BIBLIOTECA BÁSICA

INFORMÁTICA

BASIC II 6

SÉCULO FUTURO

BIBLIOTECA BÁSICA

INFORMÁTICA

BASIC II 6

SÉCULO FUTURO

Diretor editor:
M.A.Nieto
Coordenação e supervisão técnica:
Eng.º Sergio Rocha Paggioli
Tradução:
Ideli Novo
Projeto:
Rainer K.E. Ladewig
Diretor de arte:
Duilio Sarto F.º
Studio editorial:
Auro Pereira da Silva (chefe), Susana M.Amaral Couto (revisão), Luiz Carlos Siqueira Lago (prod. gráfica), Antonio Carlos Martins, Rubens Tadeu Benedito
Fotocomposição, fotolito:
Omnicolor Gráfica e Propaganda Ltda - Rua Dr. Virgilio de Carvalho Pinto, 619 Pinheiros - CEP 05415 - São Paulo
Impressão
Editora Antártica S.A. - Av. Ramon Freire, 6920 (Pajaritos) - Santiago - Chile

ÍNDICE

PREFÁCIO

No primeiro volume sobre o BASIC descrevemos as instruções principais desta linguagem, graças às quais nos "aventuramos" através da escrita de alguns programas simples.

O BASIC é uma linguagem que parece estar concebida intencionalmente para os computadores pessoais, ainda que suas origens se perderam nos primeiros tempos das calculadoras eletrônicas, quando os computadores pessoais estavam "por vir ao mundo".

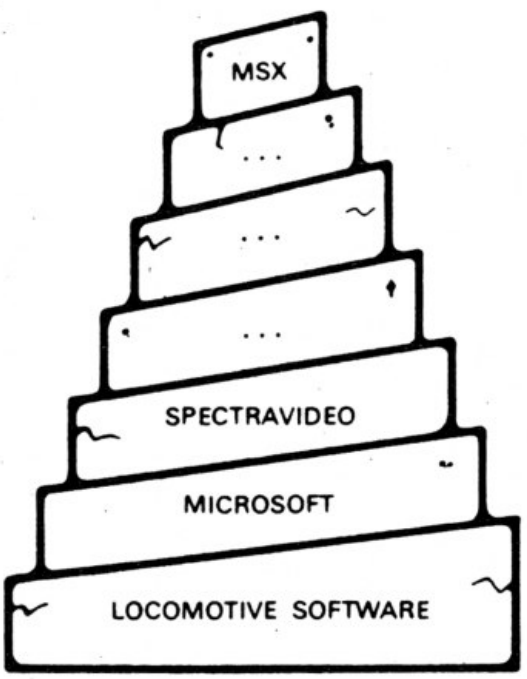
Os computadores pessoais (utilizamos o termo pessoal como equivalente e sinônimo de computador "caseiro") são máquinas flexíveis das quais podemos fazer o uso que mais nos agrade. Até metade dos anos setenta foi quando apareciam como tais as primeiras. Imediatamente se distinguiram de seus irmãos maiores, os grandes computadores tradicionais, por ser econômicos, simples e, sobretudo, por não exigir um profundo domínio da Informática.

Até esse momento os computadores eram objetos muito exclusivos, utilizados somente por grandes empresas ou por centrais atômicas muito sofisticadas. Alguns anos antes, a Hewlett Packard havia utilizado o termo pessoal para uma de suas mais características calculadoras de bolso, mas foi o Altair 8800 (vendido em kits de montagem pela MITS de Albuquerque, no Novo México) quem abriu a era do computador "em casa", com todas as conseqüências sociais e culturais que dele se haviam derivado. No transcur-

so de muito pouco tempo nasceram o Apple I, irmão maior do indiscutível lider Apple II, e o PET, que são os responsáveis da nobre dinastia dos Commodore.

O BASIC, imortal como o fênix, renasceu precisamente graças a sua adoção como principal linguagem dos computadores pessoais. Atualmente estes, como coroação de um assombroso progresso tecnológico, podem trabalhar praticamente com qualquer linguagem de programação (FORTRAN, PASCAL, COBOL, C, FORTH, para citar os nomes mais conhecidos), mas seu principal instrumento de programação continua sendo o BASIC, que constitui um padrão e um indiscutível modelo de linguagem para o computador.

O entusiasmo por haver adotado o simpático e fácil BASIC nos computadores pessoais, sem que existisse uma versão oficial patrocinada por algum ente internacional, traiu como conseqüência que cada fabricante de computadores pessoais ou cada firma fornecedora de software por encargo o adaptara a suas próprias exigências, de caráter meramente comercial. A história é velha e aconteceu também com os computadores maiores: você compra meu hardware, mas somente pode utilizar meu software. Trair a um fabricante e passar-se à competência queria dizer atirar ao lixo o software e os programas de aplicação e tornar a escrever tudo a partir de nada outra vez. Assim sucedeu com os computado-



res pessoais; cada um colocou em objetivo sua própria versão de BASIC sem deixar nenhum espaço ao cliente para orientar-se para outras máquinas.

Atualmente as coisas estão mudando. O mercado (o inefável mercado!) deseja a compatibilidade entre programas, sistemas operacionais e pacotes de software. Somente quem respeita as convenções, tácitos ou impostos, sobre as normas (os standards) supera as duras exigências dos grandes mercados internacionais. E assim, enquanto por um lado se afirma a versão do BASIC nascida na prestigiosa firma americana Microsoft, da qual surgiram tantos rebentos de "sangue azul", por outro lado se estão aclarando as posições dos sistemas operacionais: os sistemas operacionais CP/M e MS-DOS para os computadores pessoais, e o UNIX para a gama superior (por alguns chamado de "micro" e que tempos atrás compreendiam os famosos "minicomputadores"). Mas existe quem não quer se submeter a nenhuma normalização e a "torre de Babel" cresce mais outro andar. E assim chegamos a um novo pretendente à coroa de unificador: o sistema MSX. O MSX nasceu também na casa Microsoft, mas imediatamente foi adotado mais além dos oceanos pelos fabricantes do Sol Nascente, em primeiro lugar, e por alguns europeus depois. Procurava contrastar o predomínio comercial americano e propõe uma unificação que, pela primeira vez, implica não somente ao software, mas também ao hardware.

As boas intenções dos fabricantes que adotam o sistema MSX estão a favor do usuário: todos os programas escritos nos computadores MSX são transferíveis a qualquer outra máquina que adote este sistema. E também, todos os periféricos que adotam o padrão de conexão MSX podem ser conectados a qualquer outro computador que o utilize. Já não teremos necessidade, por exemplo, de joysticks diferentes para cada computador, mas cada joystick servirá para qualquer máquina.

Os fabricantes poderiam assim dedicar-se a melhorar o produto sabendo que o mercado potencial se faz cada vez mais amplo. O mesmo é aplicável aos discos flexíveis e aos cartuchos de memória ROM ou às impressoras. É uma questão de normalização que já se propôs outras vezes, mas que em muito raras ocasiões teve êxito. Somente os grandes interesses comerciais e industriais podem impor uma normalização, e em um próximo futuro veremos se o sistema MSX, ou outros, são capazes de impor-se definitivamente. Nós, os usuários do computador pessoal, não esperamos outra coisa, que estes acordos sejam uma realidade tangível.

CAPÍTULO I

PERIFÉRICOS E CONEXÕES

De qual BASIC falamos

Na parte introdutória da primeira monografia dissemos que falar de BASIC em geral quer dizer descrever sua estrutura, apresentar suas instruções principais, os conceitos que são comuns a todos os dialetos e advertir ao leitor quando existe diferenças importantes.

Também dissemos que trataremos de falar de BASIC nos termos mais comuns possíveis e que as mais importantes diferenças entre versões se encontram nas instruções de gráficos (desenhos na tela) ou da gestão de arquivos (os arquivos de dados), instruções que pretendemos apresentar nesta monografia. Para estabelecer contato com o exposto no volume anterior recordamos que as instruções de uma linguagem de programação podem ser divididas, em "grandes rasgos", em quatro famílias:

- as instruções declarativas,
- as instruções de entrada/saída,
- as instruções de cálculo e atribuição,
- as instruções de controle.

Esta subdivisão não é somente conceitual, pois existe sempre uma correspondência física com um elemento diferente do hardware do computador. Por exemplo, as instruções de entrada/saída se relacionam com a gestão ou controle das unidades pe-

riféricas (impressoras, discos, etc.), enquanto que as de atribuição se referem à atribuição de valores às células da memória central.

As instruções que nos propomos estudar nesta monografia podem incluir-se nestas famílias, mas exigem uma maior atenção e, com freqüência, uma descrição mais detalhada que as vistas no volume anterior. Trata-se de instruções que somente devem ser empregadas quando se tenha adquirido um bom domínio das demais (as descritas na primeira monografia) e quando, frente a um erro de sintaxe indicado na tela, se tenham os conhecimentos e a boa vontade de analisar o programa e detectar o erro. Assim, falaremos de GOSUB, que é a instrução de BASIC para chamar a uma subrotina. GOSUB pode classificar-se entre as instruções de controle, mas merece uma posição privilegiada porque permite dar uma certa estruturação aos programas, como veremos dentro em pouco.

Faremos também alusão a algumas instruções particulares de atribuição que, por falta de espaço, não se apresentaram na monografia anterior: RND e DEF FN. A primeira, de grande utilidade, permite gerar números ao acaso, enquanto que a segunda põe à disposição do programador a possibilidade de construir funções (em sentido matemático) novas com respeito às já proporcionadas pela linguagem BASIC. Abordaremos, também, um dos temas mais complicados, o "nível universitário" do BASIC, que é a gestão dos arquivos, como se diz na gíria Informática.

As instruções sobre arquivos adoecem muitíssimo das derivações próprias das distintas versões, porque trataremos de falar muito de conceitos e dar exemplos que se refiram somente aos computadores de maior difusão. Nos desculpamos, desde agora, por não tratar as instruções de todos os computadores existentes no mercado, mas isto seria verdadeiramente impossível. O que faremos, como no volume anterior, é apresentar uma "tabela de equivalências" das instruções de distintos computadores.

Por último, veremos como se pode desenhar com o computador, e dentro deste tema abordaremos os gráficos. Também neste caso, e quase em maior medida, se podem descrever muitos conceitos, mas daremos alguns exemplos que não sejam específicos, na medida do possível, de uma só máquina. Por desgraça, se com os arquivos se podem descrever situações que compreendem a muitos computadores similares, com os gráficos cada computador é um mundo à parte.

Com freqüência, incluindo modelos de computadores de uma mesma firma, se comportam, no que diz respeito aos gráficos, de um modo absolutamente diferente. Não existe mais remédio que

assimilar os conceitos básicos (o que nunca será mal) e logo, aplicá-los aos casos concretos. Antes de começar a falar de instruções de BASIC, e levando em consideração que esta monografia trata muitas instruções de entrada e de saída, consideramos oportuno fazer um exame das unidades periféricas mais importantes. Na verdade, os problemas de programação não são conceituais nem teóricos, derivam do desconhecimento real das unidades periféricas.

O computador e seus periféricos

Nos propusemos a falar somente de BASIC e, conseqüentemente, de software, mas não é possível prescindir de uma breve descrição das unidades periféricas do computador se quisermos compreender melhor o significado e funcionamento das instruções que tratam da entrada e da saída dos dados.

Um computador processa dados, mas para que isto seja possível é preciso que desde o exterior ditos dados sejam fornecidos junto aos programas que os devem tratar. Estas funções de comunicação com o exterior não são competência, falando em termos estritos, do computador propriamente dito (ou melhor dito, da CPU), mas que se desenvolvem mediante dispositivos adequados, chamados unidades periféricas, ou de uma forma mais simples, periféricos. Assim, quando se quer descrever de uma maneira concisa um sistema baseado em computador, se desenha a unidade central (a CPU) e ao redor dela todos seus periféricos (Fig. 1).

Os periféricos são, por exemplo, as impressoras, as unidades para discos, os digitalizadores e os joysticks, mas também o teclado e o monitor são periféricos.

Alguém poderia considerar que os periféricos têm uma importância secundária em relação com o computador. Isto não é certo em absoluto e, pelo contrário, com muita freqüência a capacidade de um computador depende, em grande medida, dos periféricos que dispõe. Basta um exemplo para constatá-lo: torna-se inútil que um computador seja capaz de calcular milhares de dados por segundo se sua impressora somente pode escrever dez ou cem por segundo.

O primeiro encargo solicitado aos periféricos consiste em permitir a comunicação, em um ou outro sentido, entre o computador e o homem.

Dentro dos periféricos de entrada (aqueles que comunicam ao homem com o computador) se encontram:

- teclados,
- mouse,

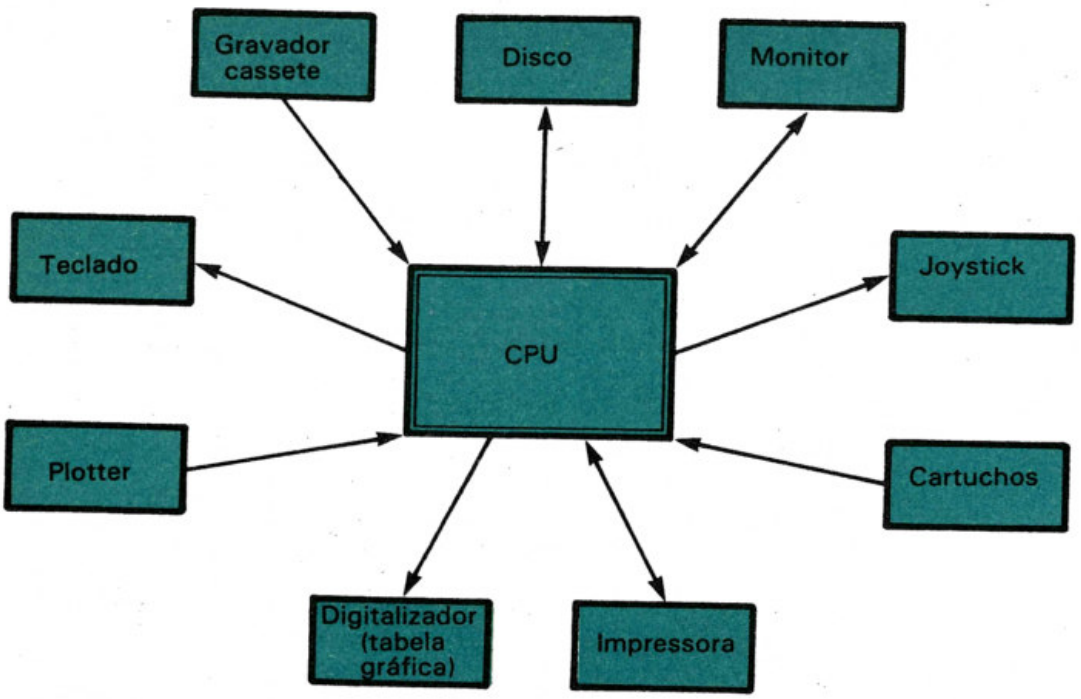


Fig. 1. — Diagrama de blocos de um sistemas de computador, com a unidade central de processamento (CPU) no centro e os periféricos ao redor.

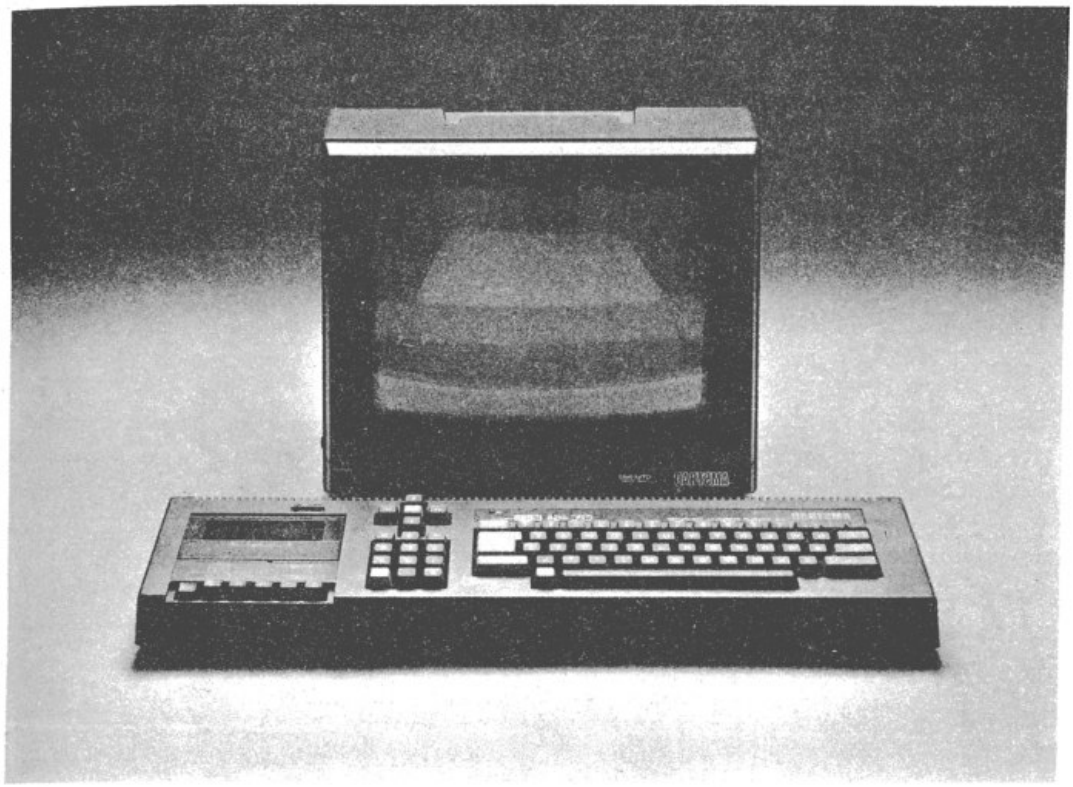
- digitalizador (tabela gráfica),
- lápis ótico ("ligh pen")
- paddle ("palhetas"),
- joystick,

e entre os periféricos de saída (enviam mensagens do computador):

- tela de apresentação visual (ou monitor),
- impressoras,
- plotter (traçador).

A lista anterior não é completa, mas o conceito está claro: todos eles são dispositivos, mais ou menos complexos, que permitem proporcionar informações ao computador (escrevendo, desenhando, etc.) ou recebê-las dele (na tela, no papel).

Além de todos os anteriores existem periféricos que intercambiam dados somente com a CPU e nos quais o homem não intervem de maneira direta. A grandes passos, se podem dividir em dois



Fotografia 1 — Equipamento que inclui em sua configuração básica: teclado, monitor e gravador cassete.

grupos: periféricos de armazenamento massivo e os dispositivos de telecomunicação. O primeiro grupo compreende:

- fita magnética,
- cassete,
- discos flexíveis (disquetes),

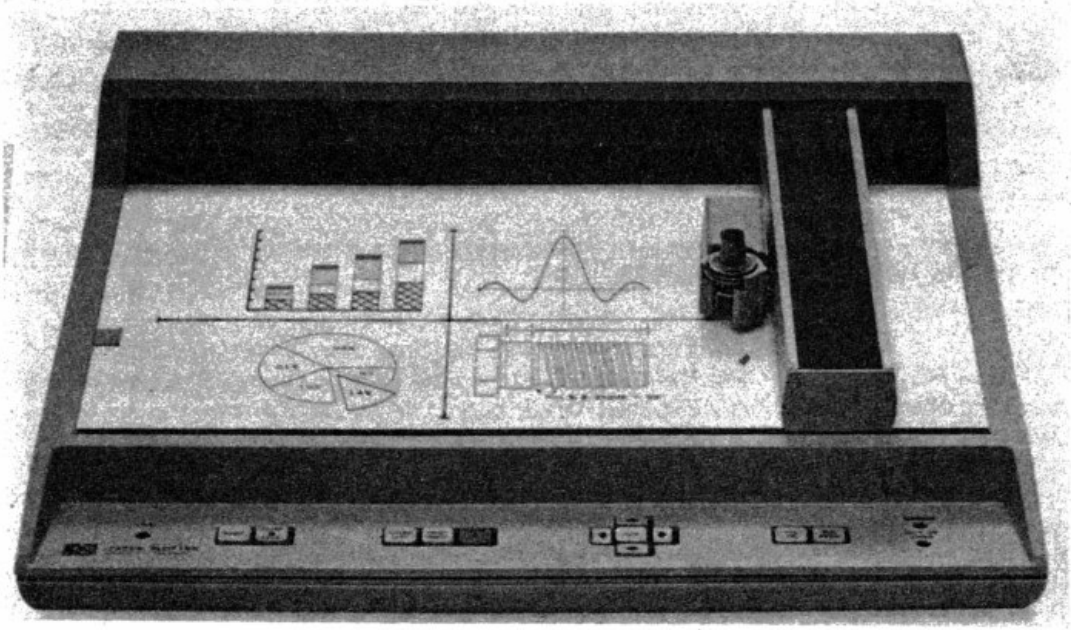
O segundo grupo está caracterizado por um único dispositivo significativo. Dito dispositivo é o modem que permite modular um sinal para sua transmissão e, logo demodulá-lo (MOdulador/DEModulador).

Periféricos de entrada/saída menos conhecidos

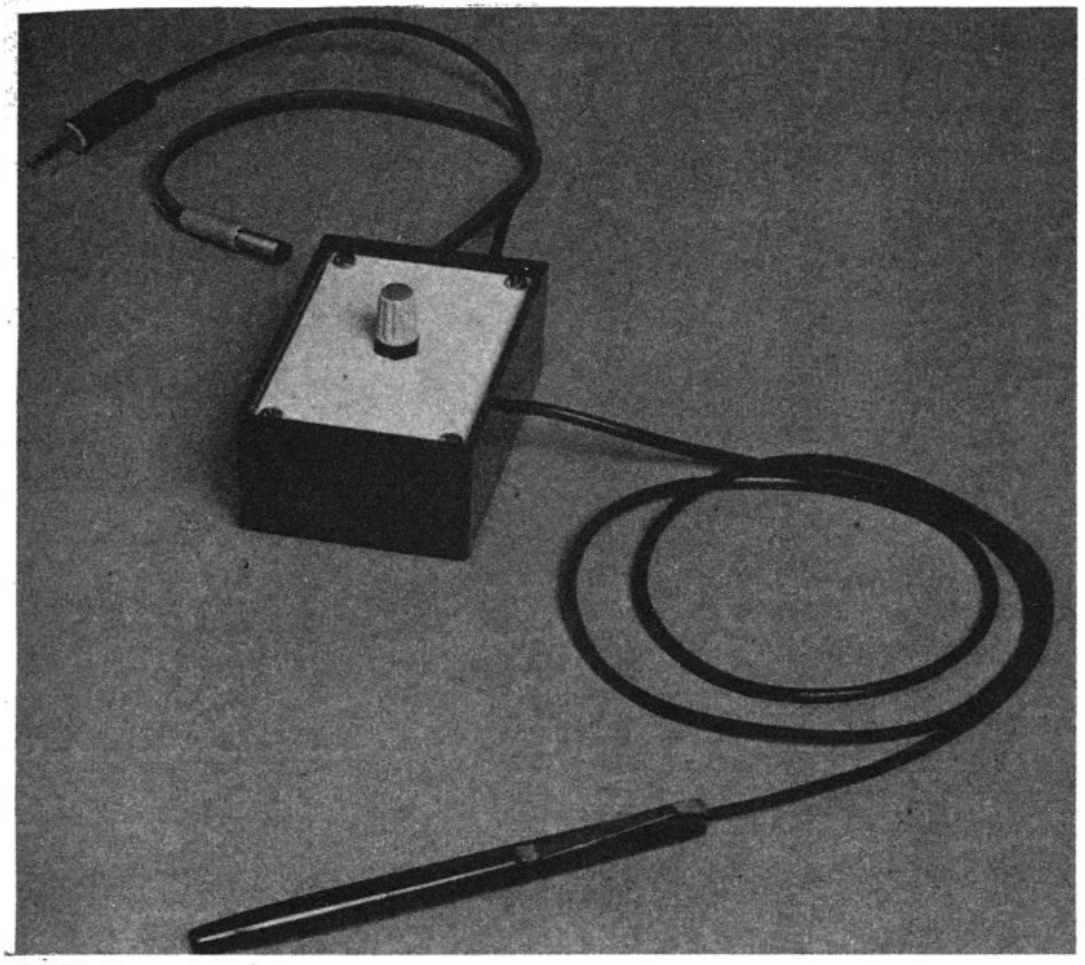
O mouse se assemelha a um maço de cigarros com um pulsador e em sua parte inferior, uma bola "aprisionada". Ao movê-la com a mão sobre a mesa se desloca em correspondência um cur-

Fotografia 2 — Caixa do mouse, com seu " rabo " característico.

sor indicador na tela. Deste modo torna-se possível distribuir ordens ao computador de uma maneira rápida, fácil e sobretudo sem conhecer nada de programação. No espírito de seus criadores trata-



Fotografia 3 - Plotter moderno de alta qualidade.



Fotografia. 4 - Periférico muito recente: o lápis óptico.

se de uma alternativa aos teclados tradicionais.

O digitalizador (tabela gráfica) está constituído por uma espécie de quadro negro e por um lápis; ao deslocar o lápis sobre o quadro se envia ao computador a forma digital de um desenho.

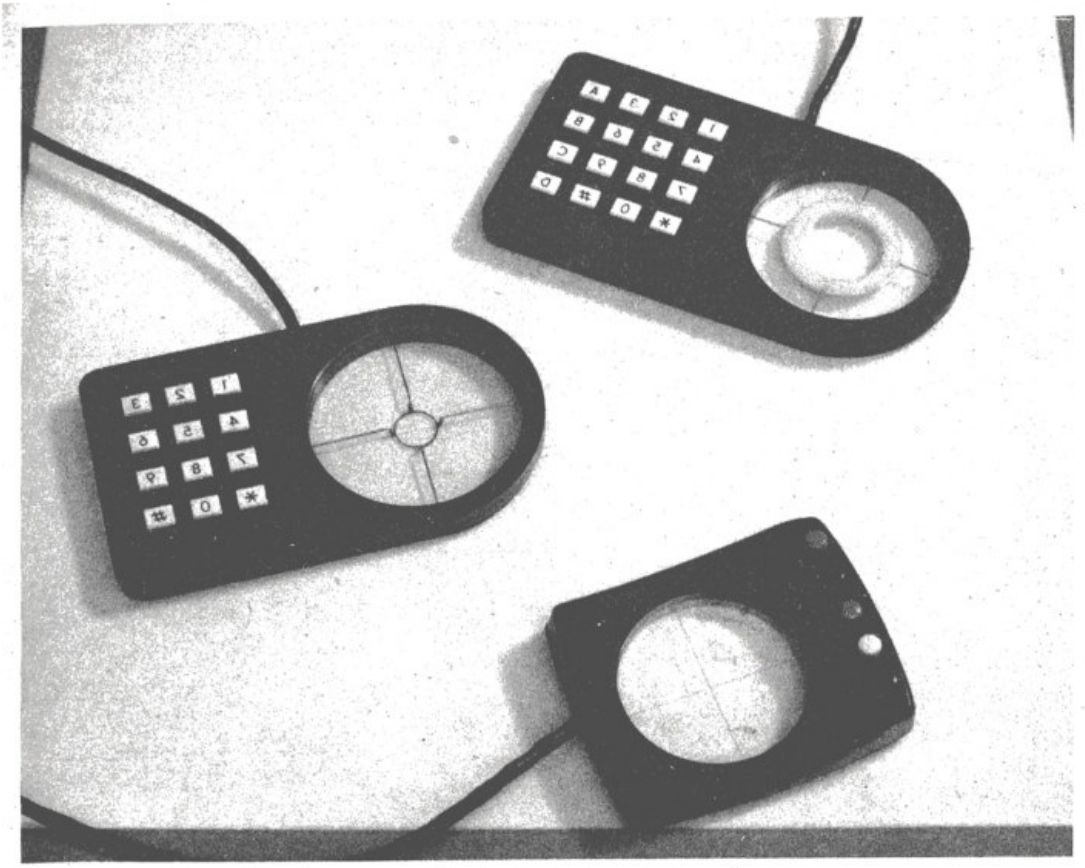
O lápis óptico funciona de modo análogo, mas se trata de um lápis provido de uma célula fotoelétrica que se tem que apoiar sobre o cristal da tela.

