MICRO COMPUTADOR

CURSO BASICO

Sumário

VOLUME 1

Chips & bytes

Conexões

Fundamentos

Hardware

Os precursores

Perspectivas

VOLUME 1

Por dentro do hardware

Programação Basic

Software

Sumário

VOLUME 2

Chips & bytes

Conexões

Fundamentos

Hardware

Os precursores

Perspectivas

Por dentro do hardware

VOLUME 2

Programação BASIC

Software

Som e luz

Às suas ordens

Se você "falar" da maneira certa com seu computador, ele fará exatamente aquilo que você pedir. Sem cometer erros.

Outras linguagens

O BASIC é a linguagem de programação utilizada na maioria dos microcomputadores. Mas não é, sem dúvida, a única linguagem no gênero. Antes do advento dos micros, quando para a computação eram utilizados grandes computadores, os cientistas e engenheiros usavam a linguagem chamada FORTRAN. No mundo dos micros, são também empregadas as linguagens PASCAL, FORTH e LOGO.

Pascal

Tal como o BASIC, o PASCAL foi desenvolvido inicialmente como linguagem para estudantes de programação. Tem ótimo conceito entre os professores, pois possibilita elaborar programas refinados. A linguagem PASCAL é geralmente fornecida em discos flexíveis e tende a ser de alto custo. A linguagem PASCAL pode ser utilizada para escrever programas sofisticados e extensos.

Forth

Os programas escritos em FORTH são menos parecidos com a língua inglesa do que os feitos em BASIC ou PASCAL. O FORTH é também mais difícil para se aprender. É uma linguagem poderosa, pois programas complexos podem ser escritos em poucas linhas. Com a linguagem você define seus próprios comandos, enquanto no BASIC esses comandos são predefinidos.

Logo

A linguagem LOGO é relativamente nova e tem-se difundido na educação. Possui a grande vantagem da simplicidade para o ensino de jovens e crianças. Auxilia no aprendizado das técnicas de programação e também possibilita um enfoque lógico no desenvolvimento de programas. Com essa linguagem, desenhos são facilmente produzidos na tela. Uma "tartaruga" mecânica pode ser conectada ao computador. Comandos simples digitados no teclado movimentam a "tartaruga", fazendo-a desenhar linhas e figuras.

É perfeitamente possível a qualquer pessoa usar um computador — em casa ou no trabalho — sem nada saber sobre a maneira como ele funciona. Aqui tem início o MICROCOMPUTADOR — CURSO BÁSICO, que explica passo a passo, desde o princípio, tudo o que você precisa conhecer para criar seus próprios programas.

Muitas pessoas, após algum tempo, acham enfadonhos os programas e jogos que compraram prontos para o microcomputador. Elas querem saber se podem modificá-los ou mesmo escrever programas próprios. Como um computador nada pode fazer por si só, ele precisa receber uma série de instruções que lhe diga exatamente, em pormenores, o que fazer e como fazer. Essas instruções são chamadas "programa" e a arte de criá-lo é denominada "programação".

Não há nada difícil em programação. Você nem mesmo precisa ser bom em matemática, a menos, é claro, que queira escrever programas que realizem tarefas matemáticas. Para começar, basta apenas conhecer o BASIC.

Sua primeira linguagem

A maioria dos microcomputadores possui uma linguagem chamada BASIC. Como o nome diz, essa linguagem foi projetada para possibilitar aos iniciantes o aprendizado rápido e fácil das noções básicas de programação. Tal como qualquer linguagem humana, o BASIC tem seu próprio vocabulário e sua gramática, embora bem menos complexos que os da língua inglesa ou portuguesa. O BASIC utiliza pequeno número de palavras em inglês, facilmente reconhecíveis e nada difíceis de aprender. Como linguagem de finalidade genérica, é adequada tanto para iniciantes quanto para os usuários mais experientes.

Há um inconveniente, porém: durante os últimos anos, diversos fabricantes de computadores introduziram suas próprias modificações. Por isso existe grande número de variações no BASIC, especialmente com relação aos comandos de controle dos aspectos mais recentemente desenvolvidos da máquina — como cores, gráficos e sons. Todas as variações do BASIC existentes nos computadores mais populares encontram-se no quadro "A propósito...".

De um computador para outro, há variações no BASIC, tornando praticamente impossível escrever um programa nessa linguagem, menos ou mais complexo, que funcione em qualquer computador. Mas a linguagem tem um núcleo comum, que é normalmente o mesmo em todas as máquinas. Vamos concentrar-nos nesse núcleo e, à medida que o curso progredir, veremos programas de maior complexidade.

Os primeiros passos

Começaremos escrevendo um pequeno programa para ver o que acontece. Esse programa mostrará que o computador aparentemente cometeu um erro. Ligue o micro e digite o programa exatamente como é apresentado, incluindo os espaços. O <CR> ao final de cada linha é para lembrá-lo de que deve bater a tecla RETURN. Nos diversos micros, essa tecla pode ser também a ENTER ou a ↵.

```
10 REM COMPUTADORES JAMAIS COMETEM
   ERROS <CR>
20 PRINT "DIGITE UM NUMERO" <CR>
30 INPUT A <CR>
40 LET A = A + 1 <CR>
50 PRINT "ACHO QUE O NUMERO QUE VOCE
   DIGITOU FOI   "; <CR>
60 PRINT A <CR>
70 END <CR>
```

Após ter digitado o programa, datilografe LIST <CR>. O programa que você acaba de digitar deve reaparecer na tela. LIST é uma instrução para o computador "imprimir" uma listagem do programa que está na memória. Se o programa apareceu na tela de modo certo após o LIST, podemos experimentar o comando RUN, para processá-lo. Se você cometeu um erro de digitação, não se preocupe. Após o programa ter sido LISTado, simplesmente redigite qualquer linha que contenha erro. Não se esqueça do número da linha. Experimente digitar

```
25 REM AQUI ESTA OUTRA LINHA "REM" <CR>
```

e, em seguida, LISTe o programa novamente. Para eliminar a linha, digite só o número dela, seguido por <CR>. Quando você considerar o programa digitado corretamente, pode processá-lo, digitando RUN <CR>. Experimente isso e verá na tela:

```
DIGITE UM NUMERO
```

Continue e digite um número. Por exemplo, o 7. (Faça uso de numerais; o computador não reconhecerá "sete" como 7, a menos que o tenhamos programado para tal.) Se você digitou 7, na tela deverá aparecer:

```
ACHO QUE O NUMERO QUE VOCE DIGITOU FOI 8
```

Será que realmente o computador cometeu um erro, ou estava simplesmente obedecendo a ordens? Se observarmos o programa, linha por linha, veremos qual instrução fez com que ele agisse assim. Eis a primeira linha:

```
10 REM COMPUTADORES JAMAIS COMETEM
   ERROS
```

REM significa observação (REMark em inglês). Tudo o que aparecer na mesma linha após REM é ignorado pelo computador. As observações são uma forma útil de lembrá-lo do que o computador está fazendo. Este REM em particular é apenas um título, não nos indica o que o computador está fazendo. Posteriormente, veremos neste curso como são úteis essas observações REM, quando escritas de forma adequada. Agora, vejamos:

```
20 PRINT "DIGITE UM NUMERO"
```

Em BASIC, a parte que se segue à palavra PRINT é "impressa" na tela do computador. Observe que a sentença está entre aspas. Uma das regras do BASIC é que os caracteres (letras) que se seguem ao comando PRINT entre aspas aparecerão exatamente como foram digitados. Veremos outra forma de utilização do PRINT na linha 60. Em seguida, temos:

```
30 INPUT A
```

Pularemos esta linha por enquanto, retomando-a após observar a linha 40.

```
40 LET A = A + 1
```

A letra A é aqui empregada como uma variável. A variável pode ser considerada uma caixa rotulada que contém um número ou alguns caracteres. Em vez de termos de lembrar o que está na caixa, devemos apenas saber o nome da caixa para consultá-la. É como dizermos: "Dê-me a caixa rotulada B", em vez de: "Dê-me a caixa que contém parafusos de 15 mm".

Nessa linha temos uma caixa chamada A, conhecida como variável, pois o valor que nela colocamos pode variar. Podemos atribuir qualquer valor a uma variável. Um valor foi dado à variável A, na linha 30; assim, vejamos como isso se fez:

```
30 INPUT A
```

Usar a palavra INPUT é uma das formas de fornecer, em BASIC, um valor específico a uma variável. Quando um programa BASIC chega a uma linha que se inicia com INPUT, ele espera que algo seja digitado no teclado. INPUT A informa ao computador que temos uma variável chamada A e tudo o que for digitado no teclado será associado a essa variável. Digitar 7 <CR> neste ponto coloca 7 na caixa A, ou, utilizando o jargão de computador, associa o valor 7 à variável A. Agora sabemos o que é uma variável e vamos novamente observar a linha 40.

```
40 LET A = A + 1
```

O nome da variável à qual um valor foi associado sempre aparece à esquerda do sinal de igualdade. Aqui, estamos dando um novo valor para A. A declaração significa "LET (Deixe) o novo valor de A igual ao antigo valor mais 1". O antigo valor era 7. Agora temos 7 + 1, o novo valor é 8.

```
50 PRINT "ACHO QUE O NUMERO QUE VOCE
   DIGITOU FOI  ";
```

Temos, novamente, o comando PRINT. Ele "imprime" o conjunto de caracteres (ou seja, as palavras ou números que você digitou) entre as aspas. Observe o ponto e vírgula no fim da linha. Ele auxilia a especificar as posições em que os itens são mostrados na tela. Neste curso, voltaremos a falar do ponto e vírgula, com maiores detalhes. Agora vejamos:

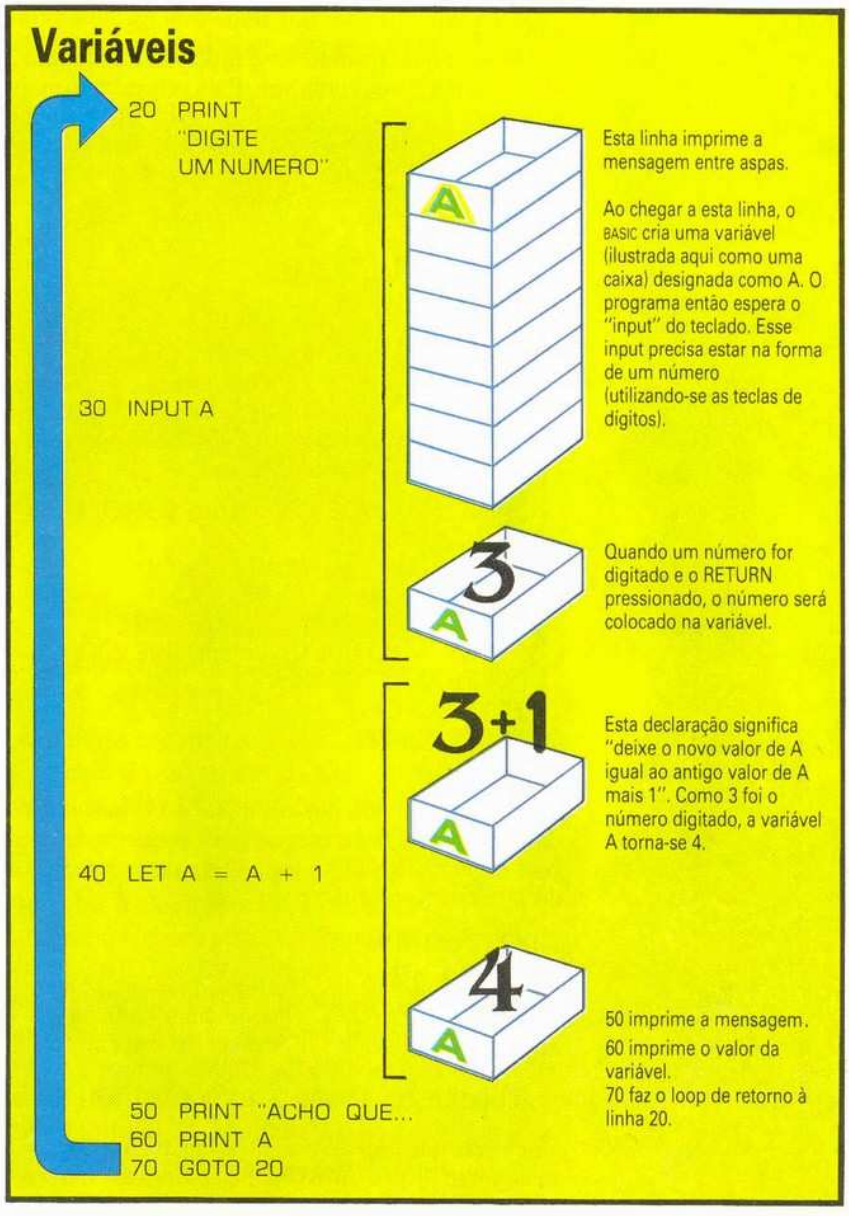
```
60 PRINT A
```

Eis outra declaração PRINT, mas aqui o A não está entre aspas. Já sabemos que o programa não mostrará um A na tela, porque precisaríamos das aspas para isso. Sem as aspas, o BASIC procura pôr uma variável com o mesmo nome do caractere após o PRINT. Se encontrar um, apresentará o valor da variável. (Se não encontrar um, dará uma mensagem de erro.) Este programa já possui uma variável chamada A e assim o BASIC mostrará seu valor. Qual é ele? Se você pensou que a resposta é 7, lembre-se de que o BASIC funciona por meio de programas, linha por linha. Quando estávamos na linha 60, o valor do A foi alterado para 8, e é isto que será mostrado. Finalmente, chegamos em:

```
70 END
```

O quadro abaixo mostra como as "variáveis" são usadas no BASIC e como o comando GOTO é empregado para formar um loop.

Variáveis



A propósito...

O TK85 e o CP 200 utilizam LET como parte da instrução. Na maioria dos outros computadores isso pode ser omitido. Por exemplo, a linha 20 pode ser escrita como A = A + 1, em vez de LET A = A + 1.

Não é utilizado no TK85 e no CP 200. A última linha do programa digitado é considerada o fim do programa.

Pode aparecer na tela como duas palavras (GO TO) no TK85 ou CP 200, embora apenas uma tecla seja pressionada. A maioria dos computadores aceitará a instrução digitada em duas palavras.

A declaração END diz ao BASIC que o fim do programa chegou. Algumas versões do BASIC insistem no fato de que todos os programas devem terminar com END, enquanto outras não (ver o quadro "A propósito...").

Observe que, quando você processa o programa, ele funciona apenas uma vez. Para processá-lo mais uma vez, você deve digitar RUN <CR> novamente. Agora, vamos analisar uma forma de obter o funcionamento do programa quantas vezes quisermos, mediante a utilização do comando GOTO.

Fazendo uso do GOTO

O mesmo programa, com uma linha a mais, é apresentado abaixo. Se você desligou o computador para descansar, digite-o novamente. Caso contrário, digite as linhas 70 e 80. Estas estão apresentadas em azul, na listagem abaixo.

```
10 REM COMPUTADORES JAMAIS COMETEM
   ERROS <CR>
20 PRINT "DIGITE UM NUMERO" <CR>
30 INPUT A <CR>
40 LET A = A + 1 <CR>
50 PRINT "ACHO QUE O NUMERO QUE VOCE
   DIGITOU FOI  "; <CR>
60 PRINT A <CR>
70 GOTO 20 <CR>
80 END <CR>
```

Após tê-lo digitado por completo e LISTado, tente descobrir o que acontece, antes de processá-lo. Depois digite RUN <CR> e, como na primeira versão do programa, você deve ver:

```
DIGITE UM NUMERO
```

Digite qualquer número (utilizando as teclas de numeral) e bata RETURN. O computador adicionará 1 ao número e o exibirá no fim da mensagem.

```
ACHO QUE O NUMERO QUE VOCE DIGITOU FOI 8
```

Você verá que isso será seguido imediatamente pela mensagem DIGITE UM NUMERO, outra vez. Introduzindo outro número e batendo RETURN novamente, você pode observar que o ciclo do programa recomeça *ad infinitum*. A razão para que isso aconteça encontra-se na linha 70:

```
70 GOTO 20
```

Quando o BASIC chega à declaração GOTO, não continua na próxima linha; em vez disso, vai para a linha de número especificado. Neste caso, está dirigido de volta à linha 20 e o programa inteiro recomeçará. Se você quiser parar o programa, perceberá que não existe forma de sair do loop. O programa simplesmente continua, aguardando sua entrada de dados.

Como seria de esperar, existem várias formas de escrever um programa, de modo que podemos sair do loop se quisermos. No próximo segmento deste curso veremos uma dessas formas. Por enquanto utilizaremos a tecla BREAK para fazê-lo. Digitar RUN <CR> fará com que o programa recomece.

Observe que ainda temos a declaração END, ao final do programa. O modo pelo qual escrevemos o programa com o comando GOTO 20 criou um loop infinito. Entretanto, algumas versões do BASIC exigem que sempre utilizemos um END no fim.

Se você não encontrou uma forma de parar o programa, tente a tecla RESET. Esta certamente vai pará-lo. Então, tente LISTá-lo novamente. Desse modo você será capaz de "editar" o programa nos exercícios abaixo. Se o programa não aparecer na tela, isso significa que o RESET em seu micro o destrói na memória e, assim, você deve digitar tudo de novo.

Exercícios

Estas questões foram cuidadosamente classificadas e pretendem ser de agradável resolução. A solução dos exercícios é uma das melhores maneiras de verificar se você compreendeu o material apresentado e está fazendo progressos reais.

Antes de começar os exercícios, procure alterar algumas linhas para ver o efeito no modo de processar-se o programa. O computador não será danificado se você cometer erros ou bater teclas erradas. Para alterar uma linha, digite o programa e depois, para verificação do resultado, LISTe-o. Todo o programa aparecerá na tela novamente. Digite o número da linha que deseja alterar seguido pela nova linha. Experimente isto:

```
10 REM COMPUTADORES ALGUMAS VEZES
   COMETEM ERROS <CR>
```

Em seguida digite LIST novamente. Observe como a primeira linha foi alterada. Se você desejar eliminar toda a linha, simplesmente digite o número da linha seguido de <CR>. Experimente:

```
10 <CR>
LIST
```

A linha 10 deve ter desaparecido. Reintroduza a linha 10, digitando a linha inteira novamente, e não se esqueça do número da linha.

■ Reescreva o programa, para que o computador re-

almente imprima o número digitado. Sugestão: retire uma linha inteira.

■ Redigite a linha 70, para que o programa vá para a linha 80. LISTe o programa. Acione o comando RUN. Por que ele não se processou do mesmo modo que antes?

■ Altere a linha 60, para que o computador imprima um A na tela em vez do valor da variável A.

■ Reescreva a linha 60, para que o computador imprima o valor da variável A outra vez. Retire a linha 10 (a linha REM) completamente. Processe o programa. Notou alguma diferença?

■ Coloque um novo REM na linha 25. Novas linhas podem ser adicionadas, simplesmente digitando o novo número seguido pela nova declaração. Coloque uma observação na linha 25 para lembrá-lo do que deve fazer a seguir (pode ser "esperar uma entrada de dados pelo teclado"). Após ter digitado a nova linha, bata <CR>, LISTe o programa novamente e verifique se suas observações aparecem no lugar certo.

■ Reescreva o programa, a fim de ele multiplicar por 10 o número que você digitou. Você precisará alterar a linha 50 para imprimir algo como o NUMERO QUE VOCE DIGITOU MULTIPLICADO POR 10 E. Desta vez, não queremos somar o valor da variável antiga, e sim multiplicá-la por 10. O BASIC faz uso do sinal * para significar "multiplicar". (Não utilize um X, porque o BASIC apenas o reconhece como uma letra e não como sinal de multiplicação.)

Até aqui, examinamos uma boa quantidade de assuntos. Vimos como escrever comentários que o BASIC chama de REM, como imprimir conjuntos de caracteres na tela, qual o modo de imprimir o valor das variáveis na tela e como fazer um programa ir a um número de linha especificada utilizando o GOTO.

Da próxima vez, veremos como sair de um loop, utilizando a declaração IF-THEN. Descobriremos como conseguir que o programa seja executado um número especificado de vezes, sem ficar em loop contínuo.

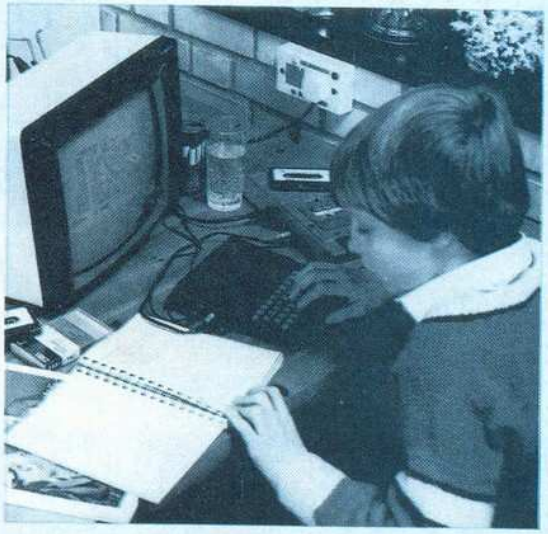
Além disso, veremos como a velocidade do programa pode ser diminuída, fazendo com que o computador pareça estar realmente pensando.

E o BASIC foi criado

Hoje em dia, o BASIC é a linguagem de programação mais popular do mundo. As linguagens de computador foram inventadas para permitir que o operador humano se comunique de modo mais fácil com a máquina. O BASIC compõe-se de instruções em inglês que foram combinadas de forma simples com símbolos matemáticos.

Em pouco tempo, utilizando o BASIC, você já conseguirá elaborar pequenos programas. O BASIC foi concebido em 1965 por dois professores da Faculdade de Dartmouth (New Hampshire, Estados Unidos), Thomas Kurtz e John Kemeny, com a finalidade de simplificar as linguagens existentes. Com o tempo, ligeiras variações na linguagem original foram ocorrendo. Mas o núcleo do BASIC permanece comum a todos os fabricantes.

Um programa é uma seqüência de instruções que o computador executa para desenvolver uma tarefa específica. O programa aparece na tela do micro como uma série de linhas numeradas. Cada linha contém uma instrução e o número permite ao computador obedecer aos comandos na ordem correta. Os comandos são rapidamente aprendidos e até mesmo programas mais complicados fazem uso apenas de combinações e repetições de comandos elementares.



O BASIC eliminou o mistério da programação e tornou a computação acessível a todos.

A maioria dos micros vem da fábrica com a linguagem BASIC embutida. Os computadores podem também ser programados em "linguagem de máquina" (designada como linguagem de "baixo" nível, pois é a que está mais próxima da estrutura da lógica encontrada nos circuitos eletrônicos). O BASIC é linguagem de "alto" nível, pois é muito semelhante ao inglês cotidiano. Existem muitas outras linguagens de alto nível planejadas para aplicações mais técnicas e especializadas, mas o BASIC é a melhor introdução a todas elas. É uma linguagem simples e poderosa.

Loops sob controle

Você pode interromper os loops e voltar a eles quantas vezes quiser. Nesta segunda parte do curso de programação você também aprende o melhor modo de numerar as linhas.

