotina encarregada de imprimir ou apagar a língua da cobra na tela. O par de registros BC é carregado com 57224, os dados para a língua da cobra. D é ajustado com 43, a cor da figura. O atraso da cobra em A é comparado com 15.

Se o atraso não tiver chegado a 15, a instrução **jr nz** leva o processador pa-

ra **ste**. Se for igual a esse valor, você precisará apagar a língua, para que ela não apareça nos próximos oito ciclos. BC é carregado com 15616, o endereço dos dados de um espaço vazio, e o registro D é carregado com 45, o código da cor do céu.

Qualquer que seja o valor do atraso da cobra (neste ponto da rotina, ele é sempre maior do que sete), o processador encontra a instrução **ld a,d**. Depois que ela carrega a cor apropriada em A, a rotina **print**, em 58217, é chamada. Com isso, imprimimos a língua — ou apagamos a que existia — para fora da boca de uma das cobras. A seguir, o processador retorna.

T

Esta rotina movimenta as cobras, acrescentando ação ao jogo.

```
10  ORG 20856
20  SNK LDA 18238
30  CMPA #2
40  BHS SKO
50  RTS
60  SKO LDY #18255
70  LDX #5095
80  JSR SKM
90  LDX #4591
100  JSR SKM
110  LDX #3833
120  JSR SKM
130  RTS
140  SKM PSHS X
```

MICRO DICAS

PROGRAMAS LONGOS: MELHORE A VELOCIDADE DE MONTAGEM

Como o leitor deve ter notado, apresentamos o programa *Avalanche* em pequenos segmentos funcionais, que podem ser testados separadamente. Essa técnica de construção de programas, chamada de *desenvolvimento modular*, é muito útil para qualquer tipo de programa, pois agiliza o processo de montagem e facilita enormemente os testes de execução. Para a programação complexa em linguagem de máquina, o uso dessa técnica é essencial, devido à dificuldade de se documentar internamente — isto é, na própria listagem do programa — o código em Assembler.

Os profissionais que criam os sofisticados videogames que vemos em fliperamas, por exemplo, dominam perfeitamente a técnica de modularização. Na realidade, eles dispõem de programas Assembler poderosíssimos, capazes de representar uma biblioteca de rotinas e ferramentas de programação que facilitam muito a implementação dos recursos normalmente usados em videogames.

Infelizmente, os Assembler que *INPUT* apresentou para os diferentes micros são muito lentos. Se você quiser melhorar a velocidade de montagem, divida um programa mais longo em segmentos menores e compile-os separadamente. Não se esqueça, porém, de calcular o endereço de origem para cada segmento e incluí-lo em um comando **org** ou equivalente, no começo de cada segmento.

```
150   LDX 18247
160   LEAX 31,X
170   CMPX ,S
180   PULS X
190   BHI SNS
200   RTS
210   SNS LDA ,Y
220   INCA
230   ANDA #$F
240   STA ,Y+
250   CMPA #7
260   BHS SCO
270   RTS
280   SCO LDU #18062
290   CMPA #15
300   BNE STE
310   LDU #1536
320   STE LEAX -256,X
330   JSR CHARPR
340   RTS
350   CHARPR EQU 19402
```

A primeira tarefa da rotina é verificar se as cobras devem entrar em ação. Para isso, o valor do nível do jogo, armazenado em 18238, é carregado no acumulador e comparado com 2.

Se você está no nível três ou quatro — ou seja, se o valor é 2 ou 3 —, as cobras serão necessárias. Nesse caso, a instrução **BHS** faz o processador pular o **RTS**, seguindo a rotina.

AS COBRAS AMEAÇAM

Y é carregado com o endereço do atraso da primeira cobra. Esta é a variável incumbida de interromper ou começar o seu movimento. Os atrasos da segunda e da terceira cobras estão logo em seguida na memória.

X é carregado com 5095, a posição na tela do primeiro buraco, e o processador vai para a rotina **SKM**.

Essa rotina atiça a primeira cobra e incrementa o conteúdo de Y antes de retornar. Assim, quando o processador salta para fazer com que a segunda e a terceira cobras se movimentem, apenas as posições do segundo e do terceiro buracos precisam ser carregadas em X. Y é incrementado automaticamente, apontando os atrasos das três cobras. Quando tiver passado por todas elas, o processador retorna.

MARÉ ALTA

A primeira coisa que o processador faz ao entrar na rotina **SKM** é colocar na pilha, para armazenamento temporário, a posição da cobra na tela.

O registrador X é carregado com o conteúdo de 18247, a posição do mar. Esse valor é somado com 31, para que o apontador de tela se mova para o canto direito da tela. X é então comparado com a posição do buraco da cobra, que está na pilha de máquina, através da instrução **CMPX ,S**. A posição da cobra é recuperada da pilha, voltando para o registrador X.

Como isso não afeta nenhuma das balizas, a instrução BHI ainda se refere à operação **CMPX ,S**. Assim, se o mar está acima do buraco — ou seja, se a posição do mar na tela não é maior do que a posição do buraco que estava na pilha —, o salto não ocorre, e o processador retorna.

Caso o mar não tenha encoberto o buraco — isto é, se a posição do mar na tela é maior do que a posição do buraco —, o processador pula a instrução **RTS**, e dá continuidade à rotina que atiça as cobras.

MEXER OU NÃO MEXER

A é carregado com o conteúdo do atraso da cobra que está apontado no registrador Y. Esse valor é então incrementado e a operação AND é feita com $F, para ajustá-lo com 0 toda vez que ultrapassar o valor 15.

O resultado é armazenado de volta no endereço apontado por Y. Esse registrador é então incrementado para apontar o atraso da próxima cobra.

O atraso da cobra que ainda está em A é comparado com 7. Se for maior ou igual, o processador vai para a rotina de impressão da língua; caso contrário, o processador retorna.

O BOTE

Ao entrar na rotina **SCO**, o processador carrega o apontador da pilha do usuário com 18062, o endereço dos dados para a língua da cobra.

A — que ainda contém o atraso da cobra que foi incrementado — é comparado com 15. Se ele não foi incrementado até esse valor, o processador pula para as instruções de impressão. Mas, se já chegou a 15, U é carregado com 1536, o endereço da parte de céu no canto superior esquerdo da tela.

Seguindo adiante, X é subtraído de 256, fazendo o apontador de tela se mover um caractere para cima do buraco da cobra. Esta é a posição da língua.

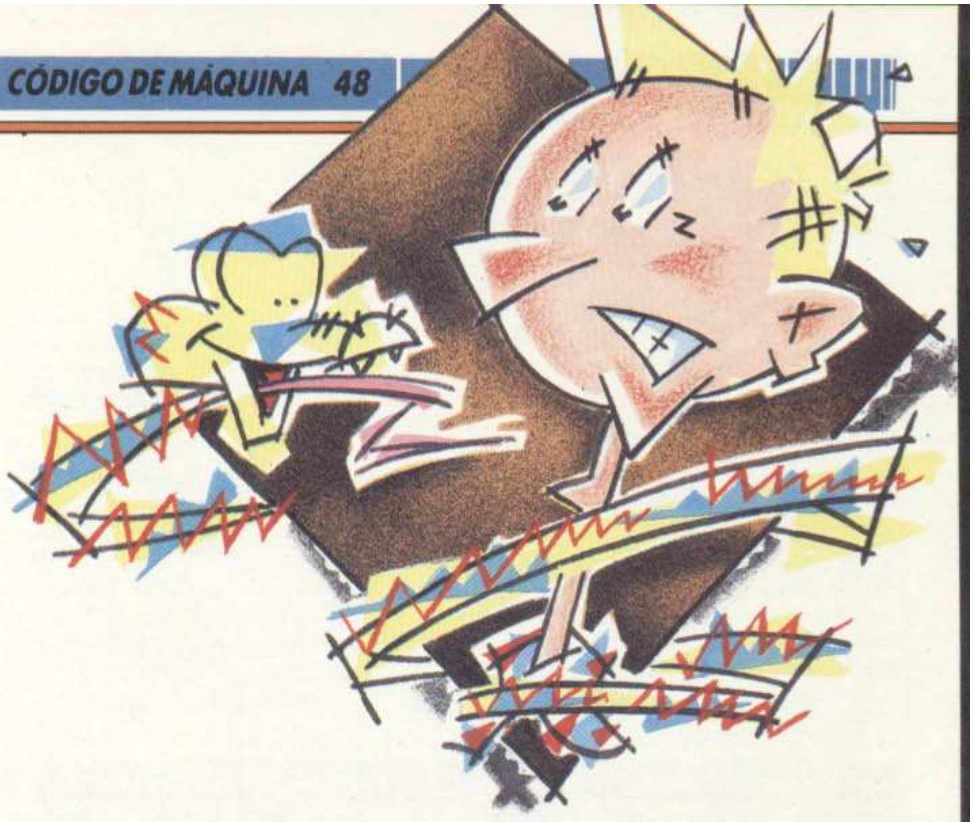
Se o atraso da cobra estiver entre 7 e 14, a rotina **CHARPR** imprimirá a língua nessa posição ao ser chamada. Mas, se o atraso tiver atingido o valor 15, a língua será apagada.

A língua permanecerá invisível, isto é, não será impressa, até que o atraso chegue novamente a 7.

A rotina apresentada a seguir atiça as cobras que até agora permaneciam quietas em seus buracos.

```
10    org 55564
20    ld a,(-5228)
30    cp 2
40    jr nc,sk
50    ret
60    sk ld bc,-5198
70    ld hl,425
80    call sm
90    ld hl,369
100    call sm
110    ld hl,282
120    call sm
130    ret
140    sm push hl
150    ld de,(-5212)
```

```
160    sbc hl,de
170    pop hl
180    jr c,sn
190    ret
200    sn ld a,(bc)
210    inc a
220    res 4,a
230    ld (bc),a
240    inc bc
250    cp 7
260    jr nc,sc
270    ret
280    sc ld de,(62407)
290    add hl,de
300    cp 15
310    ld b,36
320    jr nz,st
330    ld b,255
340    st ld a,b
350    call 77
360    ret
370    end
```



A primeira tarefa da rotina consiste em verificar se as cobras devem entrar em ação no nível atual do jogo. Para isso, o valor do nível, −5228, é carregado no acumulador e comparado com 2. Se você está no terceiro ou no quarto nível — ou seja, se o valor é 2 ou 3 — as cobras estão nos buracos e devemos atiçá-las.

A comparação é feita por meio de uma subtração cujo resultado não armazenamos. Se compararmos 2 com um número menor, teremos um resto e a baliza carry passará a conter 1. Nesse caso, o processador retornará, porque as cobras não são necessárias.

Caso a baliza carry não tenha sido afetada, a instrução **jr nc,sk** pula a instrução **ret**, porque as cobras estão presentes neste nível.

AS COBRAS AMEAÇAM

O endereço de memória −5198 contém o primeiro dos chamados atrasos das cobras, que indicam quando elas devem mostrar sua língua. O par BC é carregado com esse endereço, e HL, com 425, a posição da língua da primeira cobra. A rotina **sm**, encarregada de promover o movimento, é chamada.

A posição da língua da próxima cobra é carregada em HL, e **sm** volta a ser chamada. Em seguida, a posição da língua da terceira cobra é carregada em HL, e **sm** é chamada mais uma vez.

Depois de movimentar todas as línguas, a rotina retorna.

MARÉ ALTA

Quando a rotina **sm** é chamada, a posição da língua que está em HL é colocada na pilha para armazenamento temporário. A posição do mar em −5212 é colocada no par DE. Neste ponto do programa, precisamos verificar se a cobra se afogou com a subida da maré. Em caso afirmativo, ela não precisa mexer a língua — assim, a parte da rotina que faz isso pode ser pulada.

Para essa verificação, a posição do mar no par DE é subtraída da posição da língua em HL. Se não houver resto — ou seja, se a posição do mar estiver acima da posição da língua da cobra na tela —, a instrução **jr c,sn** não tem efeito e o processador retorna. Mas, se houver resto, a posição do mar está abaixo da posição da língua na tela. A instrução **jr c,sn** faz, então, o processador saltar a instrução **ret** e continuar a rotina.