O joystick está provido de uma alavanca móvel e às vezes de um pulsador, à diferença do paddle, que emprega um anel giratório.

Os plotters efetuam traçados sobre folhas de papel, sobre uma superfície plana ou sobre um cilindro e podem utilizar vários lápis coloridos.

Fotografia 5 - Diversos joysticks.

Como dissemos anteriormente, muitas vezes a velocidade dos periféricos que interagem com o homem não é um fator determinante para o bom funcionamento do computador. Por exemplo, é inútil que uma tela seja capaz de visualizar milhares de caracteres por segundo, posto que nós nunca os poderíamos ler. Pelo contrário, uma impressora demasiadamente lenta pode ocasionar problemas quando se tem que imprimir muitos formulários. A questão é diferente para os periféricos que se comunicam somente com o computador, tais como os discos flexíveis. Em tal caso, sempre se procura alcançar a máxima velocidade possível.



Fotografia 6 - Três modelos de digitalizadores.

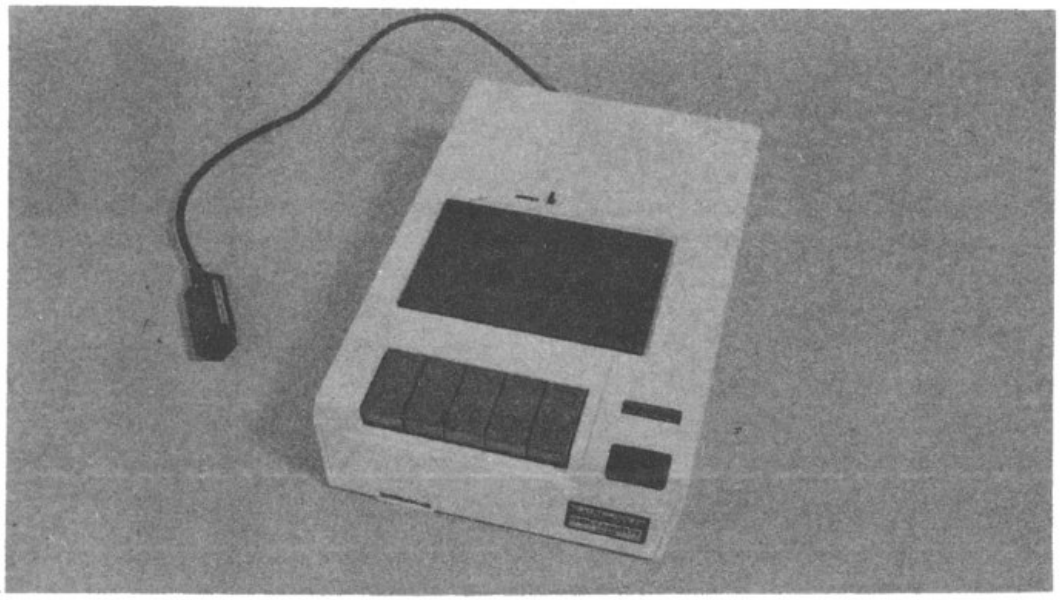
Periféricos de armazenamento massivo

Para o usuário humano, os programas e os dados são coisas conceitualmente muito distintas: os primeiros são as instruções necessárias para processar aos segundos. Para o computador, ao contrário, ambos são seqüências de números binários (fileiras longas de zeros e uns) conservados na memória central.

Lamentavelmente, a memória central, ainda que cada dia mais barata, é bastante custosa e, conseqüentemente, sua amplitude total nunca é demasiadamente "generosa". Além disso, as memórias que utilizamos na atualidade (de custo relativamente moderado) são sempre voláteis e perdem seu conteúdo ao desligar o computador. Por estes dois motivos torna-se muito importante que os programas e os dados sejam conservados em memórias não voláteis.

Os periféricos de armazenamento massivo, denominados também memórias de massa, porque podem conter grandes quantidades de informações, são precisamente do tipo não volátil. Entre os mais importantes estão:

- fitas magnéticas,
- tambores magnéticos,



Fotografia 7 - Periféricos de armazenamento massivo mais usados: cassete e disco flexível.

- fitas de cassete magnéticas,
- discos magnéticos rígidos ("hard disk"),
- discos magnéticos flexíveis ("floppy disk"),

e além destes:

- unidade de "bolhas magnéticas",
- unidade de disco óptico.

Como se observa, a tecnologia atualmente mais difundida é a magnética, na qual cada bit é armazenado sob a forma de um campo magnético microscópico em um suporte constituído por óxidos de ferro depositados sobre uma base de mylar (fitas, cassetes e disquetes) ou de alumínio (tambores e discos rígidos). Esta tecnologia é muito similar à empregada para as gravações musicais comuns, e tanto é assim que se costumam utilizar os mesmos gravadores a cassete destas últimas com os computadores, inclusive para gravar dados e programas.

Um parâmetro muito importante nas memórias de massa é o tempo de acesso aos dados, isto é, o tempo necessário para chegar a uma informação determinada. Os discos rígidos, que giram continuamente em grande velocidade, tem tempos de acesso muito pequenos; as fitas magnéticas e as fitas de cassete, ao contrário, são muito lentas porque é preciso desenrolar toda a fita que precede ao ponto procurado, isto é, se trata de dispositivos.

Conexões série e paralelo

Para compreender melhor a forma em que trabalham os periféricos com o computador e, conseqüentemente, para saber como devem ser escritas as instruções de entrada e de saída de dados nos programas, é útil esclarecer um par de conceitos de hardware. A tal fim falaremos de conexões em série e em paralelo entre os dispositivos.

Nas conexões em paralelo (Fig. 2) existe um fio elétrico para cada bit objeto de transmissão. Se temos que transferir 8 bits existirão, pois, 8 fios (mais o de massa, claro). Cada bit é transmitido como um breve impulso elétrico. De forma análoga, haverá tantos fios como bits, inclusive nos computadores maiores de 16 ou de 32 bits, se se conectam em paralelo a qualquer periférico.

No caso de conexões em série (Fig. 3), ao contrário, os bits são enviados um a um ao longo de um só fio (mais o de massa) a intervalos de tempo periódicos. A transmissão de dados em série

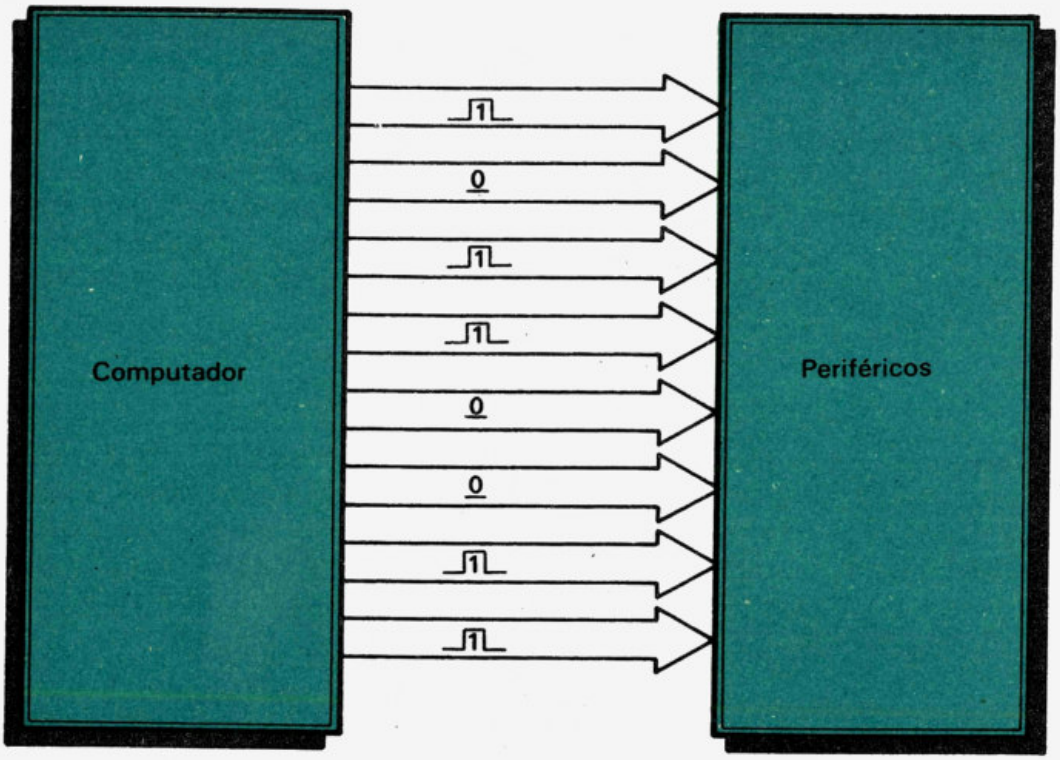


Fig. 2 - Conexão em paralelo: uma linha para cada bit.

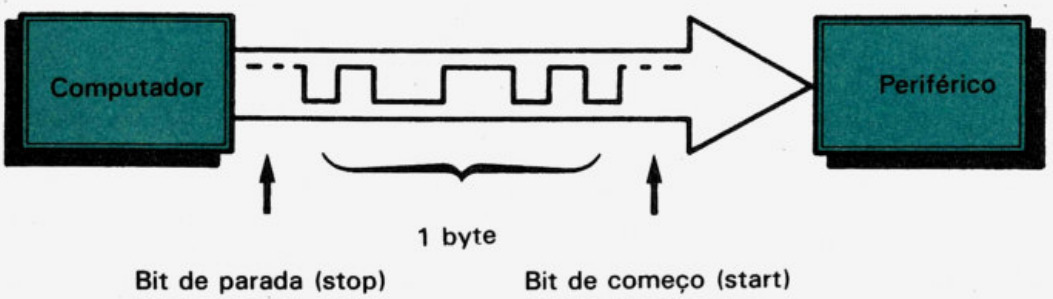


Fig. 3 - Conexão em série: por uma só linha se transmitem sucessivamente todos os bits.

é utilizada para conexões econômicas ou onde não exista alternativa (por exemplo, para poder explorar as linhas telefônicas). Este sistema é muito similar à comunicação telegráfica em código morse e é praticamente a mesma utilizada pelos teletipos comuns.

Os buffers ou memórias tampão

Um elemento importante, quando se fala de intercâmbios de dados entre dois dispositivos, é o buffer ou memória tampão. Supomos, ao falar de conexões em série e em paralelo, que os dispositivos transmissores e receptores trabalham à mesma velocidade. Por exemplo, se o computador transmite 2 Kbytes por segundo (2.000 bytes por segundo), o periférico deve poder receber à mesma velocidade.

Nem sempre é possível esta igualdade de velocidade. Os circuitos eletrônicos podem trabalhar com grande rapidez (tempos da ordem de magnitude de microsegundos ou inclusive inferiores), enquanto que os dispositivos com partes mecânicas são muito mais lentos. Se supusermos que estes últimos trabalham a uma velocidade de milisegundos, que é muito elevada para sua própria natureza, continuarão sendo mil vezes mais lentos que os dispositivos eletrônicos (em um microsegundo existe mil milisegundos).

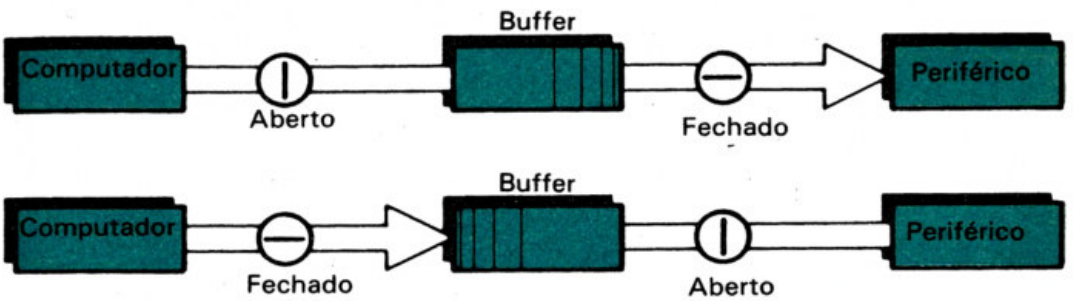


Figura 4 - Um buffer trabalha como um depósito de dados intermediário.

Por exemplo, uma unidade de disco não pode ter acesso a um dado até que o giro do disco não leve o dado desejado sob a cabeça de leitura/escrita. Se o computador tem que escrever 100 dados em um disco seria inadmissível ter que esperar cada vez uma volta completa do disco (que costuma atrasar 200 milisegundos por volta) e, conseqüentemente, uns 20 segundos no total.

Se recorre então à técnica da memória buffer. Buffer significa memória intermediária. Atua como uma espécie de amortecedor, igual, por exemplo, aos amortecedores dos vagões da ferrovia. Também a bateria do automóvel é um exemplo de buffer. Em

nosso caso, no entanto, os buffers não amortizam acumulando energia, mas dados.

Os dados produzidos pela CPU se acumulam no buffer, que não é outra coisa que uma pequena zona de memória RAM, sem fazer mais lento o trabalho da própria CPU (Fig. 4). Quando o buffer está cheio, se descarrega enviando seus dados até o periférico (às vezes, inclusive, sem necessidade de que intervenha a CPU). No exemplo da unidade de disco, o tempo necessário para a operação se reduz assim a uma só volta do disco.

Às vezes, o buffer está incorporado no próprio periférico, como no caso de algumas impressoras que admitem um milhão de bytes à velocidade máxima com que os produz a CPU; os acumulam em uma memória buffer interna e logo "com toda calma" os imprimem à velocidade permitida pela mecânica. Assim, enquanto isso, a CPU pode desempenhar outras tarefas.

CAPÍTULO II

SUBROTINAS E DEFINIÇÃO DE FUNÇÕES

Escrever progamas em BASIC, como em qualquer outra linguagem de programação, pode ser fácil e agradável. Freqüentemente se obtém resultados satisfatórios inclusive com programas somente de algumas dezenas de instruções, mas quando se escrevem programas complexos profissionais, podem ser necessárias centenas ou milhares de linhas de programa. Neste caso, o modo no qual se escreve o programa, isto é, sua estrutura como a chamam os experts, se faz fundamental tanto para a correção de seu funcionamento como para sua legibilidade. A legibilidade de um progama consiste em que qualquer um que observe sua listagem possa compreender qual é o princípio de funcionamento, ou dito de outro modo, o algorítmo base do próprio programa. Da relação entre programas e algorítmos falamos amplamente na primeira monografia sobre o BASIC.
O BASIC não é uma linguagem adaptada ao tipo de programação que os experts denominam estruturada, como é o caso da linguagem Pascal ou da mais moderna ADA. Em que consiste esta? Tentar esclarecer esta matéria, inclusive para os programadores de BASIC nunca vai mal.

Dar uma definição de programação estruturada poderia nos tomar muito tempo e, dado o caráter divulgativo destas monografias, nos complicaria as idéias em lugar de esclarecê-las. Digamos que um programa que tenha intenções estruturalistas é um programa escrito e desenvolvido com clareza que pode ser compreen-

dido por qualquer um que conheça a linguagem de programação que utiliza. Com demasiada freqüência vemos programas que não são compreensíveis nem sequer para pessoas muito versadas em BASIC. Em ocasiões até mesmo o autor, transcorrido algum tempo, não chega a compreender o que significa sua listagem, e se deve procurar um erro ou realizar uma modificação, Será "apalpando" (são os denominados "programas labirintos").

Como é possível que aconteça isto? Basta utilizar muitas instruções GOTO ou empregar o mesmo nome de variável em vários pontos do programa com significados diferentes. Um programa é um processamento lógico que deve ter um começo e um fim, e cada separação deve ser assinalada com clareza. Se um determinado grupo de instruções tem um encargo concreto colocamos adiante uma instrução de comentário (REM) para esclarecer dito encargo, ou então utilizamos os dois pontos (:) depois de um número de linha (ou outro caracter permitido pelo BASIC) para separar grupos e dar instruções e dar um aspecto arquitetônico ao programa. Vejamos um exemplo:

```
100 REM PROGRAMA PARA O CALCULO
110 REM DA PASSAGEM
120 REM DO COMETA HALLEY
130 REM
140 REM
150 REM ULTIMA PASSAGEM DO COMETA HALLEY
160 REM
170 REM
180 REM CALCULO DA ORBITA
190 REM
200 REM
210 REM
220 REM PERIELIO O DIA 9 DE FEVEREIRO DE 1986
230 REM
240 REM
250 REM PROXIMA PASSAGEM NO ANO 2062
```

Mas vamos supor agora que queremos desenvolver um novo programa. Teremos que proceder com muita ordem. Já tivemos ocasião de dizer que a pior maneira de realizar um programa é a de sentar-se imediatamente diante do computador e começar a escrever as instruções como se tratássemos de desafogar um impulso de inspiração literária. Antes de tudo, temos que escrever em um papel os passos principais do programa (ou seja, do algorítmo) com idéias muito claras sobre o que queremos; e logo partiremos desta descrição para desenvolver a estrutura do programa.

Vejamos um exemplo comum para muitos casos práticos, co-

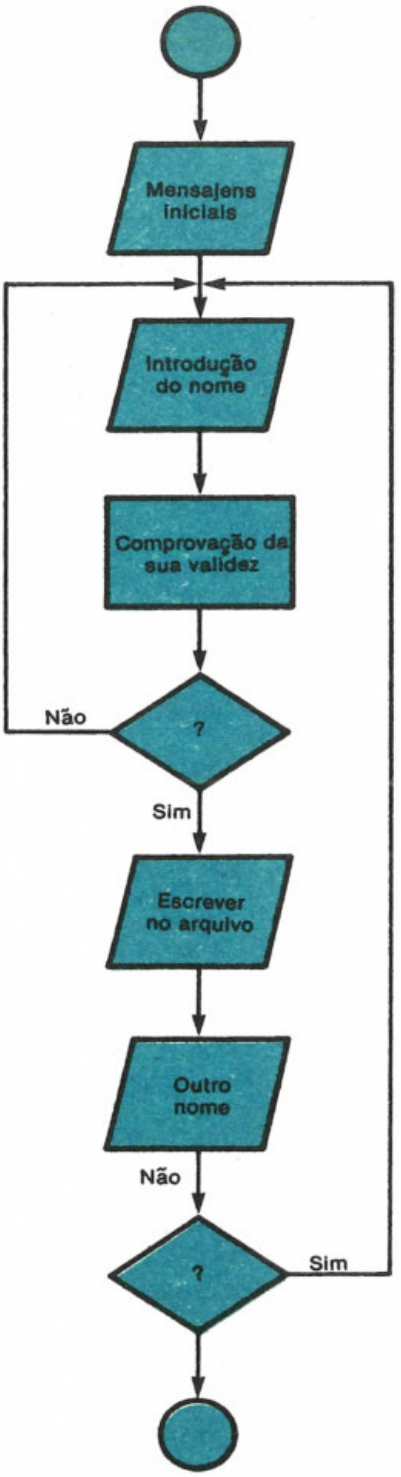


Figura 1 - Diagrama de blocos explicativos, a partir do qual se escreverá o programa.

mo é a escrita em um arquivo de nomes recebidos desde o teclado. Os passos do programa são os seguintes:

1. Enviar à tela as mensagens iniciais de aviso ao usuário.
2. Pedir um nome.
3. Comprovar a validade do nome.
4. Se é correto, gravar no nome do arquivo (disco flexível). Em caso contrário, tornar a pedí-lo (2).
5. Perguntar se existe mais nomes para introduzir e, se for assim, voltar ao ponto 2; em caso contrário, terminar o programa.

Antes de prosseguir, desenharemos o diagrama de blocos de nosso programa, tal como o definimos até agora (Fig. 1).

Para facilitar isto, entre outras coisas estão as subrotinas. Uma subrotina é, em essência, uma série de instruções, separadas da seqüência principal do programa, que cumprem uma missão específica. Tem um ponto de começo e um de final próprios. Quando se quer usá-la desde o programa "principal" se faz um salto especial ao ponto de começo e, ao chegar ao final, a subrotina faz outro salto à instrução seguinte no programa principal, àquela pela qual foi "chamada". O salto à subrotina se realiza com a instrução GOSUB (Fig. 2).

Se ao escrever o programa conseguirmos manter clara a divisão entre os cinco pontos anteriores, teremos feito um bom trabalho.

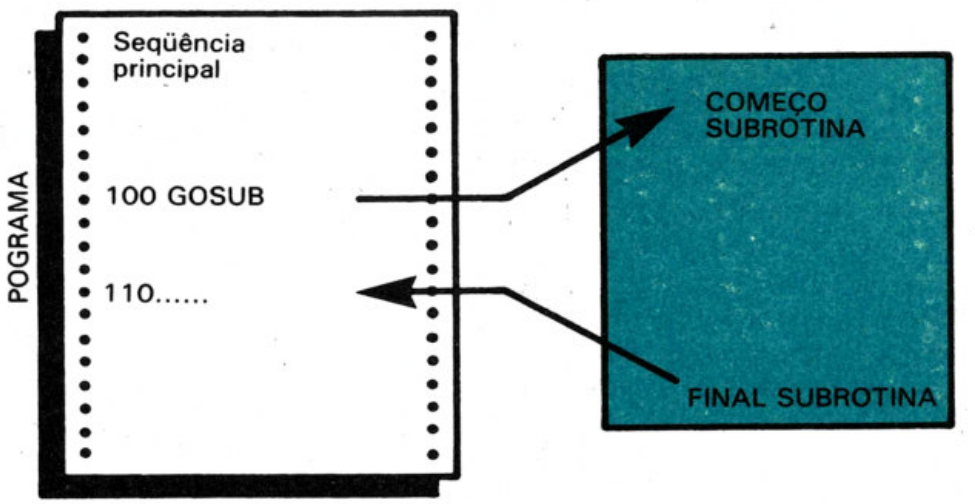


Figura 2 - Forma de empregar uma subrotina desde o " programa principal ".

GOSUB

Se fixamos no ponto 3, relativo à comprovação da validade dos dados recebidos à entrada, observamos que se trata de algo separado, do ponto de vista conceitual, do núcleo do programa. O ponto 3 poderia "confiar-se" a uma subrotina de serviço, que se ocupe exclusivamente de efetuar as comprovações, e à qual recorreríamos cada vez que necessitássemos efetuar uma.

Pensando bem, também o ponto 4 pode chegar a ser uma subrotina. Na realidade, se ocupa somente de levar a um arquivo os nomes. Nesta primeira escrita do programa consideramos oportuno gravar o arquivo em um disco flexível, mas poderíamos querer fazê-lo também em uma fita de cassete. Se utilizamos uma subrotina será muito fácil mudar somente as instruções necessárias para passar do disco à fita de cassete, enquanto que, se não a usamos teríamos que mudar o programa principal, e isto seria mais complicado.

O emprego de subrotinas, que dependem de um programa principal, nos permite a estrutura de blocos de um programa de uma maneira diferente e mais flexível. Na Figura 3 se ilustra a forma em que se modifica o diagrama de blocos anterior ao empregar subrotinas.

Vejamos agora como se utiliza a instrução GOSUB para chamar às subrotinas em BASIC. Na gíria informática se diz chamar uma subrotina, porque o FORTRAN, a primeira linguagem que utilizou as subrotinas tem uma instrução CALL (chamar) que serve precisamente para ativar a execução das subrotinas. Na prática se diz que o programa principal chama às subrotinas. Sempre por motivos históricos se fala também em BASIC de passagem dos parâmetros para indicar o intercâmbio de dados e variáveis entre o programa principal e as subrotinas, pois em FORTRAN antes de poder empregar uma subrotina é preciso "comunicar-lhe" sobre que parâmetros ou argumentos terá que atuar. Em BASIC isto não é estritamente necessário porque as subrotinas não são programas totalmente separados, mas que formam um todo junto com o próprio programa principal, dividindo a memória do computador, com o que todas as variáveis são comuns.

Um exemplo de subrotina

Para ver como se utilizam as subrotinas, tratamos de escrever um programa que faça uso delas.

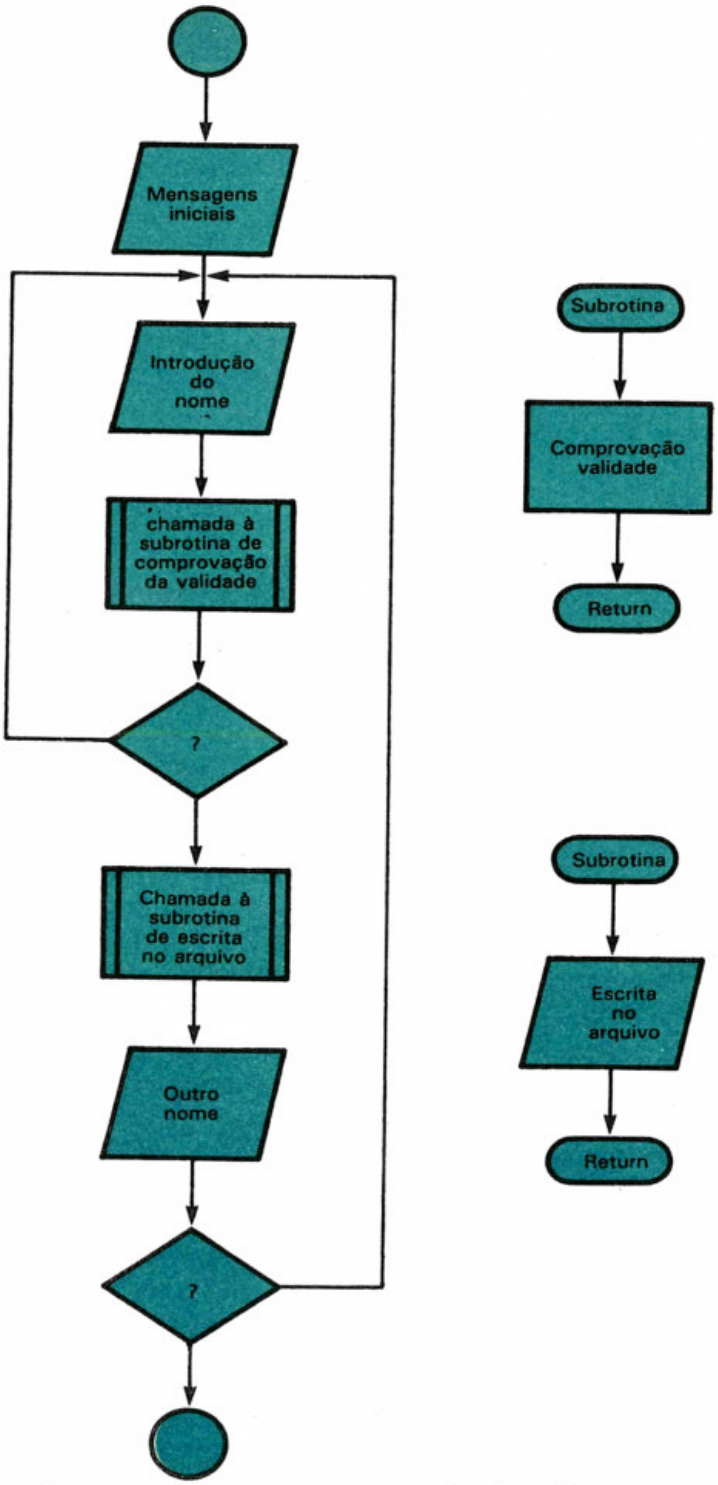


Figura 3 - Diagrama da figura 1 modificado pelo emprego de subrotinas.

Um caso muito freqüente quando se trabalha com os computadores é o de que se tenha que dar uma data. Um computador não está obrigado, a não ser que o programemos para isso, a comprovar o que lhe comunicamos, e assim, se quiséssemos caçoar e dar datas como o dia 66 do mês 25 do ano 234, o poderíamos fazer com "toda impunidade".

Em troca, é muito fácil que ao menos a correção formal de uma data seja controlada; o número do mês não deve ser superior a 12, etc. Conseqüentemente, se em um programa do tipo de controle de um registro civil tivermos que pedir muitas vezes as datas, vale a pena dedicar algum tempo para estabelecer uma subrotina de controle, chamada imediatamente depois de que seja introduzida uma data, e que comprove sua validade.