A primeira parte do curso de Programação BASIC terminou com o programa relacionado abaixo. O programa funcionou bem, mas, devido ao GOTO na linha 70, sempre voltava para o início. A única maneira de sair do loop era com o uso da tecla BREAK ou com o da tecla RESET. Conheça agora uma das maneiras de sair de um loop como este, incluindo um teste no programa.

Geralmente se testa com um número que, na verdade, nunca iríamos querer usar no programa. O programa permitia digitar um número que o computador então imprimia na tela somando-lhe 1. Poderíamos decidir que nunca seria usado um número superior a 999. Nesse caso, há possibilidade de testar para saber se o número que entrou é maior do que 999. Digite o programa e então acrescente:

```
35 IF A > 999 THEN GOTO 80 <CR>
```

Agora execute o programa novamente e ele funcionará como antes — a menos que você introduza um número maior do que 999. Tente digitar 1000 <CR> e veja o que acontece.

Por que o programa parou desta vez? O IF na linha 35é o motivo. Quando o BASIC encontra um comando IF sabe que se segue a ele um teste de lógica. O sinal > significa "maior que". A linha 35, portanto, significa IF (variável) A (é maior que) 999 THEN GOTO (linha) 80. Se você apenas digitar 1000, o valor de A será 1000, que é maior do que 999, e o programa THEN (então) GO (vai) TO linha 80, que é o fim do programa. Se A não é maior do que 999, a parte THEN da linha é ignorada e o programa continua na linha seguinte.

Ao executar este programa, você poderá introduzir quantos números quiser, desde que não sejam maiores do que 999. Logo que um número superior a 999 for introduzido, o comando IF-THEN detecta o fato e encerra o programa (GO TO END). Quando um programa BASIC atingir o fim ou for concluído, aparecerá um sinal na tela. Dependendo do seu computador, esse sinal pode ter diversas formas. No microcomputador Unitron AP II, é um sinal como este:]; no CP 500, é READY. Qualquer que seja a sua forma, este sinal é como o BASIC informa que nenhum programa está sendo executado, e aguarda novas instruções.

Há inúmeras variações na forma em que as diferentes versões do BASIC usam THEN. Mais detalhes você encontrará na seção "A propósito...", na página 40.

Outras comparações usadas em BASIC são < (menor que); = (igual a); ≥ (maior que ou igual a); ≤ (menor que ou igual a) e ≠ (diferente de). Veremos essas comparações usadas com freqüência à medida que o curso evoluir.

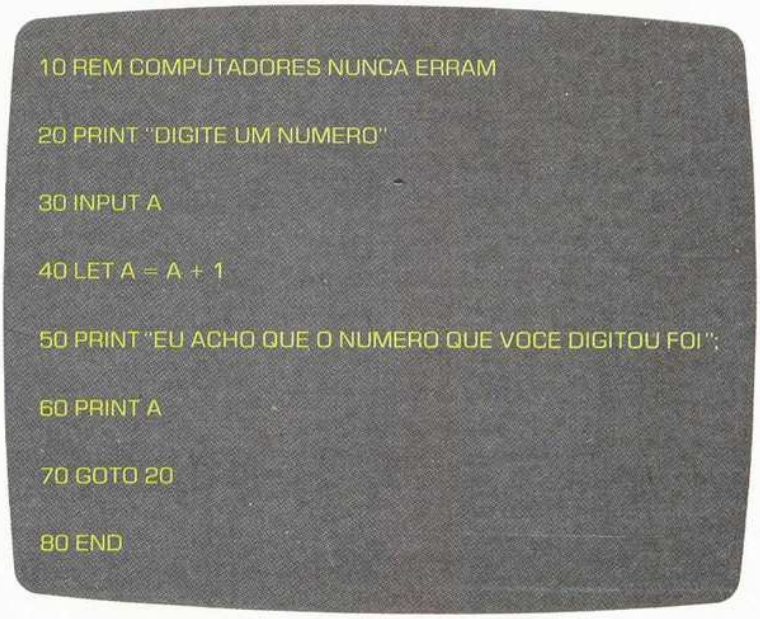
Antes de continuar, faremos alguns exercícios para aprender o uso das comparações.

Exercícios

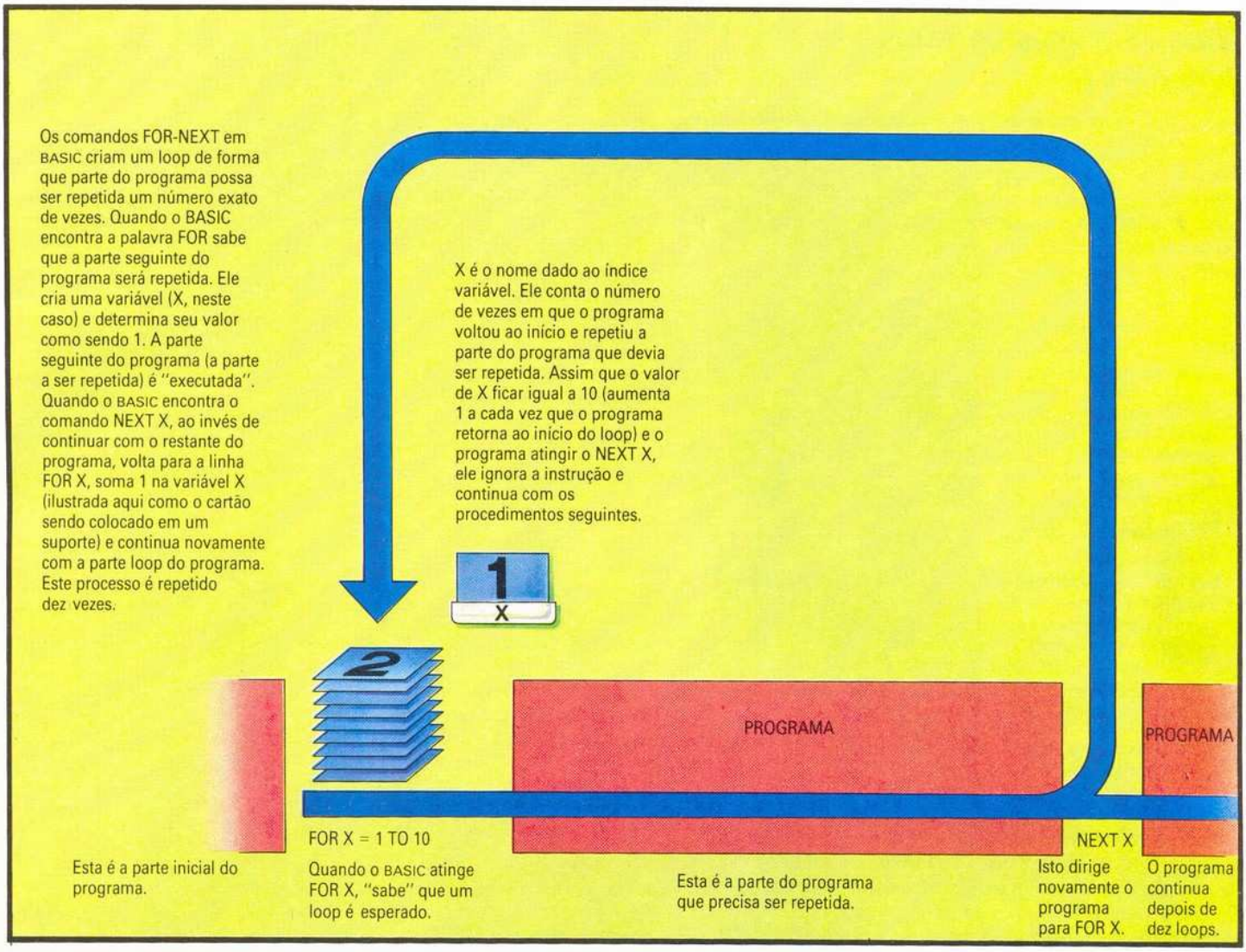
- Mude uma das linhas de forma que o programa seja interrompido se A = 1000.
- Mude uma das linhas de forma que o programa seja interrompido se o número introduzido for menor que zero.
- Mude a linha GOTO de forma que faça o programa voltar para o começo, se A for igual ou menor que 500. Sugestão: você não precisará de uma linha separada IF-THEN e de uma linha GOTO.

Descobrindo FOR-NEXT

Ao escrever programas, haverá muitas ocasiões em que você gostará de que alguns itens do programa se repitam determinado número de vezes. O GOTO na linha 70 possibilitou que o programa fizesse quantos loops quiséssemos. Mais tarde acrescentamos um comando IF-THEN na linha 35, que nos permitiu sair, introduzindo um número "fora de limite".



```
10 REM COMPUTADORES NUNCA ERRAM
20 PRINT "DIGITE UM NUMERO"
30 INPUT A
40 LET A = A + 1
50 PRINT "EU ACHO QUE O NUMERO QUE VOCE DIGITOU FOI";
60 PRINT A
70 GOTO 20
80 END
```



Entretanto, há ocasiões, como vimos na primeira parte do curso, em que usar o GOTO para fazer um loop não é a melhor maneira de agir.

Voltemos ao nosso antigo programa, desta vez modificado, realmente, para multiplicar o número que entra (input) por 10 e fazê-lo exatamente oito vezes.

```
10 REM MULTIPLICACAO POR 10
20 FOR X = 1 TO 8
30 PRINT "DIGITE UM NUMERO"
40 INPUT A
50 LET A = A * 10
60 PRINT "SEU NUMERO MULTIPLICADO POR 10 E ";
70 PRINT A
80 NEXT X
90 END
```

Digite este programa, LISTe para verificar erros e, então, RUN. Um número será solicitado apenas oito vezes. Depois disso, o programa simplesmente pára. A razão para que isso aconteça está na linha 20.

```
20 FOR X = 1 TO 8
```

Isto faz parte de um loop FOR-NEXT. É uma das estruturas mais úteis que o BASIC tem a oferecer e, por isso, merece um estudo cuidadoso.

Da forma como fizemos aqui, criamos uma variável chamada X. (As variáveis estão explicadas na primeira parte do curso, na p. 17.) Poderíamos ter-lhe dado qualquer nome (exceto A, que estamos usando para outra coisa). FOR deve sempre ser usado com o correspondente NEXT, mas NEXT aparecerá mais adiante no programa, depois da parte a ser repetida. A parte FOR de um loop FOR-NEXT sempre tem a seguinte forma:

FOR variável = valor inicial TO valor final. No nosso exemplo FOR X = 1 TO 8, chamamos a variável de X e lhe demos valor inicial de 1. A parte seguinte do programa é então executada pelo computador. O número que nós digitamos é multiplicado por 10 e então mostrado na tela. Depois disso vamos para NEXT X e o programa volta para onde a variável X está, na linha 20. Assim que isso tenha sido feito, X aumenta 1, e passa a ter o valor 2. A parte do programa dentro do loop FOR-NEXT é então outra vez executada. Ao chegar a NEXT novamente na linha 80, o programa volta e aumenta X para 3.

O programa se repete desta forma até que X tenha sido aumentado para 8. Depois disso, o loop termina. NEXT X não volta para FOR X e o programa continua na linha seguinte.

Mais usos para os loops FOR-NEXT

Os loops FOR-NEXT são freqüentemente usados para deter o programa por algum tempo. Há momentos em que você não quer que tudo seja feito com a máxima velocidade e então pode empregar esta estrutura. Você provavelmente achou que as respostas do programa MULTIPLICACAO POR 10 vieram tão depressa que pareciam instantâneas. Façamos com que o computador pareça estar precisando pensar antes de responder, usando FOR-NEXT para provocar um atraso. Acrescente as linhas indicadas em azul ao seu programa.

```
10 REM MULTIPLICACAO POR 10
20 FOR X = 1 TO 8
30 PRINT "DIGITE UM NUMERO"
40 INPUT A
50 LET A = A * 10
52 FOR D = 1 TO 1000
54 NEXT D
60 PRINT "SEU NUMERO MULTIPLICADO POR 10 E ";
70 PRINT A
80 NEXT X
90 END
```

Acrescentamos outras duas linhas, 52 e 54, ao nosso loop FOR-NEXT original. Vamos examiná-las.

```
52 FOR D = 1 TO 1000
54 NEXT D
```

D é fixado em 1 e o programa vai para a linha seguinte. Este é o comando NEXT correspondente. Na verdade, nada acontece dentro do loop. O programa simplesmente volta para a linha 52 e aumenta D para 2. Isto acontece mil vezes antes que o programa vá para a parte seguinte, que é imprimir a resposta. Os computadores são rápidos, mas tudo tem um tempo finito, de forma que fazer mil vezes um loop gasta um tempo facilmente perceptível. Os computadores variam no tempo que levam para fazer um loop. No Unitron AP II, o loop FOR-NEXT leva 1,9 segundo, enquanto no TK85 leva 4,5 segundos. Experimente, mudando o número que você usa como limite mais alto na linha 52.

Para fazer com que o computador se comporte mais como um ser humano, acrescente estas três linhas:

```
56 PRINT "AGORA DEIXE-ME VER..."
57 FOR E = 1 TO 1000
58 NEXT E
```

LISTe o programa e RUN. Temos agora dois atrasos que não levam a nada, a não ser a perder tempo.

Acrescente estas duas linhas:

```
51 REM ESTE LOOP DESPERDICA TEMPO
55 REM ISTO DESPERDICA MAIS TEMPO
```

Agora LISTe o programa e preste atenção a ele. Veja como todas as linhas extras que acrescentamos se encaixaram nos lugares certos. O que nos leva ao último ponto desta parte do curso: número de linhas.

Começamos nosso programa original com a linha 10 e fomos aumentando, em saltos de 10 para cada nova linha, terminando com a linha 90. Poderíamos ter escolhido quaisquer números, por exemplo 1, 2, 3... 9. Mas, se tivéssemos feito isso, como encaixaríamos as linhas extras? Os programadores sempre têm novas idéias e fazem melhoramentos, e por isso deixam grandes intervalos entre os números de linha na primeira versão de seus programas. Você pode até começar com o número de linha 100 e aumentar em saltos de 50 ou 100 se quiser.

Algumas versões do BASIC incluem uma instrução útil chamada AUTO. Este é o caso do BASIC do CP 500.

Já no Unitron AP II, no TK83 e 85 e no CP 200 esta instrução não está disponível. Se o seu BASIC tem AUTO, é possível economizar bastante tempo, tendo os números do linha gerados para você automaticamente. Descubra se tem AUTO digitando:

```
AUTO 100, 10 <CR>
```

Se o seu BASIC tiver AUTO, você verá na tela:

```
100
```

A tela mostra o número 100 seguido de um espaço e depois o cursor. O cursor é um sinal (às vezes um triângulo ou um quadrado) que indica na tela o lugar em que o próximo caractere vai aparecer. Você pode começar introduzindo a primeira linha do programa a partir da posição do cursor. Quando atingir <CR>, a linha seguinte aparecerá automaticamente, começando com o número de linha 110. O AUTO, se você o tem, pode ser usado sozinho ou com um ou dois argumentos. Argumento é um termo matemático. Na expressão 2 + 3 = 5, os argumentos são 2 e 3. A instrução AUTO pode ser usada sozinha (i.e., AUTO <CR>), ou com um argumento (ex.: AUTO 100 <CR>),ou com dois argumentos (ex.: AUTO 300,50). AUTO, sozinho, geralmente faz com que os números de linha comecem em 10 e aumentem em saltos de 10. Se apenas um argumento é usado (ex.: AUTO 100 <CR>), o primeiro número será 100 (neste caso) e então os números aumentarão de acordo com o "valor default", que geralmente é 10. Se você especificar dois argumentos, o primeiro número especifica o número de linha inicial e o segundo especifica o incremento. AUTO 250,50 <CR> indica um número inicial de 250; o próximo número será 300 e assim por diante, aumentando sempre 50. Mesmo no micro mais simples é improvável que a numeração das linhas chegue ao seu limite.

Na próxima parte deste curso, veremos as várias maneiras de melhorar a apresentação visual do programa na tela e diferentes formas de imprimir dados.

A propósito...

IF A maioria dos microcomputadores pode usar esta instrução na forma IF A > 999 THEN 80 ou IF A > 999 GOTO 80.

AUTO Esta instrução não existe nos computadores da linha Apple (Unitron AP II) e Sinclair (TK83).

Direto ao ponto

Esteja atento aos detalhes de pontuação quando escrever um programa para seu micro. Eles são muito importantes.

Você deve ter observado no programa da primeira parte da programação BASIC que há um ponto e vírgula no final da linha 50. A função desse sinal de pontuação não foi explicada naquele momento, mas é muito importante. O sinal é utilizado em quase todas as versões BASIC para concatenar as seções impressas. As linhas 50 e 60 da página 22 são:

```
50 PRINT "ACHO QUE O NUMERO QUE VOCE
   DIGITOU FOI ";
60 PRINT A
```

A linha 50 imprimiu as palavras contidas entre as aspas. A linha 60 imprimiu o valor da variável A. A colocação do ponto e vírgula fez com que o valor da variável A fosse impresso diretamente após as palavras entre aspas, na linha 50. Se o ponto e vírgula não fosse empregado, o valor da variável teria sido impresso na linha seguinte.

O programa a seguir foi projetado para ilustrar algumas propriedades úteis do ponto e vírgula, tal como é usado no BASIC. Experimente digitá-lo e processá-lo. De agora em diante, omitiremos o <CR> no final de cada linha, para indicar que você deve pressionar a tecla RETURN. O programa a seguir permite a você registrar uma série de temperaturas em graus centígrados (conhecidos como Celsius) e tê-los automaticamente convertidos aos graus equivalentes em Fahrenheit.

```
10 REM PROGRAMA PARA CONVERTER GRAUS C EM F
20 PRINT "FORNECA A TEMPERATURA MAIS BAIXA"
30 INPUT L
40 PRINT "FORNECA A TEMPERATURA MAIS ALTA"
50 INRUT H
60 FOR X = L TO H
70 LET F = (X * 9 / 5) + 32
80 PRINT X; " EM CENTIGRADO E "; F; " EM FAHRENHEIT "
90 NEXT X
100 END
```

Digite este programa. LISTe-o para verificação e, depois, utilize o comando RUN para fazê-lo funcionar. Em primeiro lugar lhe será solicitado fornecer a temperatura mais baixa. Tente digitar –5. Em seguida, você será solicitado a fornecer a temperatura mais alta. Tente digitar 10. O programa converterá todas as temperaturas com 1 grau de intervalo de –5 a 10 graus centígrados aos equivalentes em Fahrenheit. Você deve obter uma impressão em blocos na tela semelhante a esta:

```
-6 EM CENTIGRADO E 21.2 EM FAHRENHEIT
-5 EM CENTIGRADO E 23 EM FAHRENHEIT
-4 EM CENTIGRADO E 24.8 EM FAHRENHEIT
```

Observe que as colunas não estão uniformes, devido aos pontos decimais, mas cada valor em centígrado tem seu equivalente em Fahrenheit, em uma única linha. Após ter processado esse programa algumas vezes, redigite a linha 80, tal como é, mas substitua todos os pontos e vírgulas por vírgulas e acione o comando RUN novamente. Como você pode notar, a impressão tornou-se uma desordem.

Para verificar por que isso ocorreu, experimentemos um programa muito simples, comparando o efeito das vírgulas com o dos pontos e vírgulas. Digite NEW<CR>. Depois introduza:

```
10 REM COMPARACAO DO ; COM A ,
20 PRINT "ESTA LINHA USA PONTO E VIRGULA"
30 PRINT "H";"E";"L";"P"
40 PRINT "ESTA LINHA USA VIRGULAS"
50 PRINT "H", "E", "L", "P"
60 END
```

A linha 30 fará com que apareça na tela a palavra HELP, enquanto na linha 50 aparecerá H E L P. Reporte-se ao quadro "A propósito..." para ver as variações entre as diferentes máquinas. A vírgula possui diversos empregos em BASIC, mas nos comandos PRINT tem o efeito de fazer os itens aparecerem espaçados na tela (ou na impressão em papel), comumente entre 8 e 16 espaços, dependendo da versão do BASIC. Se a instrução PRINT for utilizada sem vírgula ou ponto e vírgula, os itens serão impressos em linhas separadas.

Além de ilustrar o emprego do ponto e vírgula no BASIC, nosso programa de conversão de temperatura fez a revisão de diversas instruções abordadas nas duas primeiras partes do curso de programação BASIC. As linhas 30 e 50 associaram as variáveis L e H aos valores mais alto e mais baixo de temperaturas que desejamos converter. A linha 60 é a primeira parte do loop FOR-NEXT. Este loop parece ser diferente dos outros que encontramos até aqui, por utilizar letras ao invés de números. Na realidade, não há diferença. As letras que estamos utilizando aqui, L e H, são variáveis com valores numéricos correspondentes aos valores digitados no estágio do programa do INPUT L e INPUT H. Se, como anteriormente foi sugerido, você introduziu –5 e 10, a declaração FOR X = L TO H é, portanto, equivalente a FOR X = –5 TO 10.

A linha 80 diz, na realidade: PRINT o valor de X (que começa na temperatura mais baixa e aumenta 1 grau de cada vez até a temperatura mais alta) segui-

do diretamente na mesma linha (por isso utilizamos o ponto e vírgula) pelas palavras entre aspas, seguido outra vez diretamente (outro ponto e vírgula) pelo valor de F. Se você olhar atentamente para F, verá que é o valor, naquele momento, da temperatura em graus centígrados, convertida em Fahrenheit. A linha NEXT X assegura que continuemos as conversões até que o limite superior no loop FOR-NEXT tenha sido alcançado.

Antes de observarmos uma variação mais sofisticada da declaração PRINT, é importante que paremos um segundo para examinar a linha 70 do nosso programa de conversão de temperatura:

```
70 LET F = (X * 9 / 5) + 32
```

Esta linha associa um valor à variável F (que significa Fahrenheit). O programa, em primeiro lugar, obtém o valor de X (a temperatura em graus centígrados), multiplica-o por 9, divide-o por 5 e, depois, soma 32. Essa mesma fórmula seria apresentada em um livro comum de matemática do seguinte modo: $F = [(C \times 9) \div 5] + 32$. O BASIC usa * para a multiplicação, / para a divisão, + para a soma e − para a subtração.

Na matemática comum, e no BASIC também, a ordem de execução das operações aritméticas é importante. A multiplicação tem sempre prioridade, seguida pela divisão, seguida pela soma, seguida pela subtração. Se partes da expressão aritmética estão encerradas entre parênteses, devem ser realizadas em primeiro lugar. Se você desejar fazer antes da multiplicação uma soma, esta deverá estar encerrada entre parênteses. Por exemplo, se você quisesse saber o equivalente em dólares da soma de suas contas bancária e de poupança, deveria expressá-la em seu programa assim:

```
D = (C + P) / taxa do dólar
```

Se na conta corrente você tem Cr$ 500.000,00 (C) e na poupança Cr$ 700.000,00 (P), deverá somar em primeiro lugar os cruzeiros (C + P) e, depois, dividir pela taxa do dólar. Sem os parênteses, o valor de sua conta em poupança seria em primeiro lugar dividido pela taxa do dólar e, em seguida, o valor de sua conta corrente seria somado ao resultado — e não é isso que desejamos! Tenha sempre certeza de haver verificado as operações aritméticas, para que sejam calculadas na ordem correta.