ESTÁ NA HORA DE MEXER?

O atraso da cobra é carregado da posição apontada por BC para o acumulador, onde é incrementado. O bit 4 também é incrementado para evitar que o atraso se torne maior que 15. O resultado dessa operação é armazenado de volta na posição apontada por BC.

O par de registros BC é incrementado para apontar o atraso da cobra seguinte. A instrução **cp 7** compara esse valor com o conteúdo do acumulador, e a instrução **jr nc,sc** manda o processador para a rotina que imprime a língua, se o atraso da cobra for maior ou igual a 7. Caso o valor seja inferior a 7, a baliza é ajustada com 1, a instrução de desvio não tem efeito e o processador retorna.

Ao ser chamada, essa rotina sempre encontra um atraso diferente nos endereços −5198, −5199 e −5220, já que o apontador BC é incrementado entre cada acesso a **sm**. A língua da cobra fica para fora oito ciclos, não aparecendo nos oito seguintes.

IMPRESSÃO

Chegamos, finalmente, à rotina que imprime ou apaga a língua da cobra na tela. O endereço inicial da Tabela de Nomes da VRAM é carregado em DE e somado a HL, que contém a posição da língua. Esse par de registros passa então a conter o endereço correspondente à posição da língua na TN.

O atraso da cobra que ainda está no acumulador é comparado com 15. O registro B é carregado com 36, o código do padrão da língua. Se o valor do atraso não for 15, a instrução **jr nz,st** leva o processador para **st**, pulando a instrução **ld b,255**. Caso contrário, a língua deve ser apagada para que não apareça nos próximos oito ciclos; o desvio não ocorre e 255, o código do padrão de céu, é colocado em B.

Neste ponto do programa, o valor do atraso da cobra será sempre superior a 7. O processador encontra a instrução **ld a,b**, que transfere o código do padrão apropriado de B para A. A rotina 77 da ROM, que coloca o código que está em A na posição da TN apontada por HL, é chamada. Com isso, imprimimos a língua — ou apagamos a que existia — para fora da boca da cobra. A seguir, o processador retorna ao laço principal do jogo.

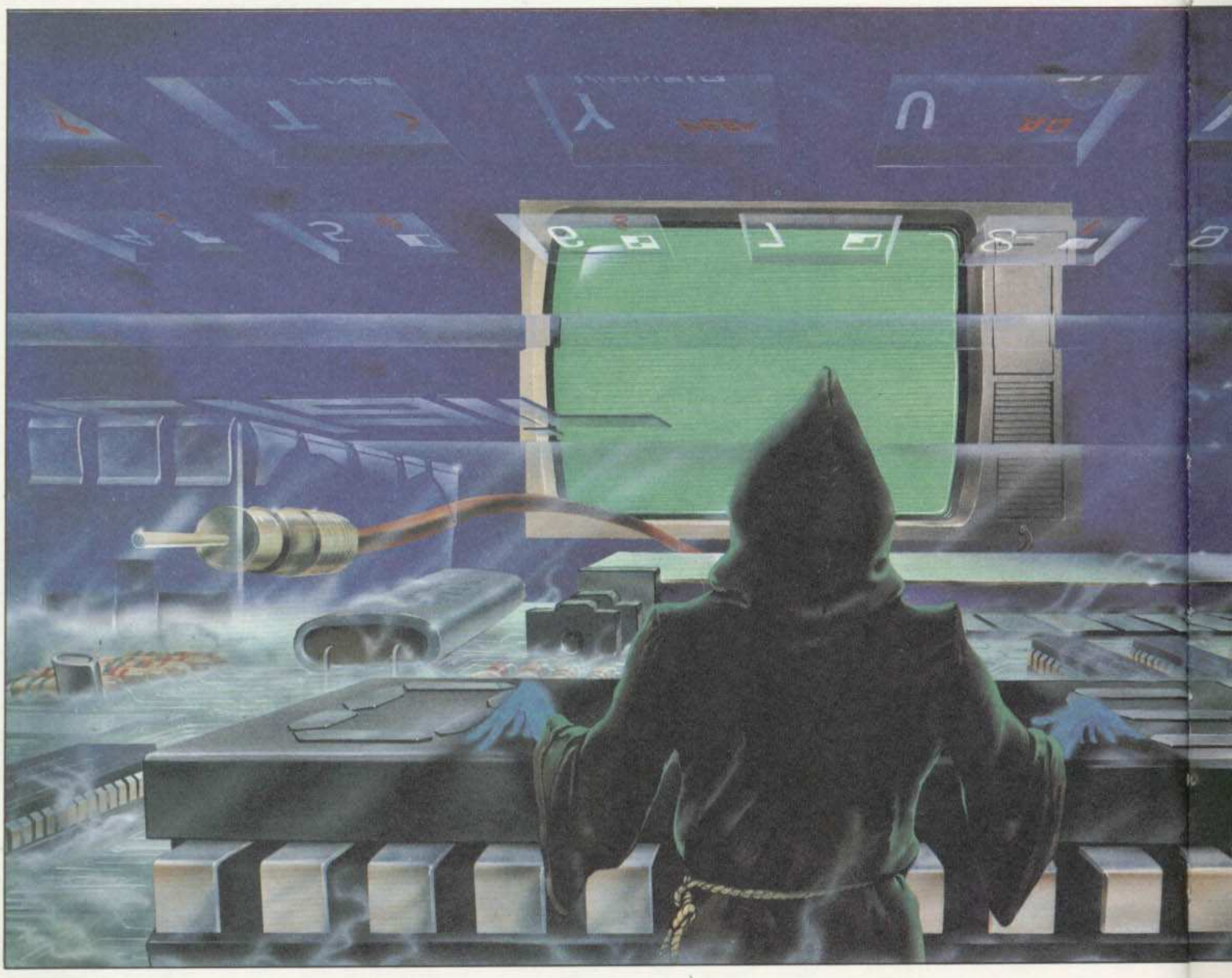
O SISTEMA OPERACIONAL

Sem o sistema operacional, seria difícil trabalhar com o computador. Veja de que modo funciona esse importante componente do seu micro e aprenda a fazer dele um aliado.

Todos os computadores trabalham e entendem apenas uma linguagem: o código de máquina, que é representado internamente por pequenas variações binárias de voltagem. Ele pode ser expresso por um conjunto de letras e números, o que facilita sua compreensão, ou por números binários, menos familiares ao usuário. De uma forma ou de outra, essa linguagem é considerada pouco acessível por muitas pessoas. O que torna mais simples a comunicação com o operador é o Sistema Operacional do computador (SO), um conjunto de rotinas em código de máquina que controlam todas as funções da máquina. Seja qual for a linguagem empregada, BASIC ou outra qualquer, essas rotinas serão sempre utilizadas para controlar as diversas funções do computador.

Conhecer o funcionamento do SO não é essencial para o usuário, porém amplia bastante suas alternativas de programação. O uso de funções internas do SO possibilita a obtenção de melhores resultados não só quanto à velocidade, mas, também, quanto à qualidade geral do programa.

Dentro dos limites impostos por cada máquina, este artigo procura mostrar



algumas aplicações úteis para as rotinas e variáveis próprias de cada sistema operacional.

O QUE É SISTEMA OPERACIONAL

O SO é apenas um tipo sofisticado de programa — escrito em código de máquina — que permite ao microprocessador existente dentro da máquina responder aos comandos básicos de operação. Ele controla a comunicação da máquina com o mundo exterior — o teclado, a tela, o alto-falante e outras portas de entrada e saída.

O SO tem ainda a função de assegurar que a memória seja utilizada de forma eficiente pelo usuário e pelo interpretador BASIC. Assim, quando ligamos o computador, ele ativa várias de suas rotinas da ROM para acertar os valores iniciais de variáveis e apontadores e, em seguida, avisa que está pronto a aceitar comandos.

Em todos os microcomputadores abordados em *INPUT*, o SO é responsável pela interpretação dos comandos em BASIC, contudo ele pode também lidar com outros tipos de linguagem.

Na maior parte do tempo, o trabalho do SO consiste em verificar e modificar os valores de determinadas variáveis e apontadores especiais. É importante, por exemplo, que o SO conheça o endereço exato tanto do início como do fim de um programa em BASIC. Se o programa for modificado, seu tamanho também mudará, e esses valores precisarão ser atualizados.

O mesmo se aplica ao processo de alocação e manutenção de espaço para as variáveis, particularmente as variáveis indexadas, ou matrizes. Quando dimensionamos uma variável desse tipo, um espaço deve ser reservado — e, quando a variável não for mais necessária, ele deve ser eliminado.

É essencial que o SO utilize racionalmente a memória, pois, caso contrário, um programa longo ou algumas variáveis mais compridas logo esgotarão a quantidade de espaço disponível. Essa administração da memória é chamada de *housekeeping* — que poderíamos traduzir livremente por "cuidar da casa". Uma boa "economia doméstica" é indispensável para que o computador funcione eficientemente.

O SO contém ainda uma série de rotinas de software montadas de modo que tanto o usuário quanto o próprio SO possam utilizá-las com facilidade. A maioria delas não é de interesse para o programador em BASIC, pois vários comandos dessa linguagem se encarregam da execução das mesmas tarefas, chamando as rotinas apropriadas. Um bom exemplo é o comando **INPUT**, que usa uma série de sub-rotinas do SO, tais como seleção e leitura da porta de entrada, varredura do teclado, uso e transferência de buffer etc.

Outro exemplo interessante é **CLEAR**. Esse comando força o SO a executar todas as rotinas necessárias à liberação da memória RAM disponível para o usuário, destruindo todas as variáveis previamente existentes.

TRABALHO SOB ENCOMENDA

Sempre que digitamos um comando, o SO seleciona e coloca em ação uma das rotinas de seu vasto acervo. Suponhamos que você tecle a letra A: o SO deverá instruir o processador a imprimir o caractere na tela, cabendo-lhe também detectar a tecla que foi pressionada. A forma como ele faz isso varia de computador para computador. Em certas máquinas, o SO "varre" periodicamente o teclado para verificar se alguma tecla foi pressionada. Em outras, seu funcionamento normal é interrompido sempre que se pressiona uma tecla, a qual é identificada, em seguida, por meio de uma varredura.

Para responder adequadamente à pressão da tecla, o SO aciona a sub-rotina que manda o caractere correspondente à tela. Sua execução, assim como a de outras rotinas em código de máquina, é muito rápida. Se pressionarmos a tecla <**ENTER**> ou a tecla <**RETURN**>, o SO ativará uma rotina que promove a mudança de linha na tela (*new line*).

O INTERPRETADOR BASIC

Quando executamos um programa codificado em uma linguagem como o BASIC, as instruções que estão na memória são interpretadas, ou seja, traduzidas para os códigos que o computador pode entender. Esses códigos modificam os registradores de modo que as rotinas do SO possam ser chamadas adequadamente. Assim, as mesmas rotinas que possibilitam a execução de comandos

entrados pelo teclado permitem ao SO a execução de um programa BASIC.

Mas por que o BASIC é tão lento, já que utiliza rotinas em código, que são rápidas? Acontece que a linguagem deve ser interpretada a partir de palavras-chave — como **PRINT** — e de outros símbolos, antes que essas rotinas internas possam ser executadas. Como a tradução do BASIC é demorada, programas que exigem rapidez, como os de videogame, por exemplo, são escritos diretamente em linguagem de máquina.

O tempo que se leva para escrever um programa em código de máquina é, em média, dez vezes maior do que o gasto em um programa BASIC. A velocidade de execução do mesmo, contudo, chega a ser cinqüenta vezes maior. Assim, a não ser que a rapidez de execução seja muito importante, a maioria das pessoas prefere utilizar o BASIC, dada a facilidade de programação.

ACESSO DIRETO AO SO

A possibilidade de ter acesso direto ao SO, sem passar pelo interpretador, seria a solução ideal para quem prefere as facilidades do BASIC, mas, por outro lado, não quer abrir mão da velocidade. Nem todas as máquinas, porém, oferecem essa alternativa.