Vamos supor que em vários pontos de um programa existe instruções, como as linhas 100 e 110 do exemplo seguinte, nas quais se visualiza uma mensagem de aviso: "dar a data como dia, mês e ano", e logo se executa uma instrução INPUT para receber a resposta à entrada (são zeradas as três variáveis D, M e A, como se pode observar na linha 110, como uma precaução para evitar que permaneçam nestas variáveis eventuais valores de uma data anterior). Torna-se conveniente sempre colocar a zero ("fazer um reset" de) as variáveis que se utilizam em operações de entrada para evitar que, se pressionarmos imediatamente a tecla Return (ou Enter ou a equivalente), estas variáveis podem conservar um antigo valor.

Imediatamente depois da introdução da data chamamos à subrotina para sua comprovação: GOSUB 230. O número 230 é o equivalente ao "nome da subrotina" e é justamente o da primeira linha da subrotina. Se tivéssemos no programa outras subrotinas, teríamos que definí-las com outros números de linha, como é lógico. O programa salta, pois, para executar a linha 230, mas em seu interior o computador "toma nota" do número da seguinte linha (170) à que retornará ao final da subrotina.

```
10 REM ************************
20 REM * EXEMPLO DE SUBROTINA *
30 REM ************************
40 REM
100 PRINT "DE A DATA COMO DD,MM,AA"
110 D=0 : M=0 : A=0 : INPUT D,M,A
120 REM
130 REM - CHAMADA DA SUBROTINA -
140 REM - DE CONTROLE DE DATA  -
150 REM
160 GOSUB 230
170 REM
180 IF CC=0 THEN PRINT "A DATA ESTA CORRETA!"
190 GOTO 100
200 END
```

```
210 REM
220 REM
230 REM  SUBROTINA DE CONTROLE DE DATA
240 REM
250 RESTORE : CC=0
260 IF M<1 OR M>12 THEN CC=1 : GOTO 330
270 FOR K=1 TO M
280 READ LM
290 NEXT K
300 IF D<1 OR D>LM THEN CC=1 : GOTO 340
310 IF A<0 THEN CC=1 : GOTO 350
320 RETURN
330 PRINT "MES EQUIVOCADO" : GOTO 200
340 PRINT "DIA EQUIVOCADO" : GOTO 200
350 PRINT "ANO EQUIVOCADO" : GOTO 200
360 REM
370 DATA 31,28,31,30,31,30
380 DATA 31,31,30,31,30,31
390 REM
400 END
```

Antes de passar a examinar como se fez a comprovação da data, façamos algumas observações. As variáveis sobre as quais atuam as subrotinas são, em princípio, comuns a todo o programa, porque tudo o que a subrotina utiliza ou calcula se põe imediatamente à disposição do programa que efetua a chamada, à diferença do que sucede em outras linguagens, tais como o FORTRAN, nos quais é necessário explicitar as variáveis ou dados divididos e enviar e receber os dados e a das subrotinas. Uma segunda observação é que toda subrotina se dá terminada com a instrução RETURN. Podem existir várias instruções RETURN em uma subrotina, como várias instruções END em um programa, mas o importante é que sempre se volte com elas ao programa principal e não com uma instrução de salto GOTO

A utilização de GOTO para sair de uma subrotina é um gravíssimo erro, ainda que talvez não o veja imediatamente, mas que destruirá um programa quando menos se espera. Deve ter presente que a instrução RETURN não tem nada que ver com a tecla RETURN (Enter ou qualquer outra equivalente), que, ao contrário, serve para "fechar" uma linha introduzida pelo teclado. Como se pode ver, no exemplo usamos a variável CC para informar ao programa que realizou a chamada se a data é correta ou não. Estas variáveis se denominam flags ou markers (bandeiras, indicadores) e proporcionam informação lógica binária muito simples, como se tudo foi correto ou houve algum equívoco, se está bem ou está mal, etc.

Dediquemos agora algumas linhas para ver como comprovamos a data. Em primeiro lugar, analisamos o mês (linha 260). Se indicamos um mês não compreendido entre 1 e 12, se põe imediatamente CC = 1 (data equivocada) e logo se imprime a mensa-

gem "mês equivocado" e com a instrução RETURN se volta ao programa principal. Aqui, antes de prosseguir, se observa o estado do indicador ("flag") CC; se é igual a zero, a data será correta e o programa o indica e prossegue (no exemplo, voltado à simplicidade, se passa a solicitar outra data).

Voltemos à subrotina. Se o mês é correto, se analisa o número do dia. Se cada mês tem uma duração diferente se recorre ao artifício de ler a duração dos meses a partir de um arquivo de instruções DATA com um loop FOR NEXT. Por exemplo, para M = 5, correspondente ao mês de maio, o loop termina com a quinta leitura, ficando em LM o valor 31. A instrução RESTORE na linha 250 serve para inicializar o arquivo das instruções DATA antes de qualquer utilização das subrotinas. Se o dia dado à entrada está compreendido dentro dos limites permitidos, se passará a examinar o ano. Aqui, o controle é muito banal: se excluem somente os anos negativos.

As comprovações de uma data poderiam ser bastante complexas; assim se poderia ter em conta os anos bissextos, excluir ou corrigir as datas anteriores ao ano 1583 (ano de início do calendário Gregoriano), excluir os números decimais, admitir os meses em letras, admitir somente duas cifras para os anos (tal como '28 ou '85), etc.

Destas últimas considerações se deduz qual é a utilidade das subrotinas. Se quisermos melhorar ou mudar nossa rotina de controle seria suficiente substituir o conjunto de suas instruções sem ter que perturbar todo o programa. Basta saber e recordar que as variáveis de entrada à subrotina são M, D, e A e que a validade da data vem indicada com CC igual a zero. Recordamos que a instrução GOSUB é indicada nos diagramas de blocos com um retângulo com os lados verticais duplos (Figs. 3 e 4). Ao contrário, a subrotina propriamente dita se representa com um diagrama independente que termina com a instrução RETURN (Fig. 4).

Níveis de subrotinas

É preciso tratar em separado o que concerne à possibilidade de que uma subrotina seja chamada por outra (na gíria informática se fala de subrotinas aninhadas, igual como já vimos ao que acontecia com os loops FOR/NEXT). Entre as instruções de uma subrotina pode existir uma nova instrução GOSUB que chame a outra. Poderia também, do ponto de vista teórico, chamar a mesma subrotina de partida (o entrelaçamento resultante se denomina recursividade mas poucas vezes se faz isto na prática).

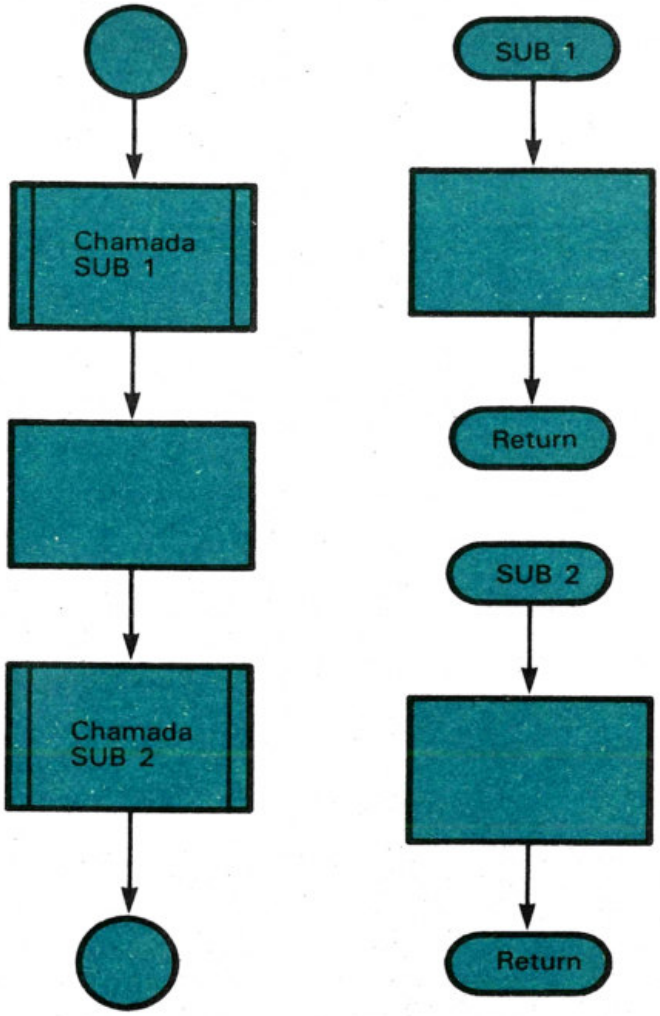


Figura 4 - Forma na qual são indicadas subrotinas nos diagramas de blocos.

O que ocorre nestes casos? Algo parecido ao que ocorria com o aninhamento dos loops FOR/NEXT. Quando o programa encontra a instrução RETURN da subrotina mais interior, prossegue a execução a partir da instrução seguinte ao último GOSUB lido. Depois da segunda instrução RETURN volta ao GOSUB precedente, a assim sucessivamente até voltar afinal ao programa principal. A Figura 5 ilustra o fluxo lógico no caso de subrotinas aninhadas. Uma subrotina pode ser chamada desde níveis diferentes; assim pode ser chamada pelo programa principal ou por outra subroti-

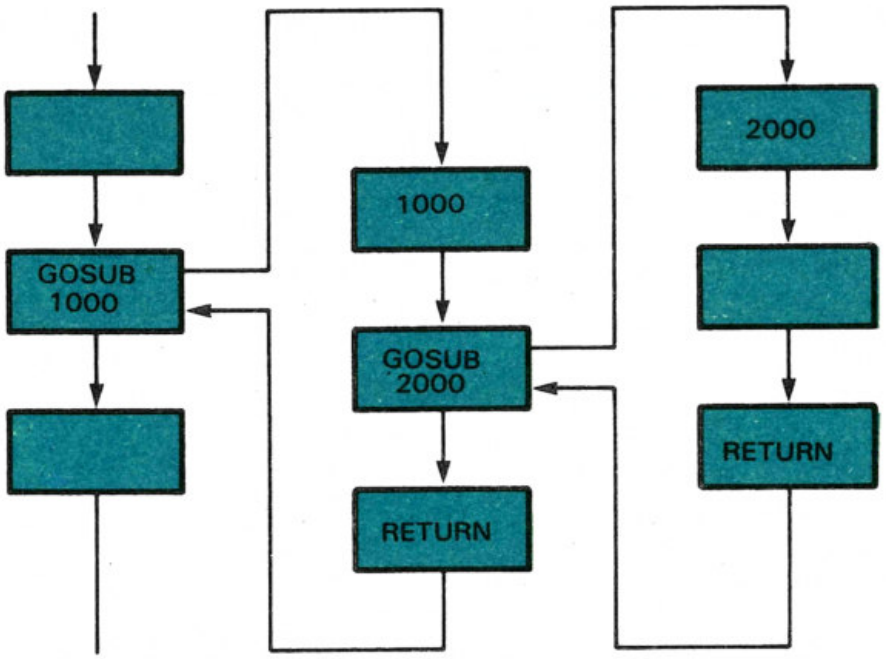


Figura 5 - Subrotinas aninhadas.

na. Em qualquer caso, voltará sempre ao ponto de onde partiu sua chamada com a instrução GOSUB.

Existem versões do BASIC nas quais uma subrotina pode chamar a si mesma um número limitado de vezes sem criar problemas; se trata de um caso de recursividade parcial. A recursividade permite aplicações muito interessantes, mas é um instrumento que somente costuma ser empregado com as linguagens estruturadas, tal como a Pascal. Em BASIC, em qualquer caso, é melhor evitar que uma subrotina chame a si mesma tanto de forma direta como através de outra subrotina.

Em geral, o BASIC estabelece um limite, bem preciso, ao número máximo de níveis de subrotinas ou, o que é o mesmo, ao número de chamadas uma dentro da outra. Ainda que este número seja bastante elevado (16 ou mais), para algumas aplicações pode não ser suficiente por depender também do número de loops FOR NEXT realizados.

Definição de novas funções DEF FN

Depois de ter falado de subrotinas e de ter visto sua utilida-

de para separar de um programa principal alguns grupos de instruções repetitivas, estudaremos as funções definidas pelo usuário, que tem um emprego similar ao das subrotinas.

Na primeira monografia, ao introduzir as expressões aritméticas indicamos que na linguagem BASIC estão disponíveis muitas funções matemáticas que se podem chamar com o simples emprego de seu nome. Por exemplo, se temos necessidade de determinar a raiz quadrada de um número X, bastará escrever SQR (X). Em BASIC existem funções para o cálculo de senos, consenos, tangentes, logarítmos, etc. Com um pouco de atenção se compreende como as funções SQR, SIN, COS, TAN, LOG, etc., são subrotinas muito particulares e privilegiadas, porque cada uma delas tem seu nome literal, igual a uma variável. Se trata de subrotinas que o fabricante do computador introduziu, de uma vez por todas, no intérprete BASIC para maior comodidade dos programadores.

Pode acontecer, ao escrever programas complexos e de tipo científico, que se tenha a necessidade de definir novas funções que interessem somente em um determinado progama. O BASIC permite fazê-lo com a instrução DEF FN (definir função). A nova função pode ser utilizada posteriormente com absoluta liberdade, mas somente no interior do programa onde se definiu, o mesmo que acontece com as subrotinas.

Vejamos um exemplo muito simples, mas de utilidade para este propósito. Vamos supor que tem que escrever um complicado programa de geometria no qual se devem calcular, muitas vezes, volumes de esferas. O volume de uma esfera, o recordamos bem, vem dado pela fórmula:

$$V = (4 \times PI \times R^3)/3$$

onde R é o raio e PI é o valor da constante matemática (3,141592...).

Ao começo do programa, antes que apareça a necessidade de determinar o volume de uma esfera, preparamos a nova função do modo seguinte:

```
50 DEF FN VOL(X) = (4*PI*R**3)/3
```

X é uma variável fictícia ("dummy") da definição que serve exclusivamente para estabelecer sobre que magnitude deve calcular-se a função (em nosso caso, o raio). Depois da linha 50, cada vez que se tem de calcular o volume de um esfera se emprega a nova função FNVOL (X), onde o prefixo FN é obrigatório e a X se atribui

o valor do raio. Deve-se ter presente que a variável argumento da função se denomina fictícia porque atua somente no interior da função, dito de outro modo, no programa poderia existir outra variável X que não tivesse nenhuma interação em absoluto com a outra. Assim, o volume da Terra, conhecendo seu raio de 6.380 quilômetros, vem dado por

```
100 PRINT FN VOL (6380)
RUN
1.0878 E + 12
```

(1087.8 milhares de milhões de quilômetros cúbicos).

Evidentemente, da mesma forma que colocamos um valor fixo, poderíamos ter usado uma variável R, por exemplo. Assim seria:

```
100 PRINT FNVOL(R)
```

O nome de uma função definida em um programa deve ir precedido por FN e segue as mesmas regras dos nomes de variáveis. A maior parte dos dialetos de BASIC permitem criar somente funções numéricas, mas em alguns casos são possíveis também as funções de cadeia. Por exemplo:

```
50 DEF FN AST$ (X$) = "***" + X$ + "***"
```

define uma função de cadeia que põe diante e detrás da dada, asteriscos.

A utilidade das funções DEF FN se aprecia em seu justo valor sobretudo quando a expressão de cálculo da função é muito complicada, pois se evita assim tê-la que voltar a escrever em mais pontos do programa.

CAPÍTULO III

OUTRAS INSTRUÇÕES DE SALTO E OS NÚMEROS ALEATÓRIOS

ON GOTO e ON GOSUB

Existe na linguagem BASIC um par de instruções que os puristas da informática chamam de "salto calculado" porque o salto, ou a chamada das subrotinas, se realiza depois de ter calculado uma determinada variável numérica. Vejamos, a título de esclarecimento, um exemplo:

```
100 ON K GOTO 150, 200, 220, 70
110...
```

Quando o programa encontra a instrução 100, a primeira coisa que faz é analisar o valor da variável K e transformá-lo em um número inteiro. Se K é igual a 1, o programa salta à linha 150; se é igual a 2, salta à segunda linha indicada (200); se é igual a 3, salta à terceira linha (220), e assim sucessivamente. Em nosso exemplo somente são indicados quatro números de linha, e, conseqüentemente, se K toma um valor superior a 4, o programa não realiza nenhum salto e prossegue na linha 110. O mesmo acontece se K é negativo ou nulo. O número de linhas que se podem indicar depois de GOTO e, conseqüentemente, de saltos possíveis, não tem praticamente nenhum limite e depende exclusivamente do

comprimento máximo que possa ter uma linha de BASIC em nosso computador.

A instrução ON GOSUB se comporta do mesmo modo que ON GOTO, com a exceção de que chama as subrotinas e ao realizar-se o "retorno" prossegue com a instrução seguinte ao ON GOSUB.

A utilização destas instruções é fácil de imaginar. Se podem realizar bifurcações múltiplas, e não somente binárias, como ocorre com a instrução IF THEN. Por exemplo, para controlar um menu de opções que ofereça quatro possibilidades, se pode pedir à entrada um número de 1 a 4 correspondente à escolha, e logo, utilizar este valor para executar a parte de programa desejada. A estrutura de dito programa aparecerá como segue:

```
100 REM  ************
110 REM  *   MENU   *
120 REM  ************
130 REM
140 PRINT "VOCE TEM QUATRO POSSIBILIDADES"
150 PRINT "TECLE UM NUMERO ENTRE 1 E 4"
160 PRINT "QUALQUER OUTRO NUMERO FAZ TERMINAR"
170 REM
180 INPUT X
190 REM
200 ON X GOTO 300,500,700,900
210 END
220 REM
230 REM
240 REM
250 REM
300 ** OPCAO 1 **
310 REM
320 REM
500 ** OPCAO 2 **
510 REM
520 REM
700 ** OPCAO 3 **
710 REM
720 REM
900 ** OPCAO 4 **
910 REM
920 REM
```

A função RANDOM (RND)

Em algumas situações pode ser muito cômodo ter à nossa disposição números aleatórios, não previsíveis a priori (em inglês, "Randon Number"). Uma aplicação típica encontramos nos jogos: o computador extrai números ao acaso que o jogador não conhece e sobre os quais está estabelecida a norma do jogo. Pode tratar-se de uma partida de "mastermind" ou do vôo de astronaves inimigas em uma guerra estrelar.

Para obter números aleatórios, a linguagem BASIC coloca à

disposição do usuário a função RND (abreviação da denominação inglesa "random"), que calcula um valor compreendido entre 0 (inclusive) e 1 (exclusive). O emprego de RND é muito fácil, porque basta utilizá-la como uma variável numérica. Qualquer uma de suas chamadas, em um mesmo programa, proporciona um valor aleatório diferente.

O comportamento da função RND não é idêntico em todas as versões de BASIC. Em geral, RND (1), ou colocando como argumento em lugar de 1 qualquer outro número positivo, calcula o valor aleatório sucessivo, enquanto que RND (0) calcula novamente o último valor extraído. Isto é:

```
PRINT RND (1)
0.725372155
PRINT RND (1)
0.384537283
PRINT RND (0)
0.384837283
```

É importante destacar que o gerador de números aleatórios contido na função RND não é, na realidade, verdadeiramente aleatório (é evidente que isto estaria em contradição com a natureza determinística do computador). É mais correto falar de seqüência de números pseudo-aleatórios, obtida aplicando um algorítmo particular; na prática, qualquer número se obtém a partir do precedente.

Como qualquer programa, também o gerador de números aleatórios (que é um programa em linguagem máquina) requer dados à entrada. Precisa um número (denominado base da seqüência) a partir do qual inicia a cadeia de números aleatórios.

Na falta desta base, o gerador partirá do número contido em uma célula da memória do computador no momento que é ligado.

Às vezes é possível fazer partir a seqüência, desde um número conhecido, isto é, se pode inicializar o programa gerador. Este último pode ser feito para obter seqüências sempre iguais ou sempre diferentes. Alguns computadores, para fazê-lo, utilizam a instrução RANDOMIZE e outros requerem um valor negativo como argumento da função RND.

Considerando que a função RND costuma gerar números no intervalo entre 0 e 1, vejamos como é possível passar a um intervalo diferente. Por exemplo, para gerar números aleatórios inteiros entre 1 e 100 se pode escrever:

INT (100*RND (1) + 1)

de fato, 100*RND gera números entre 0 e 99,9999. Ao acrescentar 1 se obtém números entre 1 e 100,9999. A função INT suprime a parte decimal e deste modo somente ficam os inteiros entre 1 e 100.

De modo análogo se podem obter números aleatórios compreendidos em qualquer intervalo, inclusive números negativos.

Um programa para embaralhar cartas

Vejamos um programa que utiliza a função RND para gerar números aleatórios. Neste caso vamos simular a operação aleatória de misturar os naipes ou embaralhar.

Vamos supor, com fins práticos, que as 52 cartas de um maço de baralho estão numeradas de 1 a 52.

Distribuir estas cartas equivale, na prática, a gerar ao acaso os números compreendidos entre 1 e 52.

Damos o programa em uma primeira versão, mais simples, que se limita a imprimir as cartas à medida que saem ou são dadas. Se quisermos conservar o maço misturado, basta utilizar um novo vetor no qual introduziríamos as cartas misturadas:

```
100 REM  **********************
110 REM  *  BARALHANDO CARTAS  *
120 REM  *  AS CARTAS VAO DO   *
130 REM  *   NUMERO 1 AO 52    *
140 REM  **********************
150 REM
160 DIM F(52) : REM  INDICADOR
    DAS CARTAS QUE APARECEM
170 REM
180 FOR I=1 TO 52
190 C=INT(52*RND(1)+1)
200 IF F(C)<>0 THEN 190
210 F(C)=1 : PRINT C : REM IMPRIME A CARTA
220 NEXT I
```

Em algumas versões de BASIC, se este programa é executado várias vezes, a sucessão de cartas é sempre a mesma, por quanto que a função RND se comporta sempre do mesmo modo e gera a mesma seqüência de números. Freqüentemente é necessário, como no caso dos jogos de azar, que os números aleatórios sejam sempre diferentes, inclusive entre sucessivas execuções do programa. Para obtê-lo basta antepor a instrução RANDOMIZE que faz partir ao acaso os números da função RND (por exemplo, poderíamos colocá-la na linha 175).

Em um programa desta classe, a única dificuldade de progra-

mação consiste em evitar gerar números iguais, pois significaria que daríamos duas vezes a mesma carta. Para solucionar este inconveniente se utiliza uma variável vetorial F ( ) como indicador ("Flag") das cartas que já foram dadas; o indicador F é controlado na linha 200. Na linha 190 se gera ao acaso um número inteiro compreendido entre 1 e 52 (uma carta de jogo) e se a carta já saiu (F< >0) se obtém outro número. Em caso contrário, o número da carta é impresso e o indicador F (C) correspondente é colocado igual a 1.

O jogador profissional

Se executar o programa que acabamos de descrever, se perceberá que a velocidade com a qual se dão as cartas é muito grande ao princípio e logo se faz menor. Isto se deve ao fato de que a função RND gera sempre um valor entre 1 e 52, pelo que, à medida que vão dando as cartas, é preciso esperar mais tempo para encontrar os números que ainda não saíram. Este problema é resolvido com um vetor M ( ) que tem em conta as cartas que já saíram e que vai encurtando de forma sucessiva.

Por este motivo, na segunda versão do programa, o loop FOR da linha 220 tem o passo (STEP) negativo. A função RND, para qualquer passo, é utilizado para extrair um número dentro de um intervalo cada vez menor: entre 1 e 52, entre 1 e 51, 50, 49 e assim, sucessivamente. O vetor M ( ) é carregado inicialmente com as 52 cartas nas linhas, 180, 190 e 200, e logo, à medida que saem as cartas, se levam a M ( ) os valores altos que vão substituir aos que acabam de sair (linha 300).

```
100 REM  ************************
110 REM  *  BARALHANDO CARTAS   *
120 REM  *  AS CARTAS VAO DO    *
130 REM  *   NUMERO 1 AO 52     *
140 REM  ************************
150 REM
160 DIM M(52)
170 REM
180 FOR L=1 TO 52
190 M(L)=L
200 NEXT L
210 REM
220 FOR J=52 TO 1 STEP -1
230 REM
240 REM  Random gera um valor
250 REM  somente entre as cartas
260 REM  que nao tenham aparecido
270 REM
280 K=INT(J*RND(1)+1)
290 PRINT M(K) : REM IMPRIME A CARTA
300 M(K)=M(J)
310 NEXT J
```

CAPÍTULO IV

DADOS E ARQUIVOS

O que são os arquivos?

Vamos falar neste capítulo de um dos temas mais atraentes e complexos da programação. Os arquivos contém grandes quantidades de dadosque, às vezes, podem chegar inclusive a milhões de informações. Tanto por suas grandes dimensões intrínsecas como para podê-los conservar durante longos períodos de tempo, os arquivos são armazenados quase sempre nas memórias exteriores do computador: fita magnética, disco rígido, cassete ou disco flexível. Estes dois últimos suportes são as memórias que costumam ser utilizadas nos computadores pessoais, tanto por seu reduzido custo como pela simplicidade de seu uso.

Como exemplo mais imediato podemos citar o fato de que também os programas são tipos particulares de arquivos. De fato, para conservá-los, quando desligamos o computador, temos que "salvaguardá-los" em uma memória externa. O mesmo pode dizer-se dos arquivos de dados, com a ressalva de que seu controle, escrita (ou gravação) e leitura, é realizada com instruções mais complexas. Lamentavelmente, a gestão dos arquivos é uma das matérias que sofre mais as diferenças entre as versões de BASIC, ainda que atualmente exista uma certa tendência a uniformizar as instruções que os controlam, graças à aceitação obtida por importantes sistemas operacionais, tais como MS-DOS ou CP/M, ou pelo BASIC da Microsoft. Mas as diferenças continuam sendo muito

grandes.
Outro motivo de disparidade no emprego dos arquivos, como veremos melhor mais adiante, se deduz do fato de que nem todos os computadores pessoais ou caseiros empregam, por motivos de custo, os discos flexíveis, que são as memórias exteriores mais cômodas e mais "naturais" para a utilização dos arquivos.

Apesar destas notáveis diversidades práticas, é possível falar de arquivos em geral e escrever programas que, com poucas modificações, são facilmente adaptáveis a muitos computadores. Como sempre em casos similares a este, se compreendermos os conceitos teóricos básicos, não fica difícil "descer" às situações concretas particulares que se encontram programando qualquer computador pessoal.

Antes de falar da programação dos arquivos, vejamos algumas idéias sobre as fitas de cassete e os discos flexíveis , assim como sobre os critérios de organização dos dados e as estruturas dos arquivos.

Fitas de cassete e discos flexíveis

Dissemos anteriormente que, entre os periférios de um computador, os que resolvem melhor o problema de conservar os dados durante um longo período de tempo são os que utilizam a gravação magnética. Todos nós sabemos quão fiel e duradoura pode ser a gravação de um concerto. O mesmo princípio é válido também para os computadores, com a única diferença de que os da-

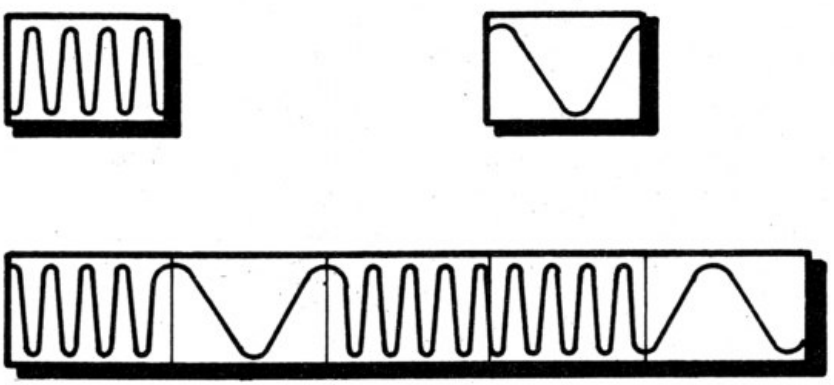


Figura 1 - Os valores 1 e 0 dos bits são armazenados como sinais senoidais de freqüência diferente.