O PRINT USING

A fim de verificar um último aspecto em nosso programa de conversão, experimente digitá-lo outra vez e acione o comando RUN. Introduza, digamos, −10, como o valor inferior de temperatura, e 10 para o valor superior. Como já vimos, a impressão em blocos da tela é muito confusa. Isso se deve aos pontos e vírgulas na linha 80, que reúnem todas as partes que estão sendo impressas, ao invés de imprimi-las em linhas separadas. O que seria bom, exceto pelo fato de que varia o espaço tomado pelos números, tanto os graus centígrados quanto os Fahrenheit. Isso tem como efeito o desalinhamento das colunas, tornando desordenada a impressão em blocos.

Quase todas as versões do BASIC possuem uma forma de PRINT denominada PRINT USING. Esta permite que a aparência dos números impressos, ou palavras, seja "formatada", ou harmonizada. Se você desejar imprimir o valor de X e souber antecipadamente que esse valor varia de, digamos, −99 a 99, os números podem ser impressos corretamente alinhados o utilizando PRINT USING "###";X. Os três sinais de numeral permitem que três dígitos, ou dois dígitos precedidos pelo sinal de menos, sejam impressos. Se mais de três dígitos forem introduzidos, eles não serão impressos corretamente. Sendo necessário, pontos decimais podem ser incluídos na posição adequada dentro dos sinais de numeral. Por exemplo, a declaração pode tomar a forma PRINT USING "###.##";X. Utilize um sinal de numeral para cada dígito. Todos os pontos decimais serão alinhados automaticamente.

Modifique o programa original alterando a linha 80 e adicionando as linhas 82, 84 e 86:

```
80 PRINT USING "###";X;
82 PRINT " EM CENTIGRADO E ";
84 PRINT USING "###.##";F;
86 PRINT " EM FAHRENHEIT"
```

LISTe o programa novamente e acione o comando RUN, para executá-lo. Todas as colunas devem agora estar alinhadas perfeitamente.

No próximo segmento do curso vamos descobrir como "guardar" programas, para que não precisem ser redigitados cada vez que se queira executá-los.

Exercícios

■ Experimente introduzir a "temperatura inferior" de −1.000. Por que o programa não funcionou desta vez? Como você modificaria a declaração PRINT USING na linha 80 para fazê-lo funcionar?

■ Altere a linha 84, de modo que apenas os números inteiros (sem frações decimais) sejam impressos.

■ Escreva um programa para converter uma determinada quantidade de valores em cruzeiros para dólares, fazendo uso da taxa de câmbio do dia.

A propósito...

PRINT USING

Esta característica não se encontra nos modelos compatíveis com o Apple (por exemplo, Unitron), nem nos compatíveis com o Sinclair (TK83 e 85, e CP 200).

VÍRGULA

A vírgula entre campos de impressao separará os itens a serem impressos, com a inserção de um número de espaços, que varia conforme o tipo de computador utilizado. Nos computadores da linha Apple, a tela é dividida em três campos, dos quais dois possuem 16 espaços e um possui 8 espaços. Nos computadores da linha Sinclair (TK's, CP 200) e TRS-80 (CP 500), a tela é dividida em dois campos de 16 espaços.

Problemas de rotina

Programas dentro de programas: conheça um novo aspecto do BASIC que tornará seus programas concisos e fáceis de ler.

Em partes anteriores deste curso de programação BASIC, pudemos digitar, processar e modificar programas, limpando depois a memória (pelo uso do comando NEW) para poder processar outros programas. Mas, sempre que havia necessidade de processar o programa anterior, era preciso digitar tudo outra vez.

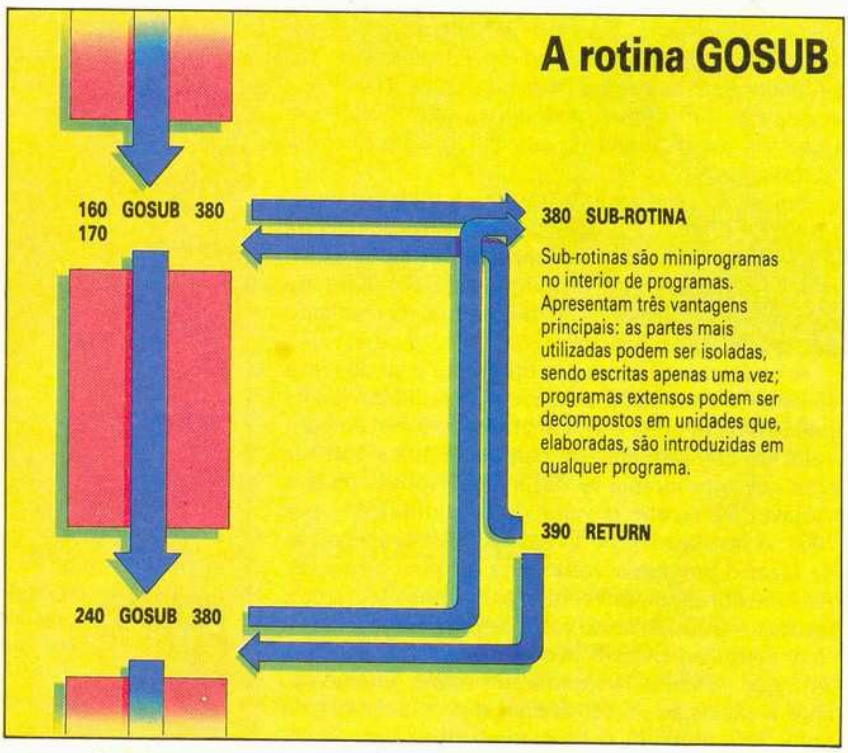
Para evitar esse trabalho repetitivo, todos os computadores que operam em linguagem BASIC são equipados com um comando que permite a armazenagem de qualquer programa em fita cassete. O programa abaixo pode ser guardado em fita usando-se o comando SAVE, seguido por um nome de arquivo. Vejamos, por exemplo, o cálculo do número de azulejos necessários para revestir uma sala.

```
10 REM ESTE PROGRAMA CALCULA O NUMERO DE
   AZULEJOS
20 REM NECESSARIOS PARA REVESTIR UMA SALA
30 PRINT "FORNECER O TAMANHO DA LATERAL DO
   AZULEJO EM MM"
40 INPUT A1
50 REM LINHA 60 CALCULA A AREA DO AZULEJO
60 LET A2 = A1 * A1
70 PRINT "FORNECER O NUMERO DE PAREDES"
80 REM W ESTABELECE OS LIMITES DO LOOP
90 INPUT W
100 FOR X = 1 TO W
110 PRINT "COMPRIMENTO DA PAREDE NO."; X;
   "EM METROS"
120 REM D E A DIMENSAO DA PAREDE
130 INPUT D
140 REM E CONVERTIDA EM MM
150 REM NA SUB-ROTINA
160 GOSUB 380
170 REM LINHA 190 ESTABELECE L COMO
180 REM COMPRIMENTO DA PAREDE EM MM
190 LET L = D2
200 PRINT "ALTURA DA PAREDE NO."; X; "EM
   METROS"
210 REM LINHAS 230 A 250 ESTABELECE H
220 REM COMO ALTURA DA PAREDE EM MM
230 INPUT D
240 GOSUB 380
250 LET H = D2
260 REM LINHA 270 ESTABELECE A3 COMO AREA
   DA PAREDE
270 LET A3 = L * H
280 REM S (SUBTOTAL) E A AREA DA PAREDE
   DIVIDIDA
290 REM PELA AREA DO AZULEJO
300 LET S = A3/A2
310 REM T (TOTAL) TEM O NOVO SUBTOTAL
320 REM ACRESCENTANDO CADA VEZ ATRAVES
   DO LOOP
330 LET T = T + S
340 NEXT X
350 REM IMPRIMIR O TOTAL
360 PRINT T
370 END
375 REM SUB-ROTINA DE CONVERSAO M × MM
380 LET D2 = D*1000
390 RETURN
```

Após digitar o programa, você usará o comando SAVE para guardá-lo em fita cassete. Naturalmente, o gravador cassete deve ser preparado de acordo com as instruções do manual de seu computador.

O comando SAVE é muito simples: digite SAVE, seguido de um nome de arquivo entre aspas. Arquivo, em computação, assemelha-se a um arquivo qualquer: um programa ou um conjunto de dados são, mediante seu nome de arquivo, armazenados ou recuperados quando necessário. É sempre melhor usar um nome de arquivo que faça você se lembrar da função do programa. Como nosso programa calcula o número de azulejos necessários para revestir uma sala, podemos chamá-lo AZULEJOS. Estando o gravador cassete preparado, introduza uma fita limpa para guardar o programa.

Os gravadores com tomada para controle remoto geralmente têm motor controlado diretamente pelo computador. Caso contrário, aperte a tecla para gra-



vação e depois a de pausa. Digite o comando SAVE e o nome do arquivo; ligue o gravador soltando a tecla de pausa e pressione a tecla RETURN.

Para certificar-se de que o programa foi gravado corretamente, limpe a memória do computador digitando o comando NEW <CR>. Retroceda a fita, aperte a tecla para reprodução e carregue novamente o computador com o programa, usando o comando LOAD seguido do nome do programa em questão. Coloque: LOAD "AZULEJOS" e pressione a tecla RETURN.

Após o computador ter recebido o programa, uma mensagem na tela, como READY ou OK, indicará que o programa já está carregado. LISTe o programa e verifique se é idêntico ao que você digitou inicialmente.

GOSUB

A instrução que desvia o fluxo de um programa para uma sub-rotina é GOSUB. A sub-rotina pode ser definida como um miniprograma ou um programa no interior de outro. No programa aqui usado para ilustração, a sub-rotina é bastante simples e sua função é mostrar o princípio do procedimento.

Nosso programa destina-se a encontrar o número de azulejos para o revestimento de uma sala. Ele calcula a área da parede, depois de o comprimento e a altura serem convertidos de metros em milímetros. O total de azulejos é encontrado pela divisão da área de cada parede pela área do azulejo e a subseqüente soma dos resultados. A conversão do comprimento e da altura da parede em milímetros é efetuada pela sub-rotina, que simplesmente multiplica o comprimento ou a altura (em metros) por 1000, para encontrar o equivalente em milímetros.

As sub-rotinas apresentam três vantagens. As partes do programa utilizadas freqüentemente podem ser separadas e necessitam ser escritas apenas uma vez — não importando quantas vezes a operação seja necessária. Também permitem que programas longos e complexos sejam decompostos em unidades com as quais é mais fácil trabalhar. Finalmente, as sub-rotinas podem ser reutilizadas em qualquer outro programa em que sua função seja adequada.

Em nosso programa, a sub-rotina inicia-se na linha 380 e consiste em apenas uma instrução: LET D2 = D * 1000. Com isto, D, a dimensão da parede (comprimento ou largura), é multiplicada por 100 para sua conversão de metros em milímetros. O resultado é atribuído à variável D2.

A instrução que induz o programa a se dirigir à sub-rotina é o GOSUB e ocorre pela primeira vez na linha 160. Na linha 130, era atribuído à variável D o valor do comprimento da parede. A linha 160 faz com que o programa se dirija à sub-rotina, onde a variável D2 recebe o valor de D multiplicado por 1000. A instrução RETURN na linha 390 tem a função de fazer o programa voltar para a rotina principal. As sub-rotinas sempre voltam para a linha seguinte à instrução GOSUB; neste caso, à linha 170.

A instrução GOSUB ocorre novamente na linha 240, que "chama" a mesma sub-rotina. Desta vez, após a execução da sub-rotina, o programa volta à linha 250. Embora o programa utilize uma única sub-rotina, é possível usarmos quantas forem necessárias. Em cada caso, a instrução GOSUB terá de incluir o número da linha da sub-rotina desejada. Observe que a instrução END ocorre na linha 370, antes da sub-rotina. END indica o fim do programa principal e serve igualmente para evitar que, após o término do programa, o processamento continue, invadindo as sub-rotinas.

Este é um programa um pouco mais extenso do que os já apresentados neste curso, mas sua complexidade não é maior. Experimente acompanhá-lo linha por linha, observando o que se passa em cada estágio. Com exceção da instrução GOSUB e das sub-rotinas, ele introduz um único conceito novo: nomes de variáveis mais extensos.

Será útil traçar quadros, escrevendo neles os nomes das variáveis e os valores que adquirem em cada estágio.

Linha 300: LET S = A3/A2 em alguns casos resultará em um número com fração decimal. Experimente processar o programa fornecendo a medida do azulejo como 110 mm e o comprimento e altura da parede como 2,3 e 1,8 m, respectivamente, calculando uma única parede: a resposta deverá ser 342, 149 azulejos. Uma vez que os azulejos não são vendidos em unidades fracionárias, este resultado não é completamente satisfatório. Na próxima oportunidade, examinaremos um dos métodos de atingir resultados satisfatórios com números inteiros.

Exercícios

■ Observe o que acontecerá se você fornecer o tamanho do azulejo como sendo 0 mm. Por que ocorre uma indicação de erro? Por que não acontece o mesmo quando você fornece o comprimento da parede como 0 m? Dica: os procedimentos de multiplicação e de divisão por zero não são os mesmos; verifique isso em sua calculadora.

■ O programa só dá certo no caso de azulejos quadrados. Tente modificar as linhas de 30 a 60 para calcular a área de azulejos retangulares (o procedimento é o mesmo usado para o cálculo da área das paredes retangulares neste mesmo programa).

Diferenças básicas

O TK83 não possui este comando; por isso é necessário outro procedimento para se desviar das linhas 380 e 390, o que é conseguido pela mudança da linha 370: 370 GOTO 400 e acréscimo da linha 400 PRINT "END".

Para o TK83 é necessário que todas as variáveis sejam definidas antes da realização de cálculos matemáticos. Para fazer com que a linha 330 não perca seu sentido, deve-se acrescentar a seguinte linha:
5 LET T = 0.

No TK83, este comando está escrito em duas palavras, bastando, entretanto, pressionar uma única tecla.

À espera do Natal

Novos comandos para trabalhar com dados e escrever um programa que calcula quantos dias faltam para o Natal.

Este programa retoma todos os tópicos vistos até agora em nosso curso de programação e também introduz algumas importantes instruções de linguagem BASIC. A finalidade do programa é calcular quantos dias faltam para o Natal do ano corrente.

Se observar a listagem do programa, você perceberá que ela se inicia por uma lista de variáveis usadas. Este procedimento não é essencial, porém é recomendável, pois pode tornar seus programas mais fáceis de ser entendidos, quando você posteriormente vier a examiná-los.

Algumas versões do BASIC permitem o uso de variáveis com nomes extensos, DIA, por exemplo, em vez de letras isoladas, como temos usado até agora. Se tiver a sorte de possuir um equipamento em linguagem BASIC, que possibilita nomes extensos de variáveis, escolha nomes sugestivos. Nomes como DIA, MES, NUMDIA são preferíveis a símbolos como A, X ou D. Se não tiver escolha, porque o BASIC que você utiliza não possibilita nomes extensos de variáveis, a anotação das variáveis no início do programa torna-o mais "legível".

Quando o programa é processado, o primeiro sinal que surge na tela são as instruções PRINT, que se iniciam na linha 230. Elas indicam resumidamente o que o programa fará e então solicitam ao usuário digitar a data no formato apresentado, separando dia, mês e ano com vírgulas.

A primeira instrução não familiar encontra-se na linha 300. É uma instrução DIMensão (DIMension), utilizada para determinar o total de itens ou elementos que entram na tabela designada como X. As tabelas, às vezes chamadas variáveis indexadas, assemelham-se a variáveis comuns, mas apresentam várias subdivisões. Na linha 300, criamos uma variável denominada X, com treze subdivisões no seu interior. Voltaremos, mais adiante, à questão das tabelas e da instrução DIM, detalhadamente.

```
310 INPUT D, M$, A
```

Esta linha é uma instrução comum para entrada de dados (INPUT), com a diferença de que prevê três entradas. D é uma variável numérica que contém a data do dia; A é outra variável numérica, para o ano. A variável M$ é um pouco diferente. É denominada variável alfanumérica (variável string), indicada pelo sinal $ (cifrão). Este tipo de variável aceita caracteres, bem como números, do teclado. Se, por exemplo, digitarmos 23, JANEIRO, 1983, à variável D será atribuído o valor 23, à variável M$ a série de caracteres JANEIRO e à variável A o valor 1983.

```
330 GOSUB 560 REM "ROTINA NO. DO MES"
```

Esta instrução faz o programa desviar-se para a sub-rotina que começa na linha 560. Observe, igualmente, que uma instrução REM foi inserida na mesma linha. Se houver espaço na linha, não será necessário colocar a instrução REM em outra linha. Esta sub-rotina em particular é usada pelo programa principal apenas uma vez e, a rigor, poderia com igual facilidade ter sido inserida nele. Fazer uma sub-rotina apenas isola esta parte do resto do programa.

Quando o programa foi inicialmente desenvolvido, um número era usado para indicar o mês e esta parte do programa não era necessária. Posteriormente, decidiu-se fazer o mês constar como uma palavra digitada por extenso. A fim de converter o mês por extenso no seu número correspondente, um programa suplementar foi desenvolvido separadamente, na forma dessa sub-rotina. A única mudança necessária no programa principal (original) foi acrescentar uma única instrução GOSUB. Esta sub-rotina exemplifica a facilidade com que os programas podem ser elaborados em blocos e unidos com a utilização dos comandos GOSUB e RETURN.

A rotina em si é muito simples, mas exemplifica a engenhosidade da linguagem BASIC em lidar com séries de caracteres. Admitamos que fornecêssemos JANEIRO como o mês solicitado na instrução de entrada (INPUT). À variável M$ seria atribuída a série de caracteres JANEIRO. A primeira linha da sub-rotina é:

```
560 IF M$ = "JANEIRO" THEN LET M = 1
```

Esta instrução compara os conteúdos da variável M$ com os caracteres entre aspas. Se forem os mesmos (como acontece neste caso), a linha prossegue a fim de colocar 1 como valor da variável numérica M. Não confunda a variável M com a variável M$. São diferentes: apenas uma pode conter a variável alfanumérica: aquela com o sinal $. Após verificar se a variável M$ corresponde a JANEIRO, o programa segue para a próxima linha e verifica se o conteúdo da variável M$ corresponde a FEVEREIRO. Caso contrário, a variável M não passa a ter valor 2. Apenas onde a correspondência for correta a variável M receberá um valor, que será correspondente ao número do mês: 1 para janeiro, 3 para março etc.

Ao chegar à linha 680, o programa retorna (RETURN) ao programa principal, na linha seguinte à instrução GOSUB: é a linha 340, que contém uma instrução REM, sem nenhuma observação; foi inserida simplesmente para dar mais espaço entre as linhas, tornando mais fácil a leitura do programa.

As linhas 350 e 370 são um loop do tipo FOR-NEXT, que incrementa o valor da variável I, iniciando com 1 e seguindo até 13. A variável I é usada

como indexador da tabela X, na linha 360, e deve ser cuidadosamente examinada.

```
360 READ X(I)
```

READ é uma instrução nova, ainda não usada anteriormente. É sempre utilizada com uma instrução DATA correspondente. A instrução DATA para esta linha está na linha 510:

```
510 DATA 31,28,31,30,31,30,31,31.30, 31, 30, 25, 0
```

Essês números, com exceção dos dois últimos, representam o total de dias em cada mês do ano. As duas linhas equivalem a treze instruções LET.

```
LET X(1)  = 31
LET X(2)  = 28
LET X(3)  = 31
LET X(4)  = 30
LET X(5)  = 31
LET X(6)  = 30
LET X(7)  = 31
LET X(8)  = 31
LET X(9)  = 30
LET X(10) = 31
LET X(11) = 30
LET X(12) = 25
LET X(13) = 0
```

O loop montado na linha 350 faz a variável I percorrer de 1 a 13; desse modo, foi possível substituir X(I) por X(1), X(2), X(3) etc.

Antes de voltar a este programa, examinemos um outro menor e bem mais simples:

```
10 READ A, B, C
20 LET D = A + B + C
30 PRINT D
40 DATA 5, 10, 20
```

Aqui, a instrução READ, na linha 10, faz a leitura do primeiro item da instrução DATA na linha 40 e "registra" seu valor para a primeira variável. Em outras palavras, atribui o valor 5 à variável A. A instrução READ lê, em seguida, o próximo item e o coloca na variável seguinte. Este programa torna A = 5, B = 10 e C = 20. Em seguida, soma-os e atribui o resultado à variável D. O resultado, 35, é impresso (PRINT) na linha 30.

Voltemos ao nosso programa principal. Na primeira passagem pelo loop, iniciado na linha 350, o valor da variável I é 1. Assim, a linha 360 equivale a READ X(1). O item correspondente, na linha 510, é 31 (o primeiro item). Conseqüentemente X(1) passa a ter 31 como valor.

Na segunda passagem pelo loop, a variável I torna-se 2 e, com isso, a linha 360 passa a equivaler a READ X(2). O próximo item na linha DATA é 28. Isto significa que a variável X(2) passa a valer 28. Desse modo, todas as treze "subdivisões" da variável X indexada são preenchidas com o número de dias de cada mês; com exceção do 12.º mês, que tem apenas 25 dias (você percebe por quê), e da décima terceira subdivisão, de que falaremos mais adiante.

```
390 GOSUB 750 REM "ROTINA DE ANO BISSEXTO"
```

Essa linha dirige o programa para uma sub-rotina que verifica se o ano fornecido é ou não um ano bissexto. Vejamos seu procedimento.

```
100 REM LISTA DE VARIAVEIS
110 REM
120 REM D = DATA DE HOJE
130 REM M$ = NOME DO MES
140 REM A = ANO
150 REM I = INDEXADOR 1
160 REM X = TABELA DE DIAS EM CADA MES
170 REM R = DIAS RESTANTES
180 REM M = NO. DO MES
190 REM L = INDEXADOR 2
200 REM Z = VALOR INTEIRO DE A/4
210 REM
220 REM
230 PRINT "ESTE PROGRAMA CALCULA"
240 PRINT "O NUMERO DE DIAS QUE FALTAM"
250 PRINT "ATE O NATAL DESTE ANO"
260 PRINT
270 PRINT "FORNECER O DIA DE HOJE, MES, ANO"
280 PRINT "EX.: 12, JULHO, 1984"
290 PRINT
300 DIM X(13)
310 INPUT D, M$, A
320 REM
330 GOSUB 560 REM ROTINA "NO. DO MES"
340 REM
350 FOR I = 1 TO 13
360 READ X(I)
370 NEXT I
380 REM
390 GOSUB 750 REM ROTINA "ANO BISSEXTO"
400 REM
410 LET R = X(M) - D
420 FOR L = M TO 11
430 LET M = M + 1
440 LET R = R + X(M)
450 NEXT L
460 REM
470 IF R = 1 THEN GOTO 500
480 PRINT "FALTAM"; R; "DIAS PARA O NATAL"
490 GOTO 520
500 PRINT "FALTA 1 DIA PARA O NATAL"
510 DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 25, 0
520 END
530 REM
540 REM
550 REM
560 IF M$ = "JANEIRO" THEN LET M = 1
570 IF M$ = "FEVEREIRO" THEN LET M = 2
580 IF M$ = "MARCO" THEN LET M = 3
590 IF M$ = "ABRIL" THEN LET M = 4
600 IF M$ = "MAIO" THEN LET M = 5
610 IF M$ = "JUNHO" THEN LET M = 6
620 IF M$ = "JULHO" THEN LET M = 7
630 IF M$ = "AGOSTO" THEN LET M = 8
640 IF M$ = "SETEMBRO" THEN LET M = 9
650 IF M$ = "OUTUBRO" THEN LET M = 10
660 IF M$ = "NOVEMBRO" THEN LET M = 11
670 IF M$ = "DEZEMBRO" THEN LET M = 12
680 RETURN
690 REM
700 REM
710 REM
720 REM NOTA: ESTA ROTINA NAO VERIFICA
730 REM       OS ANOS BISSEXTOS DO FINAL
740 REM       DE CADA SECULO
750 LET A = A/4
760 LET Z = INT (A)
770 IF A - Z = 0 THEN GOTO 790
780 RETURN
790 LET X(2) = X(2) + 1
800 RETURN
```

```
750 LET A = A/4
760 LET Z = INT (A)
770 IF A-Z = 0 THEN GOTO 790
780 RETURN
790 LET X(2) = X(2) + 1
800 RETURN
```

O ano bissexto é definido como "divisível por 4". No caso de um século, deve poder ser divisível por 400 para ser um ano bissexto. Para simplificar, não verificamos o século, apenas a divisibilidade por 4.