No TRS-80 e no TRS-Color, por exemplo, o SO faz parte do interpretador de dezesseis Kbytes — o Microsoft BASIC. No MSX, além de um conjunto de rotinas básicas, chamado BIOS (*BASIC Input/Output System*), o SO inclui também o interpretador. Os micros da linha Sinclair, como o ZX-81 e o Spectrum, são ainda mais restritivos, uma vez que não permitem acesso aos registradores internos diretamente do BASIC (o que é essencial para que se possa selecionar as rotinas existentes no SO).

Seja qual for o microcomputador e o tipo de acesso que oferece ao SO, sempre é possível contornar diversos problemas — inclusive a diminuição do tamanho das rotinas em código — por meio de alguns truques.

VARIÁVEIS DO SISTEMA

Os usuários do Spectrum podem utilizar as rotinas do SO através do comando **USR**, que também serve para executar rotinas em código de máquina colocadas temporariamente na RAM. Tente usar, por exemplo, o comando direto **RANDOM USR 0**: ele provoca um *reset* geral na máquina (reinicialização).

Para programas em BASIC, há a alternativa de se modificar determinadas variáveis por meio de comandos **POKE**.

POKE 23561, seguido de um número entre 1 e 255, altera o lapso de tempo que antecede a ativação da auto-repetição das teclas. O valor normal é dado por um **POKE 23561,35**. De modo análogo, **POKE 23562,5**, que define o valor normal do período que decorre entre duas auto-repetições sucessivas, também pode ser modificado. Tal recurso é útil em jogos, ou qualquer outro tipo de programa que exija respostas rápidas do usuário, via teclado.

As posições 23606 e 23607 contêm o endereço dos padrões de pontos dos caracteres. Se usarmos **POKE 23606,8** (byte menos significativo), o apontador será movido um caractere para cima, na tabela. Assim, qualquer letra que digitarmos aparecerá na tela como o caractere seguinte do código ASCII. Tente digitar esse **POKE** seguido de 1, 2, 3, 4. Note que este é um método simples de "criptografar" listagens em BASIC, desencorajando curiosos que tentem lê-las. Se, em vez do byte menos significativo você recorrer ao mais significativo — **POKE 23607,0** —, o apontador será dirigido para o começo da memória ROM, tornando os caracteres absolutamente ininteligíveis.

A posição de memória 23658 possibilita alterar o estado da tecla <**CAPS LOCK**> durante a execução de um programa. **POKE 23658,0** muda os caracteres de maiúsculos para minúsculos; **POKE 23658,8** admite só maiúsculos.

O comando **PLOT** permite especificar uma posição absoluta da tela, enquanto **DRAW** se refere de forma relativa à posição corrente do cursor gráfico. Se você precisar de um **DRAW** absoluto, poderá consegui-lo através das posições de memória 23677 e 23678. Para verificar seu efeito, digite **PLOT 128,85**. Esse comando colocará um ponto no centro da tela. Suponhamos que você queira traçar uma linha que vá deste ponto até o canto superior direito da tela, cuja posição absoluta é (255,175). **DRAW 255,175** indicaria um ponto fora da tela, mas **DRAW 255-PEEK 23677,175-PEEK 23678** lhe dará o resultado desejado. Ao subtrair **PEEK 23677** da coordenada X e **PEEK 23678** da coordenada Y, estamos, na verdade, efetuando uma operação para obter as coordenadas relativas do comando **DRAW**.

Um outro exemplo de utilização das rotinas do SO por meio de um comando BASIC é dado pela simulação de um relógio. No Spectrum, o contador de linhas de vídeo ocupa as posições 23672, 23673, 23674. Se colocarmos 0 nelas, usando o comando **POKE**, o contador será zerado. Depois disso, sua atualização será feita automaticamente através de interrupções. Aqui está um programa simples que emprega o contador para improvisar um relógio:

```
10 POKE 23674,0: POKE 23673,0
: POKE 23672,0
20 BORDER 0: PAPER 0: INK 6:
CLS
30 DEF FN t()=INT ((65536*
PEEK 23674+256*PEEK 23673+
PEEK 23672)/50)
40 LET t=FN t()
50 LET h=INT (t/(60*60))
60 LET m=INT (t/60)
70 LET s=t-((h*60)*60)-(m*60)
80 LET t$="["+STR$ h+":"+STR$
m+":"+STR$ s+"]"
90 PRINT AT 1,15-((LEN t$)/2)
;" ";t$;" "
100 GOTO 100
```

A linha 10 zera o contador de linhas de vídeo, enquanto a linha 30 define a função **t**, que o lê. A linha 40 armazena o valor encontrado em **t**. A partir desse valor, as horas, minutos e segundos são calculados.

ENTRADA E SAÍDA

Os microcomputadores pertencentes à linha MSX possuem cerca de 32 Kbytes de memória ROM dedicados ao SO. Os primeiros 16385 bytes são preenchidos com rotinas de entrada e saída. O interpretador BASIC começa em 16385 e termina em 32769. Existem ainda 3202 bytes dedicados às variáveis do sistema, todos eles situados no topo da memória — 62333 a 65535.

Um SO com essa extensão possui quase todas as rotinas que o usuário pode desejar. Muitas dessas rotinas têm sido aproveitadas no videogame *Avalanche*. Contudo, como a maioria delas emprega valores armazenados em registradores internos do Z-80 — o microprocessador do MSX —, poucas podem ser utilizadas diretamente por programas em BASIC. A solução consiste em escrever pequenas rotinas em código que modificam os valores dos registradores, de modo que o BASIC possa chamar a rotina desejada do SO. Um exemplo desse procedimento foi dado no artigo publicado à página 1141 de *INPUT*.

De fato, é uma pena que o BASIC não possa modificar os registradores diretamente, sobretudo porque o SO do MSX dispõe de certos artifícios projetados para facilitar a utilização de suas rotinas pelo usuário. Algumas porções

da ROM e da RAM são preenchidas com vetores que apontam para determinadas rotinas do SO, numa tentativa de simplificar seu uso.

As rotinas de entrada e saída — comunicação com a tela, com o gerador de som, com o teclado e o cassete, entre outros periféricos — estão espalhadas pelos primeiros 16K da ROM. Existe, porém, uma pequena região — que vai do endereço 59 ao 348 — que contém vetores que apontam para as principais rotinas do BIOS. Assim, podemos ter acesso a essas rotinas de duas maneiras: chamando seu endereço verdadeiro na memória ROM ou usando seu vetor na tabela. As rotinas do interpretador BASIC também são apontadas por vetores de uma tabela localizada no topo da memória — 64922 a 65535.

As rotinas mais interessantes necessitam de parâmetros colocados nos registradores internos. Como muitas delas já foram apresentadas em outros artigos publicados em *INPUT*, faremos uma pequena lista de rotinas (endereços da tabela de vetores) e endereços de variáveis úteis.

As rotinas podem ser usadas por meio dos comandos **DEFUSR** e **USR()**, e as variáveis, por meio de **PEEK** e **POKE**.

GRÁFICOS

Eis algumas rotinas de interesse na utilização da tela:

65 - Desabilita a geração de imagens na tela. Pode ser útil quando se pretende que o usuário não acompanhe a elaboração de um desenho complicado, vendo-o só depois de completo.
68 - Reabilita a geração de imagens. Faz com que o processo de desenho, que a rotina anterior oculta, apareça instantaneamente na tela.
98 - Muda as cores da tela de acordo com o valor das variáveis do sistema.
108 - Equivale a **SCREEN 0**. Os endereços das tabelas e as cores podem ser modificados através das variáveis do sistema.
111 - **SCREEN 1**.
114 - **SCREEN 2**.
117 - **SCREEN 3**.
192 - Emite um sinal sonoro.
195 - **CLS**.
207 - Mostra as teclas de função na parte inferior da tela.
204 - Apaga as teclas de função da tela.

São variáveis de especial interesse para a tela:

62387 a 62426 - Cada par de bytes corresponde ao endereço de uma **BASE**. Por exemplo, 62407 e 62408 contêm o endereço de **BASE(10)**, tabela de nomes da tela gráfica.
62431 a 62438 - Conteúdo dos registradores do VDP (*Video Display Processor*).
62441 - Cor de frente.
62442 - Cor de fundo.
62443 - Cor da borda.
64695 e 64696 - Coordenada X.
64697 e 64698 - Coordenada Y.

TECLADO E CASSETE

Apresentamos a seguir algumas rotinas de comunicação com o teclado e o gravador cassete. A maioria dessas rotinas precisa de determinados parâmetros nos registradores.

159 - Aguarda que uma tecla seja pressionada pelo usuário e devolve seu código ASCII ao registrador A. Essa rotina pode substituir a linha em BASIC que geralmente aparece depois que a mensagem "APERTE QUALQUER TECLA" é impressa:

```
100 IF INKEY$=" " THEN 100
```

225 - Lê o cabeçalho de uma gravação em fita cassete.
228 - Lê um byte da fita. O registrador A guardará o byte lido.
231 - Encerra a leitura da fita.
234 - Grava um cabeçalho na fita. Se A=0, o cabeçalho será curto; se A=1, será longo.
237 - Grava um byte na fita. O registrador A deverá conter o byte a ser gravado.
240 - Encerra a gravação.

Muitos endereços e rotinas úteis dos microcomputadores das linhas Apple e TK-2000 foram apresentados no artigo da página 261. Não voltaremos a tratar deles, nem de alguns truques já mencionados, como, por exemplo, o uso de **CALL -151** para ativar o monitor.

O SO do Apple e do TK-2000 divide-se em sistema monitor e rotina de inicialização (*Autostart ROM*). As rotinas citadas aqui são do monitor.

O SO do Apple e do TK-2000 — como o de todos os computadores que utilizam o microprocessador 6502 — tem suas variáveis armazenadas nos primeiros 256 bytes da ROM (página 0). Esses micros não permitem a modificação direta dos registradores do microprocessador 6502 através do BASIC. Assim, as rotinas que requerem parâmetros devem ser usadas com pequenas rotinas em código para leitura e/ou modificação desses registradores.

O comando **CALL** serve para chamar não só essas rotinas mas, também, as que são montadas em código, na RAM. Os endereços podem ser modificados com **POKE** e lidos com **PEEK**.

ROTINAS DO MONITOR

-528 - Imprime na tela o caractere cujo código está no acumulador (registrador A).
-384 - **INVERSE**.
-380 - **NORMAL**.
-198 - Emite um sinal sonoro. No Apple tente também -1059.
-741 - Imprime um cursor piscante e aguarda que se pressione uma tecla, colocando então seu código no acumulador. Também embaralha o gerador de números aleatórios.
-864 - Provoca um atraso de acordo com o valor do acumulador A. No Apple II a duração do atraso é de (26+27*A+5*A*A)/2 microssegundos. No TK-2000 a demora é maior.
-1948 - Determina qual será a cor do gráfico de baixa resolução de acordo com o conteúdo do acumulador.
-1953 - Adiciona o valor 3 ao código da cor atual para gráficos de baixa resolução.
-2048 - Desenha um ponto de baixa resolução. O acumulador define a coordenada vertical, e o registro Y determina a coordenada horizontal.
-2023 - Desenha uma linha horizontal em baixa resolução — coordenada vertical em A, coordenada horizontal inicial em Y e coordenada horizontal final no endereço $2C.
-2008 - Desenha uma linha vertical em baixa resolução — coordenada horizontal em Y, coordenada vertical inicial em A e coordenada vertical final no endereço $2D.
-1998 - Apaga a tela de baixa resolução. No micro Apple, preenche a tela com caracteres @ invertidos, se for chamada no modo texto.

-1994 - Igual à anterior, só que respeita as quatro linhas de texto no rodapé da tela.
-1935 - Verifica a cor de uma posição da tela de baixa resolução, usando os mesmos registros que a rotina do endereço -2048. A cor do ponto retornará no acumulador.
-182 - Grava todos os registros do microprocessador 6502. Usa os endereços $45 a $49.
-193 - Recupera os registros. Usa os endereços $45 a $49.

TELA DE ALTA RESOLUÇÃO

Os endereços que controlam o tipo de tela no Apple foram muito usados nos programas de *INPUT* e não serão repetidos. Existem, contudo, alguns outros endereços interessantes para gráficos de alta resolução.