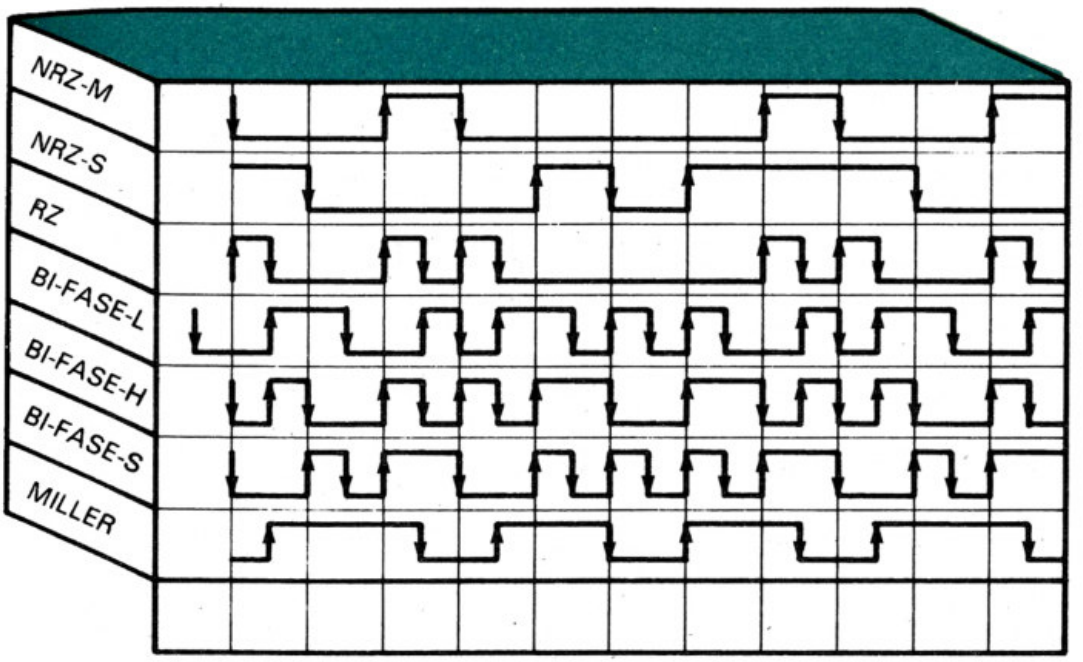


Figura 2 - Técnicas utilizadas no armazenamento magnético digital.

dos (os sinais) não são gravados de modo analógico, mas digital, como é característica da natureza do computador.

A gravação analógica é a mais imediata e fácil de perceber: um microfone gera um sinal elétrico proporcional (analógica) ao som e, por sua vez, a cabeça de gravação gera um campo magnético proporcional ao sinal elétrico. Quanto mais alta for a fidelidade com a qual o microfone converte o som e quanto mais preciso for o campo magnético gerado pela cabeça, mais alta será a qualidade da gravação.

Ao contrário, os computadores trabalham com sinais digitais (longas seqüências de uns e zeros), porque transformou-se a tecnologia magnética analógica em uma tecnologia digital. Nas fitas, nos cassetes ou nos discos flexíveis, os dados do computador são gravados como sinais que representam os valores 1 ou 0 dos bits (Fig. 1). O problema da fidelidade, tal como se considera no campo analógico, é praticamente inexistente nos computadores, porque para que uma gravação seja perfeita basta que nenhum bit 1 ou 0 se perca.

Algumas vezes, para obter resultados se recorre a técnicas complexas de gravação magnética, algumas das quais estão representadas na Figura 2.

A gravação magnética digital pode ser efetuada em duas classes de suportes: fitas e discos. As fitas para computador são muito similares às musicais. Tanto é assim, que os cassetes, que são as únicas fitas utilizadas pelos computadores pessoais, são muitas vezes as mesmas em ambos os casos. Freqüentemente, por economia, se utilizam com os computadores pessoais inclusive os gravadores em cassetes portáteis comuns.

A gravação magnética digital pode também ser realizada, e com resultados muito melhores, em suportes em forma de discos. Neste caso a organização dos dados é muito mais complexa, ainda que, em compensação, se tem um maior rendimento em seu manejo.

Gravação de dados em disco

Com independência do tipo de disco (se trate de um grosso disco rígido ou de um pequeno disco flexível), a organização física dos dados é praticamente sempre a mesma. Um disco está, em condições ideais, subdividido em muitas pistas circulares concêntricas formadas, por sua vez, por setores angulares . Os dados são gravados em blocos de comprimento constante e igual ao de um setor. Na Figura 3 se ilustra a subdivisão da superfície do disco em pistas e setores, e na Figura 4 se observa a correspondência física entre um disco e sua organização.

O conteúdo de um setor é o bloco mínimo de dados que pode ser lido ou escrito no disco. Dito de outro modo, não é possível ter acesso a um byte individual como se faz com a memória central, mas é preciso ler ou escrever um setor completo que, evidentemente, está constituído por muitos bytes.

Para ler um determinado setor, a cabeça de leitura da unidade mecânica na qual está introduzindo o disco é deslocado até o ponto correspondente à pista desejada e fica esperando que o setor procurado passe por baixo da mesma, momento no qual é lido seu conteúdo. O mesmo ocorre para gravá-lo.

Vejamos a terminologia utilizada, válida para toda classe de discos:

- face ou lado ("side") é um lado ou superfície do disco suscetível de gravação. Um disco pode utilizar uma ou duas de suas faces ou lados.
- Pista ("track") é uma circunferência imaginária na qual se gravam os dados. Em um lado do disco existe muitas pistas concêntricas (recordemos que a gravação nos discos musicais se rea-

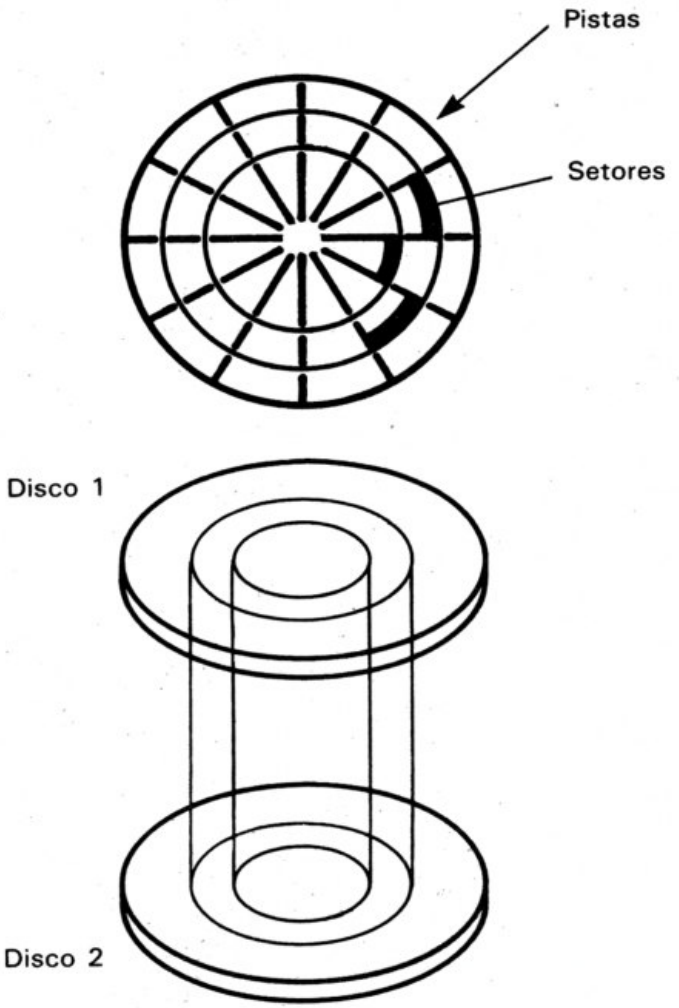


Figura 3 - Subdivisão do lado de um disco em pistas e setores.

liza, em mudança, ao longo de uma espiral).

- setor é a parte de uma pista obtida subdividindo a circunferência em um determinado número de setores angulares. Um setor contém o bloco mínimo de dados manejáveis durante uma operação de leitura ou escrita.

A capacidade total de armazenamento de uma unidade de disco depende principalmente de:

- tamanho do disco;
- número de lados utilizados. Para cada lado se deve ter uma ca-

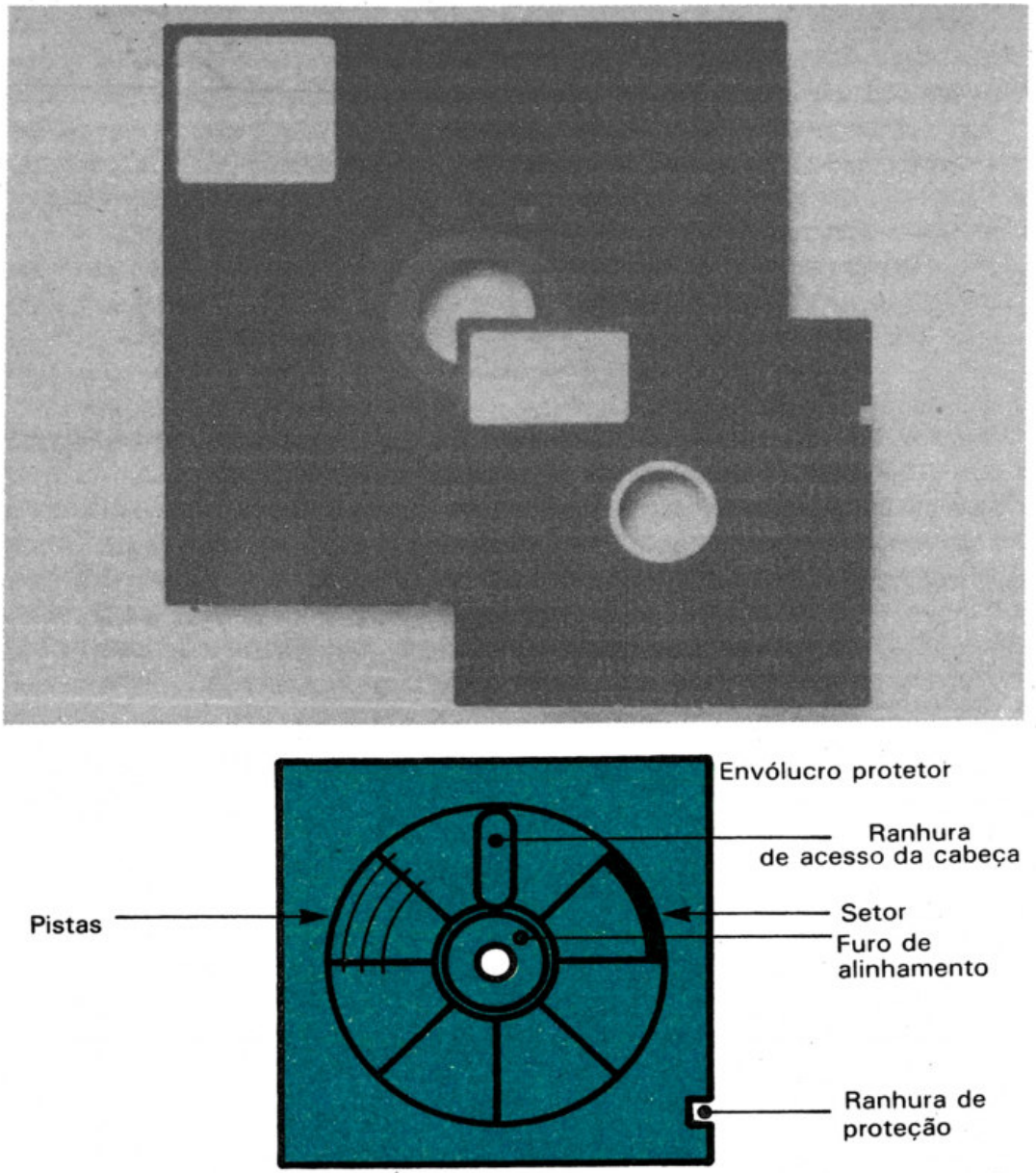


Figura 4 - Correspondência física entre o disco e sua organização.

beça e, conseqüentemente, aumentará o custo da unidade de disco;

- número de pistas por lado;
- densidade de gravação dos bits ao longo da pista, isto é, quantos bits se consegue gravar em um centímetro de comprimento.

Tipos de discos

Ainda quando muitas unidades de disco tenham um aspecto físico diferente, todas elas funcionam do mesmo modo do ponto de vista conceitual. No entanto, não devemos esquecer-nos de fazer uma pontualização que para alguns poderia parecer banal: o termo "disco" é utilizado para indicar tanto o suporte magnético no qual são gravados os dados, como a unidade periférica que efetua as gravações ou as leituras dos dados (a denominadade unidade de disco ou drive em inglês). Em qualquer caso, se não se empregar a distinção disco/unidade de disco, o significado fica claro pelo contexto no qual se emprega a denominação.

Um disco, na prática, é um conjunto de setores aos quais se pode ter acesso de modo independente e em um tempo relativamente curto. Se trata, pois, de uma memória de acesso aleatório similar, conceitualmente, a uma memória RAM, mas muito mais lenta.

No que diz respeito à capacidade de memória de um disco, um setor pode conter desde 128 a 1.024 bytes, enquanto que um disco pode ter uma capacidade total desde várias dezenas ou centenas de kilobytes (mil bytes) a milhões de bytes, segundo o número de pistas utilizadas. Passemos em revista com rapidez as classes de disco mais comuns:

- Discos rígidos fixos, construídos com a tecnologia Winchester. Costumam ser utilizados com os grandes computadores, ainda que atualmente, em suas versões menores, estão sendo difundidos também entre os computadores pessoais pela sua grande capacidade de armazenamento e sua notável fiabilidade.
- Diskpack. São pilhas de discos rígidos superpostos e solidários entre si. A gravação costuma realizar-se em todos os lados, com uma cabeça para cada lado. A capacidade destes suportes é muito grande: dezenas e centenas de megabytes. Costumam ser extraíveis da unidade de disco, como os são os discos de cartucho ("cartridge") constituídos por um só disco.
- Discos flexíveis ou disquetes ("floppy disk"). À diferença com os discos rígidos, têm o suporte constituído por uma lâmina de material plástico (mylar ou similar) recoberta pelos habituais óxidos magnéticos e estão permanentemente envolvidos por um envólucro de plástico com feltro em seu interior (ver Fig. 4). Sempre podem ser extraídos da unidade de disco. Lamentavelmente, sua duração não é comparável com a dos discos rígidos, tanto por estar em contato com a cabeça de gravação, que se deslo-

ca por cima e com o transcurso do tempo os desgasta, como por não estar protegidos contra o pó ambiental. Como máximo, armazenam, somente um milhão de bytes (ver Fig. 5), mas, em compensação, seu custo é verdadeiramente moderado. Os primeiros discos flexíveis tinham um diâmetro de 8 polegadas (como um disco musical de 45 rotações), mas imediatamente depois se produziram os de 5 1/4 polegadas (denominados minifloppy), adotados logo pelos computadores pessoais. Recentemente apareceram os "microfloppy" (de diâmetros variados, por exemplo, 3,5 polegadas), utilizados, entre outros, pelo computador Macintosh da Apple. Temos que esclarecer, de todas as formas, que não é de todo exato denominá-los "floppy" porque não são flexíveis, mas bem rígidos.

Como se observa na Figura 5, a capacidade de armazenamento de um disco, além de depender do número de lado e do número de pistas, é função também da densidade de gravação dos bits nas pistas.

Computador e sistema operacional	Lado útil	Densidade	Pistas	Setores	Bytes por setor	Bytes totais
Apple II DOS 3.3;PRODOS	1	Dupla	35	16	256	143.360
IBM-PC MS-DOS 1.1 MS-DOS 2.0	 2 2	 Dupla Dupla	 40 40	 8 9	 512 512	 327.680 368.640
TRS-80	2	Dupla	40	18	256	184.320
Apple macintosh y Lisa II	1		80	Variável	-	409.600
Sirius/Victor MS-DOS	2	Quádrupla	80	Variável	512	1.228.800
Commodore 1541 (VIC 20-C64 Commodore DOS	1	Dupla	35	Variável	256	174.648

Fig. 5. — Capacidade dos discos flexíveis empregados com alguns dos computadores pessoais, existentes no mercado mundial.

Formato e diretório

Em um disco não existe nada que indique de forma visível as pistas e os setores onde está gravada a informação. Sua superfície está recoberta por igual de óxidos magnéticos com sua cor parda característica. Como é possível, então, escrever os dados nas posições corretas?

No que diz respeito à subdivisão em pistas não tem grandes problemas, porque a mecânica de precisão da unidade de disco está em condições de deslocar a cabeça até situá-la sobre a pista desejada. O número de pistas distinguíveis depende somente da precisão da mecânica; nos discos flexíveis de 5" 1/4 são possíveis 35, 40 ou 80 pistas, enquanto que nos de 8" existe quase sempre 77 pistas. Os discos rígidos podem ter, ao contrário, centenas de pistas graças a um dispositivo que "detecta" a intensidade do campo magnético da pista localizada a partir da cabeça.

Uma vez que temos então a cabeça situada sobre a pista desejada, é preciso estabelecer o ponto de começo dos setores. Isto pode ser conseguido de duas maneiras: marcando de forma física no disco com orifícios os setores (a operação denominada "setorização" por hardware) ou então com a gravação de sinais especiais (a denominada "setorização" por software). O primeiro método é muito utilizado com os discos rígidos ("hard disk"), enquanto que o segundo é empregado em quase todos os computadores pessoais para os discos flexíveis.

Com a setorização por software os setores se identificam mediante informações (na chamada "cabeceira"), gravadas na própria pista. Estes indicadores, que não estão presentes em um disco novo (virgem) dependem do tipo de computador utilizado ou, mais exatamente, de seu software para disco, de seu sistema operacional de disco (ou DOS).

Assim, quando se quer empregar um novo disco é preciso formatá-lo (ou inicializá-lo) em primeiro lugar. Esta operação, executável também em discos antigos que se queiram reutilizar como novos, além de sinalizar os setores costuma gravar no disco a tabela de controle dos arquivos, denominada diretório.

O diretório (ou tabela índice) contém as informações necessárias para identificar todos os arquivos gravados no disco. Uma parte desta informação é possível de ser lida com o comando DIR do BASIC (ou CATALOG, segundo o computador que se trate). A principal informação contida em um diretório é:

- nomes dos arquivos,

- tipos de arquivo,
- setores nos quais se encontram os arquivos,
- dimensões dos arquivos,
- data de sua criação,
- data da última modificação,
- permissão de acesso aos arquivos.

A informação mais importante é a relativa à identificação dos setores onde se encontra o arquivo; graças a isto torna-se possível ter acesso direto aos arquivos individuais ou a seus registros.

Arquivos seqüenciais

Quase todas as versões de BASIC permitem empregar dois tipos de arquivos: seqüenciais e diretos (ou aleatórios). Nos arquivos seqüenciais, que são os mais simples, os dados estão gravados um após outro, como se o dispositivo de memória de massa fosse sempre uma fita magnética (Fig. 6). A informação contida no arquivo pode, por sua vez, subdividir-se em registros e campos, mas, em qualquer caso, se gravarão ou voltarão a ler sempre em ordem a partir do começo do arquivo. Funcionam como se fossem uma casa de vizinhos: uma vez que entramos no portal (arquivo), se quisermos chegar ao terceiro andar (registro) devemos passar

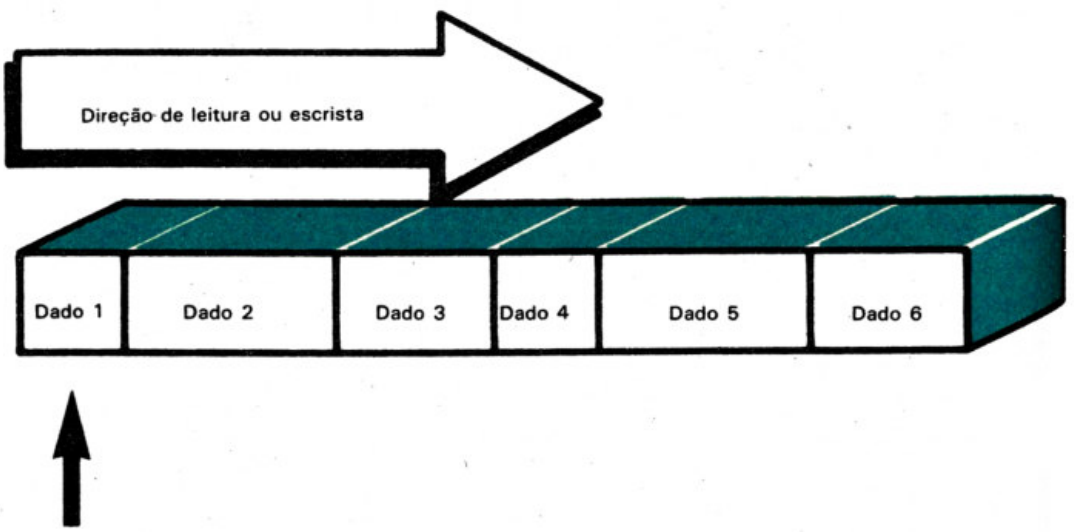


Figura 6 - Em um arquivo seqüencial, para ter acesso a um registro concreto se deve passar através de todos os precedentes.

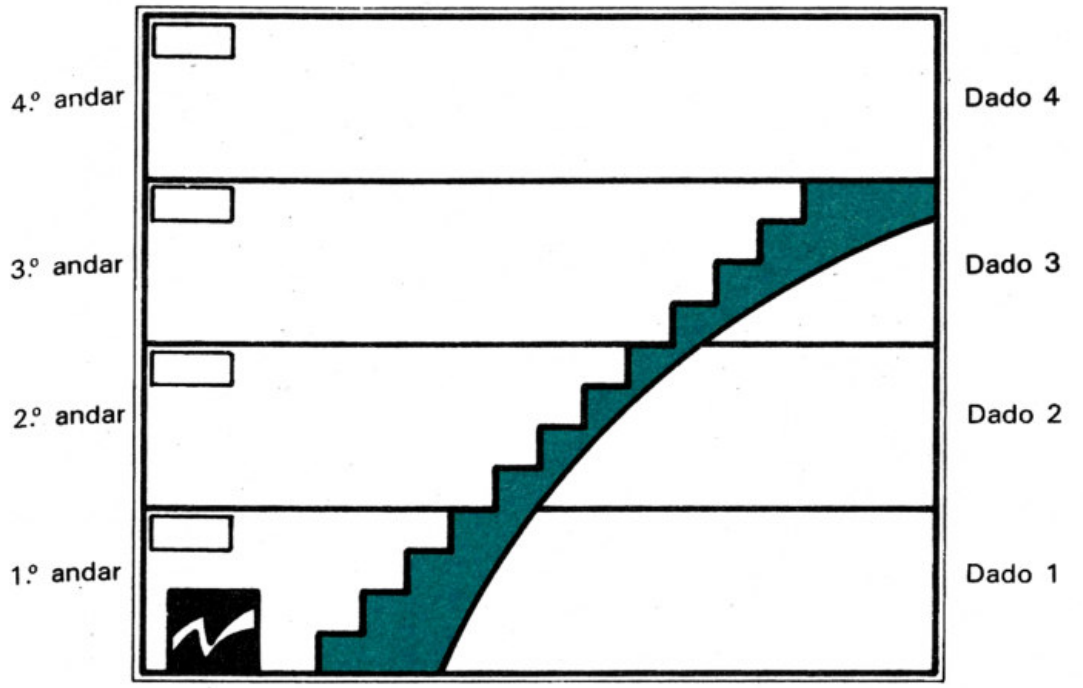


Figura 7 - Em uma casa quando queremos chegar ao 3.º andar temos que passar pelo 1.º e 2.º, em um arquivo sequencial para alcançar um dado devemos antes recorrer todos os anteriores.

antes pelo primeiro e pelo segundo (Fig. 7).

Freqüentemente, um arquivo seqüencial é criado "imprimindo" os dados no próprio arquivo com a instrução PRINT # equivalente à instrução PRINT normal, que visualiza, em câmbio, os dados na tela. Um arquivo seqüencial se assemelha, a todos os efeitos, em uma longa cadeia de caracteres.

Para ler se deve partir sempre do começo utilizando, se forma seqüencial, instruções INPUT # completamente similares às instruções INPUT. Na prática, os dados são lidos do arquivo como se fossem lidos a partir do teclado.

Um dado pode ser também um registro completo, constituído por vários campos. Em tal caso, é preciso para cada registro escrever (ou ler) um após o outro, em sua ordem, todos os campos que o compõem.

Os arquivos seqüenciais servem para todas as aplicações nas quais não é fundamental ter acesso direto aos dados situados em pontos intermediários e se pode recorrer ao arquivo do começo ao final. Um caso particular de arquivo seqüencial, ao qual já fize-

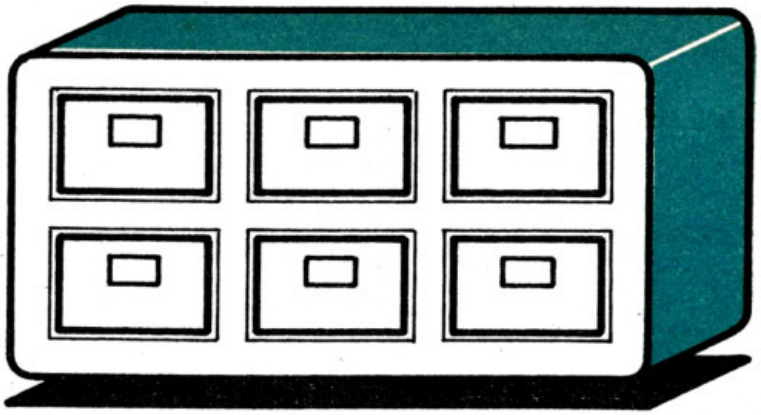


Fig. 8. — A forma de manejar um registro é parecida à que usamos quando procuramos algo em nossa cômoda: podemos abrir a gaveta que melhor nos pareça sem seguir nenhuma ordem.

mos alusão, é o dos programas. Quando se dá a ordem SAVE, o programa é armazenado da mesma forma que um arquivo seqüencial, ao menos do ponto de vista conceitual, ainda que o BASIC uti-

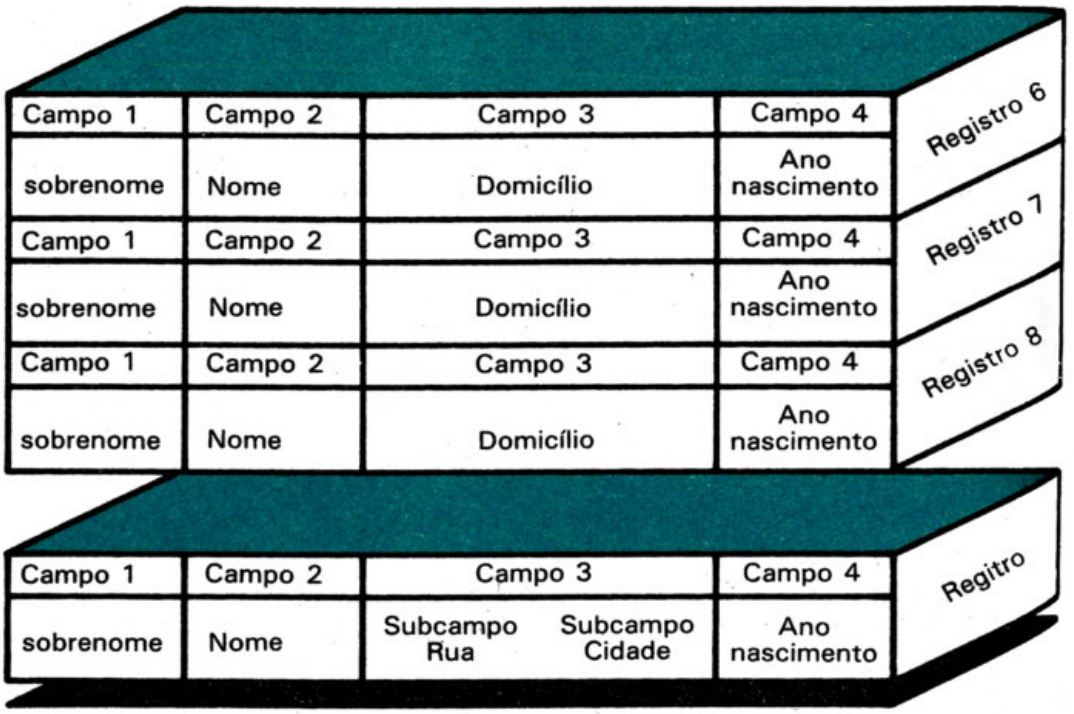


Fig. 9. — Os arquivos diretos, ou aleatórios, estão constituidos por registros que têm todos eles o mesmo comprimento.

liza ordens especiais para realizar esta operação (comandos SAVE e LOAD).