A linha 750 coloca na variável A o seu valor anterior dividido por 4. O novo valor da variável A resultará em um número inteiro, se o ano for divisível por 4. Caso não, terá uma fração decimal.

A linha 760 utiliza a função INT para calcular o valor "inteiro" de A. Se o número do ano fornecido for 1985, o novo valor da função INTeiro transforma o número com fração decimal no número inteiro imediatamente inferior. O número a ser arredondado é colocado entre parênteses após a instrução INT. Outra alternativa é a colocação do nome de uma variável entre parênteses. Assim, LET Z = INT (496,25) atribuiria à variável Z o valor 496.

A linha 770 subtrai Z de A e verifica se o resultado é 0. Se for, significa que o ano é bissexto (pois não há fração decimal no novo valor da variável A). Se for este o caso, o programa se dirige para a linha 790 pelo uso da instrução GOTO. A linha 790 acrescenta 1 ao segundo item da tabela (o segundo item era 28, o número de dias em um mês de fevereiro de ano não bissexto).

Se o resultado da subtração na linha 770 não for zero, X(2) é mantido como está e a sub-rotina RETORNA ao programa principal, na linha 400.

A linha 400 é outra instrução REM usada apenas para deixar mais espaço entre as linhas e tornar o programa mais fácil de ler. A próxima linha que efetivamente realiza alguma operação é a 410, onde R é a variável que contém o número dos dias restantes. A ela foi atribuído o número de dias no mês dado, menos o número do dia fornecido. Se tivéssemos fornecido, por exemplo, 12, FEVEREIRO, 1983, a variável D seria igual a 12 e a variável M seria 2. Assim, X(M) seria igual a X(2) e o segundo item da tabela X seria 28 (e não teria o número 1 somado a ele, pois 1983 não é ano bissexto). Por conseqüência, a variável R terá o valor de 28 − 12, isto é, 16, o número de dias restantes no mês fornecido, fevereiro

A linha 420 inicia um outro loop. Ele tem por objetivo incrementar o valor da variável M. Você pode perceber por que dizemos FOR L = 1 TO 11, em vez de FOR L = 1 TO 12? Se a variável M for 2, porque fornecemos o mês de FEVEREIRO, a linha 430 aumentará seu valor para 3. Na linha 440 o valor da variável R, que representa o número de dias restantes, passa a ser o último valor de R mais X(M). A variável X(M) agora equivale a X(3), já que M foi aumentado em uma unidade. O valor de X(3) é 31, o total de dias em março. Desse modo, na linha 440 o novo valor da variável R passa a ser 16 + 31 (16 foi o resultado da subtração 28 − 12). Na próxima passagem pelo loop, a variável M aumentará para 4 e o número de dias em abril, X(4), será somado ao último valor da variável R. Assim, esta se torna 16 + 31 + 30.

A última passagem pelo loop ocorre quando o último valor de M, 11, for alcançado (na linha 420). A linha 430 acrescenta 1 ao valor da variável M, pela última vez, e ela passa a valer 12. O valor de X(12) na instrução DATA é 25. Bem, não há 25 dias em dezembro, porém nos interessa apenas o número de dias até o Natal. Quando o mês atual for dezembro, neste loop M assumirá o valor 13 (12 + 1), e R já estará calculado corretamente com o número de dias até o Natal (R = X(12) − D). É por este motivo que a 13.ª subdivisão de X é igual a zero, para que, quando M for igual a 13, X(13) seja igual a zero, não alterando o valor de R.

```
470 IF R = 1 THEN GOTO 500
```

Esta linha verifica se falta só um dia para o Natal, de modo que obtenha uma resposta na tela gramaticalmente correta. Se a variável R não for 1, deve faltar mais de um dia e com isso a instrução PRINT da linha 480 estará gramaticalmente correta.

Isso é tudo. A versão da linguagem BASIC que usamos pode ser processada na maioria dos computadores (ver o quadro "A propósito..."), com exceção possivelmente da sub-rotina do "ano bissexto". O uso do comando LET varia muito, dependendo da versão do BASIC utilizada. Se linhas como IF M$ = "SETEMBRO" THEN LET M = 9 não derem certo em seu computador, a sub-rotina poderá ser mudada para:

```
560 IF M$ = "JANEIRO" THEN GOTO 900
570 IF M$ = "FEVEREIRO" THEN GOTO 910
580 IF M$ = "MARCO" THEN GOTO 920
:
900 LET M = 1
905 RETURN
910 LET M = 2
915 RETURN
920 LET M = 3
925 RETURN
(... e assim por diante)
```

Esta solução consome mais espaço e é mais difícil de ser acompanhada com as diversas instruções GOTO e RETURN. Todavia, ela mostra que há vários modos de resolver um mesmo problema.

A propósito...

INPUT

O TK85 ou CP 200 não são compatíveis com a entrada de séries de valores separados por vírgulas. Desse modo, a linha 310 deve ser substituída por: 310 INPUT D, com o acréscimo das linhas: 312 INPUT M$ e 314 INPUT A. Como a data vai agora ser fornecida em três estágios, deve-se substituir as instruções PRINT por: 280 PRINT "DIA?"; acrescentam-se também as linhas: 311 PRINT "MES?" e 313 PRINT "ANO?"

REM

Caso sejam usadas no TK85 ou CP 200, é aconselhável que as instruções REM estejam em linhas separadas.

END

Não existe esta instrução no TK85 ou CP 200 e, assim, a linha 520 deve ser mudada para: 520 GOTO 1000, e deve ser acrescentada a linha: 1000 REM FIM DO PROGRAMA.

Desafie os elementos

As variáveis indexadas, ao contrário das variáveis simples, podem conter qualquer quantidade de elementos.

Em nosso programa anterior, no cálculo do número de dias que faltam para o Natal, encontramos um novo tipo de variável, a variável "indexada". É diferente das variáveis comuns, ou "simples", pois pode conter um número indeterminado de subdivisões ou elementos. As variáveis simples admitem duas letras ou uma letra seguida por um dígito, de 0 a 9 (algumas versões de BASIC admitem o uso de palavras inteiras, como nomes de variáveis). As variáveis A, B, B1, C3 e R2 são todas simples. As variáveis indexadas têm esta forma: A(6), B(12) ou X(20). O indexador é o número entre parênteses. A leitura dos exemplos acima é a seguinte: "A indexado de seis", "B indexado de doze" e "X indexado de vinte".

Se imaginarmos uma variável simples como uma caixa identificada por um nome, podemos visualizar uma variável indexada como uma caixa que contém determinado número de elementos em seu interior. Se desejamos uma variável com doze elementos, nós a criamos, utilizando a instrução DIM, da seguinte forma: DIM A(12). Pode-se usar qualquer letra do alfabeto.

Atribuem-se valores a variáveis simples pelas instruções LET ou INPUT, como: LET A = 35, LET B1 = 365, ou INPUT C3. De modo semelhante, valores são atribuídos aos elementos de uma variável indexada. Vejamos como é possível atribuir valores a uma matriz indexada ("matriz" é o nome alternativo dado a conjuntos de variáveis indexadas). Por exemplo:

```
10 DIM A(5)
```

cria uma variável indexada com cinco elementos. Podemos agora atribuir um valor a cada elemento:

```
20 LET A(1) = 5
30 LET A(2) = 10
40 LET A(3) = 15
50 LET A(4) = 20
60 LET A(5) = 100
```

A fim de perceber como essas variáveis diferem das variáveis simples, experimentemos atribuir valores a algumas variáveis simples:

```
70 LET X = 5
80 LET Y = 6
90 LET Z = 7
```

Experimente fornecer estes dados a seu computador e depois confira o conteúdo de cada variável, com o comando PRINT. Muitas das instruções da linguagem BASIC também funcionam como comandos. Após ter fornecido as instruções acima, confira-as, LISTando-as. Digite em seguida a instrução PRINT X < CR >. O número 5 deverá aparecer imediatamente na tela. Em seguida, digite o comando PRINT Y. O computador responderá a este comando apresentando o número 6 na tela. Se quiser conferir os elementos da variável indexada, digite o comando PRINT A(1) para verificar o valor do primeiro elemento da tabela. O computador deverá apresentar o número 5 na tela. Experimente usar o comando PRINT para conferir os valores de A(3) e A(5).

A maior diferença entre variáveis indexadas e variáveis comuns está em que o próprio indexador pode ser uma variável. A fim de perceber o que isto significa, digite o comando PRINT A(X). A tela apresentará o número 100. Por quê?

Examine a lista que você digitou e verifique o valor da variável X. O valor é 5 e corresponde a A (o valor da variável X), o que equivale a A(5). Desse modo, digitar PRINT A(X) corresponde exatamente a digitar PRINT A(5). Que valor será obtido se você digitar PRINT A(Y−X)? Antes de fazê-lo efetivamente, experimente calcular você mesmo a resposta.

Atribuição de valores

Se houver apenas algumas variáveis simples, a instrução LET será o modo mais simples de atribuir-lhes valores. As variáveis indexadas podem ter grande número de elementos na matriz; assim, verifiquemos quais são as alternativas para o fornecimento de valores:

```
10 DIM A(5)
20 PRINT "FORNECER AS VARIAVEIS"
30 INPUT A(1)
40 INPUT A(2)
50 INPUT A(3)
60 INPUT A(4)
70 INPUT A(5)
```

Este método de digitação é tão cansativo quanto o uso das instruções LET, embora funcione. Se soubermos quantas variáveis existem realmente (neste caso, cinco), será mais fácil o uso do loop FOR-NEXT, da seguinte forma:

```
10 DIM A(5)
20 FOR X = 1 TO 5
30 INPUT A(X)
40 NEXT X
```

Este programa prevê a digitação de cinco valores no teclado do computador, ao ser processado o programa. A tecla RETURN deverá ser pressionada depois de cada novo número fornecido. Se soubermos previamente quais serão os valores incluídos, será mais fácil fornecê-los com a instrução READ combinada a uma instrução DATA, da seguinte forma:

```
10 DIM A(5)
```

```
20 FOR X = 1 TO 5
30 READ A(X)
40 NEXT X
50 DATA 5, 10, 15, 20, 100
```

Experimente este pequeno programa e verifique o conteúdo da matriz usando o comando PRINT (ou seja, use o comando PRINT após o programa ter sido processado). Por exemplo, PRINT A(1) <CR> e PRINT A(5). Agora, podemos acrescentar ao programa algumas linhas, de modo a instruí-lo a automaticamente fazer por nós o trabalho de impressão dos elementos incluídos na matriz:

```
60 FOR L = 1 TO 5
70 PRINT A(L)
80 NEXT L
90 END
```

Processe este programa e verifique se os valores corretos aparecem na tela. Em seguida, digite novamente a linha 50, usando cinco diferentes elementos DATA. Lembre-se de que os números incluídos na instrução DATA devem ser separados por vírgulas, mas não pode haver vírgulas antes do primeiro número, nem após o último.

O método mais simples de atribuição de valores consiste no uso de instruções DATA e READ. Se os valores variarem toda vez que o programa for processado, o uso de uma instrução INPUT como parte de um loop FOR-NEXT será provavelmente o melhor método. Se o número total de elementos na matriz for fixo, ele poderá ser usado para indicar o limite máximo, na instrução FOR.

Vamos recorrer a tudo que aprendemos até agora para elaborar um programa curto, porém bastante eficaz. Suponha que queremos colocar alguns números em ordem crescente. Antes de escrever o programa, o primeiro passo será elaborar sob forma lógica os procedimentos que podem resolver o problema. Quando o método para solucionar o problema estiver claro, anote as etapas sucessivas, usando sentenças curtas em português.

Vamos supor que iniciamos com cinco números: 4, 9, 2, 8, 3. Colocá-los em ordem crescente é um procedimento simples: basta examinar a linha, verificar qual o menor deles e colocá-lo à esquerda, procedendo do mesmo modo com os demais dígitos.

Todavia, o computador necessita de um conjunto muito preciso de instruções e temos de planejar muito claramente os passos exigidos. Eis uma possibilidade: compare o primeiro dígito com o segundo. Se o primeiro for maior que o segundo, inverta as posições; se for menor, mantenha-os no mesmo lugar.

Em seguida, compare o segundo dígito com o terceiro: se for menor que o terceiro, deixe-os na mesma posição; caso contrário, faça nova substituição.

Repita esse procedimento até atingir o último par de dígitos.

Não havendo necessidade de substituição na linha, todos os dígitos deverão estar na ordem correta. Se ocorreu alguma substituição, volte ao início e repita todo o processo.

Se observar este procedimento, você perceberá que, na verdade, ele pode colocar qualquer conjunto de números em ordem numérica crescente. Veja como se processa a ordenação do conjunto inicial de números, por esse processo de comparação de dígitos:

```
4  9  2  8  3
4  2  9  8  3
4  2  8  9  3
4  2  8  3  9
```

Todos os pares foram comparados e trocados, quando necessário. Uma vez que houve necessidade de pelo menos uma substituição, volte ao início e repita o procedimento:

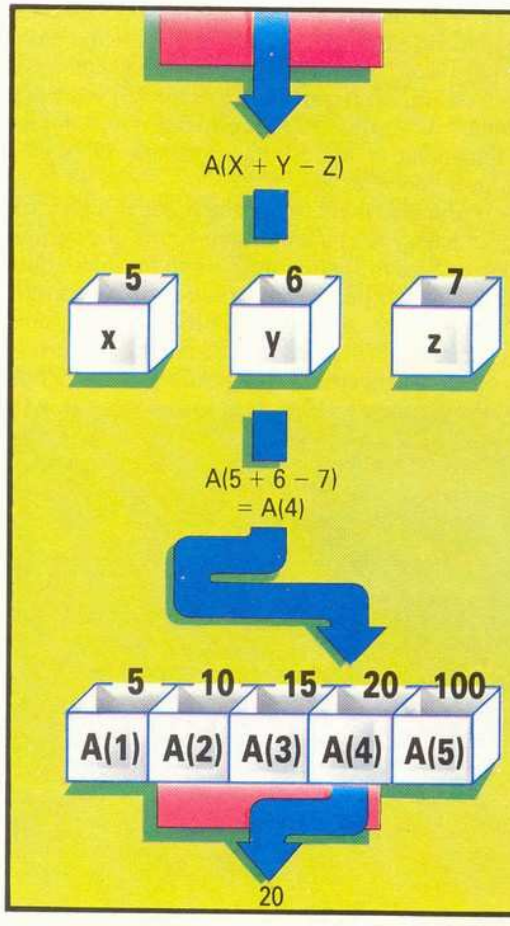
```
4  2  8  3  9
2  4  8  3  9
2  4  3  8  9
```

Ainda houve necessidade de substituições; assim, volte ao início e repita novamente:

```
2  4  3  8  9
2  3  4  8  9
2  3  4  8  9
```

Não houve necessidade de substituição, no último exame da linha; em conseqüência, todos os números devem ser menores, em comparação ao dígito à sua direita. Ou seja, os números estão em ordem crescente e a operação pode ser encerrada.

O uso de variáveis indexadas possibilita a elaboração de rotinas como essa, em linguagem BASIC, pois o indexador pode ser uma variável. Se nosso conjunto inicial de cinco números constituir os valores de uma matriz, de modo que A(1) = 4, A(2) = 9, A(3) = 2, A(4) = 8 e A(5) = 3, caso X tenha o valor 1,



Variáveis indexadas

As variáveis indexadas (variáveis com diversas subdivisões) aumentam sensivelmente a capacidade da linguagem BASIC. Aqui a variável A tem o indexador X + Y – Z. Cada um dos elementos desse indexador consiste em uma variável e o valor de cada uma delas é apresentado no interior das pequenas caixas. X tem o valor 5; Y, 6; e Z, 7. Assim, X + Y – Z equivale a 5 + 6 – 7, isto é, 4. A(4) é o quarto elemento na matriz e seu valor é 20. Desse modo, a instrução PRINT A(X + Y – Z) apresentará o número 20 na tela.

então A(X) terá o conteúdo de A(1), que é 4. Neste caso, A(X + 1) corresponderá ao conteúdo de A(2), que é 9, e assim por diante.

Examine o programa e veja se pode perceber o que exatamente está ocorrendo. A linha 20 atribui à variável N o total de números que desejamos ordenar. Suponha que decidimos ordenar cinco números: quando o programa for processado, digitaremos o número 5 e teclaremos RETURN.

A linha 30 apresenta a instrução DIMensão. Se a variável N for 5, ela estabelecerá uma matriz constituída de cinco números. Esta linha equivale a DIM A(5).

As linhas de 40 a 60 consistem em um loop que possibilita a digitação dos cinco números. A maioria das versões de BASIC alerta o usuário apresentando um ponto de interrogação na tela. A tecla RETURN deverá ser pressionada após o fornecimento de cada número. Os números podem ter mais de um dígito, bem como incluir frações decimais.

A linha 90 zera a variável S. Esta variável está sendo usada como "flag" (sinalizador). Mais adiante no programa, a variável S é testada a fim de verificar se corresponde a 1 ou não. Só corresponderá se dois números tiverem sido substituídos um pelo outro, como veremos na sub-rotina de substituição. Vamos examinar em maior detalhe o uso de "flags", mais adiante, no curso.

A linha 100 estabelece os limites de um loop; neste caso, de 1 a 4 (pois N corresponde a 5; assim, N − 1 equivale a 4). Na primeira passagem pelo loop, a variável L assume o valor 1; assim, A(L) na linha 110 corresponderá a A(1), ou seja, o primeiro elemento da matriz, e A(L + 1) corresponderá a A(2), o segundo elemento da matriz. Na próxima passagem pelo loop, o valor da variável L será aumentado para 2; assim, A(L) equivalerá a A(2) e A(L + 1) a A(3). A linha 110 verifica se A(L) é maior que o número imediatamente à sua direita na tabela. O sinal de "maior do que" é >.

Se o primeiro número for maior que o segundo, o programa desviará para uma sub-rotina que permuta números. Se o primeiro número não for maior que o seguinte, não há desvio para a sub-rotina e o programa simplesmente segue para a próxima linha, que é a instrução NEXT L. Após ter sido repetido quatro vezes, o loop se encerra e o programa se dirige à linha 130, que examina a "flag de substituição", S, a fim de verificar se possui ou não o valor 1. Se tiver o valor 1 (colocado na sub-rotina de "substituição"), o programa desvia, voltando à linha 90, para repetir o processo de comparação. Se a "flag" S não corresponder a 1, não ocorreu substituição, o que significa que todos os números estão ordenados. O resto do programa apenas realiza a impressão dos números na tela.

A sub-rotina da substituição exige uma variável que armazene temporariamente um dos números que será substituído. Após os dois números terem sido permutados, nas linhas 210, 220 e 230, a "flag de substituição" S assume o valor 1, após o que o programa retorna ao programa principal.

```
 10 PRINT "QUANTOS NUMEROS DESEJA
    ORDENAR?"
 20 INPUT N
 30 DIM A(N)
 40 FOR X = 1 TO N
 45 PRINT "PROXIMO NUMERO"
 50 INPUT A(X)
 60 NEXT X
 70 REM
 80 REM ROTINA ORDENACAO (SORT)
 90 LET S = 0
100 FOR L = 1 TO N - 1
110 IF A(L) > A(L + 1) THEN GOSUB 200
120 NEXT L
130 IF S = 1 THEN 90
140 FOR X = 1 TO N
150 PRINT "A("; X;") = ";A(X)
160 NEXT X
170 END
180 REM
190 REM
200 REM SUB-ROTINA SUBSTITUICAO
210 LET T = A(L)
220 LET A(L) = A(L + 1)
230 LET A(L + 1) = T
240 LET S = 1
250 RETURN
```

A propósito...

IF... THEN

Se este programa for processado nos computadores compatíveis com o Sinclair (TK85, CP 200), a linha 130 deverá ser alterada para: 130 IF S = 1 THEN GOTO 90.

END

Esta instrução não existe no TK82 ou CP 200. A linha 170 deve ser alterada para 170 GOTO 260, acrescentando-se a linha 260 REM FIM DO PROGRAMA.

Exercícios

■ Ampliar o programa a fim de calcular o valor médio dos números fornecidos. A média é obtida pela soma dos itens dividida pelo número total desses itens. O modo mais simples de fazê-lo consiste em colocar uma instrução GOSUB antes da instrução END, na linha 170. A sub-rotina deverá ler cada um dos elementos incluídos na tabela e somará os valores em uma variável de "soma". Após a adição de todos os elementos, o resultado deverá ser dividido pelo número total desses elementos. A soma é facilmente obtida pelo uso do número de elementos como limite máximo de um loop FOR-NEXT.

■ Altere uma linha no programa, de modo que os números sejam dispostos em ordem decrescente.

■ Este exercício é dirigido em especial a quem possui computadores que não admitem o uso de variáveis como indexadores. Refaça o programa de modo que a instrução INPUT possa prever uma quantidade determinada de números a serem fornecidos, digamos, 12. Isto eliminará a necessidade de recorrer a uma variável como indexador. As linhas 100 e 110 deverão ser alteradas, assim como a sub-rotina de substituição.

■ Um exercício difícil: nosso método para ordenação de números não é o único possível. Experimente elaborar outro procedimento.

Organize seus dados

Veja como um programa põe logo em ordem suas informações.

Este capítulo do seu curso mostra como programas relativamente complexos são subdivididos em subprogramas ou sub-rotinas simples, que podem ser desenvolvidos e testados separadamente.

Além da vantagem de serem testadas separadamente, as sub-rotinas possibilitam que a elaboração de programas siga um desenvolvimento lógico. Há muitos modos de desenvolver programas em linguagem BASIC. Um dos mais comuns é conhecido como "tentativa e erro": você se senta diante do computador e começa a fornecer instruções em linguagem BASIC, sem refletir previamente no que seria necessário para o programa funcionar. Esse procedimento resulta em programas mal estruturados, que não podem dar bons resultados na primeira tentativa. Se a estrutura do programa não estiver clara, não será fácil detectar os erros ou falhas.