224/225 - Coordenada X do último ponto plotado.
226 - Coordenada Y do último ponto plotado.
230 - Indica a página em que o ponto deve ser plotado. O valor 32 corresponde à página 1, e 64, à página 2. Note que o acesso a esse endereço permite que se desenhe na página 2, enquanto se mostra a página 1 e vice-versa.
228 - Indica a cor do gráfico. Seus valores estão na tabela apresentada a seguir.

HCOLOR	COR	BYTE 228
0	preto 1	0
1	verde	42
2	violeta	85
3	branco 1	127
4	preto 2	128
5	vermelho	170
6	azul	213
7	branco 2	255

62450 - Limpa a tela (alta resolução).
62454 - Preenche a tela de alta resolução com o byte 228. Além de permitir a seleção de cores de fundo para seus desenhos, essa rotina produz diversos padrões quando o byte 228 assume valores que não correspondem às cores da tabela anterior.

T

As rotinas do SO são acessadas pelo comando **EXEC**, que também chama rotinas em código montadas na RAM. Muitas variáveis são obtidas com **PEEK**.

COMUNICAÇÃO COM O CASSETE

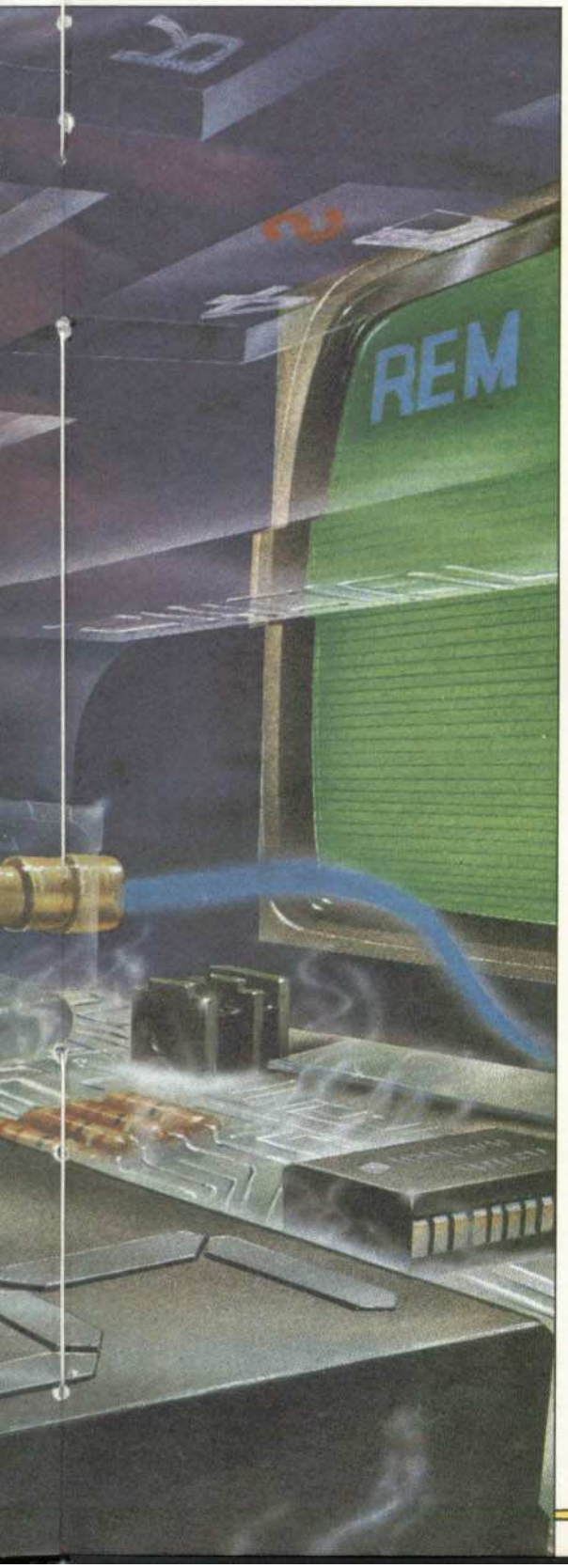
Todas as rotinas destinadas ao uso em gravador cassete empregam duas rotinas principais, a saber: **BLKIN** — no endereço 42763 —, que faz a leitura de um bloco de 255 bytes na fita; e **BLKOUT** — endereço 42996 —, que grava um bloco na fita.

Também são de utilidade para o programador BASIC os endereços da página 1, que listamos a seguir:

121 - Fornece o status da porta de entrada e saída do gravador cassete. Pode assumir os valores 0 (porta fechada), 1 (aberta para entrada) e 2 (aberta para saída). Para evitar que erros de saída destruam seu programa, verifique a porta utilizando o comando **PEEK** antes de abrir um arquivo.

144 - Empregado pelo sistema operacional para definir o comprimento do cabeçalho. Se você tem problemas com o volume do gravador, coloque um valor mais alto nesta posição por meio do comando **POKE**.

149/150 - Estas posições contêm o lapso de tempo que se segue a um comando **MOTOR ON**. **POKE 149,0:POKE 150,1** resultará em lapso zero, útil ao se usar **AUDIO ON/OFF** com controle do motor.



PROCESSAMENTO DE TEXTOS

Outra seção do sistema operacional que pode ser de interesse para o programador BASIC é aquela que executa os testes de entrada e saída. Ela inclui entrada do teclado e saída para a tela e a impressora.

Freqüentemente, a mensagem "aperte qualquer tecla" surge na tela junto com um laço do tipo:

```
100 IF INKEY$=" " THEN 100
```

Essa linha pode ser substituída por outra bem mais curta: **EXEC 44539**. Se você quiser um cursor piscante use **EXEC 36038**.

Convém anotar ainda os seguintes endereços dessa seção:

135 - Contém o código ASCII da última tecla pressionada.

338 e 345 - Colocando o valor 255 nessas posições, antes do **INKEY$**, obtém-se a auto-repetição das teclas.

282 - É a trava de letras maiúsculas. Você pode forçar as minúsculas colocando um 0 ali. O valor 255 força as maiúsculas e qualquer outro valor desabilita o uso de <**SHIFT**>0.

43304 - Contém a rotina que limpa a tela.

COMUNICAÇÃO COM A IMPRESSORA

Entre as variáveis relacionadas à impressora, são de maior utilidade para o programador aquelas que se encontram nos seguintes endereços:

153 - Determina a distância entre itens separados por vírgulas. O normal é 16.

155 - Para que a função **POS(-2)** tenha um desempenho satisfatório, esse endereço deve conter a largura do papel utilizado (40, 80 ou 132).

330 - Contém o número de caracteres enviados pelo computador para assinalar um final de linha (**EOL**). Normalmente, apenas um caractere é enviado.

331 a 334 - Contêm os caracteres do **EOL**: CR (13), LF (10), 0, 0. Esses caracteres podem ser alterados para corresponder a um certo tipo de impressora.

328 - Contém o indicador de alimentação automática de linha. Se seu valor for 0, o computador faz o papel avançar automaticamente com o retorno do carro. Qualquer outro valor leva à impressão dos caracteres do **EOL** após a do número de caracteres indicados por 155.

GRÁFICOS

O programador BASIC não tem acesso às rotinas do SO que dizem respeito aos gráficos. Mas existem alguns endereços que, junto com os comandos **PEEK** e **POKE**, podem ser úteis.

182 - **PMODE**: número do modo gráfico.
183/184 - Endereço final da tela.
186/187 - Endereço inicial da tela.
188 - Início da página 1.
200 - Coordenada X do cursor gráfico.
202 - Coordenada Y do cursor gráfico.

OUTRAS ROTINAS INTERESSANTES

EXEC 43486 - Atualiza todos os joysticks; soluciona os problemas que possam ocorrer com o comando **JOYSTK**.

EXEC 40999 - Equivale ao botão **RESET**.

EXEC 41142 - Reinicialização total: equivale a desligar e ligar a máquina novamente.

EXEC 46481 - Executa uma "coleta de lixo" controlada nas variáveis alfanuméricas, evitando as desagradáveis pausas que ocorrem inesperadamente nos programas que as incluem. A quantidade de espaço *string* remanescente pode ser calculada com

```
PEEK(35)*256+PEEK(36)-PEEK(33)*
256-PEEK(34)
```

O valor normal para o espaço string é 200. A não observância desse limite pode causar erros do tipo OS (*out of string*, que quer dizer: fim do espaço para string).

COMO LIDAR COM ARQUIVOS

Abra um canal de comunicação com o drive ou o cassete e mande seus valiosos dados para o arquivo. Saber manipular essa ferramenta é fundamental para certas aplicações.

Qualquer programa que manipule grande quantidade de informação deve armazená-la da maneira apropriada. No artigo publicado à página 128, tratamos do armazenamento de uma pequena lista de telefones em linhas **DATA**. O programa ali apresentado permite ao usuário procurar e imprimir qualquer item da lista. Porém, para qualquer modificação na lista, é necessário interromper a execução do programa e editar as linhas **DATA**. Além disso, os dados só podem ser gravados junto com o programa que os manipula.

Essa dependência dos dados ao programa muitas vezes é indesejável. Para evitá-la, convém guardar os dados em fita cassete ou disco flexível. A recuperação das informações é tão rápida quanto no caso das linhas **DATA**, mas a alteração e a edição de dados tornam-se extremamente mais fáceis. Para efetuá-las, o usuário não precisa saber nada a respeito do programa.

Como em muitos programas aplicativos listados em *INPUT*, os dados são armazenados em arquivos, na fita ou disquete. O banco de dados, o orçamento doméstico, a agenda eletrônica e a planilha são exemplos de programas que envolvem grande quantidade de informações. Se observarmos suas listagens com atenção, poderemos identificar as linhas que tratam da gravação e leitura dos dados. Elas ocupam uma porção mínima do programa, se comparadas às demais seções, encarregadas, entre outras coisas, de reservar espaço na memória, estabelecer a comunicação com o usuário e manipular os dados.

Quando armazenamos os dados à parte, as possibilidades de utilização do programa aumentam bastante, estendendo-se à criação e alteração de vários arquivos diferentes, com quantidades diversas de registros e campos.

Empregamos em nosso programa o método de armazenagem seqüencial, cujo princípio é muito simples. Como você verá, ele permite que os dados sejam convertidos a um formato que se adapta a diversos programas, e até mesmo a computadores diferentes.

O arquivo seqüencial caracteriza-se pela armazenagem dos dados em série, dispostos um após o outro e separados apenas por um byte. Toda a informação deve estar na memória antes de ser transferida ao arquivo.

O primeiro passo para a criação de um arquivo seqüencial consiste na abertura de um canal de comunicação com o dispositivo que vai armazenar os dados — cassete ou drive. Isso é feito por intermédio do comando **OPEN**, numa sintaxe que varia conforme o tipo do microcomputador. Para conhecer os comandos mais utilizados, volte ao artigo da página 688.

Aqui, explicamos em detalhe as técnicas envolvidas na criação de arquivos, de modo que você poderá incorporá-las facilmente a seus programas.

CRIANDO UM ARQUIVO

O programa a seguir mostra como criar uma versão mais sofisticada da lista de telefones mencionada anteriormente. Ao utilizá-la, você poderá modificar, sem maiores dificuldades, as mensagens dos comandos **INPUT** e o formato das matrizes, para que aceitem outro tipo de informação.

Os dados são fornecidos nesta ordem: nome, sobrenome e número do telefone. Como números de telefone podem conter espaços, hífens e parênteses, convém dar-lhes a forma de cadeia de caracteres. Se você quiser alterar o programa para lidar com outros tipos de informação, poderá usar também matrizes numéricas e alfanuméricas. Um arquivo recebe dados em qualquer formato e em qualquer ordem. O importante é que a leitura se faça nessa mesma ordem e que os dados sejam colocados no tipo adequado de variável.

Embora seja possível armazenar informações diretamente no arquivo, é mais conveniente que coloquemos os dados em uma matriz e depois guardemos toda a matriz no arquivo.

A primeira parte do programa tem um pequeno laço que possibilita a entrada dos dados. Digite quantos nomes e números quiser — o limite é a dimensão da matriz, especificada na linha 10. Modifique a dimensão, caso precise de um número de itens superior a cinqüenta. Quando tiver entrado todos os dados, digite **ENTER** ou **RETURN** e o laço de entrada será interrompido.