Arquivos diretos

O segundo tipo de arquivo é o dos arquivos diretos ou aleatórios. Estes arquivos têm a grande vantagem de que permitem o acesso direto a todos seus dados do mesmo modo e com o mesmo gasto de tempo. Por este motivo, em inglês o denominam "random" (aleatórios), precisamente por sua possibilidade de alcançar imediatamente um ponto qualquer do arquivo escolhido ao acaso, sem nenhuma limitação. É como quando (Fig. 8) nos aproximamos à nossa cômoda e podemos abrir com a mesma facilidade qualquer de suas gavetas (se nenhuma estiver impedida, claro).

Com respeito aos arquivos senqüenciais, os diretos são mais práticos e eficazes, mas têm também uma estrutura mais complexa e rígida; nos arquivos diretos a informação deve estar forçosamente organizada em blocos de comprimento fixo (os denominados registros). Dito de outro modo, os arquivos diretos estão constituídos por uma série de registros, cada um dos quais pode conter uma quantidade bem definida de informação, que se estabelece de uma vez por todas quando se cria o arquivo (Fig. 9).

Por exemplo, vamos supor que os registros de um arquivo direto tenham que conter somente nomes e sobrenomes. Podemos ter pessoas com o nome muito curto ("João Paz") e outros com o nome muito longo ("José Antonio Campuzano"). O primeiro está contituído por oito caracteres (incluindo o espaço) enquanto que o segundo tem vinte e dois. Se quisermos que o arquivo contenha estes nomes, todos os registros deverão ter um comprimento fixo (mínimo) de vinte e dois caracteres (ou vinte e três para alguns computadores que acrescentam um caracter separador entre os registros). Isto quer dizer que para o primeiro nome desperdiçamos 14 bytes, o qual é um desperdício considerável. Não existe nenhuma outra alternativa nos arquivos diretos: ou cortamos os nomes mais longos ou desperdiçamos espaço com os mais curtos.

Estruturas e acesso

Até agora utilizamos os termos seqüencial e direto para referir-se tanto à estrutura do arquivo como ao método de acesso aos dados. Na realidade, trata-se de dois conceitos bem distintos e teria

sido mais exato falar de:

- estrutura de arquivo (seqüencial ou direto). Forma na qual estão organizados, isto é, escritos, os dados;
- método de acesso (seqüencial ou direto). Forma de ter acesso aos dados do arquivo, tanto na fase da escrita como na de leitura.

É fácil perceber, e também recordar quando se escrevem os programas, que em um arquivo com estrutura seqüencial somente se pode ter um acesso seqüencial, enquanto que em um arquivo com estrutura direta se pode ter acesso de qualquer modo.

Conseqüentemente, a escolha de uma estrutura ou de outra depende não somente do tipo de dados que se tenham, como também da classe de emprego que vamos dar e do tipo de computador que utilizamos. Por exemplo, os menores computadores caseiros não costumam permitir controlar arquivos diretos e, às vezes, nem sequer seqüenciais.

Neste ponto, voltamos a falar de computador pessoal. Estes computadores costumam trabalhar somente com gravadores de cassete (como nos musicais) e, conseqüentemente, não permitem controles sofisticados dos arquivos. Para falar de arquivos, como se precaver já, é preciso, na prática, referir-nos aos computadores que utilizam discos flexíveis.

Antes de prosseguir para ver finalmente as instruções BASIC relativas aos arquivos, façamos uma pontualização técnica com respeito ao emprego dos gravadores para cassetes comuns (musicais, para entender-nos). Para controlar os arquivos seqüenciais, nem todos estes gravadores proporcionam resultados satisfatórios. De fato, um arquivo seqüencial é uma sucessão de registros independentes entre si, que devem ser escritos ou lidos de maneira seqüencial, mas não necessariamente todos com continuidade temporal: entre a escrita, ou leitura, de um registro e o seguinte pode existir uma longa pausa; basta considerar o tempo necessário para preparar um novo registro antes de escrevê-lo.

Por este motivo o motor do gravador deve poder ser controlável pelo computador, que o fará avançar ou parar quando for necessário. Os arquivos seqüenciais em cassete podem ser utilizados, pois, somente com os computadores que tenham o controle remoto do gravador. Com maior razão, os arquivos diretos nunca podem ser escritos em nenhum tipo de fita magnética, nem sequer nas empregadas pelos grandes computadores, porque estes arquivos requerem que se possa ter acesso direto a cada um de seus registros.

CAPÍTULO V

GESTÃO DE ARQUIVOS EM BASIC

Abertura e Fechamento de Arquivos

Agora que temos mais claras as idéias sobre a estrutura dos arquivos e os suportes sobre os quais criá-los (fitas ou discos) voltemos à linguagem BASIC. Vamos descrever a forma na qual temos de proceder para controlar um arquivo.

Vamos supor que existem muitos dados para armazenar: por exemplo, uma lista de endereços. Então, para utilizar um arquivo é preciso, em primeiro lugar, abrí-lo, isto é, avisar ao computador de que desejamos uma conexão "lógica" com um gravador de cassete ou um disco.

Uma analogia válida da abertura de um arquivo é a de telefonar: quando marcamos um número e a pessoa chamada atende o telefone, se estabelece uma conexão e fica "aberta" a comunicação.

Somente depois de um arquivo ser aberto é possível realizar no mesmo as operações de leitura ou gravação dos dados. Nesse preciso momento se faz acessível ("transparente") para o usuário e, em um certo sentido, pode comparar-se com uma grande expansão da memória central. Ao acabar as operações de leitura e de escrita o arquivo deverá ser fechado para cortar a conexão lógica que se havia estabelecido com a abertura inicial.

Digamos imediatamente algo muito importante: um arquivo, uma vez aberto, preenchido de dados e fechado se conservará ainda quando termine o programa que o escreveu. Os arquivos têm

"vida própria" graças ao fato de que estão depositados em memórias magnéticas e, conseqüentemente, não perdem a informação ao desligar o computador. Uma fita de cassete (ou um disco flexível) conservam dados e programas durante muitos anos, o mesmo que acontece no caso das gravações musicais. É muito raro que um suporte magnético perca a informação se forem tomadas algumas mínimas precauções, como não molhá-lo, esquentá-lo ou sujá-lo.

Um arquivo, além de ser um grande meio de armazenamento de nossos dados, pode servir também de ponte de conexão entre dois programas. Por exemplo, se dois programas são demasiadamente amplos para residir ambos na memória central (como um programa único) podem ser carregados, um a um, no computador e intercambiar seus dados através de um arquivo temporário (Fig. 1).

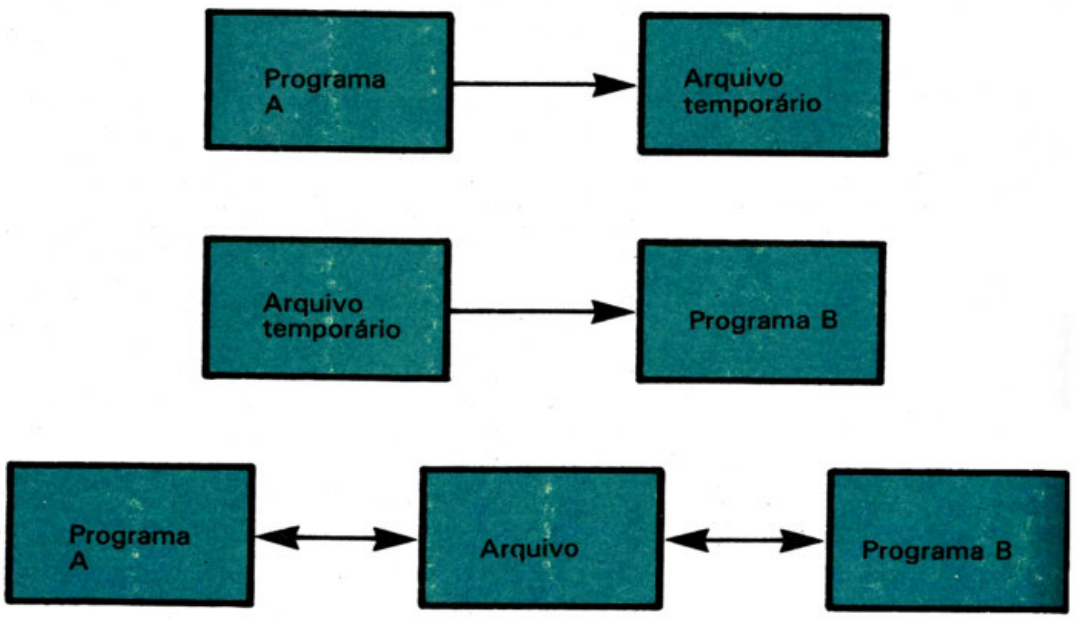


Figura 1 - Um arquivo temporário pode ser usado para transferir os dados desde um programa a outro.

Quando se trabalha com discos flexíveis se podem utilizar (abrir) simultaneamente vários arquivos e desenvolver várias operações de leitura e de escrita de dados. É evidente que cada arquivo aberto deve ter um nome diferente (um número de telefone diferente na analogia da chamada telefônica). O número máximo de

arquivos simultaneamente abertos não é o mesmo para todos os computadores.

Open

A primeira instrução de BASIC que encontramos para controlar os arquivos é OPEN. Esta instrução OPEN é a que avisa ao computador de que queremos trabalhar com um arquivo.

As regras que regem seu emprego variam muito segundo a versão de BASIC, mas todas elas têm em comum que devem estabelecer as características de arquivo e em qual periférico (cassete ou disco) deve efetuar-se a gravação ou leitura dos dados. Os parâmetros das instruções OPEN são:

- tipo de arquivo (seqüencial ou direto);
- tipo de operação (leitura ou escrita dos dados);
- um código numérico denominado canal ou arquivo lógico, que associar-se-á a dito arquivo para as sucessivas instruções de leitura ou escrita dos dados (PRINT #, INPUT #, etc.). Se forem utilizados vários arquivos simultaneamente este código serve também para distinguí-los;
- nome do arquivo, e
- periférico com o qual se deve realizar a conexão.

É muito habitual que nos computadores que admitem arquivos seqüenciais e diretos o formato das instruções OPEN não coincidam.

Vejamos alguns exemplos, com a indicação de qual computador se referem.

Commodore 64:

OPEN < n.º canal >, 1, < modo <, " < nome > ".

100 OPEN 5, 1,O, "AGENDA"

Quer dizer que o arquivo com o nome AGENDA se escreve (o que se indica por O = Output, arquivo de saída) desde a fita de cassete, única opção possível (o que se indica por 1) através do canal lógico 5.

100 OPEN 4, 1, I, "TELEFONE"

Neste caso, o arquivo TELEFONE se lê (I = input, de entrada) na fita de cassete (1) através do canal lógico 4.

Spectrum:

Este computador utiliza a instrução OPEN somente com as unidades Microdrive (pequenos cassetes que trabalham em blocos como os discos).

Nas unidades de cassete não é possível abrir arquivos, ainda que se possam gravar nelas blocos completos de dados sob a forma de vetores ou matrizes com instruções SAVE DATA. Por exemplo:

```
100 SAVE "DIREÇÕES" DATA A$( )
```

Faz com que a matriz de cadeia A$( ) se armazene na fita de cassete com o nome de DIREÇÕES. Deve levar-se em consideração que a palavra DATA significa "dados" e não deve ser confundida com a instrução DATA READ.

Para a leitura da matriz completa A$ se emprega a instrução LOAD DATA:

```
100 LOAD "DIREÇÕES" DATA A$( )
```

Apple II:

Os computadores da Apple fazem uso prodominantemente de discos flexíveis. A instrução de abertura de um arquivo como o DOS 3.3 (o software operacional para discos da Apple II) deve ser dada através de uma instrução PRINT, fazendo preceder o comando OPEN por um caracter CONTROL-D (CTRL-D). Para um arquivo seqüencial seria:

```
100 D$ = CHR$ (4) :REM  <CTRL D>

110 PRINT D$; "OPEN DIREÇÕES"
```

Com o anterior se abrirá no disco o arquivo seqüencial DIREÇÕES. O DOS 3.3 não utiliza número de canal. Desculpe a aparente dificuldade destas instruções, mas tem que aprender as coisas tais como são. Para não aprofundar-nos demasiadamente no funcionamento das máquinas individuais, não demos a explicação detalhada de todos os parâmetros das instruções OPEN: de vez em quando é necessário consultar os manuais dos computadores.

A abertura dos arquivos diretos se consegue no Apple acrescentando o parâmetro L, que indica o comprimento do registro:

```
100 D$ = CHR$ (4):    REM CTRL D

110 PRINT D$; "OPEN DIRETO, L50"
```

Isto quer dizer que o arquivo DIRETO, de estrutura aleatória tem os registros com um comprimento de 50 caracteres cada um.

IBM PC:

Este computador trabalha com o BASIC Microsoft, que costuma ser utilizado com o sistema operacional MS-DOS. As instruções relativas aos arquivos são muito simples e elegantes. Para uma seqüencial.

```
OPEN " < modo > ", # <canal> , " <nome > "
100 OPEN "O", # 5, "B:ESTUDANTES"
```

O arquivo seqüencial ESTUDANTES se abre com o número de canal 5 na unidade B. As operações são de escrita (o "O" indica saída do computador).

```
100 OPEN "I", 5, "B:ESTUDANTES"
```

Como a anterior, mas as operações são de leitura ("I" por input igual entrada).

Para os diretos:

```
OPEN "R", # <canal> , " <nome> ", <comprimento>
100 OPEN "R", # 4, "A:VÔOS.1",230
```

O arquivo direto VÔOS.1 se abre com o número de canal 4. O parâmetro "R" indica que o arquivo é direto. Os registros têm um comprimento de 230 caracteres. As operações podem ser de leitura ou de escrita.

Equipamentos MSX:

Para os seqüenciais (desde cassete) usam:

```
OPEN " <nome >" FOR <modo >AS <canal>
```

e para os diretos:

```
OPEN " <nomes>" [FOR <modo>] AS [#] <canal>
[LEN = <comprimento >]
```

CLOSE

Ao final das operações de leitura ou escrita de qualquer tipo de arquivo, é imprescindível fechar o canal ou, dito de outro modo, desconectar logicamente o gravador para cassete ou o disco flexível (... equivale a dependurar o telefone).

A instrução de BASIC é, neste caso, muito simples: CLOSE seguida pela indicação do número de canal incluído na instrução OPEN correspondente (atenção a esta correspondência). Por exemplo:

```
500 CLOSE# 1
```

Serve para fechar o canal 1.

Somente o computador Apple com DOS 3.3 apresenta uma complicação, devida à obrigação de dar o caracter Control D com PRINT:

```
500 PRINT CHR$(4); "CLOSE"
```

Quando um arquivo está aberto como de leitura, a falta de fechamento, CLOSE, não produz nenhum dano efetivo. Ao contrário no caso de escrita, a falta de fechamento pode dar lugar à perda do último bloco de dados, que não se grava no arquivo.

O BASIC e os arquivos sequenciais

Depois de ter visto as instruções de abertura e de fechamento, tanto para os arquivos seqüenciais como para os diretos, passamos às instruções necessárias para escrever e ler os registros individuais; primeiro veremos as correspondentes aos arquivos sequenciais e depois as relativas aos arquivos diretos.

PRINT

Escrever os dados em um arquivo sequencial de BASIC é muito simples. Se procede do ponto de vista conceitual, como se os dados fossem visualizados na tela: cada linha equivale a um registro.

A instrução PRINT # se comporta como PRINT, mas em lugar de enviar os dados à tela os envia ao arquivo (atenção ao sinal # denominado "grade", que distingue as duas instruções).

Para obter a separação dos registros em campos, reconhecíveis logo na fase de leitura por uma instrução INPUT #, é preciso escrever entre os valores de cada campo, de forma expressa, vírgulas, pontos e vírgula e outros separadores válidos segundo o computador de que se trate. Vejamos um exemplo no qual se "imprimem" no arquivo, aberto antes com o número de canal número 6 (# 6), dos campos C1S e C2S:

```
200 PRINT = 6, C1$: ","; C2$
```

A gravação da vírgula "," (separador de campo) é fundamental para permitir ao computador, quando as ler depois, separar as duas cadeias C1$ e C2$. Sem esta vírgula, o computador consideraria que se trata de uma cadeia única, união de C1$ e C2$.

Quando se gravam cadeias é preciso prestar atenção a seu comprimento máximo, que varia de um computador para outro. Estas limitações dependem, em grande medida, do tipo de instruções que foram utilizadas — para ler o arquivo. Por exemplo, com a instrução INPUT de Commodore não se pode ler mais de 80 caracteres, incluído o caracter Return, situado ao final de um registro como separador.

Exemplo de escrita em um arquivo seqüencial

Propomos um exemplo "sui generis" de arquivo sequencial: a escrita de um arquivo no qual estão inclusos os elementos de uma matriz retangular de números. A gravação se iniciará desde o princípio, linha por linha.

Sabemos que em BASIC as matrizes são tabeladas (ou "arrays") que residem na memória central, como todos as variáveis, mas podem existir dois motivos válidos para querer que os dados destas tabelas se incluam em um arquivo externo. O primeiro, evidentemente, é a conservação das próprias matrizes, e o segundo é que as matrizes muito grande não podem "residir" por completo na memória central. Por exemplo, uma matriz de 100x100 números inteiros ocupa 20.000 bytes (10.000 números, cada um dos quais está armazenado em 2 bytes). Muitos dos pequenos computadores não podem dispor facilmente de tanta capacidade de memória para uma só matriz, pelo que se tem que gravá-las em periféricos externos.

Vejamos, pois, como escrever em um arquivo seqüencial uma matriz, elemento por elemento e linha por linha.

Vamos supor que a matriz tenha 100 filas e 50 colunas, o que

indicamos na linha 160 com as duas variáveis TR e TC. Na linha 190 abrimos o arquivo (as instruções são as de BASIC Microsoft), onde "O" indica que o arquivo está aberto em escrita ("output"), o número de canal que escolhemos é o número 1 (# 1) e o nome que damos ao arquivo é de MAT.SEC.

Os dois loops FOR NEXT permitem tomar na entrada, desde o teclado, e logo escrever no arquivo, um por um, os elementos da matriz. Na linha 270 se apresenta na tela a mensagem de solicitação de um elemento. Por exemplo, para o primeiro elemento apareceria:

FILA 1 — COL 1:

ao qual se deve responder com o valor numérico do primeiro elemento. Tenha presente que estes números são de tipo real, e não inteiro, como dissemos anteriormente, pelo que a matriz ocupa muito mais bytes (4 bytes para cada número real no BASIC Microsoft).

Na linha 290 o número E é gravado no arquivo. Cada registro do arquivo contém, pois, um único número real.

A instrução CLOSE # 1 na linha 330 fecha o arquivo. Sua existência é fundamental, porque garante que o último bloco de dados se transfira efetivamente à memória exterior.

Tratemos de explicar melhor este ponto, a que se fez alusão anteriormente. A gravação dos dados em um disco (ou em uma fita de cassete) não tem lugar realmente cada vez que o computador encontra uma instrução PRINT #, mas somente quando o buffer preparado para dito arquivo está cheio de dados. Não tem porque se preocupar, já que estes problemas não afetam ao programador, mas que são competência exclusiva do sistema operacional DOS.

Se por exemplo, o buffer tem um comprimento de 256 bytes e se gravam números de 4 bytes aos que se acrescenta um caracter separador de registro (como Return, ASCII 13), dando um total de 5 bytes por registro (nos arquivos diretos a questão é diferente ao faltar os separadores), no buffer cabem 256/5 = 51 registros completos. O byte que sobra é inutilizado. Neste caso somente depois de 51 instruções PRINT # 1, o buffer se "transvazará" ao exterior e se verá girar o disco ou a fita de cassete. Quando se executa a última instrução PRINT # é pouco provável que coincida com a chegada do buffer, pelo que se neste momento desligarmos o computador ou finalizarmos o programa, os últimos registros que se encontram carregados no buffer se perderão. CLOSE

tem precisamente o encargo de armazenar o último resto de arquivo, isto é, "transvazar" o último buffer, além de avisar ao sistema DOS de que as operações nesse arquivo estão terminadas.

```
100 REM  ***************************
110 REM  * GRAVACAO DE UMA MATRIZ *
120 REM  *     DE NUMEROS EM UM    *
130 REM  *   ARQUIVO SEQUENCIAL    *
140 REM  ***************************
150 REM
160 TR=100 : TC=50 : REM MATRIZ DE 100 x 50
170 REM
180 REM
190 OPEN "O",#1,"MAT.SEC"
200 REM
210 REM  GRAVACAO DA MATRIZ
220 REM
230 PRINT "DAR OS VALORES DOS ELEMENTOS: "
240 REM
250 FOR J=1 TO TR
260 FOR K=1 TO TC
270 PRINT "LINHA ";J;" -COL ";K;" :"
280 INPUT E
290 PRINT #1,E
300 NEXT K
310 NEXT J
320 REM
330 CLOSE #1
340 REM
350 END
```

Mais adiante, depois de estudarmos as instruções de leitura dos arquivos seqüenciais, veremos como ler a matriz.

A modo de inciso, recordemos que o computador Spectrum realiza a gravação de uma matriz diretamente com a instrução SAVE DATA.

INPUT # e LINE INPUT #

A leitura dos arquivos seqüenciais pode ser realizada de duas maneiras: lendo um registro completo (INPUT # e LINE INPUT #) ou então lendo um caracter cada vez (GET #ou INPUT$). Deve-se prestar atenção a esta instrução GET # , que não deve ser confudida com a instruçaõ GET # do BASIC Microsoft, por exemplo que atua, ao contrário, sobre os arquivos diretos e do qual falaremos em seguida.

No primeiro caso, INPUT # se comporta como sua "prima irmã" INPUT. INPUT recebe os dados desde o teclado, enquanto que

INPUT # os recebe desde o arquivo. As variáveis indicadas na instrução, devem corresponder ao tipo de dados gravados no arquivo.

INPUT # <canal> , variável 1>\ ,<var 2> ...]

As vírgulas gravadas com PRINT # separam os campos e equivalem às introduzidas pelo teclado para separar os dados à entrada. O caracter Return que separa os registros equivale ao pressionamento de Return no teclado. Por exemplo, para ler quanto se escreveu com a instrução PRINT # indicada anteriormente se deve utilizar:

```
600 INPUT # 2,A$,B$
```

Com LINE INPUT # se lê ao contrário, um registro completo, até o caracter Return.Esta intrução é utilizada somente com variáveis de cadeia e cada eventual distribuição lógica em campos deve fazer-se em separado. A importância de LINE INPUT # está no fato de que admite uma linha completa com quaisquer caracteres (vírgulas e outros) e a atribuição a uma só variável.
Por exemplo:

```
600 LINE INPUT # 1, L$
```

GET # e INPUTS

Acabamos de fazer alusão do fato que, freqüentemente, é preferível ler um arquivo seqüencial que contenha dados do tipo de cadeia, em forma de um caracter cada vez. Isto é para evitar os incovenientes da instrução INPUT # , que exige uma correspondência rígida entre suas varíaveis e os campos de registro separados pelas vírgulas. Esta leitura caracter por caracter se obtém pela instrução INPUT$ ou com GET # . Neste caso já não existem campos e registros do arquivo, mas a distribuição lógica deve ser reconstruída pelo programa de leitura.

Assim, no exemplo seguinte, na cadeia R$ "acumulamos" um registro lendo um caracter cada vez. Ao inicio de cada campo, a cadeia R$ se colocará em zero (linha 150); atavés da instrução INPUT$ se lê e atribui à variável A$, um caracter desde o canal número 1 (#1) aberto anteriormente. O primeiro número 1, entre os parênteses da instrução, indica precisamente que se lê um só caracter; na realidade, INPUT$ permitiria que se lesse mais de um. Imediatamente depois se efetuam duas comprovações: se o carac-

ter lido é uma vírgula (ASCII 44), o programa passará ao controle de campo, e se o caracter é Return (ASCII 13), se terminará o registro. Em todos os demais casos, o caracter é acrescentado a cadeia R$ (linha 190). Depois de controle de campo de registro, o programa voltará à linha 150.

```
150 R$ = " # "
160 A$ = INPUT$(1, # 1)
165 REM
170 IF A$ # CHR$(44) THEN GOTO 200
180 IF A$ # CHR$(13) THEN GOTO 300
190 R$ = R$ + A$: GOTO 160
195 REM
200...R$ CONTEM UM CAMPO
...
300...FINAL DE REGISTRO, RS CONTÉM O ÚLTIMO CAMPO
```

Se empregarmos a instrução GET # , a linha 160 deve mudar-se como segue:

```
160 GET # 1, A$
```

Se o arquivo contém dados numéricos, gravados com variáveis alfanuméricas, se deve utilizar INPUT #, porque GET # e INPUT$, ao ler um caracter cada vez, se torna muito difícil a reconstrução dos números expressos em forma exponencial (por exemplo: 3.456 E-8).

Se poderia pensar que a leitura de um arquivo com GET # (um caracter de cada vez) é mutio mais lenta em relação com a efetuada com INPUT # (um registro completo), mas, na realidade, o tempo empregado é praticamente igual. GET # oferece, no entanto, a vantagem de qualquer coisa que se escreva no arquivo pode ser lida como se fosse programada em linguagem máquina. Exige, pois um pouco de domínio e de atenção.

Exemplo de leitura em um arquivo seqüencial

Passemos agora a ler o arquivo seqüencial escrito anteriormente para "magnetizar" uma matriz numérica.

Inclusive neste caso a primeira instrução operacional de programa deve ser a abertura de arquivo. Na linha 190 utilizamos OPEN com o parâmetro "I" (INPUT # de entrada) seguido pelo nú-

mero de canal (1).

Este número que usamos não depende, em absoluto, do número de canal empregado no programa de escrita, e somente por casualidade se escolheu também o 1. Ao contrário, o nome do arquivo MAT.SEC. deve ser exatamente o utlizado na escrita, porque somente assim o sistema DOS pode buscar no disco o arquivo correto que contém a matriz desejada.

A leitura da matriz é muito simples e basta utilizar uma instrução INPUT # com uma variável numérica (variável que pode ser diferente da empregada na escrita). Os dois loops FOR NEXT aninhados têm como abjetivo ler os elementos da matriz linha por linha. Na linha 310, CLOSE fecha a comunicação com o arquivo. Neste caso, sua eventual omissão não produziria danos nos dados como teria acontecido, pelo contrário, no caso da escrita.

```
100 REM  ************************
110 REM  *   MATRIZ DE NUMEROS   *
120 REM  * EM ARQUIVO SEQUENCIAL *
130 REM  *        LEITURA        *
140 REM  ************************
150 REM
160 TR=100 : TC=50 : REM MATRIZ DE 100 x 50
170 REM
180 REM
190 OPEN "I",#1,"MAT.SEC"
200 REM
210 REM  IMPRESSAO DA MATRIZ
220 REM
230 FOR L=1 TO TR
240 FOR M=1 TO TC
250 INPUT #1,C
260 PRINT "LINHA ";L;" -COL ";M;
270 PRINT " : ";C
280 NEXT M
290 NEXT L
300 REM
310 CLOSE #1
320 REM
330 END
```

Se em lugar de uma matriz numérica se tratasse de uma matriz de cadeias, não teria diferenças importantes. Em lugar da variável C teríamos empregado uma varíavel de cadeia tal como E$. As únicas complicações ocorreriam se, em lugar de gravar um só número por registro quísessemos gravar mais. Neste caso teríamos forçado a gravação de "vírgulas separadoras", as quais antes fizemos alusão, deste modo:

290 PRINT # 1, A; ", "; B; ", "; C

no programa de escrita.

Isto teria permitido, na leitura, empregar varíaveis separadas entre si:

```
250 INPUT # 1,D,E,F
```

As vírgulas utilizadas em escrita se comportam exatamente igual as vírgulas que se tem de teclar para separar a entrada de vários dados quando se utiliza a instrução INPUT.

O BASIC e os arquivos diretos

Falemos agora de aquivos diretos, que somente podem ser gravados em discos, como explicamos anteriormente. Os arquivos diretos, além de ter a grande vantagem de permitir o acesso diretamente a seus dados, permitem também reescrever ou ler, sem modificar o OPEN, cada registro de forma individual.