Um método muito melhor consiste, antes de tudo, em elaborar numa folha de papel a estrutura do programa, em etapas de detalhamento cada vez maior, até chegar ao desenvolvimento de um programa correto e eficaz. Os diagramas de bloco também poderão ser muito úteis (ver p. 104). Vejamos como isso é realizado.

Problema: desenvolver um programa que deverá receber uma série de nomes seguidos pelo sobrenome. Você deverá inverter a ordem de cada nome, de modo que o sobrenome venha em primeiro lugar, seguido de uma vírgula, de um espaço e do nome. Após isto, deverá ordenar os nomes em ordem alfabética (pelo sobrenome) e imprimi-los.

Por exemplo, se os nomes JOSÉ MARTINS e LUÍS LOPES forem fornecidos (nessa ordem), o programa imprimirá:

LOPES, LUÍS
MARTINS, JOSÉ

Antes mesmo de tentar desenvolver o programa que vai executar essas tarefas, anote os procedimentos de entrada e saída necessários, em termos genéricos:

Estágio 1
Entrada dos nomes em ordem aleatória, nome em primeiro lugar
Saída dos nomes em ordem alfabética, sobrenome em primeiro lugar

Esse procedimento mostra claramente o que desejamos executar. É o primeiro estágio, essencial para obtermos um programa bem elaborado. O passo seguinte consiste em aprimorar os procedimentos do primeiro estágio, mantendo-se sempre atento à eficácia do programa. Não se aprofunde em detalhes, apenas especifique um pouco mais os procedimentos necessários:

Estágio 2
Determinar o total de nomes a ser fornecido
Fornecer os nomes
Inverter os nomes
Ordenar os nomes
Imprimir os nomes

Observe a lista acima e verifique se é adequada. Percebe algum erro nela? Há algum problema de lógica? Se não há, prossiga para o estágio seguinte de detalhamento.

Os procedimentos a que chegamos no Estágio 2 são simples e distintos e podem ser desenvolvidos separadamente, como pequenos subprogramas. Em linguagem BASIC, subprogramas são denominados sub-rotinas. Vamos dar nomes às sub-rotinas para ficar fácil a identificação. A sub-rotina 1, para determinar o número de nomes que será fornecido, pode ser designada por NUMNOM. A sub-rotina 2, onde serão solicitados os nomes, pode ser chamada DADOS. A sub-rotina 3, para invertê-los, pode ser denominada INVERTER. A sub-rotina 4, para ordená-los, chamaremos ORDENAR. Finalmente, a sub-rotina 5, para imprimi-los, será IMPRINOMES.

Estágio 3.1 NUMNOM
Solicitar ao usuário a entrada do número desejado
Obter o número N solicitado
Usar o número N para montar a tabela alfanumérica
Estágio 3.2 DADOS
Se o total de nomes for menor que o número N, sugerir ao usuário o fornecimento de outro nome
Acrescentar o nome à tabela
Estágio 3.3 INVERTER
Calcular a extensão do nome
Localizar o "espaço" no conjunto de caracteres
Colocar os caracteres até o "espaço", em uma variável alfanumérica auxiliar
Colocar os caracteres do "espaço" até o final em outra variável auxiliar
Acrescentar vírgula e espaço para completar a variável
Colocar a segunda variável auxiliar, seguida da primeira, na série original
Estágio 3.4 ORDENAR
Comparar o primeiro componente da tabela com o seguinte
Se o primeiro componente for maior que o seguinte (i.e., posterior na seqüência alfabética), substituir um pelo outro
Comparar o segundo componente com o terceiro
Permutá-los se necessário
Repetir o procedimento até que todos os pares tenham sido comparados

Retornar ao início da tabela e repetir o procedimento de comparação dos pares, até que nenhuma permuta seja necessária, por toda a tabela.

Nota: essa rotina de ordenação é exatamente idêntica à usada na unidade anterior do curso de programação em linguagem BASIC. A seção da "permuta" será usada como uma sub-rotina chamada do interior da sub-rotina ORDENAR.

Estágio 3.5 IMPRINOMES
Imprimir cada nome da tabela até que todos os componentes tenham sido impressos

Todos os estágios necessários à elaboração desse programa estão agora formulados em um nível conveniente de detalhamento. A rotina ORDENAR foi apenas esboçada, pois a estudamos em detalhe anteriormente. A rotina PERMUTA, que é chamada do interior desta sub-rotina, foi deixada completamente de lado. Examinemos agora a facilidade com que programas em linguagem comum podem ser convertidos em linguagem BASIC.

Estágio 4
1. NUMNOM

As três linhas do estágio 3.1 traduzem-se facilmente em instruções em linguagem BASIC. O usuário recebe uma sugestão, através da instrução PRINT; o número é fornecido pelo uso de uma instrução INPUT e uma instrução DIM determina a extensão da tabela.

```
PRINT "QUANTOS NOMES DESEJA FORNECER?"
INPUT N
DIM A$(N)
RETURN
```

A variável N contém agora o número máximo de nomes a serem fornecidos. A instrução DIM dimensiona a tabela de variáveis alfanuméricas. Os nomes dessas variáveis sempre terminam com o sinal de "cifrão". A variável A$ sozinha pode conter apenas um conjunto de caracteres. A instrução DIM A$(N) cria uma tabela que pode conter um número "N" de séries. Tratamos de variáveis indexadas anteriormente, no curso.

A instrução RETURN transfere o controle de volta para o programa principal, na linha seguinte à chamada da sub-rotina. Os valores atribuídos a variáveis na sub-rotina serão "transportados de volta" ao programa principal e poderão ser retomados em alguma outra parte do programa e mesmo em outras sub-rotinas.

2. DADOS

Se o total de nomes for menor que o número N, o programa deverá indicar ao usuário a necessidade de fornecer um nome que deve ser acrescentado à tabela. Isso pode ser feito através de um loop FOR-NEXT. Sabemos que o primeiro nome na tabela será seu primeiro elemento e que o último será o enésimo, deste modo:

```
FOR X = 1 TO N
PRINT "FORNECER O NOME"
INPUT A$(X)
NEXT X
RETURN
```

Isto deve bastar para o fornecimento de todos os nomes para a tabela. Leitores atentos terão percebido o que vai acontecer quando precisarmos inverter a ordem do primeiro e do último nome na sub-rotina INVERTER. Cada componente (nome) da tabela terá de ser obtido novamente, em seguida invertido e recolocado na tabela. Para não aumentar a extensão do programa com esse procedimento, seria mais simples chamar a sub-rotina INVERTER, no interior da sub-rotina DADOS, após a digitação de cada nome. O nome poderá então ser invertido antes de ser colocado na tabela. Para obter esse resultado, temos apenas de acrescentar uma linha, assim:

```
FOR X = 1 TO N
PRINT "FORNECER O NOME"
INPUT A$(X)
GOSUB [INVERTER]
NEXT X
RETURN
```

Todos os nomes da tabela se apresentarão agora em ordem inversa (sobrenome seguido do nome), prontos para ser ordenados.

3. INVERTER

Para inverter a ordem dos nomes, necessitamos saber onde se localiza o "espaço" que separa o primeiro nome do sobrenome. Sabendo sua localização, poderemos utilizar várias funções a fim de obter partes da variável e colocá-las em outras áreas. As funções em linguagem BASIC são comandos que executam operações predeterminadas sobre o valor apresentado após o nome da função. Este valor é sempre apresentado entre parênteses. Muitas funções vêm incorporadas, mas você também pode definir suas próprias funções. Um exemplo caracterís-

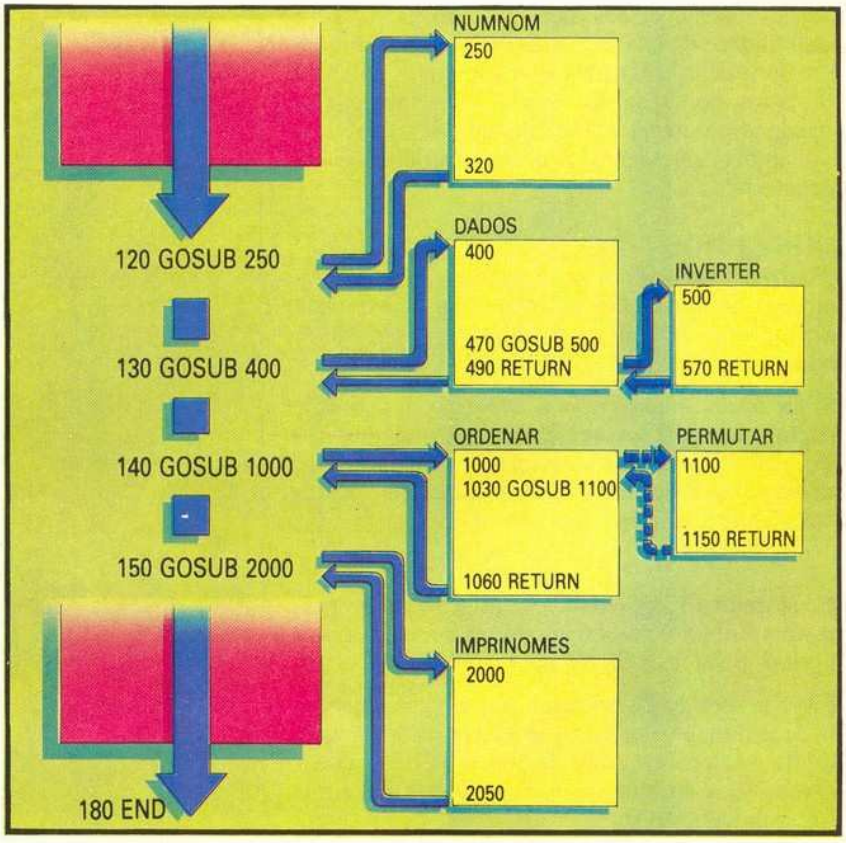
Programas incluídos em programas

Desta vez, o programa principal é muito pequeno. Todo trabalho é, na realidade, realizado pelos subprogramas (chamados sub-rotinas, em BASIC). Cada um dos procedimentos necessários para fazer o programa funcionar está separado e desenvolvido como "miniprogramas", unidos entre si pelo programa principal.

Quando é processado, toda vez que chega a uma instrução GOSUB o programa desvia para o número de linha da sub-rotina especificada e essa seção do programa é, então, executada. O final da sub-rotina é indicado pela instrução RETURN. Ao alcançá-la, o programa retorna à linha imediatamente seguinte à instrução GOSUB que chamou a sub-rotina.

As sub-rotinas podem ser incluídas em outras sub-rotinas. A sub-rotina DADOS chama uma outra sub-rotina, chamada INVERTER, e a rotina ORDENAR algumas vezes chama uma outra, denominada PERMUTAR.

Subdividir um problema em sub-rotinas separadas e unidas por um programa principal torna mais fácil o desenvolvimento e teste dos programas.



tico de função "incorporada" é a função SQR ( ). Esta função calcula a raiz quadrada do valor apresentado entre parênteses. Assim: na instrução LET A = SQR (9), PRINT A imprimirá o número 3.

A sub-rotina INVERTER utiliza a função LEN (para calcular a extensão da variável), a função INSTR (para localizar a posição do espaço), a função LEFT$ (para obter um número determinado de caracteres à esquerda) e a função RIGHT$ (para obter um número determinado de caracteres à direita). Por enquanto, não discutiremos de modo detalhado como exatamente essas funções funcionam. Faremos um exame mais minucioso das funções em linguagem BASIC na próxima parte do curso.

4. ORDENAR

A sub-rotina ORDENAR e a sub-rotina PERMUTAR, nela incluída, são muito semelhantes às rotinas usadas na unidade anterior do curso.

5. IMPRINOMES

Este procedimento é o seguinte:

```
FOR Q = 1 TO N
PRINT A$(Q)
NEXT Q
RETURN
```

Agora, resta apenas desenvolver o programa principal, que é bastante simples:

```
REM PROGRAMA PRINCIPAL
GOSUB [NUMNOM]
GOSUB [DADOS]
GOSUB [ORDENAR]
GOSUB [IMPRINOMES]
END
```

Colocamos os "nomes" das sub-rotinas entre colchetes. Poucas versões da linguagem BASIC têm a possibilidade de chamar as sub-rotinas por nomes. A maioria deles tem de recorrer a números de linha, colocados no lugar dos nomes das sub-rotinas, quando o programa é efetivamente desenvolvido. São também acrescentadas instruções REM e PRINT adequadas.

Exercícios

Agora que examinamos quase todos os aspectos importantes da linguagem BASIC, é hora de verificar seu progresso, fazendo os exercícios apresentados a seguir.

■ **Variáveis.** Alguns nomes de variáveis, abaixo, podem armazenar valores numéricos e alguns não são válidos para uso como nomes de variáveis. Faça um círculo ao redor das variáveis numéricas válidas e um X sobre os nomes das não válidas.

A B6 2Z D$ 15 X$ A12 D9 Q81 Q5 6F H$

■ **Aritmética 1.** Desenvolver um pequeno programa para atribuir o valor 6 à variável B e em seguida imprimir o valor de B.

■ **Aritmética 2.** Escrever um pequeno programa a fim de atribuir o valor 5 à variável A; o valor 7 à variável B e o valor 9 à variável C. Somar os valores dessas variáveis e atribuir a soma à variável D. Imprimir o valor da variável D.

```
10 REM ESTE PROGRAMA COLOCA NOMES
20 REM EM ORDEM ALFABETICA
30 PRINT "PRIMEIRAMENTE DETERMINE QUANTOS"
40 PRINT "NOMES VOCE DESEJA FORNECER"
50 PRINT "ENTAO FORNECA OS NOMES"
60 PRINT "NOME(ESPACO)SOBRENOME"
70 PRINT "ORDENAR"
80 REM
90 REM ESTE E O PROGRAMA PRINCIPAL
100 PRINT
110 PRINT
120 GOSUB 250
130 GOSUB 400
140 GOSUB 1000
150 GOSUB 2000
160 REM
170 REM FIM DO PROGRAMA PRINCIPAL
180 END
250 REM SUB-ROTINA PARA DETERMINAR QUANTOS
260 REM NOMES SERAO FORNECIDOS
270 PRINT "QUANTOS NOMES VOCE"
280 PRINT "DESEJA FORNECER?"
290 PRINT
300 INPUT N
310 DIM A$ (N)
320 RETURN
400 REM SUB-ROTINA PARA FORNECER NOMES
410 PRINT "FORNECER O NOME, DESTA FORMA:"
420 PRINT "NOME(ESPACO)SOBRENOME(CR)"
430 PRINT "EX. RAUL AGUIAR"
440 FOR X = 1 TO N
450 PRINT "FORNECER NOME"
460 INPUT A$ (X)
470 GOSUB 500
480 NEXT X
490 RETURN
500 REM SUB-ROTINA PARA INVERTER A ORDEM DOS NOMES
510 LET L = LEN(A$(X))
520 LET S = INSTR(A$(X)," ")
530 LET C$ = LEFT$(A$(X),S - 1)
540 LET F$ = RIGHT$(A$(X),L - S)
550 LET F$ = F$ +" , "
560 LET A$(X) = F$ + C$
570 RETURN
1000 REM SUB-ROTINA ORDENAR
1010 LET S = 0
1020 FOR P = 1 TO N - 1
1030 IF A$(P) > A$(P + 1) THEN GOSUB 1100
1040 NEXT P
1050 IF S = 1 THEN GOTO 1000
1060 RETURN
1100 REM SUB-ROTINA PERMUTAR
1110 LET T$ = A$(P)
1120 LET A$(P) = A$(P + 1)
1130 LET A$(P + 1) = T$
1140 LET S = 1
1150 RETURN
2000 REM SUB-ROTINA IMPRIMIR
2010 PRINT
2020 FOR Q = 1 TO N
2030 PRINT A$(Q)
2040 NEXT Q
2050 RETURN
```

■ **Aritmética 3.** Observe as seguintes linhas em linguagem BASIC e calcule o valor de C.

```
LET C = 5 + 4 * 3
PRINT C
```

■ **Aritmética 4.** Qual será o resultado impresso pelo seguinte programa?

```
LET A = 3
LET B = 2
LET C = 9
LET D = 4
LET E = (A + B) * (C - D)
PRINT E
LET E = 5
LET E = E * E
PRINT E
```

■ **Comparações 1.** Qual será o valor de X necessário para que a mensagem PRINT seja impressa?

```
70 LET A = 5
80 LET B = X
90 LET R = B - A
100 IF R = 0 THEN GOTO 120
110 GOTO 10
120 PRINT "PARABENS! VOCE ACERTOU"
999 END
```

■ **Comparações 2.** Qual o menor valor de X que fará o programa desviar para a linha 300?

```
250 IF X > 6 * 100 THEN GOTO 300
```

■ **Comparações 3.** Qual o menor valor de Z que fará o programa desviar para a mensagem "Parabéns"?

```
340 IF Z < 10000 THEN GOTO 500
350 IF Z > = 10000 THEN GOTO 520
:
:
500 PRINT "SEU NUMERO ESTA MUITO BAIXO.
TENTE NOVAMENTE"
510 GOTO 600
520 PRINT "PARABENS. VOCE AGORA E UM
MESTRE"
530 GOTO 700
```

■ **Print 1.** Suponha que o valor da variável T seja 50. Escreva uma instrução PRINT que imprima "O VALOR DE T E 50". Sugestão: coloque a "mensagem" entre aspas, use um sinal de ponto e vírgula e o nome da variável.

■ **Print 2.** Examine o pequeno programa abaixo e complete a instrução PRINT, para que se imprima uma mensagem de avaliação como esta:

SINTO, MAS SUA CLASSIFICACAO DE 175 FOI MUITO BAIXA

Complete a linha de modo que o valor real da avaliação possa variar em cada vez.

```
610 REM A VARIAVEL S E A CLASSIFICACAO
ATE AGORA
620 IF S < = 500 THEN GOTO 640
630 GOTO 700
640 PRINT "SINTO"
```

■ **Print 3.** Que mensagem será impressa quando o programa for processado?

```
200 LET A$ = "MICROCOMPUTADOR — CURSO
BASICO?"
210 LET B$ = "O QUE VOCE ACHA DO "
220 PRINT B$; A$
```

■ **Input 1.** A instrução INPUT é um modo de atribuição de valores a uma variável. Se o programa a seguir for processado, qual tecla deverá ser pressionada a fim de que o programa imprima a resposta "12"?

```
60 INPUT N
70 LET N = N * 2
80 PRINT N
```

■ **Input 2.** O que será impresso neste caso?

```
100 PRINT "POR FAVOR DIGITE SEU NOME"
110 INPUT N$
120 PRINT "OLA"; N$; " SOU SEU COMPUTADOR"
```

A propósito...

DIM

Os compatíveis Sinclair (TK82, TK83, TK85, CP 200) possuem esta instrução, mas sua utilização não segue o padrão; assim, elimine a linha 310 e a substitua por:

```
310 DIM A$(N, 30)
```

GOTO

Na linha 1050, o comando GOTO 1000 vem logo após a palavra THEN. Nesse caso, a maioria dos computadores admite a omissão da palavra GOTO; assim, a linha 1050 pode ser escrita da seguinte forma:

```
1050 IF S = 1 THEN 1000
```

INSTR

Os equipamentos das linhas Apple e TRS-80 (Micro Engenho, Unitron, Maxxi, CP 500, etc.) não possuem esta função. Assim, suprima a linha 520 e a substitua por estas cinco linhas:

```
515 FOR P = 1 TO L
520 CH$ = MID$(A$(X),P,1)
522 LET S = 0
523 IF CH$ =" " THEN LET S = P:
LET P = L
525 NEXT P
```

INSTR

Os equipamentos compatíveis com o Sinclair (TK82, TK83, TK85, CP 200, Ringo) não possuem esses comandos, mas você pode criar sua própria versão, substituindo a linha 460 e as linhas de 510 a 570 por:

LEFT$

RIGHT$

```
460 INPUT Z$
510 LET L = LEN(Z$)
520 FOR P = 1 TO L
530 LET S = 0
540 IF Z$(P ) <>" " THEN GOTO 570
550 LET S = P
560 GOTO 580
570 NEXT P
580 LET C$ = Z$(TO S-1)
590 LET F$ = Z$(S+1 TO)
600 LET A$(X) = F$ + "," + C$
610 RETURN
```

END

Os equipamentos da linha Sinclair (TKs, CP 200 e Ringo) não possuem este comando; assim, substitua-o por STOP.

Descubra as funções

O BASIC apresenta algumas funções incorporadas. Assim, boa parte do trabalho de programação já está pronta. Conhecer o uso dessas funções facilita a elaboração de seus programas.

Vamos admitir que em um de seus programas você queira calcular a raiz quadrada de um número. Há vários métodos para fazer isso. O mais simples e menos satisfatório seria organizar uma tabela dos valores de raiz quadrada, a ser consultada sempre que necessário. Provavelmente, você aprendeu a efetuar esse cálculo na escola. Um método alternativo seria utilizar a "função" de raiz quadrada, existente na maioria das versões da linguagem BASIC. Aqui, a parte aritmética da operação é assumida pelo programa BASIC — o programador não precisa ocupar-se disso. Vejamos como funciona:

```
10 REM ESTE PROGRAMA CALCULA A RAIZ
   QUADRADA
20 REM DE UM NUMERO
30 PRINT "FORNECER O NUMERO QUE VOCE
   DESEJA"
40 PRINT "CALCULAR A RAIZ QUADRADA"
50 INPUT N
60 LET A = SQR(N)
70 PRINT "A RAIZ QUADRADA DE ";N;" E ";A
80 END
```

Digite esse pequeno programa e observe como ele de fato lhe fornece a raiz quadrada de qualquer número digitado. Examinemos as regras do método de utilização dessa "função" de raiz quadrada.

"Funções" em linguagem BASIC são geralmente palavras de comando (neste caso, SQR) seguidas por parênteses que encerram a expressão a ser manipulada. Nesse programa, a variável N representa o número fornecido através do teclado, do qual queremos extrair a raiz quadrada. A linha 60 significa "atribuir a raiz quadrada de N à variável A". A linha 70 imprime o valor de A.

A expressão entre parênteses chama-se "argumento" da função e não é necessariamente uma variável: também é possível utilizar números. Digite o seguinte programa e observe o que acontece quando processado:

```
10 PRINT SQR(25)
20 END
```

Você perceberá que este programa funciona tão bem quanto o primeiro. De modo análogo, podemos utilizar argumentos mais complexos entre os parênteses: Experimente este:

```
10 LET A = 10
20 LET B = 90
30 LET C = SQR(A+B)
40 PRINT C
50 END
```

Esse pequeno programa pode ser reduzido, pela combinação das linhas 30 e 40, assim:

```
10 LET A = 10
20 LET B = 90
30 PRINT SQR(A+B)
40 END
```

As funções devem ser entendidas como pequenos programas incorporados ao BASIC, que podem ser utilizados pelo programador a qualquer momento. A maioria das versões da linguagem BASIC possui grande quantidade de funções e ainda oferece ao programador a possibilidade de definir outras que poderão ser utilizadas em qualquer programa. Posteriormente, veremos como isso é realizado. Aqui, vamos examinar algumas das funções mais comuns. Há dois tipos: funções numéricas, nas quais o argumento (a parte entre parênteses) consiste em um número, variável numérica ou expressão numérica, e funções alfanuméricas, nas quais o argumento é uma série de caracteres, ou expressão constituída de séries de caracteres. Primeiramente, examinemos algumas das funções numéricas.