A segunda parte do programa — da linha 100 em diante — grava os dados no arquivo. Daremos explicações referentes a seu funcionamento nas várias versões do programa.

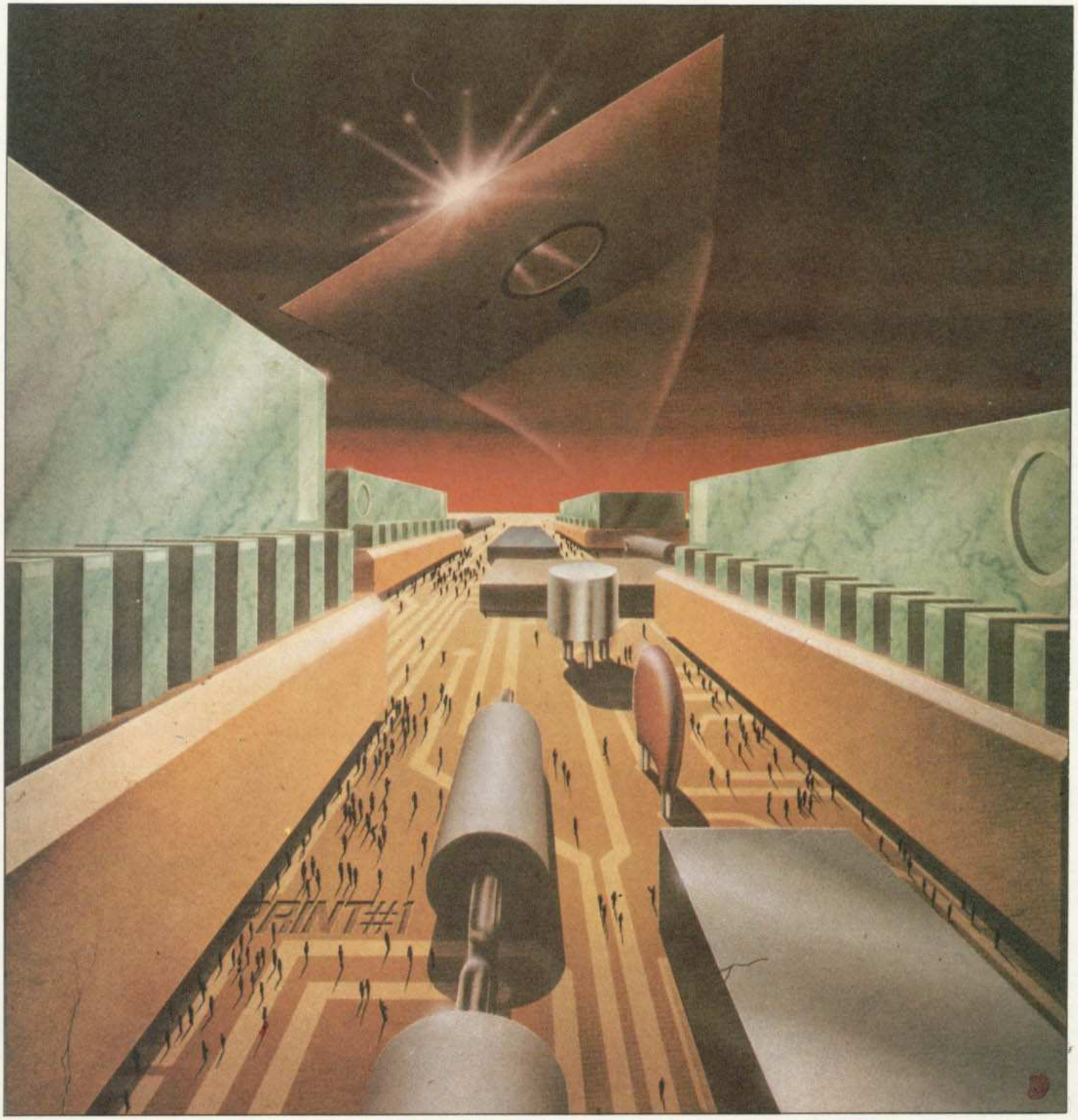
```
10 DIM A$(50,15): DIM B$(50,
15): DIM T$(50,12): DIM N(1)
20 LET N=0
30 LET N=N+1
40 INPUT "PRIMEIRO NOME ";A$(
N)
50 INPUT "SEGUNDO NOME ";B$(N
)
60 INPUT "NUMERO DO TELEFONE
";T$(N)
70 IF A$(N)<>"
" AND N<50 THEN  GOTO 30
80 CLS : PRINT "SALVANDO DADO
S AGORA"
100 SAVE "CONT" DATA N()
110 SAVE "P.NOMES" DATA A$()
120 SAVE "S.NOMES" DATA B$()
130 SAVE "N.TELEF." DATA T$()
140 PRINT "DADOS GRAVADOS"
150 STOP
```

Para a gravação, o Spectrum utiliza o comando **SAVE** seguido do nome do arquivo. Note que cada matriz é gravada como um arquivo distinto, recebendo um nome especial. A instrução **DATA** seguida do nome da matriz deve vir depois do nome do arquivo.

```
10 DIM A$(50),B$(50),T$(50)
30 N=N+1
40 INPUT "PRIMEIRO NOME";A$(N)
50 INPUT "SEGUNDO NOME";B$(N)
60 INPUT "TELEFONE";T$(N)
70 IF A$(N)<>"" AND N<50 THEN 3
0
80 CLS:PRINT "GRAVANDO OS DADOS
"
100 OPEN "CAS:ARQ" FOR OUTPUT A
S #1
110 PRINT #1,N
120 FOR L=1 TO N
130 PRINT #1,A$(L),B$(L),T$(L)
140 NEXT
150 CLOSE #1
160 PRINT "DADOS GRAVADOS"
170 END
```

Os dados entrados na primeira parte do programa são colocados em três va-

- VANTAGENS DA UTILIZAÇÃO DE ARQUIVOS
- ARQUIVOS EM FITAS E DISQUETES
- COMO ABRIR E FECHAR ARQUIVOS
- GRAVAÇÃO E RECUPERAÇÃO DE DADOS
- FIM DE ARQUIVO
- O USO DOS DADOS



Como gravar dados em fita cassete no Apple e no TK-2000?

Para contornar os problemas de arquivamento no Apple e no TK-2000, a melhor solução é comprar um drive de discos flexíveis. Mas existem algumas alternativas — nenhuma milagrosa, podemos adiantar.

Esses micros só permitem a gravação de uma variável indexada unidimensional e não montam na fita um arquivo propriamente dito. O leitor pode achar que isso já é suficiente para algumas aplicações. Porém, a lentidão e o grande espaço que uma pequena variável ocupa na fita certamente logo vão desencorajar os mais afoitos.

Em nossa opinião, a saída mais conveniente para quem não pode adquirir um drive é colocar os dados em um *buffer* no alto da memória, usando o comando **POKE**. Em seguida, basta entrar no monitor de linguagem de máquina — com **CALL -151** — e gravar o buffer por intermédio do comando **W** do monitor.

riáveis indexadas: **A$( ), B$( )** e **T$( )**. Na linha 100, abrimos um arquivo para a gravação de dados em fita cassete por intermédio da instrução **OPEN "CAS: nome do arquivo" FOR OUTPUT AS # número do arquivo**.

A instrução **PRINT #1** significa "imprima o próximo item no arquivo". Assim, a linha 110 grava o valor de **N** — total de números e nomes da lista —, para que seja usado posteriormente na leitura do arquivo.

O laço entre as linhas 120 e 140 utiliza a mesma instrução **PRINT #1** para guardar o conteúdo das três variáveis indexadas. A instrução **CLOSE #1**, por sua vez, fecha o arquivo.

```
10  DIM A$(50),B$(50),T$(50)
20 D$ =  CHR$ (4)
30 N = N + 1
40  INPUT "PRIMEIRO NOME ";A$(N
)
50  INPUT "SEGUNDO NOME ";B$(N)
60  INPUT "TELEFONE ";T$(N)
70  IF A$(N) <  > "" AND N < 50
 THEN 30
80  HOME : PRINT "GRAVANDO OS D
ADOS"
100  PRINT D$;"OPEN ARQUIVO"
110  PRINT D$;"WRITE ARQUIVO"
120  PRINT N
130  FOR L = 1 TO N
140  PRINT A$(L): PRINT B$(L):
PRINT T$(L)
150  NEXT L
160  PRINT D$;"CLOSE ARQUIVO"
170  PRINT "DADOS GRAVADOS"
180  END
```

Trataremos apenas da gravação em disquete, pois a armazenagem de arquivos em fita no Apple e no TK-2000 é muito precária. Como você deve se lembrar, para controlar o drive sem sair do BASIC Applesoft utilizamos o caractere de controle **CHR$(4)**, que foi colocado na variável **D$** (linha 5).

A primeira parte do programa, que cuida da entrada dos dados, coloca-os em três variáveis indexadas: **A$( ), B$( )** e **T$( )**. Um arquivo é aberto na linha 100 através do caractere de controle, seguido do comando **OPEN** e do nome do arquivo (entre aspas).

O comando **WRITE**, na linha 110, permite a gravação de dados no arquivo que foi aberto. Todos os comandos **PRINT** que aparecem em seguida escrevem no arquivo e não na tela. O comando **CLOSE** fecha o arquivo e devolve a função original ao **PRINT**.

```
10 DIM A$(50),B$(50),T$(50)
30 N=N+1
40 INPUT"PRIMEIRO NOME ";A$(N)
50 INPUT"SEGUNDO NOME ";B$(N)
60 INPUT"NUMERO DO TELEFONE ";T
$(N)
70 IF A$(N)<>"" AND N<50 THEN 3
0
80 CLS:PRINT"SALVANDO DADOS AGO
RA"
100 OPEN "O",#-1,"ARQUIVO"
110 PRINT #-1,N
120 FOR L=1 TO N
130 PRINT #-1,A$(L),B$(L),T$(L)
140 NEXT
150 CLOSE #-1
160 PRINT"DADOS GRAVADOS"
170 END
```

Use esta seção para arquivos em fita cassete:

Utilize esta seção para arquivos em disquete:

```
100 OPEN "O",#1,"ARQUIVO"
110 PRINT #1,N
120 FOR L=1 TO N
130 PRINT #1,A$(L),B$(L),T$(L)
140 NEXT
150 CLOSE #1
160 PRINT"DADOS GRAVADOS"
170 END
```

A primeira seção do programa, que trata da entrada dos dados, é comum aos dois tipos de dispositivo, ao contrário da segunda seção, incumbida de gravar os dados.

Na versão para fita cassete, um arquivo é aberto pela instrução **OPEN "O"** que estabelece uma linha de comunicação com o dispositivo de armazenamento para a saída de dados. Ela é seguida por um **#−1**, que diz ao computador que se trata de gravador cassete, e pelo nome do arquivo entre aspas.

O comando **PRINT #−1** significa "imprima o próximo valor no arquivo". Assim, a linha 110 grava o valor de **N** — o número de telefones da lista. Para o processo de leitura, que descrevemos a seguir, é importante que esse valor encabece o arquivo.

A linha 130 escreve cada um dos itens no arquivo — observe que eles são separados por meio de vírgulas. Finalmente, o comando **CLOSE #−1** fecha o arquivo (linha 150).

A criação de arquivos em discos flexíveis é muito semelhante, empregando os mesmos comandos. A única diferença é o número utilizado após o sinal #. **OPEN "O", #1**, seguida do nome do arquivo entre aspas, abre um arquivo; **PRINT #1** guarda um item nele e **CLOSE #1** trata de fechá-lo.

Como você pode notar, na linha 130 os itens estão separados por ponto e vírgula. Embora se permita o uso de vírgulas, aquele é o modo mais econômico de armazenar dados no disco.

Os disquetes admitem a criação de mais de um arquivo ao mesmo tempo, pois, ao contrário do gravador cassete, o drive é capaz de procurar a porção adequada para ali gravar os dados. Assim, podemos numerar os arquivos com **#1, #2, #3** e assim por diante, enquanto o cassete é sempre numerado com **#−1** — um **PRINT #−2** mandará o dado para a impressora.