Com os arquivos seqüenciais (repetir estes conceitos nunca faz mal) se podem fazer, ao contrário, somente operações de leitura ou de escrita, de forma separada. Por estes motivos, mas também por sua mais compacta gravação nos discos, os arquivos diretos são quase sempre preferíveis.

Façamos primeiro uma consideração. Temos utilizado o termo arquivos diretos ou de acesso direto, mas existe quem distingue mais precisamente entre arquivos diretos e arquivos relativos. A diferença é somente do tipo prático e não de natureza conceitual, porque, depois de tê-la esclarecido, seguiremos utilizando a primeira denominação. Um arquivo se denomina direto porque ao permitir apontar diretamente a cada um dos seus registros está em contraposição com os arquivos seqüenciais. Os americanos utilizam o termo "random" (aleatório) pretendendo indicar que o tempo de acesso a qualquer de seus elementos é sempre igual por termo médio.

A distinção entre arquivos diretos e relativos depende, ao contrário, do critério com o qual se identificam os registros individuais. Os arquivos diretos propriamente ditos são aqueles nos quais para apontar a um registro se dão, na prática, as indicações de pista e setor do disco. Nos arquivos relativos, a cada registro se associa um número crescente relativo (primeiro, segundo, terceiro, etc.) graças ao qual se pode apontar diretamente ao registro. Neste segundo caso corresponde ao sistema operacional disco do computador (o DOS) converter a indicação relativa do registro nos valores, a baixo nível de pista e setor.

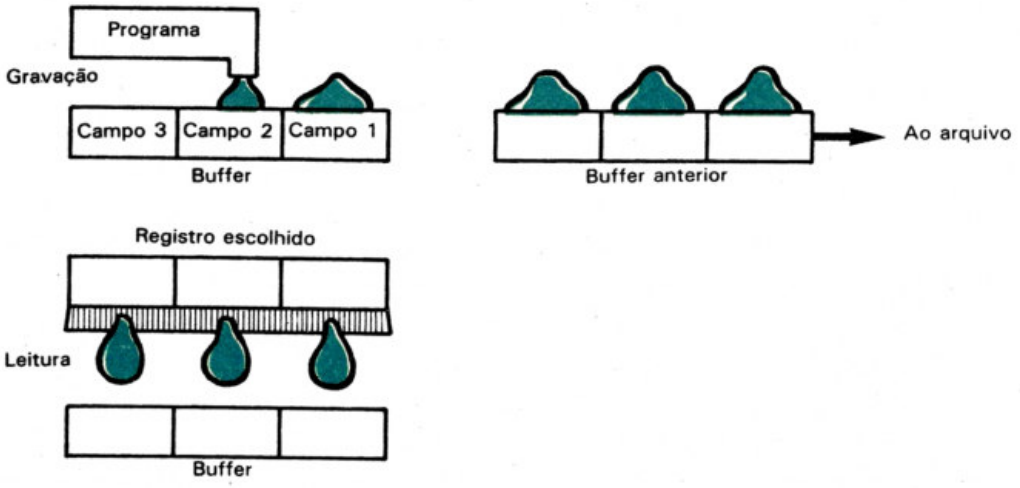


Figura 2 - Funcionamento do buffer na gravação e leitura de arquivos diretos.

Falemos então dos arquivos diretos. Considerando que estes arquivos costuman ser empregados somente nos computadores pessoais maiores, que freqüentemente trabalham com o BASIC Microsoft, vamos ver imediatamente as instruções desta linguagem. Entre os computadores caseiros são poucos os que permitem os arquivos diretos.

O ponto crucial na gestão dos arquivos diretos é a preparação da estrutura de registro. Dissemos anteriormente que, nos arquivos diretos, os registros têm todos o mesmo comprimento. Mas comprovar somente que os dados têm este comprimento não basta. E preciso imaginar um registro como uma caixa na qual situamos distintos divisores: em cada compartimento (campo) se colocam os dados efetivos. Uma vez que a caixa esteja preparada, isto é, quando o programa tiver enchido de dados os compartimentos, se enviará ao aquivo. O nome correto desta caixa é buffer, termo que encontramos anteriormente.

Um buffer é uma zona de memória temporária que se carrega e se descarrega de forma alternativa. Durante a fase de escrita do arquivo o buffer se enche passo a passo, campo por campo, e logo todo o conjunto se transmite com rapidez ao arquivo (Figura 2).

Ao contrário, durante a leitura, depois de ter apontado a um registro, este é copiado no buffer, colocando à disposição do programa seus dados, tal e como se vê na Figura 2.

A instrução do BASIC Microsoft que prepara e controla o buffer é a instrução FIELD, e veremos em seguida como funciona.

FIELD, LSET, RSET e PUT

A instrução FIELD determina uma zona de memória para o buffer indispensável na gestão de um arquivo. Com FIELD se define exatamente a estrutura de registro, indicando quantos caracteres tem de comprimento e quais são seus campos. FIELD, que deve ser utilizado tanto em escrita como em leitura, especifica quantos e quais são os campos de registro e atribui a cada um, um nome de variável. Estes nomes de variável não representam variáveis ordinárias das quais estamos acostumados a empregar em BASIC, mas variáveis muito especiais as quais atribui um valor mediante as instruções LSET ou RSET e na prática são, mais que variáveis, ponteiros dirigidos aos campos de registro.

FIELD #<CANAL >, <COMPRIMENTO 1 > AS <VARIÁVEL 1>
[, <COMP.2 > AS <VAR.2 >...]

LSET e RSET são instruções de carregamento do buffer (utilizadas somente na escrita do arquivo). A primeira, LSET ("leftset") enche os campos do buffer a partir da esquerda, como estamos acostumados para formar em colunas as cadeias, e a segunda, RSET ("right set"), realiza dita operação desde a direita, como se costuma fazer com os números.

LSET <varíavel campo > = <varíavel >
RSET <variável campo > = <varíavel >

Talvez neste ponto tenha as idéias mais confusas que nunca. Na realidade, o emprego da instrução FIELD e das intruções LSET e RSET é muito simples. Um exemplo servirá para esclarecê-lo definitivamente (RSET se utiliza em mais raras ocasiões, mas do mesmo modo).

Vamos supor que se quer escrever um arquivo direto de nomes próprios, cujo registros estão divididos em dois campos: nome e sobrenome. Antes de tudo, devemos estabelecer qual será o comprimento de cada registro; se prevemos 30 caracteres para os nomes e 20 para os sobrenomes, os registros terão um comprimento de 50 caracteres. Repetimos que esta delimitação do comprimento dos registros é característica dos arquivos diretos, en-

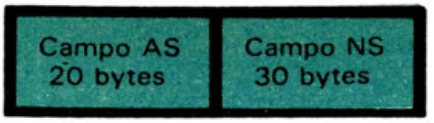


Figura 3 - Estrutura de registro.

quanto que os arquivos seqüenciais são livres de ter registros longos ou curtos, indistintamente. Vejamos as duas primeiras instruções:

```
100 OPEN "R", # 8, "NOMES PROPRIOS", 50
110 FIELD # 8,30 AS N$, 20 AS A$
```

OPEN abre um arquivo relativo ou direto ("R") com o número de canal 8, escolhido livremente entre 1 e 255. O nome do arquivo é NOMES PRÓPRIOS e 50 é o comprimento total estabelecido para todos os registros. A instrucões OPEN para os arquivos seqüenciais teria exigido em um lugar de R, um 0 para a escrita e um I para a leitura do arquivo (a leitura e a escrita dos arquivos diretos podem ser simultâneas).

A instrução FIELD, da linha 110, indica que se refere ao arquivo associado ao canal 8. Se poderiam abrir simultaneamente vários arquivos, contanto que tenham números de canal e nomes diferentes. Segundo definimos em FIELD, o registro está dividido em dois campos, chamados N$ E A$, o primeiro com um comprimento de 30 caracteres e o segundo com 20 caracteres (Fig. 3).

Neste ponto, a título de exemplo, vamos supor que se pedem à entrada 10 nomes próprios e que se queiram escrever no arquivo. Teremos que definir instruções de entrada desde o teclado, comprovar que cada nome não tenha mais de 30 caracteres de comprimento e cada sobrenome não mais de 20 caracteres, e logo, através das instruções LSET, "atribuir" a cada campo dos registros. Com a instrução PUT # os valores dos campos (no buffer) são passados ao arquivo como veremos no seguinte item. Uma vez concluidas todas as operações, se fechará o arquivo. Vejamos o programa completo:

```
100 OPEN "R",#1,"NOMES PROPRIOS",50
105 REM
110 FIELD #1,30 AS N$,20 AS A$
120 REM
```

```
130 PRINT "QUE NOME QUER LER : 1-10 (0 FINAL)"
135 REM
140 INPUT NR : REM NUMERO DE REGISTRO
150 IF NR=0 THEN 200
160 IF NR<1 OR NR>10 THEN 130
165 REM
170 GET #1,NR : REM LEITURA DO REGISTRO
175 REM
180 PRINT N$,A$ : REM IMPRESSAO DO REGISTRO
190 GOTO 130
195 REM
200 CLOSE #1 : END
```

PUT# E GET#

São as duas instruções que permitem escrever e ler um aquivo direto. Ambas as instruções, além de indicar o arquivo a que se referem (no exemplo, o associado ao canal núm. 8), têm como único parâmento o número de registro que deve ser escrito (PUT #) ou lido (GET #). Sua sintaxe é:

PUT # <canal>, <número registro>

GET # <canal>, <número registro>

Neste caso, o recordamos que a palavra reservada GET não tem nada a ver com a GET que usam alguns computadores com os arquivos seqüenciais. Este GET # pode ser lido assim: extrair o registro n-ésimo do arquivo aberto com o número de canal indicado no buffer definido pela instrução FIELD correspondente. PUT # transfere, pelo contrário, o conteúdo do buffer ao registro n-ésimo do aquivo.

GET # é algo mais simples de utilizar porque não requer as instruções LSET ou RSET. Depois de que se tenha lido um registro com GET #, as variáveis citadas na instrução FIELD podem ser utilizadas imediatamente como variáveis normais. Vejamos escrevendo o programa de leitura do arquivo NOMES PRÓPRIOS.

```
110  OPEN "R",#1,"NOMES PROPRIOS",50
105 REM
110 FIELD #1,30 AS N$,20 AS A$
120 REM
130 FOR K=1 TO 10:REM 10 NOMES PROPRIOS
```

```
140 INPUT "NOME ";DN$
150 IF LEN(DN$)>30 THEN 140
160 INPUT "SOBRENOME ";DA$
170 IF LEN(DA$)>20 THEN 160
180 LSET N$=DN$:REM CAMPO DO NOME
190 LSET A$=DA$:REM CAMPO DO SOBRENOME
195 REM
200 PUT #1,K:REM ESCRITA DO REGISTRO K-ESIMO
210 NEXT K
220 REM
230 CLOSE #1
```

A instrução PRINT na linha 180 faz uso diretamente das duas variáveis N$ e A$ da instrução FIELD, sem passar através de LSET ou RSET.

A instrução FIELD faz referência sempre a variáveis de cadeia. Se queremos gravar números em um arquivo direto, não podemos fazê-lo diretamente mediante variáveis numéricas, mas de forma indireta através de variáveis de cadeia. Para converter dados numéricos em uma cadeia se pode recorrer á função STR$, e para o contrário, a VAL, mas muitas versões de BASIC, como a de Microsoft, proporcionam um conjunto de funções especiais que fazem mais imediata a conversão de números em cadeias para gravá-los nos arquivos diretos. Estas funções são: MKI$, MKS$ eMKD$. A primeira se refere aos números inteiros; a segunda, aos números de simples precisão (números com seis cifras significativas), e a terceira, aos números de dupla precisão (com dezesseis cifras significativas).

Ao contrário, quando se lê um arquivo numérico se convertem as cadeias em números com CVI, CVS e CVD, que tratam aos números inteiros, aos de simples precisão e aos de dupla precisão, respectivamente.

Uma vez mais, com um pequeno exemplo consideramos que se esclarecerão os conceitos:

```
80 REM ***************************
90 REM *  ABERTURA DO REGISTRO  *
95 REM ***************************
100 OPEN "R",#1,"NUMERICO",8
105 REM ***************************
110 REM * DEFINICAO DO REGISTRO *
115 REM ***************************
120 FIELD #1,4 AS X$,4 AS Y$
125 REM ********************
130 REM * ESCREVO VALORES *
135 REM ********************
140 X1=32.345:Y1=23.0033
145 REM
```

```
150 RSET X$=MKS$(X1)
160 RSET Y$=MKS$(Y1)
165 REM ***************************
170 REM * INTRODUCAO NO ARQUIVO *
175 REM ***************************
180 PUT #1,1
185 REM ************************
190 REM * LEITURA DOS VALORES *
200 REM ************************
210 GET #1,1
220 REM ********************************
230 REM * TRANSFORMACAO DOS VALORES *
240 REM ********************************
250 R1=CVS(X$)
260 R2=CVS(Y$)
270 PRINT R1,R2
280 REM ***************************
285 REM * FECHAMENTO DO ARQUIVO *
290 REM ***************************
300 CLOSE #1
```

O arquivo tem um só registro com dois campos, nos quais são escritos dois números reais de precisão simples, que ocupam 4 bytes cada um, e logo são lidos.

Uma matriz numérica em arquivo direto

Nos itens anteriores vimos o exemplo de um arquivo seqüencial no qual se gravava uma grande matriz numérica. Como prometemos então, vamos resolver o mesmo problema utilizando um arquivo direto. No mesmo programa inserimos tanto a parte de escrita como a de leitura da matriz.

Também neste caso supomos que a matriz é numérica e que tem 100 filas e 50 colunas (linha 170), mas poderia ser muito mais ampla para sua gravação em disco, ou então ter dimensões variáveis e determinadas por uma sentença INPUT anterior que substituiria a linha 170.

Na linha 200 se realiza a abertura do arquivo com o parâmetro "R" (arquivo relativo ou direto), o número de canal # 1 e o nome do arquivo MAT. DIR. O número 4 (comprimento do registro) é obrigatório e sabemos que determina o comprimento de todos os registros do arquivo. O valor dado (4) pode parecer pequeno, pois os arquivos têm registros que costumam ser muito mais longos, mas o nosso é somente um exemplo que se refere a gravação de números reais (de simples precisão) que ocupam 4 bytes cada um.

Imediatamente depois da abertura do arquivo se deve préestabelecer a estrutura do registro com a instrução FIELD. Em nosso caso, o registro tem um só campo de 4 bytes de comprimen-

to. O nome do campo é F$.

Antes de explicar o significado da instrução da linha 240 e da subrotina que começa na 520, vamos ver o resto do programa. Nas linhas 260 a 300 se pergunta se pretendemos ler ou escrever o arquivo ou se queremos terminar. No terceiro caso, antes de executar END, o arquivo se fecha com CLOSE.

Se queremos ler um elemento da matriz se terá que proporcionar os números de fila e de coluna (R e C). Para passar destes dois valores ao de P, que indica a posição relativa de registro no arquivo, se utiliza a pequena fórmula da linha 360. Não esqueçamos que um arquivo é sempre uma estrutura linear, enquanto que nossa matriz é bidimensional. Para compreender como se calcula P basta pensar no corte, linha por linha, da matriz e na posterior disposição destas "fatias" em fila.

Na linha 370 GET # lê o registro P-ésimo, que se converte em um número mediante a função CVS (CV$ para os números reais com precisão simples, CVI para os inteiros e CVD para os números reais de precisão dupla).

Ao contrário, se escolhemos escrever um elemento da matriz (subrotina 420) se calculará o ponteiro P como se viu antes (linha 450) e logo se transformará em cadeia o elemento numérico com MKS$. Este valor se atribuirá ao campo F$ mediante a instrução LSET (linha 470). Finalmente, PUT # transfere o registro P-ésimo ao arquivo.

Vejamos agora o significado da linha 240. Para explicar-nos melhor, vamos supor que esta linha falta e que, antes de ter escrito nenhum registro, se trata de ler algum. À primeira vista poderíamos pensar que o computador nos deveria avisar de que esse registro não existe ou então dar-nos como resposta um valor nulo. Na realidade, o computador tenta ler a parte de disco onde deveria encontrar-se o registro e lê os bytes que encontra nesse ponto (bytes que fazem parte de qualquer outra gravação anterior). Esta situação absurda de leitura de dados "sujos" pode evitar-se de uma só maneira: preparando (inicializando) toda a zona de disco na qual se gravará o arquivo. Esta operação é precisamente a que desenvolve a subrotina na linha 520. A função STRING$ na linha 540 prepara uma cadeia de 4 caracteres "nulos" (ASCII Ø). O duplo loop FOR NEXT aponta sucessivamente a todos os registros do arquivo nos quais se escreve estes quatro caracteres nulos. Deste modo, se em qualquer momento tratar de ler um registro não escrito, o computador lhe proporcionará uma cadeia nula.

Para chamar a subrotina de inicialização do arquivo se recorre à função LOF(LOF < canal > que nos indica quantos blocos de

um arquivo relativo estão "comprometidos" com antecedência. Se este valor é nulo (linha 240) quer dizer que não se fez nenhuma escrita, e então se salta a linha 520.

```
100 REM ***************************
110 REM *   ESCRITA E LEITURA     *
120 REM * DE UMA MATRIZ NUMERICA *
130 REM *   EM ARQUIVO DIRETO     *
140 REM ***************************
150 REM
160 REM
170 TR=100:TC=50 : REM MATRIZ DE 100 X 50
180 REM
190 REM
200 OPEN "R",#1,"MAT.DIR",4
210 REM
220 FIELD #1,4 AS F$ : REM BUFFER
230 REM
240 IF LOF(1)=0 THEN GOSUB 520
250 REM
260 INPUT "LEIT/ESCR/FINAL (L,S,F)";Y$
270 IF Y$="L" THEN GOSUB 330
280 IF Y$="S" THEN GOSUB 420
290 IF Y$="F" THEN CLOSE #1 : END
300 GOTO 260
310 REM
320 REM
330 REM ----LEITURA DA MATRIZ----
340 REM
350 INPUT "LINHA E COLUNA : ";R,C
360 P=(R-1)*TC+C
370 GET #1,P
380 PRINT "ELEMENTO: ";CVS(F$)
390 RETURN
400 REM
410 REM
420 REM ----ESCRITA DA MATRIZ----
430 REM
440 INPUT "LINHA E COLUNA : ";R,C
450 P=(R-1)*TC+C
460 INPUT "ELEMENTO: ";E : REM NUMERO REAL
470 LSET F$=MKS$(E)
480 PUT #1,P
490 RETURN
500 REM
510 REM
520 REM ----INICIALIZACAO DA MATRIZ----
530 REM
540 LSET F$=STRING$(4,CHR$(0))
550 REM
560 FOR R=1 TO TR
570 FOR C=1 TO TC
580 P=(R-1)*TC+C
590 PUT #1,P
600 NEXT C
610 NEXT R
620 REM
630 RETURN
```

Gestão dos clientes de um hotel

Depois de ter falado tanto de arquivos, lhe propomos um programa, mais completo, no qual se controla a presença de clientes de um hotel. O núcleo fundamental do programa é um arquivo de acesso direto no qual se incluem os dados mais importantes de cada hóspede:

- nome e sobrenome,
- data de chegada e
- preço do apartamento por dia.

Para ter acesso aos registros individuais é utilizado como "chave" o mesmo número dos apartamentos. Na hipótese de que o hotel tenha 100 apartamentos, utilizaremos os números de 1 a 100 para apontar aos registros. Evidentemente este número pode ser modificado mudando a linha 170.

As funções desempenhadas pelo programa são:

- registro de uma nova chegada, com indicação no arquivo do preço do apartamento, da data de chegada e do nome do hóspede;
- impressão da lista de nomes dos hóspedes alojados no hotel;
- em caso de partida, cálculo da importância a pagar, e, finalmente,
- fechamento do programa.

As datas de chegada e partida, proporcionadas na forma de dia e mês, são convertidas em número de dias do ano (1-365) para facilitar o cálculo dos dias de estadia. Para simplificar não se tem em conta o ano e, conseqüentemente, o programa não funcionaria para clientes que estiveram no hotel entre dois anos. Os dias de estadia são calculados como diferença entre a data de partida e a de chegada mais um (linha 4120). Isto é assim para ter em conta que se paga o apartamento ainda que somente se permaneça no hotel desde a manhã até a tarde; se não está de acordo pode suprimí-lo.

Como vimos em casos análogos, quando o programa é utilizado pela primeira vez se deve inicializar o arquivo direto. Assim, na linha 250 comprovamos com LOF(1) se o arquivo não ocupou nenhum bloco; em tal caso saltamos a subrotina da linha 1000, que grava 100 registros com 38 caracteres nulos cada um (CHR$ (0)). Dê-se conta que isto é feito com a variável IN$ que se definiu como buffer na linha 230, enquanto que na 220 usamos PR$, AR$

e NC$. Isto é totalmente corrento sempre que em uma mesma operação somente usemos um dos dois buffers criados. Quando utilizamos normalmente o programa, além de arquivo principal (HOTEL), se deve utilizar outro pequeno arquivo (que escolhemos seqüencial) no qual indicar os apartamentos livres (APARTAMENTOS). Este arquivo contém uma cópia do vetor LIV (100) no qual estão indicadas os apartamentos livres com um Ø, e os ocupados com um 1. Assim, quando se inicializa o arquivo HOTEL se inicializará também o arquivo APARTAMENTOS, todo ele com valores Ø (linha 1170).

Quando se executa o programa, o primeiro que acontece é que o arquivo APARTAMENTOS é copiado em LIV (linhas 270-340). Em seguida aparece o menú, que pede ao funcionário do hotel a operação que deseja efetuar.

Se é a chegada de um novo cliente, o programa procura, no vetor LIV, se existe apartamento livre (linhas 2020-2120). CAM é o número do apartamento que tem de atribuir-se ao cliente (com o número Ø todos os apartamentos estão ocupados). O loop estabelecido nas linhas 2060-2080 procura o primeiro elemento nulo do vetor LIV:

```
2070 IF LIV(K) = Ø THEN CAM = K:K = CA
```

Em caso afirmativo, K se faz igual ao número do apartamento, e logo, para sair corretamente do loop, se estabelece K = CA (CA é o número total dos apartamentos).

Se não existe nenhum apartamento livre, o programa imprimirá um aviso (linha 2100).

Uma vez encontrado um apartamento livre (CAM) se devem dar os dados de entrada: nome do cliente, data de chegada e preço. A data se converte no dia do ano DA (subrotina da linha 5000). A escrita no buffer definido na linha 220 com a instrução FIELD é realizada com as três instruções LSET (linhas 2230-2250). As variáveis numéricas PREC e DA são convertidas em cadeias com MKS$. Na linha 2270 é gravado no arquivo o registro do apartamento CAM e imediatamente depois (seria lamentável que um funcionário o esquecesse!) é indicado automaticamente que o apartamento já está ocupado LIV (CAM) = 1.

A subrotina para a leitura dos hóspedes alojados no hotel é muito simples. Com a instrução IF THEN na linha 3030 são procurados os apartamentos ocupados; a partir delas é lido o registro com GET # e logo são impressos os dados.

Quando um hóspede vai embora, através de seu número de

apartamento PA é lido o registro, a partir do qual se obtém o preço do apartamento e o dia de chegada. Com estes e a data de partida que é solicitada se calcula a importância total a pagar (linha 4180). Imediatamente depois se põe em "zero" o registro (linha 4250-4260) e se indica que está livre o apartamento correspondente no vetor LIV.

O fechamento do programa copia o vetor LIV no arquivo APARTAMENTOS para permitir o correto controle do hotel na seguinte execução do programa, e logo fecha os dois arquivos HOTEL e APARTAMENTOS.

Damos em seguida as variáveis utilizadas no programa:

- CA — número de apartamentos do hotel,
- PR$, PREC — preço do apartamento,
- DD, MM — dia e mês,
- AR$, DA — dia do ano,
- NC$, NOM$ — nome e sobrenomes do hóspede,
- IN$ — cadeia nula de inicializaçao
- LIV — vetor de ocupação dos apartamentos do hotel,
- CAM — primeiro apartamento livre,
- PA — apartamento do hóspede,
- DE — dias de estadia,
- PAG — importância do serviço.

```
100 REM ************************
110 REM *  RECEPCAO DE HOTEL   *
120 REM ************************
130 REM
140 REM
150 REM
160 REM
170 CA=100 : REM NEMERO DE APARTAMENTOS DO HOTEL
180 DIM LIV(CA) : REM VETOR APARTAMENTOS LIVRES
190 REM
200 OPEN "R",#1,"HOTEL",38
210 REM
220 FIELD #1,4 AS PR$,4 AS AR$,30 AS NC$
230 FIELD #1,38 AS IN$
240 REM
250 IF LOF(1)=0 THEN GOSUB 1000
```

```
260 REM
270 REM ----- PREPARACAO DO VETOR LIV -----
280 REM ----- DOS APARTAMENTOS LIVRES -----
290 REM
300 OPEN "I",#2,"APARTAMENTOS"
310 REM
320 FOR J=1 TO CA
330 INPUT #2,LIV(J)
340 NEXT J
350 CLOSE #2
360 REM
370 REM ----- MENU DE RECEPCAO -----
380 REM
390 HOME : REM APAGAR A TELA
400 PRINT
410 PRINT "NOVA CHEGADA ----------> N"
420 PRINT "CLIENTES DO HOTEL ------> C"
430 PRINT "PARTIDA DO CLIENTE -----> P"
440 PRINT "FIM DO TRABALHO --------> F"
450 REM
460 INPUT "ESCOLHA SUA OPCAO";R$
470 REM
480 IF R$="N" THEN GOSUB 2000
490 IF R$="C" THEN GOSUB 3000
500 IF R$="P" THEN GOSUB 4000
510 IF R$="F" THEN GOSUB 8000
520 GOTO 410
530 REM
540 REM
550 REM
1000 REM ----- SUBROTINA DE INICIALIZACAO -----
1010 REM
1020 REM ---- PREPARACAO DO ARQUIVO HOTEL ----
1030 REM
1040 LSET IN$=STRING$(38,CHR$(0))
1050 REM
1060 FOR K=1 TO CA
1070 PUT #1,K
1080 NEXT K
1090 REM
```

```
1100 REM --- PREPARACAO DO ARQUIVO APARTAMENTOS ---
1110 REM
1120 LIVRE=0
1130 REM
1140 OPEN "O",#2,"APARTAMENTOS"
1150 REM
1160 FOR J=1 TO CA
1170 PRINT #2,LIVRE
1180 NEXT J
1190 REM
1200 CLOSE #2
1210 REM
1220 RETURN
1230 REM
1240 REM
1250 REM
2000 REM ---- SUBROTINA DE NOVA CHEGADA ----
2010 REM
2020 REM --- PROCURA DE APARTAMENTO LIVRE ---
2030 REM
2040 CAM=0 : REM NUMERO DO PRIMEIRO APARTAMENTO
2050 REM  LIVRE  JU TODO OCUPADO)
2060 FOR K=1 TO CA
2070 IF LIV(K)=0 THEN CAM=K : K=CA
2080 NEXT K
2090 REM
2100 IF CAM=0 THEN PRINT "TODO OCUPADO!" : RETURN
2110 REM
2120 PRINT "APARTAMENTO ";CAM;" LIVRE"
2130 REM
2140 INPUT "NOME E SOBRENOME DO HOSPEDE (30 CAR. MAX.):";NOM$
2150 IF LEN(NOM$)>30 THEN 2140
2160 INPUT "DATA DE CHEGADA (DD,MM):";DD,MM
2170 INPUT "PRECO DO APARTAMENTO:";PREC
2180 REM
2190 GOSUB 5000 : REM CONVERSAO DA DATA
2200 REM
2210 REM ESCRITA NO BUFFER
2220 REM
2230 LSET PR$=MKS$(PREC)
2240 LSET AR$=MKS$(DA)
2250 LSET NC$=NOM$
2260 REM
2270 PUT #1,CAM : REM CAM E TAMBEM O PONTEIRO DO REGISTRO
2280 LIV(CAM)=1 : REM APARTAMENTO OCUPADO
2290 REM
2300 RETURN
2310 REM
2320 REM
2330 REM
3000 REM ---- SUBROTINA DE CLIENTES DO HOTEL ----
3010 REM
3020 FOR K=1 TO CA
3030 IF LIV(K)=0 THEN 3070
```

```
4120 DE=DA-CVS(AR$)+1
4130 REM
4140 PRINT "SR. ";NC$;" PERMANECEU ";DE;" DIAS"
4150 REM
4160 REM PAGAMENTO DO SERVICO
4170 REM
4180 PAG=DE*CVS(PR$)
4190 REM
4200 PRINT "O PRECO DO APARTAMENTO E : ";PAG
4210 PRINT "OBRIGADO E ATE BREVE"
4220 REM
4230 LIV(PA)=0 : REM DEIXAR O APARTAMENTO LIVRE
4240 REM
4250 LSET IN$=STRING$(38,CHR$(0))
4260 PUT #1,PA
4270 REM
4280 RETURN
4290 REM
4300 REM
4310 REM
5000 REM SUBROTINA DE CALCULO DO NUMERO DE DIAS DE ESTADIA
5010 RESTORE
5020 DA=DD
5030 FOR J=1 TO MM
5040 READ ND
5050 DA=DA+ND
5060 NEXT J
5070 REM
5080 DATA 0,31,28,31,30,31,30
5090 DATA 31,31,30,31,30,31
5100 REM
5110 RETURN
5120 REM
5130 REM
5140 REM
3040 GET #1,K : REM LEITURA DO REGISTRO
3050 REM
3060 PRINT "SR. ";NC$,"APARTAMENTO ";K
3070 NEXT K
3080 REM
3090 RETURN
3100 REM
4000 REM ---- SUBROTINA DE PARTIDA ----
4010 REM
4020 INPUT "QUE APARTAMENTO FICA LIVRE";PA
4030 INPUT "DATA DE PARTIDA (DD,MM) :";DD,MM
4040 GOSUB 5000
4050 REM
4060 REM LEITURA DO ARQUIVO
4070 REM
4080 GET #1,PA
4090 REM
4100 REM DIAS DE ESTADIA
4110 REM
```

```
8000 REM ---- ENCERRAR O PROGRAMA ----
8010 REM
8020 REM ---- ARMAZENAMENTO DO VETOR ----
8030 REM --- DOS APARTAMENTOS LIVRES ---
8040 OPEN "O",#2,"APARTAMENTOS"
8050 REM
8060 FOR J=1 TO CA
8070 PRINT #2,LIV(J)
8080 NEXT J
8090 REM
8100 CLOSE #2 : REM FECHAMENTO DO ARQUIVO DOS APARTAMENTOS
8110 REM
8120 CLOSE #1 : REM FECHAMENTO DO ARQUIVO DO HOTEL
8130 REM
8140 PRINT "FINAL DO TRABALHO"
8150 REM
8160 END
```

CAPÍTULO VI

INSTRUÇÕES GRÁFICAS E ANIMAÇÃO

Por que os GRÁFICOS?