Anteriormente, na página 77, apresentamos um programa que calculava o total de azulejos necessários para o revestimento de uma sala. Um pequeno inconveniente nesse programa era que a resposta poderia resultar em frações de azulejos. O programa poderia ter como resultado o número 988,24, por exemplo. Em casos como esse, você necessita arredondar a resposta para o número inteiro mais próximo, e esse arredondamento é uma das funções encontradas na linguagem BASIC. Eis como se processa:

```
10 PRINT "FORNECER UM NUMERO QUE
   CONTENHA UMA FRACAO DECIMAL"
20 INPUT N
30 PRINT "A PARTE INTEIRA DO NUMERO E";
40 PRINT INT(N)
50 END
```

Se esse programa for processado e o número que você fornecer for 3,14, o programa apresentará na tela:

```
A PARTE INTEIRA DO NUMERO E 3
```

Naturalmente, lidando com azulejos, é preciso acrescentar 1 à resposta, para garantir a compra de maior quantidade do que o total necessário.

Podemos também querer encontrar o "sinal" de um número — determinar se é negativo, zero ou positivo. Para fazê-lo, a maioria das versões da linguagem BASIC tem incorporada a função SGN. Experimente este programa:

```
10 PRINT "FORNECER UM NUMERO"
20 INPUT N
30 LET S = SGN(N)
```

```
40 IF S = -1 THEN GOTO 100
50 IF S = 0 THEN GOTO 120
60 IF S = 1 THEN GOTO 140
100 PRINT "O NUMERO E NEGATIVO"
110 GOTO 999
120 PRINT "O NUMERO E ZERO"
130 GOTO 999
140 PRINT "O NUMERO E POSITIVO"
150 GOTO 999
999 END
```

Observando os valores para a variável S, apresentados na função SGN, na linha 30 (testados nas linhas 40, 50 e 60), você verifica que há três valores possíveis. Se o argumento entre parênteses for um número negativo, o número apresentado será −1; se for 0, o número apresentado será 0, e se for positivo, será 1. O uso da função SGN na linha 30 economiza várias linhas de programação. Poderíamos ter escrito da seguinte forma:

```
IF N < 0 THEN LET S = -1
IF N = 0 THEN LET S = 0
IF N > 0 THEN LET S = 1
```

É sempre possível obter as mesmas operações realizadas pelas funções BASIC através da programação normal; o recurso às funções apenas economiza tempo, espaço e trabalho.

Eis aqui mais algumas funções numéricas. A função ABS apresenta o valor "absoluto" de um número, isto é, seu valor real, mediante a eliminação do sinal. Deste modo, o valor absoluto de −6 é 6. Experimente:

```
10 LET X = -9
20 LET Y = ABS(X)
30 PRINT Y
40 END
```

A função MAX calcula, entre dois números, aquele que tem valor maior. Assim:

```
10 LET X = 9
20 LET Y = 7
30 LET Z = X MAX Y
40 PRINT Z
50 END
```

A função MIN é análoga à MAX, porém calcula entre dois números aquele que tem valor menor. Experimente:

```
10 PRINT "FORNECER UM NUMERO"
20 INPUT X
30 PRINT "FORNECER OUTRO NUMERO"
40 INPUT Y
50 LET Z = X MIN Y
60 PRINT Z
70 END
```

Observe que estas duas últimas funções possuem dois argumentos em vez de um, que não necessitam ser colocados entre parênteses. Estas funções provavelmente ainda não existem no seu microcomputador. Consulte o manual de instruções BASIC para certificar-se. A maioria das versões de BASIC tem também várias outras funções numéricas, inclusive a função LOG, que calcula o logaritmo do número fornecido; TAN calcula a tangente; COS calcula o coseno; e SIN calcula o seno. Mais adiante, veremos como algumas dessas funções "trigonométricas" podem ser usadas.

Várias das funções incorporadas na linguagem BASIC operam com séries de caracteres. Utilizamos algumas em nosso programa para ordenar nomes (p. 135); porém, na ocasião, não vimos em detalhe como funcionavam, bem como algumas outras funções alfanuméricas.

Uma das funções alfanuméricas mais úteis é a LEN, que efetua a contagem do número de caracteres presentes em uma série colocada entre aspas ou, ainda, a contagem do número de caracteres atribuídos a uma variável alfanumérica. Experimente:

```
10 LET A$ = "COMPUTADOR"
20 LET N = LEN(A$)
30 PRINT "O NUMERO DE CARACTERES NA SERIE
   E ";N
40 END
```

Mas por que precisaríamos saber quantos caracteres estão incluídos em uma variável alfanumérica? Para entender por quê, digite o programa seguinte, elaborado para construir um "nome em triângulo". Ele vai imprimir a primeira letra da palavra, depois a primeira e a segunda, e assim sucessivamente, até que a palavra esteja impressa por inteiro.

```
5  REM IMPRIME UM "NOME EM TRIANGULO"
10 LET A$ = "MARIA"
20 FOR L = 1 TO 5
30 LET B$ = LEFT$(A$,L)
40 PRINT B$
50 NEXT L
60 END
```

Agora processe este programa. Você imagina qual será a disposição resultante? Será assim:

```
M
MA
MAR
MARI
MARIA
```

Este pequeno programa utiliza a função LEFT$, que extrai os caracteres de uma determinada série. Esta função possui dois argumentos. O primeiro especifica o conjunto de caracteres e o segundo (que vem após a vírgula) especifica o número de caracteres a ser extraído da série, iniciando pela esquerda. A série "MARIA" foi atribuída à variável A$; assim, LEFT$(A$,1) deve apresentar a letra M. A instrução LEFT$(A$,2) deverá representar as letras MA. O pequeno programa acima utiliza um índice, L, que varia de 1 a 5, de modo que o segundo argumento na função LEFT$ varia de 1 a 5, a cada passagem pelo loop. Sabíamos exatamente quantos caracteres havia na palavra que queríamos imprimir (MARIA), e foi fácil prever que o limite máximo do loop FOR-NEXT seria 5. Mas o que faríamos se não soubéssemos o número de caracteres que estariam incluídos no loop?

É aqui que a função LEN entra em cena. Ela toma uma série (entre aspas) ou uma variável alfanumérica como seu argumento. Eis alguns exemplos de como ela procede:

```
10 REM PROGRAMA PARA TESTAR A FUNCAO
   "LEN"
20 PRINT LEN("COMPUTADOR")
30 END
```

Ao ser processado, o programa deve imprimir o nú-

mero 10. Ele executa a contagem do número de caracteres que constituem a palavra COMPUTADOR e apresenta esse valor. Vamos fazer o mesmo, de forma ligeiramente diferente:

```
10 REM CALCULO DA EXTENSAO DE UMA SERIE
20 LET A$ = "CURSO DE COMPUTACAO"
30 LET L = LEN(A$)
40 PRINT L
50 END
```

Processado o programa, o computador deve imprimir o número 19 na tela. Há dezenove caracteres nesta série, não dezessete; não esqueça que para o computador os espaços entre palavras também são considerados caracteres. Agora utilizamos a função LEN, alterando nosso programa inicial, para impressão do "nome em triângulo"

```
10 REM ESTE PROGRAMA IMPRIME UM "NOME
   EM TRIANGULO"
20 PRINT "DIGITE O NOME"
30 INPUT A$
40 LET N = LEN(A$)
50 FOR L = 1 TO N
60 LET B$ = LEFT$(A$,L)
70 PRINT B$
80 NEXT L
90 END
```

Toda vez que este loop se realiza, o valor da variável L aumenta de 1 até N (que representa a extensão da série). Se você fornecer o nome ALFREDO, a linha 40 equivalerá a LET N = LEN ("ALFREDO"); assim, N assumirá o valor 7. Na primeira passagem pelo loop, a linha 50 atribuirá o valor 1 à variável L e a linha 60 corresponderá a LET B$ = LEFT$ ("ALFREDO",1); desse modo, um caractere da série será atribuído à variável B$, iniciando pela esquerda — e este será o caractere A.

Na segunda passagem pelo loop, a variável L assumirá o valor 2; assim, a linha 60 se tornará equivalente a LET B$ = LEFT$ ("ALFREDO",2), o que fará com que os dois primeiros caracteres da série sejam atribuídos à variável B$. Ela passará a ter como conteúdo os caracteres AL.

A função LEN encontra sete caracteres na série ALFREDO e atribui este valor à variável N; desse modo, na última passagem pelo loop, os sete caracteres da série serão atribuídos à variável B$ e toda a série será impressa.

Observe que a função LEFT$ tem uma correspondente, a função RIGHT$, que toma os caracteres da variável alfanumérica, da direita para a esquerda, exatamente do mesmo modo.

Finalmente, examinaremos uma última função alfanumérica, também utilizada no programa para ordenação de nomes. É a função INSTR, utilizada para localizar a posição de uma série específica (chamada "subsérie") na primeira vez em que ela ocorre no interior de uma série. No programa para ordenação de nomes, a função INSTR foi utilizada para localizar a posição do espaço entre o nome e o sobrenome. Vejamos como ela funciona:

```
10 LET A$ = "CORREDOR"
20 LET P = INSTR(A$,"DOR")
30 PRINT P
40 END
```

Antes de fornecer e processar o programa, verifique se pode prever o valor que será impresso para a variável P. Lembre-se de que a função INSTR localiza a posição de início da "subsérie" dentro da série, na primeira vez em que ela ocorre. Se a série for CORREDOR, a posição inicial da subsérie DOR será 6 — o caractere D da série DOR é a sexta letra em CORREDOR. Algumas versões da linguagem BASIC não utilizam a função INSTR, mas possuem, em seu lugar, uma função semelhante, chamada INDEX. Eis como utilizar a função INSTR (ou INDEX) para localizar um espaço no interior de uma série:

```
10 REM CALCULAR A POSICAO DE UM ESPACO
   EM UMA SERIE
20 LET A$ = "PROGRAMACAO BASIC"
30 LET P = INSTR(A$, " ")
40 PRINT P
50 END
```

Observe que o segundo argumento na função INSTR (linha 30) é " ". As aspas contêm um espaço — o caractere a ser encontrado. O programa imprimirá o número 12 como valor de P, uma vez que o espaço é a décima segunda posição na série. Calcule o que será impresso se a linha 30 for alterada para:

```
LET P = INSTR(A$,"B")
```

Finalmente, uma função útil usada com a instrução PRINT. Veja o que acontece quando você processa o seguinte programa:

```
10 PRINT "ESTA LINHA NAO E DESEJADA"
20 PRINT TAB(5); "ESTA LINHA E DESEJADA"
30 END
```

Você percebe o que aconteceu? A segunda linha foi impressa a cinco espaços da margem esquerda. A função TAB procede de modo análogo ao tabulador de uma máquina de escrever. Eis aqui outro pequeno programa que utiliza a função TAB:

```
10 REM UTILIZACAO DA FUNCAO TAB
20 PRINT "FORNECER O VALOR TAB"
30 INPUT T
40 LET W$ = "TABULACAO"
50 PRINT TAB(T);W$
60 END
```

Agora você pode retornar ao programa de ordenação de nomes, na página 136, e veja como algumas destas funções foram nele usadas.

Exercícios

■ **Loop 1.** O que será impresso, quando este programa for processado?

```
10 LET A = 500
20 FOR L = 1 TO 50
30 LET A = A - 1
40 NEXT L
50 PRINT "O VALOR DE A E ";A
```

■ **Loop 2.** O que aparecerá na tela, quando este programa for processado?

```
10 REM
20 REM ESTE E UM LOOP DE CRONOMETRAGEM
30 REM VERIFIQUE SUA DURACAO
40 REM
```

A propósito...

Nos computadores compatíveis com o Sinclair (TKs, CP 200, Ringo), substituir o TAB (número) por TAB número.

Os computadores compatíveis com o Sinclair não possuem quaisquer destes três comandos, mas você pode elaborar suas próprias versões deles com o uso de "Slicing" ou "partição de variáveis alfanuméricas". Isto pode ser feito com a seguinte equivalência de instruções:

```
LEFT$(X$,N) equivale a X$(TO N)
RIGHT$(X$,N) equivale a
X$(LEN(X$) - N + 1 TO)
MID$(X$,P,N) equivale a
X$(P TO P+N-1)
```

Os computadores compatíveis com Sinclair Apple e TRS80 (TKs, CP 200, CP 500, Unitron, Micro Engenho etc.) não possuem esta função, mas você pode desenvolver um programa substitutivo. Supondo que a linha original seja:

```
20 LET P = INTR(A$, "DOR")
```

Substitua-a por:

```
20 LET X$ = A$ : LET Z$ = "DOR":
     GOSUB 9930: P = U
```

e acrescente as seguintes linhas:

```
9929 STOP
9930 LET U = 0: LET X = LEN(X$):
     LET V = LEN(Z$)
9940 FOR W = 1 TO L-V+1:
     IF MID$ (X$,W,V) = Z$THEN LET
     U = W
9950 IF U <> 0 THEN LET W = L-V+1
9960 NEXT W: RETURN
```

Nos computadores compatíveis com o Sinclair, substituir a instrução MID$ (X$,W,V), na linha 9940 e seguir as instruções fornecidas acima.

```
50 PRINT "INICIO"
60 FOR X = 1 TO 5000
70 NEXT X
80 PRINT "FIM"
90 END
```

■ **Loop 3.** Que resultado será impresso se este programa for processado e você digitar o número 60, quando for solicitado?

```
10 PRINT "PENSE EM UM NUMERO E DIGITE-O"
20 INPUT N
30 LET A = 100
40 FOR L = 1 TO N
50 LET A = A + 1
60 NEXT L
70 PRINT "O VALOR DE A AGORA E ";A
80 END
```

■ **Loop 4.** O que acontecerá quando este programa for processado?

```
10 PRINT "GOSTO DE BASIC"
20 GOTO 10
30 END
```

■ **Loop 5.** O que aparecerá na tela quando este programa for processado?

```
10 FOR Q = 1 TO 15
20 PRINT "CHEGA DE LOOPS"
30 NEXT Q
40 END
```

■ **Read-Data 1.** Que resultado será impresso?

```
10 READ X
20 READ Y
30 READ Z
40 PRINT "ESTAMOS TESTANDO A INSTRUCAO
   READ"
50 DATA 50,100,20
60 PRINT X + Y + Z
```

■ **Read-Data 2.** O que será impresso na tela quando o seguinte programa for processado?

```
100 FOR L = 1 TO 10
110 READ X
120 PRINT "X = ";X
130 NEXT L
140 DATA 1,2,3,5,7,11,13,17,19,23
```

Respostas no próximo fascículo.

Respostas aos "Exercícios" das pp. 136-137.

Variáveis

(A) (B6) ~~(Z$)~~ DS ~~(X$)~~ XS (A12) (D9) (Q81) (Q5) ~~(H$)~~ HS

Aritmética 1

```
10 LET B = 6
20 PRINT B
```

Aritmética 2

```
10 LET A = 5
20 LET B = 7
30 LET C = 9
40 LET D = A + B + C
50 PRINT D
```

Aritmética 3

17

Aritmética 4

25
25

Comparações 1

5

Comparações 2

601 (considerados só os números inteiros)

Comparações 3

10000

Print 1

```
PRINT "O VALOR DE T E "; T
```

Print 2

```
640 PRINT "SINTO, MAS SUA CLASSIFICACAO DE ";S; "FOI
    MUITO BAIXA"
```

Print 3

```
O QUE VOCE ACHA DO MICROCOMPUTADOR—CURSO
BASICO?
```

Input 1

6

Input 2

```
POR FAVOR, DIGITE SEU NOME
OLA (SEU NOME) SOU SEU COMPUTADOR
```

Observe que as respostas às "Variáveis" serão diferentes nos equipamentos que só admitem um caractere alfabético (isto é, sem sufixos numéricos).

Tentando a sorte

Continuando o estudo das funções em BASIC, chegamos à função RND, que produz números aleatórios (ou quase aleatórios) para uso em jogos ou programas estatísticos.

Já vimos como várias das funções em linguagem BASIC operam, e agora vamos examinar uma das mais freqüentemente empregadas — a função RND, usada para gerar números aleatórios. É utilizada também em jogos, sempre que haja algum componente aleatório.

Infelizmente, RND é uma das "siglas" de uso mais variável em BASIC: nossa descrição dela poderá diferir do modo como essa função está adaptada ao seu microcomputador. Vamos esclarecer algumas das diferenças entre a linguagem BASIC usada no curso de programação BASIC e a que existe em seu computador.

A maioria de nossos programas segue a linguagem BASIC da Microsoft (ou MBASIC). A Microsoft é uma empresa dos Estados Unidos e sua linguagem BASIC foi uma das primeiras a se tornar amplamente acessível ao público. A linguagem BASIC não possui padrão oficial, mas sua versão Microsoft é a que está mais próxima do que se poderia chamar de um padrão: muitas outras marcas seguem o modelo da Microsoft, e essa empresa foi encarregada de produzir versões para diversos computadores mais populares.

A principal diferença entre o MBASIC e a maioria das versões mais recentes está em que os microcomputadores agora possuem alta capacidade para a elaboração de gráficos, não existente quando o MBASIC foi desenvolvido. As versões mais recentes de BASIC em geral incluem uma série de comandos e instruções para gráficos. Para obter o máximo rendimento do computador, é necessário explorar totalmente sua capacidade de elaboração de gráficos, o que exige um estudo cuidadoso do manual de instruções do equipamento.

Entre as várias versões de BASIC adaptadas a microcomputadores, a da Sinclair (usada no TK83, TK85, CP 200, Ringo) é provavelmente a que mais difere do MBASIC. A versão da Texas Instruments (usada no TI99/4A) tem igualmente diferenças significativas. Até onde for possível, indicamos como fazer determinadas modificações, nos quadros "A propósito...", que você deve observar, sempre que encontrar algum problema no processamento de programas.

Como dissemos, a função RND difere de uma versão para outra. Verifique no manual de instruções de seu equipamento como ela funciona. Aqui explicamos o modo de usá-la em um jogo de dados muito simples. Conforme programas anteriores, realizamos a maior parte do trabalho em sub-rotinas. Esta técnica tem a vantagem de tornar o programa mais legível, mais fácil de escrever e de corrigir.

O programa principal começa com a instrução RANDOMIZE, na linha 20. A maioria das versões de BASIC, mas não todas, precisa dessa instrução para restaurar a função RND. Na verdade, é muito difícil obter dos computadores a apresentação de números genuinamente aleatórios. Sem esta operação de restauração, a mesma seqüência de números supostamente aleatórios será produzida, toda vez, pela função RND. A linha 50, a seguir, chama uma sub-rotina que utiliza a função RND para atribuir um número aleatório à variável D. Escreveremos assim:

```
320 LET D = INT(10 * RND)
```

Esta é a linha que mais provavelmente exigirá alterações para que você possa processar o programa. Os detalhes de como operam diferentes versões da função RND estão no quadro "A propósito...". Assim, vejamos o que acontece no BASIC da Microsoft. A instrução RND utiliza uma expressão (entre parênteses, usual em funções) como alternativa para alterar levemente a seqüência dos números gerados. Sem essa expressão — por exemplo, LET A = RND —, o valor da variável A será um número entre 0 e 1. Não queremos um número menor que 1 e, assim, multiplicamos o número por 10. Isto é feito da seguinte forma: LET A = 10 * RND. Se, por exemplo, a função RND apresentar o valor 0,125455, o valor de A será, então, 1,25455.

Para eliminar a parte fracionária do número e reter apenas a parte inteira, usamos a função INT (inteiro) da seguinte forma: LET A = INT(10 * RND).

Para obter números inteiros aleatórios que variem entre 1 e 6 com o BASIC Microsoft, temos de verificar se os números apresentados foram maiores que 6 ou menores que 1, já que tais números não seriam apropriados para o jogo de dados. Isto é feito nas linhas 330 e 340:

```
330 IF D > 6 THEN GOTO 320
340 IF D < 1 THEN GOTO 320
```

Se a variável D estiver fora dos limites de 1 a 6, as instruções GOTO farão com que o programa retorne e proceda a nova tentativa.

Tendo escolhido um valor aleatório entre 1 e 6 para a variável D, a sub-rotina de lançamento de dados retorna ao programa principal. Este imprime a mensagem: SUA CONTAGEM DE PONTOS E, seguida da figura de um dado. Observe como o desenho correspondente é escolhido e feito na sub-rotina ESCOLHER. Por exemplo, se o dado (e, desse modo, a variável D) tiver o número 1, a linha 410 chamará a sub-rotina que se inicia na linha 530, assim:

```
410 IF D = 1 THEN GOSUB 530
```

Esta sub-rotina é simplesmente uma série de instruções PRINT, elaboradas para produzir gráficos simplificados na tela. Seu BASIC pode dispor de gráficos de tela muito melhores e, se este for o caso, será melhor substituir as nossas sub-rotinas pelas instruções apropriadas de gráficos.

Tendo escolhido aleatoriamente um dado para você, o programa repete o procedimento para selecionar e apresentar um dado para o computador. A parte do programa que decide quem venceu foi incorporada ao programa principal; poderia igualmente ter sido escrita como uma sub-rotina, mas isto dificilmente valeria a pena, já que consiste apenas em quatro linhas. A linha 200 compara a variável M (meu dado) com a variável C (o dado do computador) para verificar se são iguais. Se forem, a palavra EMPATADO será atribuída à variável alfanumérica S$. A linha 210 verifica se a variável M é maior que a variável C. Se for, ela atribuirá as palavras VOCE GANHOU à variável S$. A linha 220 verifica se M é menor que C. Se for, ela atribuirá as palavras O COMPUTADOR GANHOU à variável S$. A linha 240 apenas imprime o resultado e o jogo termina aí. Embora bastante longo, este programa é muito simples: utiliza apenas uma função, RND, e não recorre a loops ou variáveis indexadas.

Uma vez que a função RND pode variar, e algumas versões do BASIC requerem a instrução RANDOMIZE para dar origem a seqüências de números aleatórios, há algum meio de gerar números efetivamente aleatórios (isto é, que não podem ser previstos), sem recorrer a essas funções? Há várias técnicas possíveis.