A LEITURA DO ARQUIVO

O programa que recupera as informações gravadas no arquivo é exatamente o inverso do programa anterior. Ele imprime os dados lidos na tela para que possamos conferi-los.

```
200 CLEAR
210 DIM N(1): LOAD "CONT" DATA
 N()
```

```
215 DIM A$(N(1),15): DIM B$(N(
1),15): DIM T$(N(1),12)
220 LOAD "P.NOMES" DATA A$()
230 LOAD "S.NOMES" DATA B$()
240 LOAD "N.TELEF." DATA T$()
250 FOR L=1 TO N(1)
260 PRINT A$(L),B$(L),T$(L)
270 NEXT L
```

Para utilizar esse programa, rebobine a fita até o começo do arquivo gravado pelo primeiro programa. Digite **RUN 200** para recuperar os dados; não rode o programa inteiro novamente.

A linha 200 apaga os antigos valores das variáveis, para que só os obtidos no arquivo sejam impressos. A linha 210 dimensiona uma nova variável **N( )** com um elemento. Ela será usada para recuperar o valor de **N** gravado anteriormente. As outras variáveis são dimensionadas de acordo com o valor de **N( )**. Conhecendo o número exato de elementos do arquivo, não tentaremos ler além do seu final.

O resto do programa é fácil de entender: simplesmente substituímos os **SAVE** do primeiro programa por **LOAD**. É preciso fechar um arquivo aberto para gravação, mas não para leitura.

```
200 CLEAR 2000:OPEN"CAS:ARQUIVO
"FOR INPUT AS #1
210 INPUT#1,N:DIM A$(N),B$(N),T
$(N)
220 FOR L=1 TO N
230 INPUT #1,A$(L),B$(L),T$(L)
235 PRINT A$(L);B$(L);T$(L)
240 NEXT L
250 CLOSE #1
```

Para usar esse programa, rebobine a fita até o começo do arquivo gravado pelo primeiro programa. Digite **RUN 200** para ler o arquivo; não rode o programa inteiro novamente.

A linha 200 apaga as variáveis originais; dessa maneira, o programa imprime só aquilo que obtiver no arquivo. Este é aberto para leitura pela instrução **OPEN "CAS:" FOR OUTPUT AS**, seguida do valor adequado.

A linha 210 lê a quantidade de nomes armazenada no arquivo — **N** — e utiliza esse valor para dimensionar as variáveis. **N** também controla o laço de leitura do arquivo, para que ele seja recuperado até o final. Note que os valores lidos são colocados nas variáveis na ordem em que foram gravados, separados pelo mesmo tipo de sinal — no caso da fita, a vírgula. O arquivo deve ser fechado após ser lido.

```
200 D$ =  CHR$ (4): PRINT D$;"O
PEN ARQUIVO"
210  PRINT D$;"READ ARQUIVO"
220  INPUT N: DIM A$(N),B$(N),T
$(N)
230  FOR L = 1 TO N
240  INPUT A$(L): PRINT A$(L)
250  INPUT B$(L): PRINT B$(L)
260  INPUT T$(L): PRINT T$(L)
270  NEXT L
280  PRINT D$;"CLOSE ARQUIVO"
```

A linha 200 é encarregada de apagar os antigos valores das variáveis; dessa maneira, o programa imprime somente os valores que obtiver no arquivo. Este é aberto para leitura pela instrução **OPEN** seguida de **READ**.

A linha 210 lê a quantidade de nomes armazenada no arquivo — **N** — e utiliza esse valor para dimensionar as variáveis. **N** também tem como função controlar o laço de leitura do arquivo, para que ele seja recuperado até o final. Note que os valores lidos são colocados nas variáveis na mesma ordem em que foram gravados, separados pelo mesmo tipo de sinal — no caso da fita, a vírgula. O arquivo deve ser fechado após ser lido.

Para fita cassete, use esta seção:

```
200 CLEAR 2000:OPEN"I",#-1,"ARQ
UIVO"
210 INPUT#-1,N:DIM A$(N),B$(N),
T$(N)
220 FOR L=1 TO N
230 INPUT #-1,A$(L),B$(L),T$(L)
235 PRINT A$(L);B$(L);T$(L)
240 NEXT L
250 CLOSE #-1
```

No caso de discos flexíveis, recorra a esta outra:

```
200 CLEAR 2000:OPEN "O",#1,"ARQ
UIVO"
210 INPUT#1,N:DIM A$(N),B$(N),T
$(N)
220 FOR L=1 TO N
230 INPUT #1,A$(L),B$(L),T$(L)
235 PRINT A$(L);B$(L);T$(L)
240 NEXT L
250 CLOSE #1
```

Para executar essa parte do programa, rebobine a fita até o início do arquivo. Digite o comando **RUN 200** para recuperar os dados; não rode o programa inteiro novamente.

A linha 200 apaga os antigos valores das variáveis; dessa maneira, o programa imprime apenas os valores que obtiver no arquivo. Este é aberto para leitura pela instrução **OPEN "I"** seguida do valor adequado — **# −1** para fita e **#1** para disquete.

A linha 210 lê a quantidade de nomes armazenada no arquivo — **N** — e utiliza esse valor para dimensionar as variáveis. **N** também controla o laço de leitura do arquivo, para que ele seja recuperado até o final. Note que os valores lidos são colocados nas variáveis na ordem em que foram gravados, separados pelo mesmo tipo de sinal — no caso da fita, a vírgula; no do disco, ponto e vírgula. O arquivo deve ser fechado após ser lido.

FIM DE ARQUIVO

O último programa usou a variável **N** para controlar a leitura do arquivo. Contudo, nem sempre é possível saber de antemão quantos itens devem ser lidos. Um modo de contornar o problema é usar um marcador para sinalizar o fim do arquivo. O TRS-Color e o MSX colocam automaticamente um sinal desse tipo — **EOF** — no final, sempre que uma instrução **CLOSE** é encontrada.

Assim os usuários dos microcomputadores TRS-Color e MSX podem modificar o programa anterior para que ele leia o arquivo desde o início, parando sempre que encontrar o sinal que indica fim de arquivo — **EOF**.

Listamos o programa a partir da linha 100 para que você possa compará-lo à versão já dada e escolher o método que considerar melhor.

Modifique as linhas conforme a listagem abaixo e, depois, tente gravar e ler o arquivo novamente.

```
100 OPEN "CAS:ARQUIVO"FOR OUTPU
T AS #1
110 N=0
120 N=N+1
130 PRINT #1,A$(N),B$(N),T$(N)
140 IF A$(N)<>"" AND N<50 THEN
120
150 CLOSE #1
160 PRINT "DADOS GRAVADOS"
170 END
200 CLEAR 2000;DIM A$(50),B$(50
),T$(50):OPEN "CAS:ARQUIVO" FOR
 INPUT AS #1
230 N=N+1:INPUT #1,A$(N),B$(N),
T$(N)
235 PRINT A$(N);B$(N);T$(N)
240 IF EOF(1)=0 THEN 230
250 CLOSE #1
```

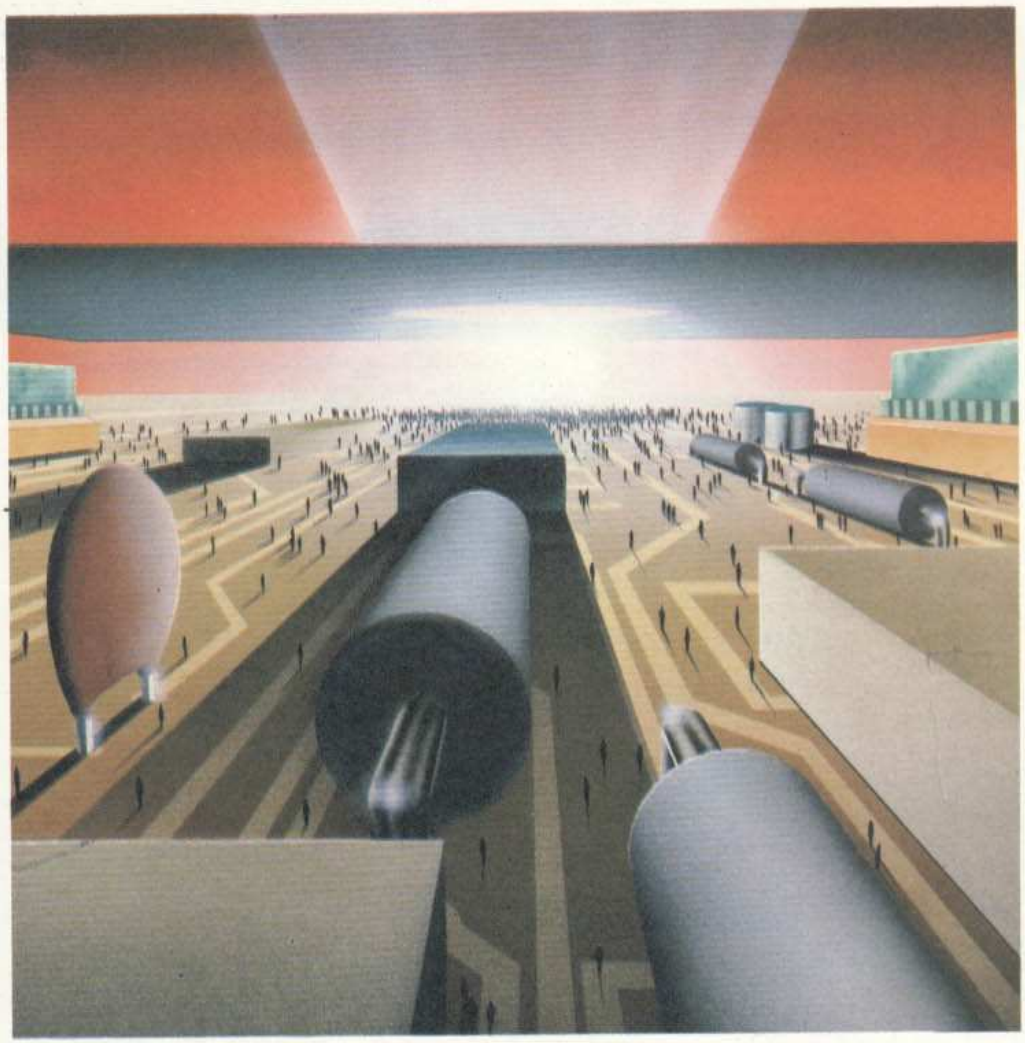
Apague as linhas 210 e 220 e modifique as demais. Para a fita:

```
100 OPEN "O",#-1,"ARQUIVO"
110 N=0
120 N=N+1
130 PRINT #-1,A$(N),B$(N),T$(N)
140 IF A$(N)<>"" AND N<50 THEN
120
150 CLOSE #-1
160 PRINT "DADOS GRAVADOS"
170 END
200 CLEAR 2000:DIM A$(50),B$(50
),T$(50):OPEN "I",#-1,"ARQUIVO"
230 N=N+1:INPUT #-1,A$(N),B$(N)
,T$(N)
235 PRINT A$(N);B$(N);T$(N)
240 IF EOF(-1)=0 THEN 230
250 CLOSE #-1
```

Para o disquete:

```
100 OPEN "O",#1,"ARQUIVO"
110 N=0
120 N=N+1
130 PRINT #1,A$(N),B$(N),T$(N)
140 IF A$(N)<>"" AND N<50 THEN
120
150 CLOSE #1
160 PRINT "DADOS GRAVADOS"
170 END
200 CLEAR 2000:DIM A$(50),B$(50
),T$(50):OPEN "O",#1,"ARQUIVO"
230 N=N+1:INPUT #1,A$(N),B$(N),
T$(N)
235 PRINT A$(N);B$(N);T$(N)
240 IF EOF(1)=0 THEN 230
250 CLOSE #1
```

A linha 240 de ambos os programas verifica a ocorrência de um sinal de fim de arquivo — **EOF**.