Sabemos que os computadores dialogam conosco facilmente mediante os carecteres alfabéticos e numéricos normais. Um programa é escrito como se fosse uma carta a um amigo e o computador responde com mensagens que são lidas como se estivessem escritas em um jornal. Junto a este "canal" alfanumérico podemos comunicar-nos também com outros meios, como são a imagem e o som. As imagens são tudo o que o olho vê e a mão pode traçar, e os sons são tudo o que ouvimos e que nossa garganta ou outro instrumento pode gerar. Quando utilizamos as imagens para comunicar-nos com o computador fazemos uso de gráficos No segundo caso não existe um termo específico, salvo falar de música com computador ou síntese vocal.

Com os gráficos do computador, o conceito de processamento de dados se amplia e assume aspectos muito diferentes e fascinantes. Se proporcionamos a um computador imagens, por exemplo, o desenho de um automóvel, nos pode responder com outros desenhos, como a vista frontal ou lateral, do mesmo automóvel e, ao mesmo tempo, nos pode calcular a área de uma de suas superfícies.

Para obter isto devemos dispor de programas que sejam capazes de realizar o processamento de dados gráficos; estes programas constituem o que se denomina software gráfico.

Os gráficos permitem empregar o computador de um modo completamente novo. Junto aos periféricos tradicionais encontra-

Figura 1 - O UNITRON, capaz de comunicar-se mediante gráficos e sons.

mos os adaptados à entrada e à saída de imagens:plotter, tabelas gráficas, lápis ópticos, mouse, joystick, track ball, etc. Uma versão diferente, mais reduzida, mas muito importante, dos gráficos é a que se refere aos computadores pessoais.

Para efetuar gráficos de tipo profissional é necessário dispor de computadores de dimensões médias ou grandes, de periféricos gráficos muito caros e de um software gráfico um tanto complexo. Um requisito muito importante é o da velocidade de cálculo da CPU, que deve ser muito elevada. Para compreender isto considere que, com freqüência, para situar um só ponto de uma imagem se devem efetuar primeiro, longos e complexos cálculos. Em uma imagem constituída por mil pontos, se o cálculo de um deles não é realizado em um tempo muito breve, não se conseguirá nenhum resultado prático.

É precisamente esta a limitação dos computadores pessoais em relação com os gráficos profissionais: são demasiadamente lentos e não têm suficiente memória central, mas, sobretudo, o preço dos bons periféricos gráficos é desproporcional com respeito ao de qualquer computador pessoal.

Por tudo isto, com os computadores pessoais nasceram uns novos tipos gráficos. Alguns gráficos econômicos e simples, mas de grande utilidade, que os coloca á altura de seus irmãos maiores. Os gráficos dos computadores pessoais são uma forma de apresentar os resultados de um programa e constituem um modo para fazer mais agradável a visualização dos dados, mas são também um meio de realizar programas de jogos (os denominados video-jogos") ou para fazer mais próximos e "amigáveis" muitos programas que por si só seriam aborrecidos, tais como os programas didáticos.

Isto é uma necessidade e uma exigência dos computadores pessoais e justifica a existência, tanto no BASIC como em outras linguagens para computadores pessoais, de instruções gráficas que não existem nas versões para grandes computadores das mesmas linguagens. Trata-se de instruções que permitem colorir a tela, traçar pontos, retas, curvas ou programar figuras (os denominados "sprites") que se movem e animam com facilidade na tela.

Como confirmação de que os gráficos "pessoais" têm seu papel autônomo basta observar que os computadores mais ricos nestas opções são precisamente os computadores pessoais da gama mais baixa e de maior êxito comercial. Os gráficos em computadores pessoais "maiores" não são proporcionados para "jogar", mas para enriquecer a visualização de resultados nos programas profissionais.

O "sketchpad" de Ivan Sutherland

Qualquer alusão histórica aos gráficos nos leva a muitos anos atrás. A necessidade de comunicar-se com o computador, não somente no modo alfanumérico, mas também com a possibilidade de efetuar algorítmos em conjuntos geométricos, se sentiu imediatamente pelos primeiros usuários dos computadores.

O primeiro sistema gráfico foi desenvolvido no ano 1963 no prestigioso MIT de Boston por Ivan Sutherland. O sistema, que se chamava SKETCHPAD (tabuleiro de desenho), era interativo, permitia desenhar na tela figuras elementares e estava provido do primeiro lápis óptico ("light pen"). Mediante comandos situados próximos da tela (Fig. 2) era possível obter figuras elementares, que logo se deslocavam e se agrupavam na tela com o lápis óptico.

Os sistema Sktchpad foi o primeiro exemplo de "trabalho com gráficos" e requeria fundamentalmente uma tela gráfica, um lápis óptico e o software necessário.Junto a este tipo de hardware se desenvolveram, a partir dos anos cinquenta, outros periféricos. O

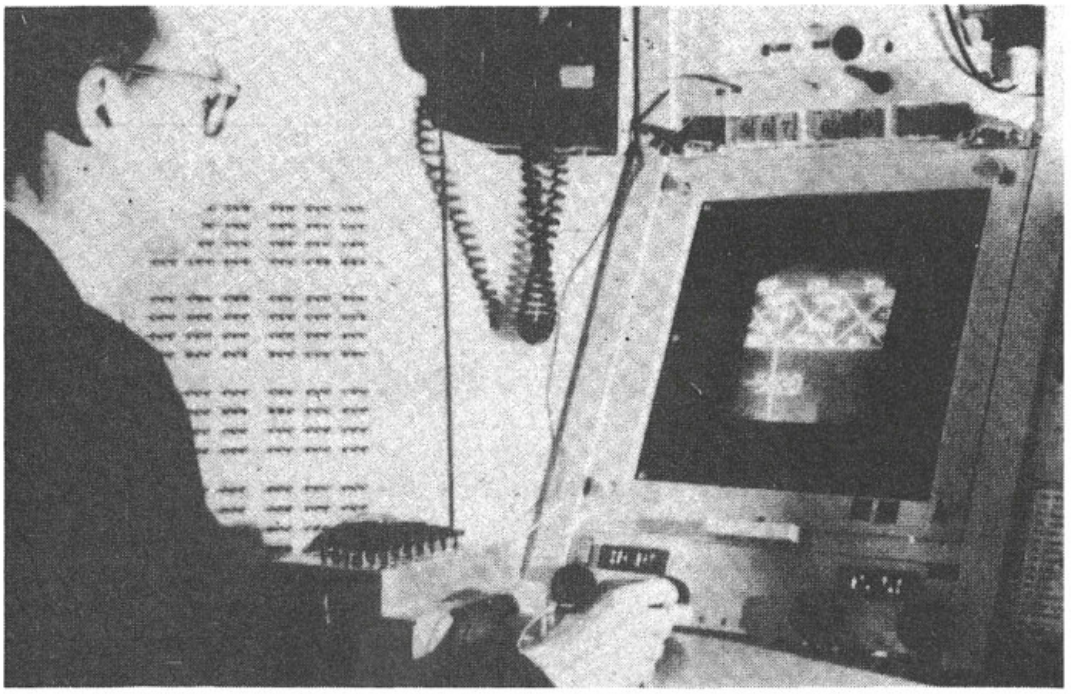


Figura 2 - O computador Sketchpad de Ivan Sutherland.

mais importante foi o plotter digital(digitalizador), graças ao qual se podem fazer alguns maravilhosos e perfeitos desenhos constituídos por pontos.

Programação de gráficos nos computadores pessoais

Todas as versões de BASIC permitem realizar gráficos, mas as instruções disponíveis para dita tarefa são muito diferentes de um computador a outro. Se quiséssemos converter um programa de Aplle que controla arquivos em um programa para o Commodore encontraríamos, quase com toda segurança, as correspondências entre as instruções dos dois computadores. Dito de outro modo, o algorítmo concebido para controlar arquivos com o computador da Apple continua sendo o mesmo no caso do Commodore e somente tem que voltar a escrevê-lo fazendo uso de instruções com diferentes sintaxe. Com os gráficos isto é praticamente impossível. É precisamente o algorítmo gráfico o que muda por completo de um computador Apple para um Commodore ou de um Spectrum para um IBM PC ou a qualquer outro computador.

Com um pouco de boa vontade, como fizemos nos outros ca-

sos, é possível determinar alguns métodos comuns para fazer gráficos. Fundamentalmente existem três:

- emprego de caracteres semigráficos,
- criação de sprites (objetos móveis),
- controle direto dos pontos de tela ("pixels").

Caracteres semigráficos

Trata-se da programação gráfica mais simples e mais comum, mas também a que proporciona os resultados menos atrativos. O conjunto dos caracteres utilizados pelo computador se amplia com novos caracteres que representam figuras elementares, tais como traços verticais, horizontais, diagonais, quadrados recheados, etc. (Fig. 3). Estes caracteres estão codificados em ASC II, tal como os alfanuméricos e são tratados de um modo idêntico. Pa-

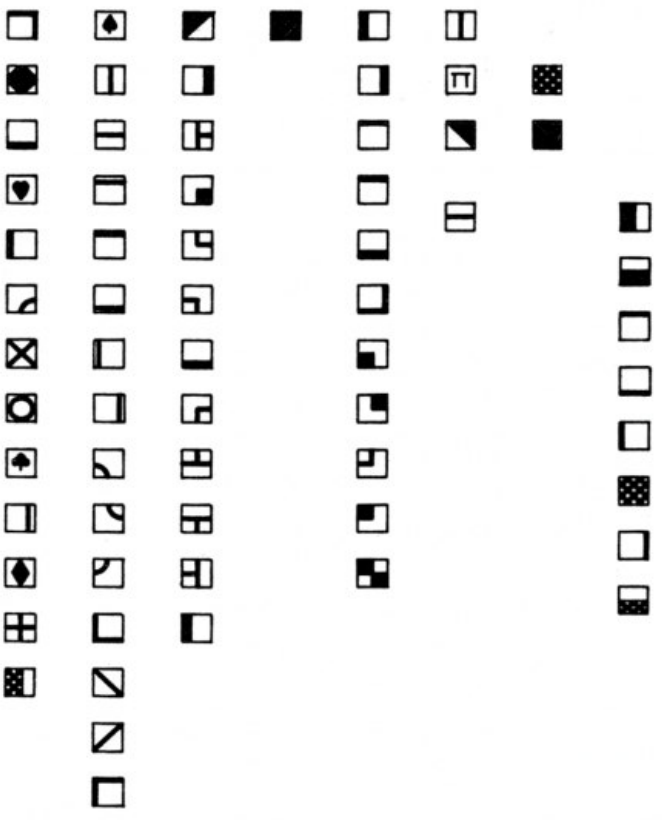


Figura 3 - Conjunto de caracteres semigráficos.

ra traçar um desenho se faz uso de instruções PRINT que imprime cadeias com estes caracteres.

Muitas vezes se traçam desenhos elementares sem utilizar nem sequer os caracteres semigráficos, ajudados somente por caracteres simples, tais como o astericos, o sinal menos ou o sinal de exclamação. Um desenho desta classe é realizável por qualquer computador pessoal, e este é o único caso de compatibilidade completa entre computadores.

Por exemplo, com as instruções:

```
PRINT "**********"

PRINT "*          *"
```

se pode desenhar um quadrado

```
**********
*        *
*        *
*        *
*        *
*        *
**********
```

Se quiser desenhar um círculo ou uma curva mais complexa é fácil imaginar que os resultados deixariam bastante a desejar.

O primeiro tipo de gráfico tem uma limitação notável, e é que permite traçar somente desenhos simples e modulares, em função do conjunto de caracteres semigráficos que possua o computador. Em compensação, é muito fácil de programar porque, como dissemos anteriormente, emprega as instruções PRINT normais.

Sprites

Os sprites são um passo adiante na programação gráfica e permitem obter resultados muito satisfatórios. Um sprite é uma pequena imagem móvel que se constrói pelo programador. Cada sprite definido em um programa tem seu nome (como uma variável) e pode visualizar-se à vontade, deslocar-se e ampliar-se na tela com as instruções adequadas. Com a alternância de sprites diferentes se podem conseguir efeitos de movimento, e no caso limi-

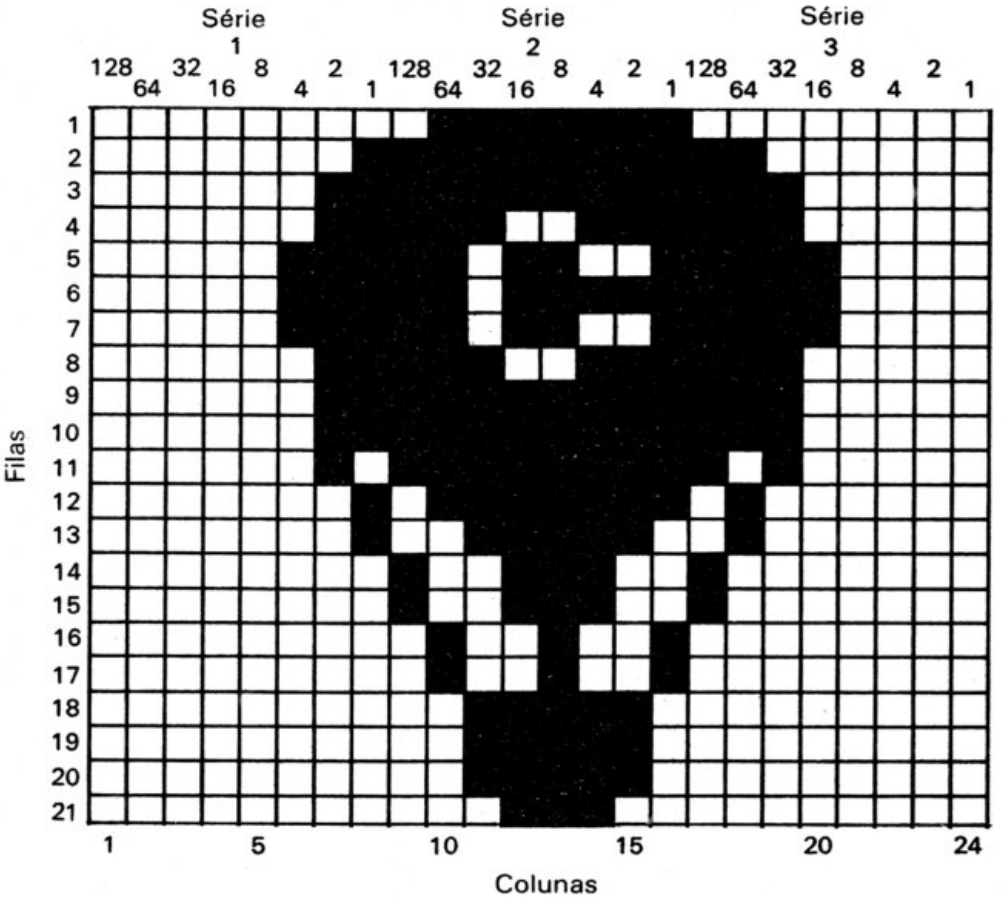


Figura 4 - Matriz de criação de um sprite.

te, criar algo muito similar às imagens dos vídeo-jogos.

Os sprites são definidos ao início do programa mediante uma matriz (Fig. 4). Cada quadrado elementar da matriz define o estado (ativado ou desligado) de um ponto da tela.

O número máximo de sprites, seu tamanho e outras características dependem do computador. Em geral, cada sprite se move em um plano próprio (como se fossem lâminas plásticas superpostas), cada um com sua prioridade. Assim, por exemplo, os equipamentos MSX admitem 32 planos mais o fundo (Fig. 5).

Gráficos por pontos

O terceiro tipo de gráficos é o mais complexo, mas também o mais aperfeiçoado. Neste caso, o programador pode controlar

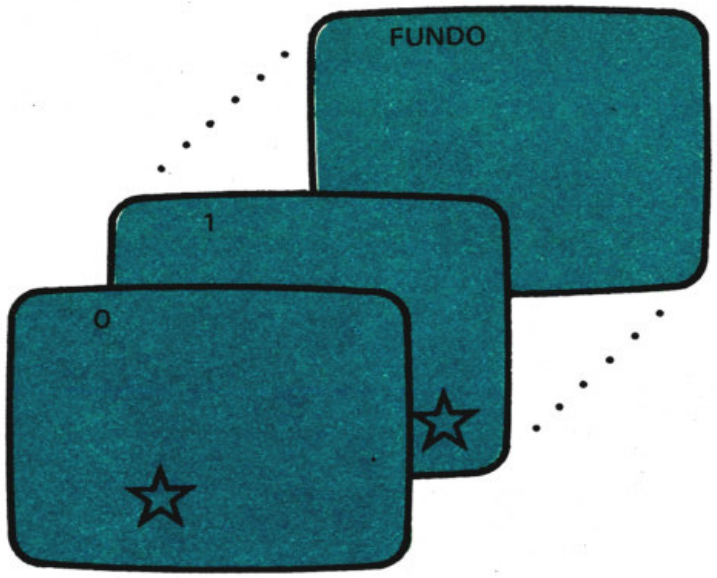


Figura 5 - Cada sprite é como se fosse movido em um plano distinto da tela. Segundo o computador, se admitem mais ou menos " planos ".

cada elemento de imagem individual da tela (pixel), escolhendo também a cor que vai lhe dar.

Os elementos de imagem, também denominados pixels (pela contração da denominação inglesa "picture cell") são os pontos unitários de uma imagem digital. Um pixel equivale a um bit: ponto luminoso ligado ou desligado. Quanto mais pixels existirem na tela (seu número não depende do software, mas do hardware), tanto mais definida e agradável será a imagem obtida. Atualmente, nos sistemas gráficos profissionais se podem obter imagens de qualidade superior à de uma boa fotografia.

Muitos destes sistemas têm 1024 x 1024 pixels, que são mais de um milhão na tela completa e cada um pode iluminar-se com qualquer cor dentre 16 milhões de cores. Não pense que este elevadíssimo número de cores é desproporcional, pois compreende os matizes de intensidade das cores de base.

Os computadores pessoais trabalham com um número de pixels inferior (por exemplo, o Commodore 64 tem 320 x 200 = 64.000 pixels). Se recordamos que as telas mais usuais de apresentação visual não têm memória e que sua imagem deve ser continuamente "refrescada", isto significa que os pixels de uma imagem devem armazenar-se em uma memória RAM que funciona co-

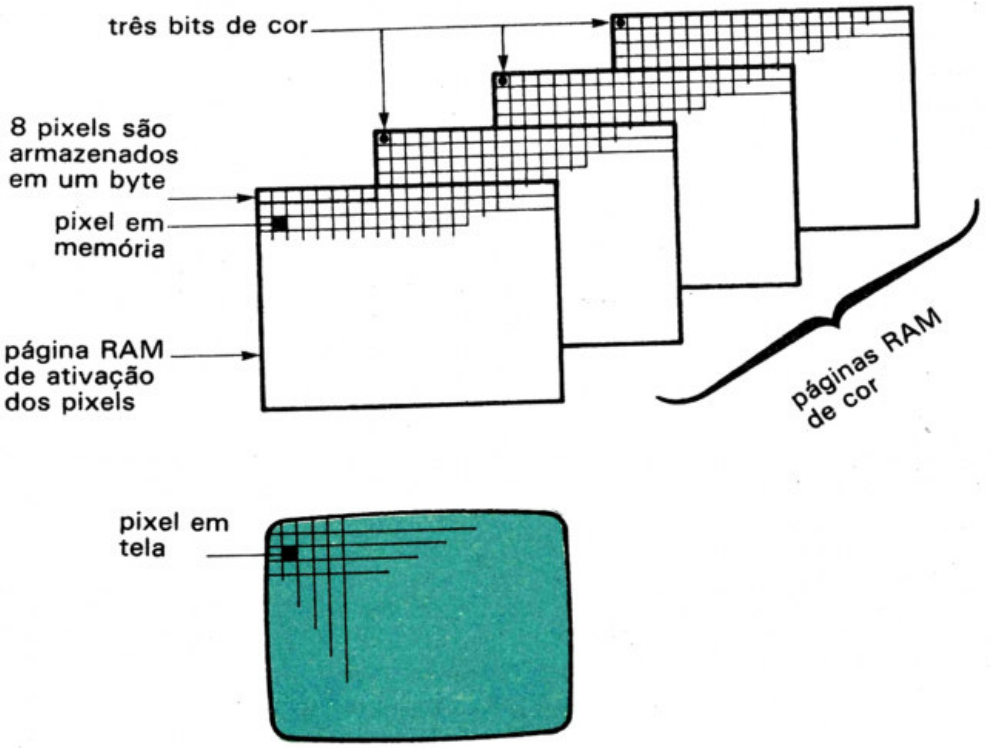


Figura 6 - Páginas gráficas de um computador.

mo um buffer (Fig. 6). Esta zona de memória RAM ou "página gráfica", contém mapa dos pontos da tela, e por este motivo, se denomina também memória de "mapeamento" de bits (em inglês, bit-mapping memory). Se cada byte de memória contém 8 bits, se pode comprovar que o Commodore 64 utiliza para sua página gráfica 8 kbytes (64000/8 = 8 k.).

Quanto maior for o número de pixels, tanto maior deve ser a quantidade de memória RAM necessária para armazenar a imagem. Se compreende, pois, porque os computadores pessoais não podem ter muitos pixels. Se são gráficos em cor, junto a cada bit que indica um pixel ativo devemos dar também a informação de cor. Para escolher entre as 8 cores habituais, na maior parte dos computadores pessoais são necessários 3 bits (3 bits permitem selecionar um entre oito casos possíveis). No caso de uma página gráfica de 8 kbytes se devem acrescentar, do ponto de vista teórico, outros 24 kbytes para a cor.

Os gráficos por pontos, que também se denominam de alta resolução, fazem uso de instruções que controlam qualquer pon-

to individual da tela, estando definido cada ponto por suas coordenadas X, Y. Com freqüência, para fins práticos, existem instruções que fazem traçar uma reta entre dois pontos (um vetor) ou então um círculo ou uma elipse. Outras instruções são capazes de preencher com cor uma curva fechada.

Um exemplo gráfico

Não é fácil dar exemplos de gráficos sem cair na banalidade ou sem recorrer a instruções demasiadamente específicas (e "obscuras"). No entanto, consideramos que para fazer algo mais concreto quanto dissemos, é interessante mostrar um programa, escrito para o computador Apple II no qual se desenha um quadrado que se move rebatendo-se de um extremo a outro da tela. Com respeito às três classificações antes comentadas, este exemplo entra dentro da terceira categoria, a de gráficos de alta resolução.

O lado L do quadrado, que queremos fazer deslocar, é solicitado como dado à entrada (linha 180). Se trabalhamos em alta resolução, este valor corresponde ao número de pixels utilizados pelo computador para desenhar cada lado do quadrado. A limitação a 100 se escolheu para ter um quadrado aceitável, considerando que existem 280 x 160 nas coordenadas X e Y da tela do Apple II. A instrução HGR na linha 220 chama a "página 1" dos gráficos de alta resolução do computador Apple II (na realidade, este computador têm duas páginas gráficas de alta resolução). A instrução HPLOT é a que traça o quadrado como sucessão de quatro segmentos L consecutivos:

```
HPLOT VERTICE1 TO VERTICE2
      TO VERTICE3 TO VERTICE4
      TO VERTICE1
```

A repetição final do vértice 1 serve para fechar o quadrado. O programa escolhe ao acaso a coordenada Y do primeiro vértice do quadrado, o situado na parte superior esquerda, enquanto que o X inicial é nulo. Os incrementos DTX e DTY, para o deslocamento do quadrado são escolhidos ao acaso nas linhas 260 e 270. As instruções 310 e 320 servem para controlar que o quadrado não salte da tela, mas que rebata; de fato, ao tocar as bordas, os incrementos DTX e DTY mudam de sinal e assim se diminuem ao invés de somar-se, ou vice-versa.

Nas linhas 340 e 410 se escolhe a cor do traço. Com HCOLOR = 3 o quadrado é traçado em branco, enquanto que com o valor

0, a cor é preta como o fundo da tela. Na prática, o quadrado é desenhado primeiro em branco e logo em preto e isto equivale a apagá-lo, mas depois de alguns segundos, que são suficientes para que nossos olhos possam vê-lo. Repetindo este duplo traço se obtém um efeito visual de movimento. O pequeno loop de espera na linha 390 serve para fazer mais suave o movimento.

```
100 REM ******************
110 REM *  UM QUADRADO   *
120 REM *  QUE REBATE    *
130 REM ******************
140 REM
150 REM
160 TEXT : HOME
170 REM
180 INPUT "LADO DO QUADRADO (1-100): ";L
190 IF L<1 OR L>100 THEN 180
200 REM
210 REM
220 HGR : REM ATIVACAO DA ALTA RESOLUCAO
230 REM
240 X=0 : Y=RND(1)*(150-L)
250 REM
260 DTX=RND(1)*5
270 DTY=RND(1)*5
280 REM
290 REM --- INICIALIZACAO DO MOVIMENTO ---
300 REM
310 IF X+DTX+L>279 OR X+DTX<0 THEN DTX=-DTX
320 IF Y+DTY+L>159 OR Y+DTY<0 THEN DTY=-DTY
330 REM
340 HCOLOR=3 : REM TRACO DE COR BRANCA
350 REM
360 X=X+DTX : Y=Y+DTY
370 HPLOT X,Y TO X+L,Y TO X+L,Y+L TO X,Y+L TO X,Y
380 REM
390 FOR K=1 TO 50 : NEXT K : REM LOOP DE RETARDO
400 REM
410 HCOLOR=0 : REM TRACO EM COR PRETA
420 REM
430 HPLOT X,Y TO X+L,Y TO X+L,Y+L TO X,Y+L TO X,Y
440 REM
450 GOTO 310
```

Como já observou, o desenho de gráficos, ainda que não seja difícil, torna-se complicado, especialmente pela diversidade de instruções de cada computador. Mas não se preocupe demasiadamente: haverá mais livros de nossa biblioteca que falarão sobre estes temas.