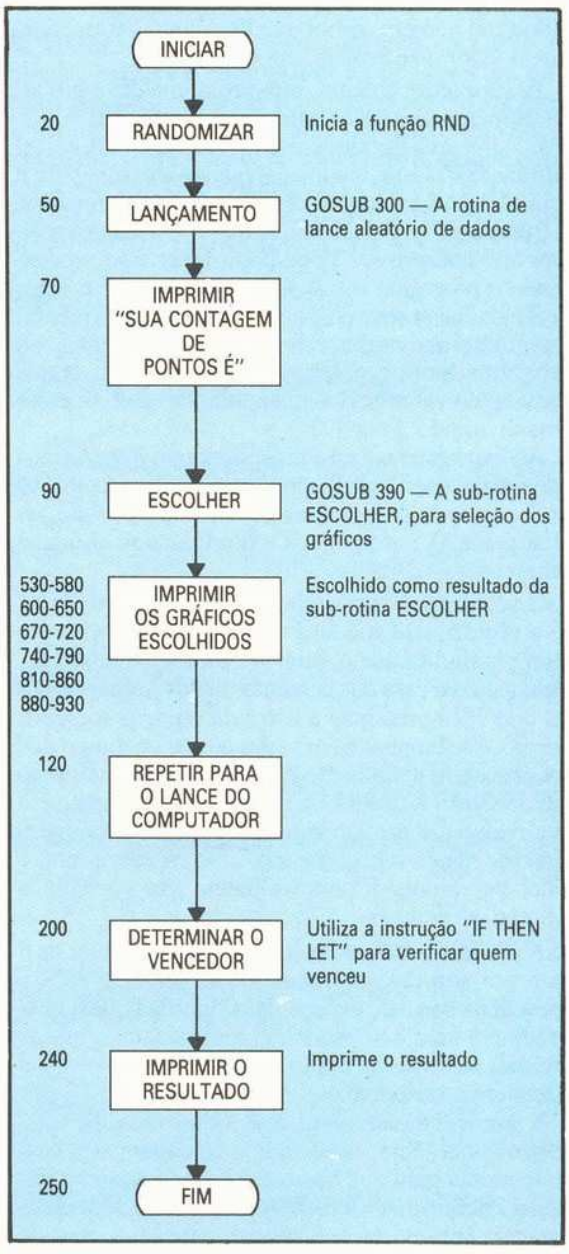
Uma das funções que até agora não examinamos foi a INKEY$ (dita "inkey-string"). Toda vez que o termo INKEY$ é encontrado, o programa examina o teclado e verifica se alguma tecla foi pressionada. O programa não aguarda que se dê entrada a um caractere, como acontece quando é usado o comando INPUT. Por isso, o comando INKEY$ é em geral colocado em um loop. Desse modo, o programa examina continuamente o teclado, aguardando que algo seja introduzido. Geralmente, há um teste incluído no loop, para concluí-lo, quando o caractere adequado for fornecido. Isto torna possível escrever um programa para formar um loop que se encerrará quando um caractere específico for digitado. O que acontecerá se usarmos o seguinte programa?

```
10 PRINT "PRESSIONAR A BARRA DE ESPACO"
20 FOR X = 0 TO 1
30 LET R = R + 1
40 LET A$ = INKEY$
50 IF A$ = " " THEN GOTO 80
60 LET X = 0
70 NEXT X
80 FOR Q = 0 TO 1
90 IF R < 10 THEN GOTO 130
100 LET Q = 0
110 LET R = R/10
120 NEXT Q
130 PRINT INT (R)
140 END
```

Neste caso, a variável R é um número aleatório? Provavelmente, sim; examinemos por quê.

A linha 10 imprime a indicação PRESSIONAR A BARRA DE ESPACO. Antes de termos tempo de atender a esta indicação, o programa terá iniciado o loop FOR X = 0 TO 1, na linha 20. Os números 0 e 1 parecem ser limites estranhos para o loop, mas vejamos resumidamente como esta estrutura é usada. A linha 30 atribui o valor 1 à variável R na primeira passagem pelo loop. A linha 40 atribui à variável A$ qualquer caractere que for digitado no teclado. Isto é realizado pelo uso da função INKEY$. Se você digitar a letra R, esta será a letra atribuída à variável A$. A linha 50 testa esta variável a fim de verificar se corresponde a um espaço (o que é representado em BASIC como um espaço entre aspas, assim: " "). Se A$ for um espaço, o programa se desviará, usando a instrução GOTO, mas, se não o for, seguirá para a linha seguinte.

Esta será a linha 60, que contém a instrução LET X = 0, e a variável X é o contador do loop. A instrução NEXT X, na linha 70, faz com que o programa retorne ao início do loop, na linha 20. Uma vez que X assume novamente o valor 0, o loop repete o procedimento. Deste modo, o loop FOR X = 0 TO 1 será repe-



Fluxo do programa
O fluxograma ao lado mostra, de forma simplificada, as principais atividades realizadas pelo programa. À esquerda, são apresentados os números das linhas correspondentes e à direita estão pequenas notas de esclarecimento. Este fluxograma não está completo, pois muitas das "decisões" e desvios do programa não são apresentados.

tido indefinidamente, enquanto o teste IF A$ = " " não for verdadeiro.

Se em algum estágio a barra de espaço for pressionada, o caractere que representa um espaço será atribuído à variável A$ e, assim, o programa se desviará para a linha 80 e o loop não será repetido.

O que acontecerá, porém, enquanto o loop se repetir? A linha 30 aumenta o valor de R em cada repetição do loop. Na primeira passagem, a variável R assumirá o valor 1, na segunda o valor 1 + 1 e assim sucessivamente. Quando o loop for interrompido pelo teste da variável A$, poderemos ler a variável R e ver até onde chegou nossa contagem.

Todavia, o computador opera muito rapidamente, atingindo a casa das centenas, quando você pressionar a barra de espaço. O que faremos se necessitarmos de valores de R entre 1 e 10? A linha 80 monta outro loop que nos permite testar a variável R e dividi-la por 10 se for maior que este número. Sendo R maior que 10, o teste na linha 90 falha, a variável Q assume novamente o valor 0 e o loop é repetido. A linha 110 divide o valor da variável R por 10 e o resultado só será impresso quando seu valor for reduzido a um número menor que 10. A linha 30 assegura que o valor de R jamais seja 0.

Assim, teoricamente, este programa deve produzir números aleatórios entre 1 e 9, inclusive. A instrução INT assegura que as frações decimais sejam eliminadas; desse modo, os valores possíveis de R serão 1,2,3,4,5,6,7,8,9. A média entre esses números é 5 (porque sua soma é 45, e 45 ÷ 9 = 5). Experimente e comprove. Você pode fazer isso processando o programa várias vezes, registrando o valor de R todas as vezes e em seguida calculando a média. Outra alternativa é acrescentar algumas linhas ao programa, para processá-lo, digamos, 100 vezes, somando o valor de R a uma outra variável, S, e depois dividindo S por 100.

Ao experimentar esse procedimento, verificamos que o valor médio de R apresentava-se muito abaixo de 5 e, assim, os números obtidos não podiam ser aleatórios. Vamos agora examinar por que isso acontece.

O problema é que, embora o programa em BASIC seja rápido, não o é suficientemente. O primeiro loop possibilita que o valor da variável R aumente, alcançando a casa das centenas ou dos milhares, antes que você pressione a barra de espaço. A menos que você se empenhe em variar o total de tempo que decorre entre o momento em que recebe a indicação PRESSIONAR A BARRA DE ESPAÇO e o momento em que realmente pressiona a barra, é provável que o lapso de tempo seja quase o mesmo. A esta altura, o valor da variável R provavelmente terá aumentado em várias centenas.

As divisões realizadas para redução do valor de R para um número abaixo de 10 só ocorrem após a pressão na barra de espaço. Isto significa que R quase sempre atinge as primeiras centenas antes que as divisões se efetuem; assim, o valor final da variável R tenderá a ser baixo.

É possível desenvolver uma rotina que supere esse problema? Sim, desde que a contagem seja bastante rápida para que nosso tempo de reação à indicação PRESSIONAR A BARRA DE ESPAÇO seja efetivamente imprevisível. A solução é fazer um teste de

```
10 REM JOGO DE DADOS — PROGRAMA PRINCIPAL
20 RANDOMIZE
30 REM SEU LANCE
40 REM GOSUB ROTINA "LANCE"
50 GOSUB 300
60 LET M = D
70 PRINT "SUA CONTAGEM DE PONTOS E"
80 REM GOSUB ROTINA "ESCOLHER"
90 GOSUB 390
100 PRINT
110 REM LANCE DO COMPUTADOR
120 REM GOSUB ROTINA "LANCE"
130 GOSUB 300
140 LET C = D
150 PRINT "A CONTAGEM DE PONTOS DO COMPUTADOR E"
160 REM GOSUB ROTINA "ESCOLHER"
170 GOSUB 390
180 PRINT
190 REM QUEM GANHOU?
200 IF M = C THEN LET S$ = "EMPATADO"
210 IF M > C THEN LET S$ = "VOCE GANHOU"
220 IF M < C THEN LET S$ = "O COMPUTADOR GANHOU"
230 REM IMPRIMIR O RESULTADO
240 PRINT S$
250 END
260 REM
270 REM
280 REM
290 REM
300 REM SUB-ROTINA LANCE ALEATORIO DE DADOS
310 REM
320 LET D = INT(10*RND)
330 IF D > 6 THEN GOTO 320
340 IF D < 1 THEN GOTO 320
350 RETURN
360 REM
370 REM
380 REM
390 REM SUB-ROTINA ESCOLHER
400 REM
410 IF D = 1 THEN GOSUB 530
420 IF D = 2 THEN GOSUB 600
430 IF D = 3 THEN GOSUB 670
440 IF D = 4 THEN GOSUB 740
450 IF D = 5 THEN GOSUB 810
460 IF D = 6 THEN GOSUB 880
470 RETURN
480 REM
490 REM
500 REM
510 REM SUB-ROTINA "GRÁFICOS"
520 REM
530 PRINT "         "
540 PRINT " --------- "
550 PRINT "|         |"
560 PRINT "|    *    |"
570 PRINT "|         |"
580 PRINT " --------- "
590 RETURN
600 PRINT "         "
610 PRINT " --------- "
620 PRINT "|       *|"
630 PRINT "|         |"
640 PRINT "|*        |"
650 PRINT " --------- "
660 RETURN
670 PRINT "         "
680 PRINT " --------- "
690 PRINT "|       *|"
700 PRINT "|    *    |"
710 PRINT "|*        |"
720 PRINT " --------- "
730 RETURN
740 'PRINT "         "
750 PRINT " --------- "
760 PRINT "|*       *|"
770 PRINT "|         |"
780 PRINT "|*       *|"
790 PRINT " --------- "
800 RETURN
810 PRINT "         "
820 PRINT " --------- "
830 PRINT "|*       *|"
840 PRINT "|    *    |"
850 PRINT "|*       *|"
860 PRINT " --------- "
870 RETURN
880 PRINT "         "
890 PRINT " --------- "
900 PRINT "|*       *|"
910 PRINT "|*       *|"
920 PRINT "|*       *|"
930 PRINT " --------- "
940 RETURN
```

limite superior, no primeiro loop. Observe este programa:

```
10 PRINT "PRESSIONAR A BARRA DE ESPACO"
20 FOR X = 0 TO 1
30 LET R = R + 1
40 IF R > 9 THEN LET R = 1
50 IF INKEY$ = "      " THEN GOTO 80
60 LET X = 0
70 NEXT X
80 PRINT R
90 END
```

Neste programa, a variável R jamais poderá ser menor que 1 ou maior que 9. No momento em que a barra for pressionada (e reconhecida pela função INKEY$ na linha 50), a variável R terá um valor situado entre 1 e 9, inclusive.

Este programa foi testado mil vezes e o valor médio da variável R foi 5,014. O fato de a média perfeita ser 5 e o erro aqui ter sido da ordem de apenas 0,28% sugere que o programa efetivamente gera números aleatórios muito próximos à média teórica. Evidentemente, a questão aqui está no fato de que, mesmo quando o programa parece razoável no papel, nele podem surgir falhas imprevistas. Assim, é interessante a realização de um teste.

Alguns leitores terão notado que esses programas de números aleatórios poderiam ser reduzidos pelo recurso às instruções GOTO em lugar do loop FOR-NEXT. O motivo por que evitamos as instruções GOTO se evidenciará mais adiante no curso de programação em linguagem BASIC.

Exercícios

■ **Função RND**. Alterar o último programa do texto, para obter um número aleatório que varie de 1 a 6 (inclusive).

■ **Loop e média**. Acrescentar linhas ao último programa do texto, de modo a fazê-lo repetir 100 vezes e apresentar o valor médio dos 100 resultados.

■ **Substituir com sub-rotina**. Substituir as linhas 50 e 130 do programa principal (a sub-rotina lançamento de dados) por uma instrução GOSUB que chame seu "gerador de números aleatórios" no primeiro exercício.

■ **INKEY$.** Usando esta função, escreva um programa que leia qualquer tecla digitada e imprima: A TECLA PRESSIONADA FOI: * (* representa a tecla que você pressionou).

■ **Loop de controle de tempo**. Escreva um loop de controle de tempo (loop de cronometragem) e utilize a função INKEY$ para saber o valor alcançado por uma variável após 10 segundos (observe no relógio). Desenvolva o programa de modo que a leitura final seja:
VALOR DE R APOS 10 SEGUNDOS E: * (* representa o valor de R).

■ **Testes IF-THEN**. Escreva um programa de jogo simples em que o computador dê origem a um número aleatório entre 1 e 100 (inclusive) e o jogador tenha de adivinhar o número. O jogador faz cinco tentativas. Em cada vez, o programa responde com as mensagens: SEU PALPITE ESTA MUITO ALTO, SEU PALPITE ESTA MUITO BAIXO, VOCE ACERTOU, ou SEM DIREITO A OUTRA TENTATIVA. VOCE PERDEU.

Respostas no próximo fascículo.

A propósito...

RANDOMIZE

Nos computadores compatíveis com o Apple II Plus (Unitron, Maxxi, Micro Engenho etc.), a linha 20 deve ser eliminada e a linha 320 alterada para:

```
320 LET D = INT(10*RND(1))
```

RND

Nos computadores compatíveis com o TRS-80 (CP 500, Digitus, etc.), pode-se eliminar a linha 20 e substituir a linha 320 por:

```
320 LET D = RND(6)
```

Nos computadores compatíveis com o Sinclair (TK83, TK85, CP 200, Ringo etc.), o termo RANDOMIZE está abreviado no teclado como RAND.

INKEY$

Nos computadores compatíveis com o Apple II Plus (Unitron, Micro Engenho, Maxxi etc.), deve-se substituir o comando INKEY$ por GET. Assim, no primeiro programa do texto, substituir a linha 40 por:

```
40 GET A$
```

A linha 50 do mesmo programa pode ser substituída por:

```
50 GET A$: IF A$ = " " THEN GOTO 80
```

Respostas aos "Exercícios" das pp. 148-149

Loop 1
O VALOR DE A E 450

Loop 2
INICIO
FIM

Loop 3
O VALOR DE A AGORA E 160

Loop 4
GOSTO DE BASIC
GOSTO DE BASIC
GOSTO DE BASIC
:
:
:
:
Até que as teclas RESET ou BREAK sejam pressionadas.

Loop 5
CHEGA DE LOOPS
(15 vezes)

Read-Data 1
ESTAMOS TESTANDO A INSTRUCAO READ
170

Read-Data 2
X = 1
X = 2
:
:
:
X = 23

Segunda dimensão

Matrizes unidimensionais armazenam conjuntos de dados que apresentam características comuns. Matrizes bidimensionais, por sua vez, são utilizadas para tabelas de dados e diagramas.

Até agora, examinamos duas espécies de variáveis: as simples e as indexadas. As variáveis simples são semelhantes às posições de memória em que os números (ou séries de caracteres) são armazenados e manipulados pela referência ao "nome" da variável. Também podem armazenar apenas um valor ou série e ter nomes "simples" de variáveis — N, B2, X, Y3 são exemplos. As variáveis indexadas, às vezes chamadas matrizes unidimensionais, podem armazenar uma lista completa de valores ou séries. A quantidade de valores ou séries que o programa comporta é especificada no início do programa, com a instrução DIM. Por exemplo, a instrução DIM A(16) determina que a matriz denominada A pode conter dezesseis valores separados. Deve-se notar, entretanto, que muitas versões do BASIC admitem a variável A(0) como o primeiro elemento, de modo que a instrução DIM A(16) efetivamente define dezessete elementos. A referência a essas posições é feita usando-se o indexador apropriado. A instrução PRINT A(1) imprimirá o primeiro elemento da matriz; a instrução LET B = A(12) atribuirá o valor representado pelo décimo segundo elemento da matriz à variável B; a instrução LET A(3) = A(5) atribuirá o valor do quinto elemento ao terceiro elemento.

Todavia, às vezes precisamos manipular dados que se apresentariam melhor sob a forma de tabelas de dados. Observe como este procedimento é muito semelhante ao das folhas eletrônicas (ver p. 158). Tais dados podem variar desde tabelas para resultados de futebol a desdobramentos de vendas por itens e por seções em uma loja. Examine o desdobramento de despesas domésticas durante um ano, como exemplo típico de uma tabela de dados:

	ALUGUEL	TELEFONE	LUZ	ALIMENT.	CARRO
JAN	200000	25000	30000	160000	80000
FEV	200000	25000	30000	160000	80000
MAR	200000	25000	35000	200000	80000
ABR	200000	35000	35000	200000	130000
MAI	200000	35000	35000	230000	130000
JUN	200000	35000	45000	230000	130000
JUL	200000	40000	45000	260000	130000
AGO	450000	40000	45000	260000	180000
SET	450000	40000	60000	300000	180000
OUT	450000	50000	60000	300000	180000
NOV	450000	50000	60000	350000	220000
DEZ	450000	50000	70000	350000	220000

Tal organização de dados permite que estes sejam manipulados de vários modos. É fácil, por exemplo, saber as despesas de março pela simples adição de todos os números da linha desse mês. É fácil, também, obter as despesas de telefone e carro, somando-se as respectivas colunas. As médias anuais e mensais são igualmente fáceis de ser calculadas. Esta tabela de dados é chamada matriz bidimensional. Tem doze linhas e cinco colunas.

Matrizes bidimensionais como esta podem também ser representadas, em BASIC, de modo muito semelhante ao das matrizes unidimensionais; a diferença é que a variável, neste caso, necessita de dois indexadores para remeter a qualquer posição.

Se escrevermos um programa em BASIC utilizando essa tabela de dados, o procedimento mais simples será lidar com a tabela como se fosse uma matriz bidimensional única. Assim, como no caso de matrizes indexadas comuns, nós lhe atribuímos um nome de variável. Vamos chamá-la de A. Novamente, como acontece com as variáveis indexadas comuns, ela precisará ser DIMensionada. Como são doze linhas e cinco colunas, a variável deve ser dimensionada assim: DIM A(12,5). A ordem de colocação dos dois indexadores é importante: está convencionado que as linhas são especificadas em primeiro lugar e as colunas em segundo. Nossa tabela tem doze linhas (uma para cada mês) e cinco colunas (para as cinco categorias de despesas) e, portanto, é uma matriz 12-por-5.

A instrução DIM atende a duas funções básicas: reserva posições de memória suficientes para a matriz na memória do computador e permite que cada posição seja indicada pelo nome da variável, seguido pelas posições de linha e de coluna entre parênteses. A instrução DIM, por exemplo DIM X(3,5), cria uma variável X, que representa uma matriz com três linhas e cinco colunas.

Observe o quadro, admitindo que os dados tenham sido fornecidos como elementos de uma matriz bidimensional, denominada A. Calcule os valores de A(1,1), A(1,5), A(2,1), A(3,3) e A(12,3).

Pode-se fornecer uma tabela de dados sob a forma de matriz em uma parte do programa, usando-se, por exemplo, instruções LET.

```
30 LET A(1,2) = 25000
40 LET A(1,3) = 30000
50 LET A(1,4) = 160000
.
.
.
610 LET A(12,5) = 220000
```

Evidentemente, porém, este procedimento é trabalhoso. Muito mais simples são as instruções READ e DATA ou a instrução INPUT com loops FOR-NEXT incluídos. Vejamos como isto é feito com a instrução READ:

```
10 DIM A(12,5)
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
```

```
40 READ A(R,C)
50 NEXT C
60 NEXT R
70 DATA 200000, 25000, 30000, 160000, 80000,
   200000, 25000, 30000
80 DATA 160000, 80000, 200000, 25000, 35000,
   200000, 80000, 200000
90 DATA 35000, 35000, 200000, 130000, 200000,
   35000, 35000, 230000
100 DATA 130000, 200000, 35000, 45000, 230000,
   130000, 200000, 40000
110 DATA 45000, 260000, 130000, 450000, 40000,
   45000, 260000, 180000, 450000
120 DATA 40000, 60000, 300000, 180000, 450000,
   50000, 60000
130 DATA 300000, 180000, 450000, 50000, 60000,
   350000, 220000
140 DATA 450000, 50000, 70000, 350000, 220000
150 END
```

Há detalhes importantes a notar neste programa. O primeiro deles é que a instrução DIM inicia o programa. A instrução DIM deve ser executada só uma vez no programa e costuma-se colocá-la logo no início ou antes da execução dos loops. O segundo ponto a observar é que há dois loops FOR-NEXT, um para determinar a "linha" do indexador e outro para determinar a "coluna". Esses dois loops não se seguem um após o outro, mas, sim, um está "alojado" no outro. Observe os limites escolhidos. A instrução FOR R = 1 TO 12 aumentará o valor da linha de 1 até 12; a instrução FOR C = 1 TO 5 aumentará o valor de 1 até 5.

Exatamente na metade do loop incluído encontra-se a instrução READ. A parte fundamental do programa é:

```
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
40 READ A(R,C)
50 NEXT C
60 NEXT R
```

Na primeira passagem após as instruções das linhas 20 e 30 terem sido executadas, os valores das variáveis R e C serão, ambos, 1. Assim, a linha 40 equivalerá a READ A(1,1). O primeiro item de dados na instrução DATA é 200000; portanto, este valor será atribuído à primeira linha e coluna da tabela. A escolha de oito elementos para cada instrução DATA faz-se de maneira arbitrária.

Executadas essas operações, a instrução NEXT C remete o programa outra vez à linha 30 e o valor da variável C aumenta para 2. A linha 40 passa então a equivaler a READ A(1,2) e o item seguinte de dados, 25000, será atribuído à primeira linha e segunda coluna da matriz. Este processo se repete até que a variável C tenha alcançado o valor 5. Após isso, a instrução NEXT R, na linha 60, faz o programa retornar à linha 20 e o valor de R aumenta para 2. A linha 30 atribuirá novamente o valor 1 para a variável C e, desse modo, a nova linha 40 equivalerá a READ A(2,1).

A inclusão de loops por este procedimento é muito conveniente, mas preste atenção! Cada loop deve ser alojado de modo completo no interior de outro loop e a ordem das instruções NEXT deve ser cuidadosamente observada. Repare que o primeiro loop, FOR R, tem a segunda instrução NEXT. Quando há dois loops, um alojado no outro, o primeiro é chamado loop interior, e o segundo, loop exterior. O loop interior deverá estar inteiramente concluído antes de o loop exterior ser desenvolvido. É possível efetuar o alojamento de loops em tantos "níveis" quantos o programa exigir, mas tais programas se tornam complexos, difíceis de acompanhar e corrigir. É um mau hábito de programação incluir instruções de desvio no interior de loops; igualmente, o uso de comandos GOTO deve ser evitado.