OUTROS SINALIZADORES

Para indicar o final do arquivo, os usuários dos microcomputadores da linha Apple podem colocar ali um sinalizador arbitrário. Já no caso do Spectrum, detectamos o fim do arquivo de outra maneira: após entrar todos os dados no laço inicial, digitamos uma senha arbitrária. Esta pode ser **ZZZ**, **–999**, a palavra **FIM** ou qualquer outra coisa que normalmente não se inclui entre os dados de um arquivo.

Altere as próximas linhas e veja como utilizar um sinalizador desse tipo. Ao digitar os dados do arquivo, entre a palavra **FIM** após o último número de telefone e **ENTER** ou **RETURN** após as outras duas solicitações.

```
70 IF A$(N)<>"FIM" AND N<50 THE
N 30
140 IF A$(N)<>"FIM" AND N<50 TH
EN 120
240 IF A$(N)<>"FIM" AND N<50 TH
EN 230
```

```
10  DIM A$(50),B$(50),T$(50)
20 D$ =  CHR$ (4)
30 N = N + 1
40  INPUT "PRIMEIRO NOME ";A$(N
)
50  INPUT "SEGUNDO NOME ";B$(N)
60  INPUT "TELEFONE ";T$(N)
70  IF A$(N) <  > "FIM" AND N <
 50 THEN 30
80  HOME : PRINT "GRAVANDO OS D
ADOS"
100  PRINT D$;"OPEN ARQUIVO"
110 N = 0
120 N = N + 1
130  PRINT D$;"WRITE ARQUIVO"
140  PRINT A$(N): PRINT B$(N):
PRINT T$(N)
150  IF A$(N) <  > "FIM" AND N
< 50 THEN 120
160  PRINT D$;"CLOSE ARQUIVO"
170  PRINT "DADOS GRAVADOS"
180  END
200 D$ =  CHR$ (4): PRINT D$;"O
PEN ARQUIVO"
210  PRINT D$;"READ ARQUIVO"
220  DIM A$(50),B$(50),T$(50)
225 L = 0
230 L = L + 1
240  INPUT A$(L): PRINT A$(L)
250  INPUT B$(L): PRINT B$(L)
260  INPUT T$(L): PRINT T$(L)
270  IF A$(L) <  > "FIM" AND L
< 50 THEN  GOTO 230
280  PRINT D$;"CLOSE ARQUIVO"
```

```
70 IF A$(N)<>"FIM" AND N<50 THE
N 30
140 IF A$(N)<>"FIM" AND N<50 TH
EN 120
240 IF A$(N)<>"FIM" AND N<50 TH
EN 230
```

O USO DOS DADOS

Armazenados em um arquivo, ou seja, fora de um programa BASIC, os dados podem ser utilizados por vários programas diferentes. Qualquer uma das rotinas de ordenação apresentadas nos artigos das páginas 468 e 738, por exemplo, servem para ordenar alfabeticamente nossa lista telefônica. Uma vez feito isso, podemos localizar rapidamente determinado item da lista, por meio de uma busca binária, como mostra o artigo da página 930.

PROGRAMA PARA TESTE DO VÍDEO

- A IMAGEM ESTÁ CENTRADA?
- DISTORÇÕES JUNTO ÀS BORDAS
- DEFINIÇÃO DA IMAGEM
- RAZÃO DE ASPECTO
- DEFINIÇÃO DAS CORES

Se você sentir a vista cansada após fixar por algum tempo o vídeo do micro, talvez seja a hora de providenciar um outro monitor. Use este programa para testar a qualidade da imagem.

Até pouco tempo atrás, o preço de um monitor de vídeo colocava esse periférico totalmente fora do alcance da maioria dos proprietários de micros, obrigando-os a utilizar aparelhos comuns de TV como saída de vídeo.

Os televisores apresentam vários inconvenientes: a imagem pisca e oscila continuamente de modo imperceptível, o que provoca dores de cabeça e cansaço visual após um certo tempo. Outro problema é causado pela baixa resolução da tela, que resulta em perda de detalhes e definição da imagem em textos e gráficos.

Entretanto, o custo relativo dos monitores profissionais de vídeo tem caído bastante nos últimos anos, tornando esse periférico cada vez mais acessível. E, se você ainda não tem um, vale a pena ir pensando em adquiri-lo, pois a qualidade da imagem é bem superior à de um televisor.

No artigo da página 851, descrevemos as características técnicas que diferenciam um televisor de um monitor e falamos dos aspectos que devem ser considerados na escolha de um bom monitor. Não há dúvida, porém, de que o melhor critério para essa seleção ainda é o "Teste de São Tomé" — ou seja, ver como fica a imagem no vídeo, quando o conectamos ao nosso computador. Como há problemas de compatibilidade entre monitores e micros, é fundamental que você use o seu modelo na hora desse teste.

O programa listado neste artigo foi projetado para testar as características do vídeo em diferentes linhas de computadores. A versão para o Spectrum mostra um único padrão visual na tela; já as versões para o MSX, Apple, TK-2000 e TRS-Color mostram duas ou três telas sucessivas, cada qual projetada para testar um aspecto do funcionamento do vídeo.

Os padrões visuais variam de computador para computador, mas contêm basicamente os elementos utilizados em testes como os que são transmitidos em horas mortas pelas emissoras de TV: círculos e linhas espaçadas a curta distância, para testar o poder de definição de tela; padrões colocados nos cantos da tela, para testar distorções marginais da imagem, e barras ou faixas de todas as cores disponíveis.

Ao rodar o programa de teste, observe primeiro se a imagem está bem centrada. Em alguns computadores, o gerador de vídeo não deixa margem na tela, e, se ela estiver mal ajustada, haverá uma perda de gráficos e textos junto à borda. Muitos monitores e televisores dispõem de botões giratórios de ajuste de centralização e expansão de imagem, localizados atrás do aparelho. Você poderá usá-los para conseguir um resultado melhor.

Verifique em seguida se as linhas dos retângulos junto às bordas da imagem não sofrem uma grande distorção (as linhas retas se transformam em arcos convexos: quanto menor o raio destes, maior a probabilidade de que o monitor seja de baixa qualidade).

Todas as linhas aparecem nitidamente separadas umas das outras? Se isso não ocorrer, há problemas de definição na imagem, o que poderá prejudicar a legibilidade de textos. Em muitos monitores e televisores, a resolução é melhor no sentido horizontal do que no vertical, ou vice-versa, devido à maneira como os pontos coloridos de fósforo são dispostos na tela. Monitores de boa qualidade têm resolução igual em todas as direções.

Os círculos que o programa desenha não são perfeitos, tendendo a assumir uma forma ovalada? Este é um bom teste para uma das características dos vídeos de boa qualidade, a chamada *razão de aspecto*: para obtê-la, dividimos o diâmetro transversal do círculo pelo diâmetro vertical. Idealmente, é claro, o resultado deverá ser 1.

Verifique também se as linhas aparecem traçadas com nitidez ou se as bordas e cantos parecem borrados. Um problema desse tipo pode prejudicar a legibilidade de um texto na tela.

Se o monitor ou TV for colorido, examine as cores das barras (ou das letras, no programa para o TRS-Color) na segunda parte do teste. Os cantos e bordas das barras coloridas devem ser bem definidos, e a cor de uma barra não deve extravasar para outra. Como a densidade, brilho e a saturação das cores podem variar de vídeo para vídeo, experimente mudar os controles do aparelho, até obter a melhor combinação. Lembre-se de que diferentes ajustes de cor costumam ser necessários, conforme a aplicação: texto puro, por exemplo, requer pouca cor e brilho médio; um jogo exige muita cor e brilho.

Os programas para o MSX e o TRS-Color permitem variar a cor de fundo na tela pressionando seguidamente a tecla <**ENTER**>, na última parte do programa. Para interrompê-lo, pressione <**BREAK**> ou <**CTRL**><**STOP**>.

No Apple e no TK-2000, os caracteres de texto não podem ser coloridos. Assim, para testar a qualidade do vídeo quanto a este aspecto, interrompa o programa e liste-o na tela. Observe se as letras aparecem em branco puro e se não há nenhum efeito de "arco-íris" (letras multicoloridas e borradas).

Se você conhece um pouco de programação gráfica em BASIC, não terá nenhuma dificuldade em modificar os programas aqui apresentados para torná-los mais completos ou atraentes. Procure alterar a combinação de cores de frente e de fundo, para verificar o efeito, gere mais uma tela de textos e caracteres gráficos etc.

```
10 PMODE 4,1:PCLS:SCREEN 1,1
20 FOR K=0 TO 1:FOR J=0 TO 15 S
TEP K+2
30 LINE (K*16+J,K*16+J)-(255-K*
16-J,191-K*16-J),PSET,B
40 NEXT J,K
.50 FOR K=100 TO 154 STEP 2:LINE
(K,85)-(K,105),PSET:LINE(113,K-
32)-(141,K-32),PSET:NEXT
60 FOR K=1 TO 100 STEP 10:CIRCL
E(127,95),K,5:NEXT
70 IF INKEY$="" THEN 70
80 PMODE 1,1:PCLS:SCREEN 1,0
90 FOR J=0 TO 191 STEP 48:COLOR
 J/48+1
```

Padrões de teste de vídeo utilizados em diferentes computadores.

```
100 LINE(0,J)-(255,J+47),PSET,B
F:NEXT
110 FOR J=0 TO 255 STEP 63:FOR
K=0 TO 63 STEP 4:COLOR J/63+1
120 LINE(K+J,0)-(K+J,191),PSET
130 NEXT K,J
140 IF INKEY$="" THEN 140
150 S=1-S:SCREEN 1,S:GOTO 140
```

```
 5 FOR p=0 TO 7: FOR i=0 TO 7
 : IF p=i THEN  GOTO 200
 6 PAPER p: INK i: BORDER 0:
 CLS
10 FOR n=0 TO 12 STEP 2: PLOT
n,n: DRAW 0,175-2*n: DRAW 255
-2*n,0: DRAW 0,-(175-2*n):
DRAW -(255-2*n),0: NEXT n
20 FOR n=10 TO 20 STEP 2:
CIRCLE  INK i;128,85,n: NEXT
n
30 FOR n=5 TO 7: PRINT  INK i
;AT n,8;" ";AT n+10,8;" ":
NEXT n
40 PRINT AT 4,8;: FOR m=0 TO
1: FOR n=0 TO 7: PRINT  PAPER
n;" ";: NEXT n: NEXT m
50 PRINT AT 18,8;: FOR m=0 TO
1: FOR n=0 TO 7: PRINT  PAPER
 n;" ";: NEXT n: NEXT m
55 PRINT AT 2,4;: FOR n=0 TO
2: FOR m=0 TO 7: PRINT  INK m
; PAPER p;CHR$ (64+n*7+m);:
NEXT m: NEXT n
 57 PRINT AT 19,4;: FOR n=0 TO
 2: FOR m=0 TO 7: PRINT
 BRIGHT 1; INK m; PAPER p;CHR$
 (64+n*7+m);: NEXT m: NEXT n
 60 FOR n=0 TO 7: BORDER n:
 PAUSE 50: NEXT n: PAUSE 50
200 NEXT i: NEXT p
```

```
10 S=S+1:COLOR 15,S,S:SCREEN 2
15 CL=15:IF S=15 THEN CL=1
20 FOR K=0 TO 1:FOR J=0 TO 15 S
TEP K+2
30 LINE(K*16+J,K*16+J)-(255-K*1
6-J,191-K*16-J),CL,B
40 NEXT J,K
50 FOR K=100 TO 154 STEP3:LINE(
K,85)-(K,105),CL:NEXT
55 FOR K=65 TO 135 STEP3:LINE(1
40,K)-(114,K),CL:NEXT
60 FOR K=1 TO 100 STEP 10:CIRCL
E(127,95),K,CL:NEXT
70 IF INKEY$="" THEN 70
80 COLOR 15,S,S:SCREEN 2
90 FOR J=0 TO 191 STEP 13
100 LINE(0,J)-(255,J+11),J/13+1
,BF:NEXT
110 IF INKEY$="" THEN 110
130 GOTO 10
```

```
10  HGR2 :LX = 279:LY = 180
20  FOR X0 = 0 TO 20 STEP 4
25  LET Y0 = X0:LX = LX - 10:LY
 = LY - 10
30  HPLOT X0,Y0 TO X0 + LX,Y0 T
O X0 + LX,Y0 + LY TO X0,Y0 + LY
 TO X0,Y0
40  NEXT X0
50  FOR Y = 60 TO 100 STEP 2
55  HPLOT 110,Y TO 150,Y
57  HPLOT 50,Y TO 70,Y: HPLOT 1
90,Y TO 210,Y
60  NEXT Y
65  FOR X = 95 TO 165 STEP 3
70  HPLOT X,68 TO X,93
75  NEXT X
85  FOR R = 10 TO 60 STEP 10
86  HPLOT 130 + R,80
87  FOR A = 0 TO 6.4 STEP .1
90  HPLOT  TO 130 + R *  COS (A
),80 + R *  SIN (A)
95  NEXT A: NEXT R
100  GET A$
105  TEXT : HOME
110  GR :C = 0
120  FOR I = 0 TO 36 STEP 3
130  LET C = C + 1: COLOR= C: V
LIN 1,15 AT I
135  VLIN 1,15 AT I + 1: VLIN 1
,15 AT I + 2
140  NEXT I:C = 0
145  FOR J = 18 TO 36 STEP 2
150  LET C = C + 1: COLOR= C
155  HLIN 0,38 AT J: HLIN 0,38
AT J + 1
160  NEXT J
190  GET A$
200  TEXT : HOME : END
```

CONTROLE A ENTRADA DE DADOS

- UMA ROTINA BÁSICA
- TESTANDO A ROTINA
- OUTRAS TECLAS DE RETORNO
- ENTRADAS SECRETAS
- ROTINA DE ENTRADA NUMÉRICA

"Envenene" seus programas de entrada de dados com uma rotina de grande versatilidade. Ela substitui com vantagem o comando INPUT, superando várias de suas limitações.