APÊNDICE A

RESUMO DAS INSTRUÇÕES ESTUDADAS

Para facilitar ao leitor a rápida solução de qualquer dúvida sobre a sintaxe das instruções vistas neste volume da B.B.I. fizemos um pequeno resumo-dicionário, que esperamos lhe seja útil.
Além de definir a função da instrução e sua sintaxe, inclui alguns exemplos de amostra. Em nenhum caso incluímos o número de linha, pois damos por acertado que o leitor sabe que se a instrução faz parte de um programa deve levá-lo e, em caso contrário (execução direta), não.

Nas definições que seguem usaremos os símbolos.

[ ] elemento opcional
{} possibilidades alternativas
<> devem ser determinados pelo usuário.

CLOSE

Fecha o canal lógico aberto mediante um OPEN.

CLOSE # <n.º canal>

```
CLOSE # 1
```

CVI, CVS, CVD

Convertem uma cadeia no número que representa (inteiro, de

simples ou de dupla precisão, respectivamente).

CVI < variável de cadeia >
CVS < variável de cadeia >
CVD < variável de cadeia >

```
E = CVI (INTEIRO$)
S = CVS (SIMPLESPRE$)
D = CVD (DUPLAPRE$)
```

DEF FN

Cria uma nova função, que poderá ser usada onde se inclua esta definição.

DEF FN (<parâmetro mudo 1 > [,...]) =

```
DEF FN (H) = 2*H+5
DEF FN (R) = 2*π*R
```

FIELD

Determina uma zona de memória para buffer de um arquivo direto, especificando comprimentos de cada campo e as variáveis associadas.

FIELD#<n.º canal >, <comprimento 1> AS > variável de cadeia 1 > [, <comp 2 > AS < var 2 >,...]

```
FIELD # 1, 27 AS NOMES, 10 AS DNIS
```

GET

Carrega nas variáveis dadas em FIELD os valores de registro selecionado de um arquivo direto.

GET # <n.º canal>, <n.º registro>

```
GET # 1, I
GET # 3, PP + 1
```

GOSUB

Cede o controle a uma subrotina que começa no número de linha

especificado.

GOSUB <n.º de linha >

GOSUB 5000

INPUT

Lê os campos do seguinte registro de um arquivo seqüencial, carregando seus valores nas variáveis explícitas.

INPUT #<n.º canal >, <variável de cadeia 1 >[, <var 2> ...]

INPUT # 5, P$, VALOR$

LINE INPUT

Lê um registro completo de um arquivo sequencial até o caracter RETURN

LINE INPUT #<n.º cancal >, <variável de cadeia>

LINE INPUT # 1, L$

LOF

Indica os blocos usados de um arquivo direto.

LOF (<n.º canal)>

A = LOF (1)

LSET

Carrega as variáveis de campo do buffer de um arquivo direito da esquerda à direita.

LSET <variável campo > = <variável >

LSET NOME$ = PEDRO$

MKI$, MKS$, MKD$

Carregam em uma cadeia o valor de um número inteiro, de simples ou de dupla precisão, respectivamente.

MKI$ (<variável>)
MKS$ (<variável>)
MKD$ (<variável>)

PEDRO$ = MKS$ (DNI)
A$ = MKI$ (INTEIRO)

ON GOSUB

Salta à subrotina cuja linha de começo está na posição da listagem cujo número coincide com o valor que toma a variável (de 1 em diante).

ON <variável> GOSUB < n.º linha SUB1> ,[< SUB2> ,...]

ON POS GOSUB 1000, 2000, 3000

ON GOTO

Salta a linha cuja posição na lista coincide com o valor que toma a variável (de 1 em diante).

ON <variável> GOTO <n.º linha 1>, [, < n.º 2 > ...]

ON LIN GOTO 100, 200, 300, 400

OPEN

Estabelece a abertura de um canal lógico. Existe muitas sintaxes possíveis. As mais comuns são:

- Arquivos diretos

OPEN "R", # <n.º canal>, " <nome> ", <comprimento registro
OPEN " <nome> ", A$ [#] <n.º canal>, LEN = < comprimento registro>

OPEN "R", # 1, "DISCO.1", 150

- Arquivos seqüenciais

OPEN " < modo > ",# <n.º canal>, " < nome > "
OPEN " < nome > FOR " <modo> " A$ < n.º canal >

```
OPEN "I", # 1, "PEDRO"
OPEN "O", # 2, SAÍDA$
```

PRINT #

Escreve no seguinte registro de um arquivo seqüencial o conteúdo da lista de variáveis.

PRINT#<n.º canal>, <variável de cadeia 1 ,>[,< var2 >...]

```
PRINT # 1, NOM$; ","; APELL$
PRINT # 3, AS; B$
```

PUT #

Escreve no registro dado de um arquivo direto o valor das variáveis definidas em FIELD

PUT# < n.º canal >, <n.º registro>

```
PUT # 5,J
```

RND

Obtém um número pseudo-aleatório. Se a base é nula repete o último número

RND (n.º inteiro)

```
VALOR = RND (5)
REPETE = RND(0)
```

RSET

Carrega as variáveis de campo do buffer de um arquivo direto da direita à esquerda.

RSET <varirável campo > = <variável>

```
RSET VAR$ = JOAO$
```

RETURN

Devolve o controle ao programa que chamou à subrotina exatamente na linha seguinte àquela desde a que realizou a chamada.

RETURN

APÊNDICE B

TABELAS DE CONVERSÃO

Se você conseguiu a listagem de um programa que lhe interessa, mas que está escrita em uma versão de BASIC que não é a que tem seu próprio computador, ou então simplesmente deseja ver de que outras formas se podem expressar determinadas instruções, as tabelas que incluímos lhe serão de grande utilidade.

Estão incluídas nelas as equivalências de todas as instruções vistas nos dois volumes da BASIC, ordenadas alfabeticamente, para os seguintes equipamentos:

- APPLE
- COMMODORE 64
- MSX
- IBM-PC
- SPECTRUM
- TRS-80
- MACINTOSH

APPLE	ABS(<NUMERO>)	ACS<X>	ASC(<CADEIA DE CARACTERES>)	ASN<X>	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
COMMODORE 64	ABS(<NUMERO>)	–	ASC(<CADEIA DE CARACTERES>)	–	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
MSX	ABS(<NUMERO>)	–	ASC(<CADEIA DE CARACTERES>)	–	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
IBM - PC	ABS(<NUMERO>)	–	ASC(<CADEIA DE CARACTERES>)	–	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
SPECTRUM	ABS(<NUMERO>)	ACS(<X>)	CODE(<CADEIA DE CARACTERES>)	ASN(<X>)	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
TRS-80	ABS(<NUMERO>)	–	ASC(<CADEIA DE CARACTERES>)	–	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)
MACINTOSH	ABS(<NUMERO>)	–	ASC(<CADEIA DE CARACTERES>)	–	ATN(<X>)	CHRS(<NUMERO DE CODIGO ASCII>)

	CLOAD ["<NOME>"]	CLOSE#<NUMERO DE CANAL>	COS(<X>)
APPLE	–	CLOSE "<NOME DE ARQUIVO>	COS(<X>)
COMMODORE 64	–	CLOSE#<NUMERO DE CANAL>[#<NUMERO DE CANAL>,...]	COS(<X>)
MSX	CLOAD ["<NOME>"]	CLOSE#<NUMERO DE CANAL>[#<NUMERO DE CANAL>,...]	COS(<X>)
IBM - PC	–	CLOSE#<NUMERO DE CANAL>[#<NUMERO DE CANAL>,...]	COS(<X>)
SPECTRUM	–	–	COS(<X>)
TRS-80	CLOAD ["<NOME>"]	CLOSE <NUMERO DE CANAL>	COS(<X>)
MACINTOSH	CLOAD ["<NOME>"]	CLOSE#<NUMERO DE CANAL>[#<NUMERO DE CANAL>,...]	COS(<X>)

	CSAVE ["<NOME>"]	CVD(<VARIAVEL DE CADEIA>)	CVI(<VARIAVEL DE CADEIA>)	CVS(<VARIAVEL DE CADEIA>)
APPLE	–	–	–	–
COMMODORE 64	–	–	–	–
MSX	CSAVE ["<NOME>"]	CVD(<VARIAVEL DE CADEIA>)	CVI(<VARIAVEL DE CADEIA>)	CVS(<VARIAVEL DE CADEIA>)
IBM - PC	–	CVD(<VARIAVEL DE CADEIA>)	CVI(<VARIAVEL DE CADEIA>)	CVS(<VARIAVEL DE CADEIA>)
SPECTRUM	–	–	–	–
TRS-80	CSAVE ["<NOME>"]	CVD(<VARIAVEL DE CADEIA>)	CVI(<VARIAVEL DE CADEIA>)	CVS(<VARIAVEL DE CADEIA>)
MACINTOSH	–	CVD(<VARIAVEL DE CADEIA>)	CVI(<VARIAVEL DE CADEIA>)	CVS(<VARIAVEL DE CADEIA>)

	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOME DA FUNÇÃO>(<PARAMETRO>)=<EXPRESSÃO DE CADEIA>
APPLE	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	–
COMMODORE 64	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	–
MSX	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOME DA FUNÇÃO> <PARAMETRO>) = <EXPRESSÃO DE CADEIA>
IBM - PC	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOME DA FUNÇÃO> <PARAMETRO>) = <EXPRESSÃO DE CADEIA>
SPECTRUM	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOME DA FUNÇÃO> <PARAMETRO>) = <EXPRESSÃO DE CADEIA>
TRS-80	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	–
MACINTOSH	DATA<CONSTANTE1>[,<CONSTANTE2>,...]	–

	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
APPLE	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
COMMODORE 64	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
MSX	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
IBM - PC	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
SPECTRUM	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
TRS-80	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END
MACINTOSH	DIM(<NUMERO DE FILAS, NUMERO DE COLUNAS>)END

	FIELD# <NUMERO DE CANAL> <COMPR. 1> AS VARIAVEL DE CADEIA 1>[,<COMPR. 2>AS< VARIAVEL DE CADEIA 2>,..]
APPLE	–
COMMODORE 64	–
MSX	FIELD# <NUMERO DE CANAL> <COMPR. 1> AS <VARIAVEL DE CADEIA 1>[,<COMPR. 2>AS< VARIAVEL DE CADEIA 2>...]
IBM - PC	FIELD# <NUMERO DE CANAL> <COMPR. 1> AS <VARIAVEL DE CADEIA 1>[,<COMPR. 2>AS<VARIAVEL DE CADEIA 2>,..]
SPECTRUM	–
TRS-80	FIELD <NUMERO DE CANAL> <COMPR. 1> AS<VARIAVEL DE CADEIA 1>[<COMPR. 2>AS<VARIAVEL DE CADEIA 2>,..]
MACINTOSH	FIELD# NUMERO DE CANAL> <COMPR. 1> AS <VARIAVEL DE CADEIA 1>[,<COMPR. 2>AS< VARIAVEL DE CADEIA 2>,..]

	FOR<VARIAVEL>=<INICIAL>TO<FINAL STEP<SALTO>NEXT<VARIAVEL>	GET#
APPLE	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	GET<VARIAVEL> (*)
COMMODORE 64	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	GET#<N. DE CANAL>,<VARIAVEL> (** **)
MSX	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	–
IBM - PC	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	GET#<N. DE CANAL>,<N. DE REGISTRO>
SPECTRUM	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	–
TRS-80	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	GET<N. DE CANAL>,<N. DE REGISTRO>
MACINTOSH	FOR<VARIAVEL>=<INICIAL>TO<FINAL>STEP<SALTO>NEXT<VARIAVEL>	GET#<N. DE CANAL>,<N. DE REGISTRO>

(*) No Apple II os comandos de manejo de caracteres, precedidos de CTRL-D.

(**) No Commodore 64 o GET# toma um só caracter cada vez e se atribui à variável indicada.

	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>
APPLE	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN	INPUT<VARIAVEL>
COMMODORE 64	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>
MSX	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>
IBM - PC	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>
SPECTRUM	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN	INPUT<VARIAVEL>
TRS-80	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>
MACINTOSH	GOSUB<NUMERO DE LINHA>	GOTO<NUMERO DE LINHA>	GOTO<VARIAVEL>	IF/THEN[/ELSE]	INPUT<VARIAVEL>

	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT#	INT(〈NUMERO〉)
APPLE	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT"〈MENSAGEM〉"(,;)〈VARIAVEL〉[,〈VARIAVEL2〉,...]	INT(〈NUMERO〉)
COMMODORE 64	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT"〈MENSAGEM〉"(,;)〈VARIAVEL〉[,〈VARIAVEL2〉,...]	INT(〈NUMERO〉)
MSX	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT #〈N. DE CANAL〉,〈VARIAVEL1〉[,〈VARIAVEL2〉,...]	INT(〈NUMERO〉)
IBM - PC	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT #〈N. DE CANAL〉,〈VARIAVEL1〉[,〈VARIAVEL2〉,...]	INT(〈NUMERO〉)
SPECTRUM	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	—	INT(〈NUMERO〉)
TRS-80	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	—	INT(〈NUMERO〉)
MACINTOSH	INPUT〈MENSAGEM〉(;,)〈VARIAVEL〉	INPUT #〈N. DE CANAL〉,〈VARIAVEL1〉[,〈VARIAVEL2〉,...]	INT(〈NUMERO〉)

	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
APPLE	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
COMMODORE 64	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
MSX	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
IBM - PC	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
SPECTRUM	—	LEN(〈CADEIA DE CARACTERES〉)
TRS-80	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)
MACINTOSH	LEFTS(〈CADEIA〉,〈NUMERO DE CARACTERES A EXTRAIR〉)	LEN(〈CADEIA DE CARACTERES〉)

	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	LINE INPUT #
APPLE	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	—
COMMODORE 64	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	—
MSX	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	LINE INPUT #⟨N. DE CANAL⟩,⟨VARIAVEL1⟩[,⟨VARIAVEL2⟩,...]
IBM-PC	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	LINE INPUT #⟨N. DE CANAL⟩,⟨VARIAVEL1⟩[⟨VARIAVEL2⟩...]
SPECTRUM	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	—
TRS-80	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	LINE INPUT #⟨MENSAGEM⟩;⟨VARIAVEL⟩
MACINTOSH	LET⟨VARIAVEL⟩=⟨EXPRESSÃO⟩ ⟨VARIAVEL⟩=⟨EXPRESSÃO⟩	LINE INPUT #⟨N. DE CANAL⟩,⟨VARIAVEL1⟩[,⟨VARIAVEL2⟩,...]

	LOAD	LOF	LSET⟨VARIAVEL DE CAMPO⟩=⟨VARIAVEL⟩
APPLE	LOAD "⟨NOME⟩"	—	—
COMMODORE 64	LOAD "⟨NOME⟩",⟨PERIFERICO⟩	—	—
MSX	LOAD "⟨PERIFERICO⟩:⟨NOME⟩"	—	LSET⟨VARIAVEL DE CAMPO⟩=⟨VARIAVEL⟩
IBM-PC	LOAD "⟨PERIFERICO⟩:⟨NOME⟩"	LOF(⟨NUMERO DE CANAL⟩)	LSET⟨VARIAVEL DE CAMPO⟩=⟨VARIAVEL⟩
TRS-80	LOAD "⟨PERIFERICO⟩:⟨NOME⟩"	—	LSET⟨VARIAVEL DE CAMPO⟩=⟨VARIAVEL⟩
SPECTRUM	LOAD "⟨NOME⟩"	—	—
MACINTOSH	LOAD "⟨NOME⟩"	LOF(⟨NUMERO DE CANAL⟩)	LSET⟨VARIAVEL DE CAMPO⟩=⟨VARIAVEL⟩

	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]
APPLE	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]
COMMODORE 64	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]
MSX	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]
IBM-PC	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]
SPECTRUM	—
TRS-80	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[<NUMERO DE LINHA2>,..]
MACINTOSH	ON<EXPRESSÃO>GOSUB<NUMERO DE LINHA1>[,<NUMERO DE LINHA2>,..]

	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]
APPLE	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]
COMMODORE 64	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]
MSX	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]
IBM-PC	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]
SPECTRUM	—
TRS-80	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>...]
MACINTOSH	ON<EXPRESSÃO>GOTO<NUMERO DE LINHA1 [,<NUMERO DE LINHA2>,..]

	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	PRINT#< NUMERO DE CANAL> <DADO>
APPLE	PINRT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>	WRITE#<NOME>
COMMODORE 64	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	PRINT#< NUMERO DE CANAL> < DADO >
MSX	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	PRINT#< NUMERO DE CANAL> <DADO >
IBM - PC	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	PRINT#< NUMERO DE CANAL> <DADO >
SPECTRUM	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	SAVE#<NOME DE PROGRAMA>" DATA <ARRAY >
TRS-80	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT " VARIAVEL>"	PRINT#<NUMERO DE CANAL> <DADO [,<DADO>,...]
MACINTOSH	PRINT PRINT, PRINT; PRINT "<MENSAGEM>" PRINT X PRINT "<VARIAVEL>"	WRITE#< NUMERO DE CANAL> <DADO1 [,<DADO2>,...]

	PRINT #	PUT#<N. DE CANAL>, < N. DE REGISTRO>
APPLE	PRINT #"<MENSAGEM> ,; < VARIAVEL 1>[, < VARIAVEL 2>,...]	
COMMODORE 64	PRINT #, "<MENSAGEM>" ,; < VARIAVEL>[[< VARIAVEL 2>,...]	
MSX	PRINT #<NUMERO DE CANAL>, "<MENSAGEM>" ,;<VARIAVEL 1>[,<VARIAVEL 2>,...]	
IBM - PC	PRINT #<NUMERO DE CANAL>,"<MENSAGEM> ,; < VARIAVEL 1>[,<VARIAVEL2>,...	PUT #<N. DE CANAL>, <N. DE REGISTRO>
SPECTRUM		
TRS-80		PUT #< N. DE CANAL> < N. DE REGISTRO>
MACINTOSH	PRINT# < NUMERO DE CANAL> <MENSAGEM> ,; < VARIAVEL 1>[,< VARIAVEL 2>,...	PUT #<N. DE CANAL>,<N. DE REGISTRO>

	OPEN (PARA ARQUIVO DE ACESSO DIRETO)
APPLE	OPEN "<NOME DO ARQUIVO>", Ln1, Sn2, Dn2, Vn4 ( )
COMMODORE 64	
MSX	
IBM - PC	OPEN "R", <N. CANAL>, "<NOME DO ARQUIVO>";<COMP. REGISTRO>
SPECTRUM	
TRS-80	OPEN "R", <N. CANAL> , "<NOME DO ARQUIVO>", <COMP. REGISTRO>
MACINTOSH	OPEN "R",#< N. CANAL>, "<NOME>",<COMP. REGISTRO>

(*) NO APLLE II OS COMANDOS DE MANEJO DE ARQUIVOS SÃO UTILIZADOS COMO CADEIAS DE CARACTERES, PRECEDIDOS DE CTRL-D.

	OPEN (PARA ARQUIVOS SEQÜENCIAIS)
APPLE	OPEN<NOME DO ARQUIVO>( )
COMMODORE 64	OPEN "<MODO>",#<NUMERO DE CANAL>, "<NOME DO ARQUIVO>"
MSX	OPEN "[<DISPOSITIVO>:] <NOME DO ARQUIVO> ' FOR <MODO> AS#<NUMERO DE CANAL>
IBM - PC	OPEN "<MODO>", <NUMERO DE CANAL> , "<NOME DO ARQUIVO>"
SPECTRUM	
TRS-80	OPEN "<MODO>", <NUMERO DECANAL>, "NOME DO ARQUIVO>"
MACINTOSH	OPEN "MODO>",#<NUMERO DE CANAL>, "<NOME DO ARQUIVO>"

(*) NO APPLE II OS COMANDOS DE MANEJO DE ARQUIVOS SÃO UTILIZADOS COMO CADEIAS DE CARACTERES, PRECEDIDOS DE CTRL-D.

	READ <VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE [<NUMERO DE LINHA>]	RETURN
APPLE	READ <VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE	RETURN
COMMODORE 64	READ< VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE	RETURN
MSX	READ< VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE [<NUMERO DE LINHA>]	RETURN
IBM-PC	READ< VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE [<NUMERO DE LINHA>]	RETURN
SPECTRUM	READ< VARIAVEL 1> [,< VARIAVEL 2>,...]	RESTORE	RETURN
TRS-80	READ< VARIAVEL 1> [,< VARIAVEL 2>,...]	RESTORE	RETURN
MACINTOSH	READ< VARIAVEL 1>[,< VARIAVEL 2>,...]	RESTORE [<NUMERO DE LINHA>]	RETURN

	RIGHT$[(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	RSET< VARIAVEL DE CAMPO>=<VARIAVEL>
APPLE	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	–
COMMODORE 64	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	–
MSX	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	RSET<VARIAVEL DE CAMPO>=<VARIAVEL>
IBM - PC	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	RSET<VARIAVEL DE CAMPO>=<VARIAVEL>
SPECTRUM	–	RND[<NUMERO POSITIVO>]	
TRS-80	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RND[<NUMERO POSITIVO>]	RSET<VARIAVEL DE CAMPO>=<VARIAVEL>
MACINTOSH	RIGHT$(<CADEIA> <NUMERO DE CARACTERES A EXTRAIR>)	RANDOM[<NUMERO POSITIVO>]	RSET<VARIAVEL DE CAMPO>=<VARIAVEL>

	RUN	RUN <NUMERO DE LINHA>	SAVE
APPLE	RUN	RUN <NUMERO DE LINHA>	SAVE "<NOME>"
COMMODORE 64	RUN	RUN <NUMERO DE LINHA>	SAVE "<NOME>", <PERIFERICO>
MSX	RUN	RUN <NUMERO DE LINHA>	SAVE "<PERIFERICO> .<NOME>"
IBM - PC	RUN	RUN <NUMERO DE LINHA>	SAVE "<PERIFERICO> .<NOME>"
SPECTRUM	RUN	RUN <NUMERO DE LINHA>	SAVE "<NOME>"
TRS-80	RUN	RUN <NUMERO DE LINHA>	SAVE "<NOME>"<PERIFERICO>"
MACINTOSH	RUN	RUN <NUMERO DE LINHA>	SAVE "<NOME>"

	SIN (<X>)	SQR (<X>)	TAN (<X>)	VERIFY "<NOMBRE>"
APPLE	SIN (<X>)	SQR (<X>)	TAN (<X>)	VERIFY "<NOME>"
COMMODORE 64	SIN (<X>)	SQR (<X>)	TAN (<X>)	VERIFY <NOME>"[, "<PERIFERICO>"]
MSX	SIN (<X>)	SQR (<X>)	TAN (<X>)	CLOAD? "<NOME>"
IBM-PC	SIN (<X>)	SQR (<X>)	TAN (<X>)	–
SPECTRUM	SIN (<X)	SQR (<X>)	TAN (<X>)	VERIFY "<NOME>"
TRS-80	SIN (<X)	SQR (<X>)	TAN (<X>)	–
MACINTOSH	SIN (<X)	SQR (<X>)	TAN (<X>)	–

	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	MKD$(<VARIAVEL>)	MKI$(<VARIAVEL>)
APPLE	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	–	–
COMMODORE 64	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)		
MSX	MID$(<CADEIA>,<PRIMEIR CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	MKD$(<VARIAVEL>)	MKI$(<VARIAVEL>)
IBM - PC	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	MKD$(<VARIAVEL>)	MKI$(<VARIAVEL>)
SPECTRUM	–	–	–
TRS-80	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	MKD$(<VARIAVEL>)	MKI$(<VARIAVEL>)
MACINTOSH	MID$(<CADEIA>,<PRIMEIRO CARACTER A EXTRAIR>,<NUMERO DE CARACTERES A EXTRAIR>)	MKD$(<VARIAVEL>)	MKI$(<VARIAVEL>)

	MKS$(<VARIAVEL>)	ON ERROR GOTO <NUMERO DE LINHA >	ON ERROR GOSUB<NUMERO DE LINHA>
APPLE	–	ONERR GOTO<NUMERO DE LINHA>	ONERR GOSUB<NUMERO DE LINHA >
COMMODORE 64	–	–	–
MSX	MKS$(<VARIAVEL>)	ON ERROR GOTO<NUMERO DE LINHA>	ON ERROR GOSUB<NUMERO DE LINHA>
IBM - PC	MKS$(<VARIAVEL>)	ON ERROR GOTO<NUMERO DE LINHA>	ON ERROR GOSUB<NUMERO DE LINHA>
SPECTRUM	–	–	–
TRS-80	MKS$(<VARIAVEL>)	ON ERROR GOTO<NUMERO DE LINHA>	ON ERROR GOSUB<NUMERO DE LINHA>
MACINTOSH	MKS$(<VARIAVEL>)	ON ERROR GOTO<NUMERO DE LINHA>	ON ERROR GOSUB<NUMERO DE LINHA >

BIBLIOGRAFIA

Livros recomendados para o BASIC, a informática e os computadores pessoais

Esperamos que a leitura das monografias relativas ao BASIC em geral tenham conseguido o objetivo que pretendiam: proporcionar-lhes um certo conhecimento desta linguagem e, talvez, animar-lhes a escrever por si mesmos algum pequeno programa. Se, ao contrário, não se sente ainda com o domínio suficiente para desenvolver um programa, ao menos lhe será mais fácil compreender as listagens dos inúmeros programas que encontrará publicados nas revistas especializadas ou dos quais tenha uma cópia em fita.

Estamos convencidos, no entanto, de que seu interesse pela informática dos computadores pessoais está somente em seu começo e gostará de saber muito mais. Estas monografias satisfazem dita necessidade e, sem um compromisso de leitura profunda, poderá encontrar nelas um pouco de tudo o que é atualidade nos computadores pessoais.

Se, ao contrário, despertamos em você um maior interesse pelo BASIC, então lhe aconselhamos ler outras obras que, com maior dedicação, aprofundam melhor nas matérias e situações diversas. Uma recomendação que sempre repetimos é a de ler com atenção os manuais que são fornecidos com os computadores. Somos conscientes de que freqüentemente são muito resumidos, redigidos em inglês ou, pior então, mal traduzidos. Mas ao menos em ditos manuais encontrará as instruções específicas do computador que utiliza.

Em seguida sugerimos um série de obras interessantes que se referem a quase todos os fundamentos da informática e também indicamos alguns textos de fácil consulta que se referem à programação em BASIC de computadores particulares.

Programas usuais em BASIC - TRS-80
Poole

Programas usuais em BASIC - Apple II
Poole

Programas usuais em BASIC
Poole

Programas práticos em BASIC
Poole

Programação com BASIC
Gottfried

Computadores e Programação
Scheid

Ciência dos Computadores
Tremblay

Iniciação ao BASIC
Fox

Manual de BASIC para o Apple II
Peckham

IBM PC e Compatíveis - Guia do Usuário
Sachs

Apple II - Guia do usuário
Poole

Programas práticos em BASIC - IBM PC e seus compatíveis
Poole

Manual do Apple Macintosh
Sanders

BASIC no TK-90X
Mirshawka

Brincando com o TRS Color
Mirshawka

Conceitos de processamento de dados em BASIC
Stern

Enciclopédia da linguagem | BASIC
Pereira

Primeiros passos na programação em linguagem de máquina
Silveira

Simulação em BASIC
MacNitt

Gráficos no Apple e TK-2000 - Conceitos e programas
Cavanha

Técnicas de gerenciamento de arquivo
Claybrook

BASIC Manual de Conversões - Apple x TRS x PET
Bank

NOTAS

Temos a segurança de que muitos de nossos leitores estavam esperando ansiosos por este segundo volume dedicado ao BASIC de nossa coleção. Efetivamente, em "BASIC I" ficaram muitos temas por tratar: subrotinas, funções, instruções de salto, arquivos, etc., que agora vamos estudar. Também incluímos uma tabela de equivalências das instruções vistas para distintos computadores.

Dada a amplitude da temática irão aparecendo sucessivamente outros volumes da B.B.I. dedicados ao aprofundamento de muitos dos aspectos que apareceram nos volumes BASIC I e BASIC II.

Cz$ 49,50