O comando **INPUT** satisfaz a maioria das necessidades de programação de entrada de dados. Em certas situações, porém, a utilização desse comando apresenta alguns sérios inconvenientes. Entre eles, podemos citar:

- toda vez que o programa executa um comando **INPUT**, o cursor muda de linha na tela quando o usuário pressiona a tecla <**ENTER**>;
- um ponto de interrogação é sempre colocado na tela (nos micros pertencentes às linhas TRS-80, TRS-Color e MSX), ou força a entrada na última linha (caso dos micros da linha Sinclair: ZX-81 e Spectrum);
- todos os caracteres digitados aparecem na tela: é impossível ocultá-los (como se faz, por exemplo, na entrada de uma senha secreta);
- é necessário pressionar <**ENTER**> para terminar a entrada;
- não existe nenhum controle sobre os caracteres que estão sendo digitados. Na linha TRS, por exemplo, se pressionarmos a tecla de alimentação de linha (representada pela flecha para baixo), prejudicaremos a formatação de uma tela de entrada.

É possível contornar todos os problemas mencionados recorrendo a uma rotina especial de entrada, que substitui o comando **INPUT**.

Neste artigo, ensinaremos como desenvolver essa rotina usando a função **INKEY$**, disponível no BASIC dos micros pertencentes às linhas Sinclair, TRS e MSX, ou o comando **GET**, nos micros das linhas Apple e TK-2000.

UMA ROTINA SIMPLES

Inicialmente, vamos programar a rotina de entrada de dados da forma mais simples possível, para que você possa entendê-la sem dificuldade:

```
1000 LET S$=""
1010 LET C$=INKEY$
1020 IF C$="" THEN GOTO 1010
1022 LET C=ASC(C$)
1025 IF C=13 THEN RETURN
1030 LET S$=S$+C$
1040 PRINT C$;
1060 GOTO 1010
```

Modifique as seguintes linhas, para o micro ZX-81:

```
1022 LET C=CODE C$
1025 IF C=118 THEN RETURN
```

Suprima a linha 1020 e substitua a linha 1010, para o programa funcionar no Apple e no TK-2000:

```
1010 GET C$
```

Teste a rotina, acrescentando a este pequeno programa:

```
10 PRINT
90 PRINT "ENTRE: ";
100 GOSUB 1000
110 PRINT
120 PRINT S$
130 GOTO 10
```

Para evitar problemas de "estouro" da área alfanumérica, acrescente esta linha para o TRS-80 e TRS-Color:

```
5 CLEAR 1000
```

FUNCIONAMENTO DA ROTINA

A linha 1000 inicializa a cadeia de saída, **S$**, que conterá tudo o que for digitado. As linhas 1010 e 1020 verificam se alguma tecla foi pressionada. Se a resposta for negativa, o programa continua esperando. Caso contrário, o programa prossegue para a linha 1030, que testa se a tecla pressionada foi <**ENTER**> ou <**RETURN**>, que assinalam o fim da entrada de dados. Se foi outra tecla qualquer, ela é concatenada à cadeia de saída, na linha 1030, e impressa na linha 1040. Finalmente, o programa retorna à linha 1010, para aguardar a próxima tecla.

LIMITE PARA A ENTRADA

Ao retornar da rotina, o cursor continua na mesma linha de entrada. A linha 100 força sua mudança para a linha seguinte, mas pode ser retirada, se necessário — por exemplo, se você estiver escrevendo um programa de formatação de tela de entrada, com vários campos da mesma linha.

O programa principal pode ser modificado de modo a colocar o cursor num ponto determinado da tela, antes de chamar a sub-rotina da linha 1000. Acrescente as seguintes linhas para verificar o efeito obtido:

MICRO DICAS

APLICAÇÕES PARA A ROTINA

A rotina apresentada neste artigo tem variadas aplicações. Usando-a criativamente, você poderá trabalhar com diversas funções impossíveis de se obter com o BASIC. Essa rotina equivale, de fato, a um poderoso comando **LINE INPUT** que alguns interpretadores possuem.

Em programas educativos e de jogos, você pode incluir uma função de atraso máximo de tempo dentro da rotina de entrada — ou seja, a cada repetição do laço de varredura (apenas para as versões com o comando **INKEY$**), uma variável **T** de tempo pode ser incrementada. Ao sair da rotina, o programa tem acesso ao **T** total, informando ao usuário o tempo que levou para entrar os dados.

Outra aplicação interessante consiste em sair automaticamente da rotina quando um tempo máximo estipulado para a resposta foi ultrapassado — o que é útil em jogos de rapidez de raciocínio, por exemplo.

```
80 PRINT AT 10,10;
```

```
80 PRINT @ 128,"";
```

```
80 LOCATE 10,10
```

```
80 HTAB 10:VTAB 10
```

APERFEIÇOAMENTOS

Como está, a rotina oferece poucos recursos e não apresenta vantagens de ordem significativa em relação ao comando **INPUT**. Mas existem várias maneiras de aperfeiçoá-la. Podemos, por exemplo, estabelecer um limite máximo para o número de caracteres que devem ser digitados, o que nos permitirá entrar dados sem a necessidade de pressionar a tecla <**ENTER**>.

Acrescente esta linha à sub-rotina:

```
1050 IF LEN(S$)=MX THEN RETURN
```

A variável **MX**, que indica o tamanho máximo do campo, deve ser especificada antes de se chamar a rotina. Para testá-la, acrescente estas linhas ao programa principal:

```
20 PRINT "TAMANHO DO CAMPO ";
30 INPUT MX
```

Podemos ainda utilizar uma outra tecla — como <**SCAPE**> — para concluir a entrada de dados.

Substitua a linha 1025 por:

```
1025 IF C=TX THEN RETURN
```

Note que **TX** deve ser especificada pelo programa que chama a sub-rotina.

Acrescente estas linhas, para testar a nova opção:

```
40 PRINT "CODIGO DE TERMINO ";
50 INPUT TX
```

Uma rotina de entrada de dados também não estaria completa se não oferecesse ao usuário a possibilidade de corrigir erros, usando a tecla <**BACKSPACE**> ou <-. Adicione ao programa as seguintes linhas:

```
1026 IF C=8 AND LEN(S$)>0 THEN
1070
1070 LET S$=LEFT$(S$,LEN(S$)-1)
1080 GOTO 1040
```

```
1026 IF C=8 AND LEN(S$,>0 THEN
1070
1070 LET S$=S$(TO LEN(S$)-1)
1080 GOTO 1040
```

```
1026 IF C=114 AND LEN(S$)>0
THEN 1070
1070 LET S$=S$(TO LEN(S$)-1)
1080 GOTO 1040
```

A linha 1026 verifica se a variável de saída já tem no mínimo um caractere e se a tecla de correção foi pressionada (código ASCII 8; em se tratando do ZX-81, deve ser pressionada a tecla <-, cujo código é 114).

Em caso positivo, a rotina desvia para a linha 1070, que extrai o caractere apagado e vai para a linha 1040. Esta faz retornar o cursor, imprimindo-a sobre a posição anterior.

ENTRADAS SECRETAS

Se quiser que os caracteres digitados não apareçam na tela — para a entrada de uma senha secreta, por exemplo —, substitua a linha 1040 por:

```
1040 IF SX=1 THEN PRINT C$;
```

A variável **SX** indica se o caractere digitado deve ser mostrado (**SX** = 1) ou não (**SX** < >1). Para testar essa modificação, acrescente as próximas linhas ao programa principal:

```
60 PRINT "ENTRADA INVISIVEL
(SIM=0) ";
70 INPUT SX
```

ENTRADAS NUMÉRICAS

Podemos tornar nossa rotina de entrada ainda mais versátil. Uma opção interessante consiste em utilizar a chamada *censura de entrada*, por intermédio da qual fazemos com que a rotina ignore automaticamente toda tecla que não esteja incluída dentro de um determinado conjunto de entrada. Essa alternativa nos permite estabelecer que todas as teclas sejam maiúsculas, ou que não ocorram certos caracteres, como vírgulas, ou, também, que a entrada seja apenas numérica.

O procedimento é, na verdade, bastante simples. Para obter uma rotina de entrada numérica, por exemplo, basta que acrescentemos:

```
1027 IF NOT (C>=43 AND C<=46)
AND NOT (C>=48 AND C<=57) THEN
GOTO 1020
```

```
1027 IF C<>21 AND C<>22 AND
C<>26 AND C< >27 AND NOT (C>=28
AND C<=37) THEN GOTO 1020
```

Essa linha verifica se o caractere entrado está entre 0 e 9 ou se é um sinal de mais, de menos, vírgula ou ponto. Se não for um desses, retorna para a linha 1020, de espera.

P&R

O que faz a instrução LINE INPUT?

Esta resposta é dirigida aos usuários de computadores TRS-80 (com BASIC nível II), ZX-81, Sinclair, Apple e TK-2000. O BASIC padrão de suas máquinas não dispõe da instrução **LINE INPUT**, mas a sub-rotina apresentada neste artigo pode substituí-la muito bem.

Como o nome sugere, a instrução **LINE INPUT** permite a entrada de uma linha de texto pelo teclado. Essa linha é armazenada em uma variável literal ou *string*, nomeada pela instrução. Por exemplo:

```
LINE INPUT X$
```

Não podemos colocar outras variáveis para entrada pela mesma instrução.

Ao contrário da instrução **INPUT**, **LINE INPUT** não imprime na tela o ponto de interrogação — sinal de prontidão para o usuário —, o que dá uma maior liberdade de programação.

O **LINE INPUT** aceita uma linha de texto contendo vírgulas, pontos e vírgulas e outras pontuações — que nos obrigariam ao uso de aspas, para delimitar o string de entrada, caso utilizássemos uma instrução **INPUT**. Assim, é ideal para a entrada em processamento de texto.

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

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

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

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

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

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


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

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

NO PRÓXIMO NÚMERO

APLICAÇÕES

Você quer saber o que o destino lhe reserva? Consulte os astros... ou melhor, o programa de horóscopo de *INPUT*.

CÓDIGO DE MÁQUINA

Avalanche está completo. Para que o jogo comece, falta apenas a rotina final, que comanda toda a ação.

PROGRAMAÇÃO DE JOGOS

Você é capaz de pintar uma parede sem deixar cair tinta no carpete? Teste sua habilidade no jogo *O Pintor Aloprado*.

PROGRAMAÇÃO BASIC

Conheça algumas rotinas capazes de simplificar a manipulação de datas em BASIC.