Examinemos as instruções DATA. Observe que as vírgulas são usadas para separar itens de dados, mas não deve haver vírgulas antes do primeiro item de dados ou após o último. Inserimos espaços entre cada item de dados; este procedimento, entretanto, não é comum. Com freqüência cometem-se erros ao fornecer os dados, e é difícil detectá-los posteriormente. Pode-se recorrer a tantas instruções DATA quantas forem necessárias. Cada nova linha deve começar pela instrução DATA. Os dados são lidos, um de cada vez, a partir do início da primeira instrução DATA, prosseguindo até que todos os itens tenham sido lidos. Assegure-se de que esteja correta a numeração de itens de dados, senão você receberá uma mensagem de erro, quando o programa for processado.

O programa apresentado, até aqui, nada realiza, exceto a conversão dos dados correspondentes para a matriz bidimensional. Após a entrada e o processamento do programa, nada aparentemente acontece. A fim de testar se os dados estão distribuídos de modo correto, experimente usar alguns comandos PRINT (comando, em BASIC, é uma palavra-chave que pode ser imediatamente executada, sem estar incluída em um programa e, portanto, não necessita de um número de linha. São exemplos de comandos: LIST, RUN, SAVE, AUTO, EDIT e PRINT). O comando PRINT A(1,1) <CR> deverá fazer com que o número 200000 apareça na tela. O que será impresso pelos seguintes comandos?

```
PRINT A(12,1)
PRINT A(1,5)
PRINT A(5,1)
PRINT A(5,5)
```

Para que o programa realize uma operação que tenha utilidade efetiva, será necessário ampliá-lo. Do modo como se apresenta, ele forma uma base adequada para um "programa principal". A fim de utilizá-lo como parte de um programa mais amplo e de maior utilidade, pode-se desenvolver módulos sob a forma de sub-rotinas a serem chamadas por instruções GOSUB inseridas em pontos convenientes, antes da instrução END.

Nos primeiros estágios de elaboração de um programa de contabilidade doméstica, é melhor iniciar com uma simples descrição, escrita, das várias exigências. Possivelmente, desejamos obter totais e médias das despesas mensais ou de uma categoria (luz, por exemplo). Calcularemos os detalhes para obtenção desses resultados em um estágio posterior. Se, no interior do programa, houver uma escolha das sub-rotinas que devem ser executadas, receberemos a indicação, talvez por um "menu", que diri-

girá o controle para as sub-rotinas correspondentes em resposta aos dados que fornecemos. Um esboço inicial do programa neste estágio seria semelhante ao seguinte:

```
PROGRAMA PRINCIPAL
(ENTRADA DE DADOS)

MENU
(SUB-ROTINAS PARA ESCOLHA)

END
```

Um pequeno trabalho de aprimoramento deste programa nos mostrará que necessitaremos de sub-rotinas para calcular os totais dos meses e das categorias (TOTALMES e TOTALCAT), calcular a média de despesas mensais (MEDMENSAL) e a média de despesas anuais por categoria (MEDCAT). O motivo da utilização, nessas sub-rotinas, de nomes constituídos por uma só palavra está em que isto nos ajuda a planejar o programa sem preocupar-nos, nesta etapa, com detalhes como números de linha. Pensando melhor, chegaremos à conclusão de que mesmo a escolha do principal ''menu'' do programa deve ser executada como uma sub-rotina, a fim de manter a parte principal do programa como um módulo separado. O próximo estágio de elaboração será semelhante ao seguinte:

```
PROGRAMA PRINCIPAL (ENTRADA DE DADOS)
  MENU (CHAMAR SUB-ROTINA)
END

**SUB-ROTINAS**

1 MENU
2 TOTAIS
3 MEDIAS

(2) TOTAIS
4 TOTALMES
5 TOTALCAT

(3) MEDIAS
6 MEDMES
7 MEDCAT
```

Este esboço do programa mostra como a sub-rotina MENU nos permitirá a escolha entre TOTAIS e MEDIAS. Estas mesmas serão, ambas, sub-rotinas. A sub-rotina TOTAIS permitirá uma escolha ulterior entre TOTALMES e TOTALCAT. Estas operações se constituirão de sub-rotinas que realizarão os cálculos efetivos.

A sub-rotina MEDIAS permitirá a escolha das operações MEDMES ou MEDCAT, que serão também sub-rotinas para realização dos cálculos correspondentes. Nesta etapa, é possível perceber se nosso ''programa'' realizará as operações que desejamos, sem precisar recorrer a alguma codificação efetiva (detalhado programa escrito em BASIC). Se ficarmos satisfeitos com um ''até aqui, tudo bem'', estaremos em condição de enfrentar o desenvolvimento dos próprios módulos (sub-rotinas). A única alteração exigida pelo programa principal será uma chamada de sub-rotina, antes da instrução END; assim, podemos acrescentar:

```
145 GOSUB **MENU**
```

Observe que ainda estamos utilizando ''nomes'' para as sub-rotinas, em vez de números de linha. Outras linguagens admitem que os subprogramas sejam chamados pelo nome, mas a maioria das versões da linguagem BASIC não; assim, é necessário substituí-los pelos números das linhas.

Examinemos o modo pelo qual a sub-rotina MENU poderia ser escrita.

```
REM A SUB-ROTINA **MENU**
PRINT "VOCE DESEJA T(OTAIS) OU M(EDIAS)?"
PRINT "DIGITE T OU M"
INPUT L$
IF L$ = "T" THEN GOSUB *TOTAIS*
IF L$ = "M" THEN GOSUB *MEDIAS*
RETURN
```

Observações: estamos assinalando as sub-rotinas chamadas, encerrando-as entre os sinais *—*, os quais você deverá substituir por números de linha, quando for possível saber quais são eles.

Admitamos que você digite T, para TOTAIS. O programa, então, chamará a sub-rotina TOTAIS, que apresentará um outro menu, semelhante ao seguinte:

```
REM A SUB-ROTINA **TOTAIS**
PRINT "VOCE DESEJA OS TOTAIS DE"
PRINT "M(ES) OU C(ATEGORIA)?"
PRINT "DIGITE M OU C"
INPUT L$
IF L$ = "M" THEN GOSUB *TOTALMES*
IF L$ = "C" THEN GOSUB *TOTALCAT*
RETURN
```

Admitamos que você esolha M para representar o TOTALMES. Vejamos como escrever um módulo para calcular a despesa total de qualquer mês no ano.

```
REM A SUB-ROTINA **TOTALMES**
REM ESTA CALCULA A DESPESA TOTAL EM
REM QUALQUER MES
PRINT "ESCOLHER MES"
PRINT "1-JAN 2-FEV 3-MAR 4-ABR 5-MAI"
PRINT "6-JUN 7-JUL 8-AGO 9-SET"
PRINT "10-OUT 11-NOV 12-DEZ"
PRINT "DIGITE O NUMERO CORRESPONDENTE AO
  MES"
LET T = 0
INPUT M
FOR C = 1 TO 5
LET T = T + A(M,C)
NEXT C
PRINT "A DESPESA TOTAL NO MES "
PRINT "NUMERO ";M;" E ";T
RETURN
```

O número que representa o mês é digitado e a instrução INPUT o atribui à variável M(MES). A variável M é utilizada para especificar o indexador de ''linha'' da matriz bidimensional A. O loop FOR-NEXT aumenta o valor da variável C (que representa a coluna) de 1 a 5; assim, na primeira passagem pelo loop, se escolhermos o número 3 — que corresponde a março, a instrução LET equivalerá a LET T = T + A(3,1). Na passagem seguinte pelo loop, ela equivalerá a LET T = T + A(3,2) e assim sucessivamente.

Desta vez, deixaremos a você a tarefa de desenvolver as outras sub-rotinas ou experimentar outros exercícios. As matrizes bidimensionais são convenientes para programas que incluem tabelas de dados estatísticos, financeiros ou de outra natureza.

Respostas aos Exercícios da p. 175

Função RND

```
40 IF R > 6 THEN LET R = 1
```

Loop e média

```
5 FOR L = 1 TO 100
:
:
80 LET T = T + R
90 NEXT L
100 LET A = T/100
110.END
```

Substituir com sub-rotina

Elimine as linhas 5, 80, 90, 100 e 110 da resposta acima. Altere as linhas de 10 a 70 para (digamos) 1000 a 1070. Certifique-se de que a linha 40 está igual à da resposta ao exercício de Função RND, acima. A seguir, acrescente a instrução 1080 RETURN. Incorpore o resultado ao programa principal. Altere as linhas 50 e 130 do programa principal, de modo que se apresente da seguinte forma: 50 GOSUB 1000 e 130 GOSUB 1000.

INKEY$

```
10 PRINT "DIGITE QUALQUER TECLA"
20 LET A$ = INKEY$
30 IF A$ = "  " THEN GOTO 20
40 PRINT "A TECLA PRESSIONADA FOI ";A$
50 END
```

Loop de controle de tempo

```
5 PRINT "PRESSIONAR A BARRA DE ESPACO APOS 10
  SEGUNDOS"
10 FOR L = 0 TO 1
20 LET R = R + 1
30 IF INKEY$ = "  " THEN GOTO 60
40 LET L = 0
50 NEXT L
60 PRINT "O VALOR DE R APOS 10 SEGUNDOS E ";R
70 END
```

IF-THEN

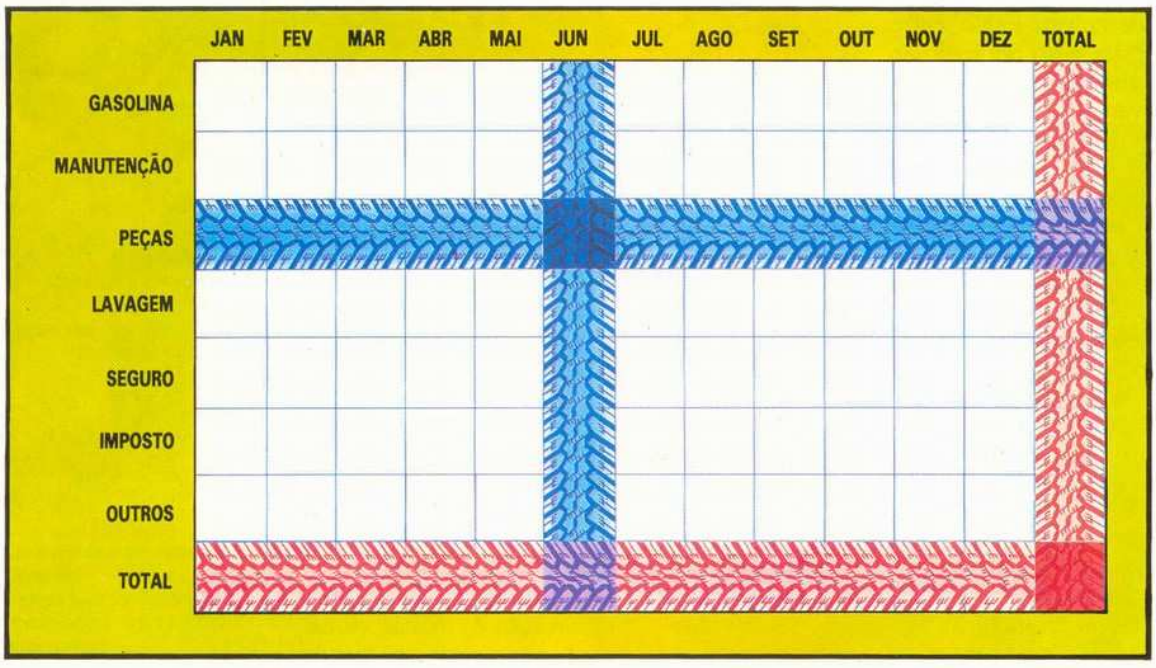
```
10 GOSUB 1000
20 PRINT "SUGIRA UM NUMERO"
30 FOR G = 1 TO 5
40 INPUT N
50 IF N > R THEN GOTO 110
60 IF N < R THEN GOTO 130
70 IF N = R THEN GOTO 150
80 NEXT G
90 PRINT "SEM DIREITO A OUTRA TENTATIVA. VOCE
   PERDEU"
100 GOTO 500
110 PRINT "SEU PALPITE ESTA MUITO ALTO"
120 GOTO 80
130 PRINT "SEU PALPITE ESTA MUITO BAIXO"
140 GOTO 80
150 PRINT "VOCE ACERTOU"
500 END
1000 REM **SUB-ROTINA DO NUMERO ALEATORIO**
(introduza sua sub-rotina aqui)
1020 RETURN
```

Exercícios

■ **Atribuição de valores.** Desenvolva um programa que atribua valores aos elementos (''gasolina'', ''manutenção'' etc.) da matriz (ver ilustração abaixo). A seguir, escreva uma sub-rotina que solicite o nome de um mês e um título para despesas e imprima os conteúdos do quadro assim especificado. Finalmente, escreva uma sub-rotina que calcule a soma de cada coluna, coloque o resultado no quadro abaixo, realize o mesmo procedimento com relação às linhas, e calcule, em seguida, o total geral, a ser armazenado no quadro inferior da direita.

■ **Erros.** O programa apresentado a seguir não poderá ser processado convenientemente e produzirá uma mensagem de erro. Localize o erro e efetue a correção necessária.

```
10 DIM A(3,4)
20 FOR R = 1 TO 3
30 FOR C = 1 TO 4
40 READ A(R,C)
50 NEXT C
60 NEXT R
70 FOR X = 1 TO 3
90 FOR Y = 1 TO 4
100 PRINT A(Y,X)
110 NEXT Y
120 NEXT X
130 DATA 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22
140 END
```



Despesas com o carro
A ilustração apresenta oito quadriculados verticais e treze horizontais, com os diversos itens de despesas de um carro, durante um ano. Acompanhe os dados do exercício "Atribuição de valores", para fazer os cálculos.

Novas estruturas

Todas as versões da linguagem BASIC apresentam estruturas chamadas de controle. E alguns equipamentos possibilitam grande variedade de alternativas, com diferenças sutis.

As primeiras dez unidades do curso de programação BASIC abrangeram quase todos os aspectos mais importantes desta linguagem. Agora apresentamos um apanhado geral dos tópicos já vistos, focalizando algumas questões interessantes, com indicações do que faremos mais adiante.

Lembramos em primeiro lugar: linguagens de alto nível como o BASIC proporcionam ao usuário um conjunto de instruções traduzidas internamente para uma forma compatível com o computador. Qualquer programa para o computador pode ser escrito usando-se apenas duas construções simples: as seqüenciais e as estruturas de controle. Em BASIC, são construções essenciais: IF-THEN-ELSE e WHILE-DO. A maioria das outras linguagens de computação apresenta muitas outras estruturas.

A construção seqüencial permite que uma tarefa seja dividida em um conjunto de subtarefas que realizam a tarefa principal, quando executadas em seqüência. O tamanho das subtarefas depende da linguagem; em BASIC, as subtarefas são representadas pelas instruções escritas em cada linha e a seqüência, por números de linha. Desse modo, se a tarefa for multiplicar por 10 o valor atribuído a uma variável, a seqüência poderá ser a seguinte:

```
110 INPUT N
120 LET N = N * 10
130 PRINT N
```

Além das construções seqüenciais, também necessitamos de estruturas de controle. Estas são construções que alteram a ordem de execução das instruções no programa.

A estrutura de controle mais simples da linguagem BASIC é a instrução GOTO, que provoca um salto (ou desvio) incondicional, ou seja, sem condições determinadas. A execução do programa é desviada para um número específico de linha, sem necessidade de satisfazer a testes ou condições. A instrução GOSUB é também um desvio sem condições determinadas, mas, com ela, o programa sempre retorna ao ponto imediatamente seguinte à instrução GOSUB e admite-se seu uso em programação estruturada.

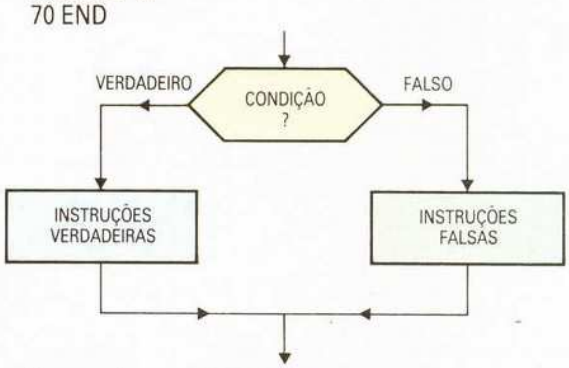
A estrutura de controle IF-THEN-ELSE também existe em BASIC, sob a forma da instrução IF-THEN, e apresenta a seguinte sintaxe (''sintaxe'' é o jargão de computação que corresponde à ''forma''):

> SE (IF) a condição especificada for verdadeira, ENTÃO (THEN) executar a instrução especificada; CASO CONTRÁRIO (ELSE), executar a próxima instrução

Observe que em linguagem BASIC standard, o elemento ELSE da instrução IF-THEN-ELSE está implícito. Em algumas versões ou adaptações de BASIC e em outras linguagens, como PASCAL, o elemento ELSE faz parte da instrução.

A instrução IF-THEN-ELSE (IF-THEN, em BASIC) executa uma das duas subtarefas, dependendo de ser verdadeira ou não certa condição. Observe o seguinte programa, elaborado para determinar a raiz quadrada de números fornecidos através do teclado, a menos que se forneça um valor ''flag'' de −9999 (para encerrar o programa):

```
10 PRINT "FORNECER UM NUMERO"
20 INPUT N
30 IF N = −9999 THEN GOTO 70
40 LET S = SQR(N)
50 PRINT "A RAIZ QUADRADA DE "; N; " E "; S
60 GOTO 10
70 END
```



Estrutura de controle IF-THEN-ELSE
Se a condição for Verdadeira, as instruções Verdadeiras serão executadas. Se a condição for Falsa, as instruções Falsas serão executadas.

A linha 30 está efetivamente indicando: ''SE (IF) é verdade que a variável N = −9999, ENTÃO (THEN) vá ao final do programa; CASO CONTRÁRIO (ELSE) (se não é verdade que N = −9999), execute a linha seguinte para calcular a raiz quadrada''.

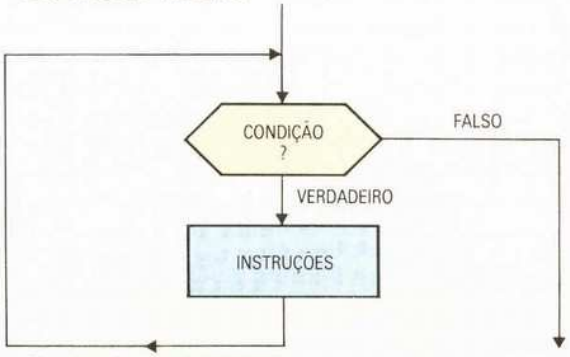
Outra estrutura de controle fundamental (WHILE-DO) não está diretamente disponível em BASIC, mas pode ser simulada com facilidade. WHILE-DO é um tipo de ''loop de execução'' e significa ''repetir uma instrução ou conjunto de instruções, ENQUANTO (WHILE) uma determinada condição for verdadeira''; ou seja, ''ENQUANTO (WHILE) uma condição for verdadeira, EXECUTAR (DO) determinado procedimento''.

WHILE-DO sempre testa a condição antes de executar as instruções; assim, se o teste for negativo na primeira passagem, as instruções (chamadas corpo do loop) não serão executadas. Como exemplo, observe um programa de jogos que indica ao jogador "PRESSIONAR A BARRA DE ESPAÇO, QUANDO PRONTO". Esta parte do programa poderia ser desenvolvida (em ''pseudolinguagem'', ou inglês ou português simplificado) como:

ENQUANTO (WHILE) a barra de espaço não for pressionada,
FAÇA (DO) a sondagem do teclado
começar o jogo

Em BASIC, isto seria escrito:

```
250 PRINT "PRESSIONAR A BARRA DE ESPACO
    QUANDO ESTIVER PRONTO"
260 IF INKEY$ <> " " THEN GOTO 260
270 GOSUB *INICIAR*
```



Estrutura de controle DO-WHILE
O loop é repetido enquanto a condição for Verdadeira. As instruções podem não ser executadas (se a condição inicial for Falsa).

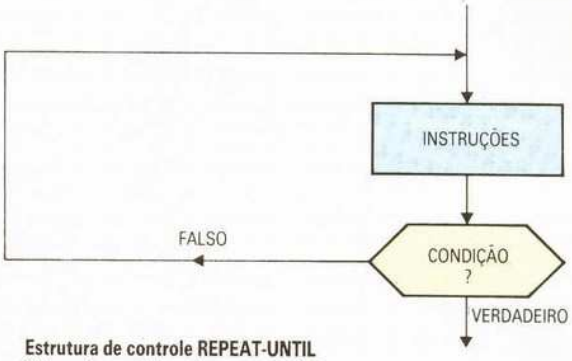
A linha 260 indica que SE (IF) INKEY$ não é igual (<>) a um espaço (" "), ENTÃO (THEN) retorne e verifique o teclado novamente. Um modo um pouco mais elegante de desenvolver esse procedimento seria:

```
250 PRINT "PRESSIONAR A BARRA DE ESPACO
    QUANDO ESTIVER PRONTO"
260 FOR X = 0 TO 1 STEP 0
270 IF INKEY$ = " " THEN LET X = 2
280 NEXT X
290 GOSUB *INICIAR*
```

Neste fragmento de programa, o loop (para sondagem do teclado) é executado somente se a barra de espaço não for pressionada. Se a barra tiver sido pressionada (i.é, INKEY$ = " "), então o programa vai do loop FOR-NEXT para a linha 290, que é a chamada para a sub-rotina INICIAR. (Obs.: Estamos utilizando "títulos" ou nomes para as sub-rotinas. Muitas versões de BASIC não admitem a chamada de sub-rotinas pelo nome; assim, você terá de utilizar os números de linha das sub-rotinas.)

Não apresentamos antes a instrução STEP, e o exemplo dado é uma aplicação pouco comum. Quando usada em um loop FOR-NEXT, a instrução STEP permite que o "contador" seja aumentado em unidades diferentes de um. A instrução FOR I = 1 TO 10 STEP 2 fará com que a variável I tenha o valor 1 na primeira passagem pelo loop, seguido dos valores 3, 5, 7 e 9. O aumento seguinte (para 11) ultrapassará o limite de 10 e, desse modo, o loop estará encerrado. Pode-se fazer com que o contador execute a contagem regressiva. A instrução FOR I = 10 TO 1 STEP−1 fará a variável I executar regressivamente a contagem de 10 para 1. A utilização da instrução STEP 0 é um artifício engenhoso para assegurar que o loop nunca termine, a menos que a variável X seja "aumentada artificialmente", como no caso da instrução IF-THEN.

Outra estrutura de controle muito útil, também



Estrutura de controle REPEAT-UNTIL
O loop é repetido até que a condição se torne Verdadeira. As instruções terão de ser executadas, pelo menos uma vez.

não encontrada diretamente em BASIC, mas simulável com facilidade, é a instrução REPEAT-UNTIL. Aqui, o teste de condição se apresenta após a parte principal do loop; assim, a instrução ou instruções no corpo principal do loop serão repetidas, pelo menos uma vez. Observe o seguinte programa "gerador de números aleatórios":

```
10 PRINT "PRESSIONAR A BARRA DE ESPACO"
20 FOR X = 0 TO 1 STEP 0
30 LET R = R + 1
40 IF R > 9 THEN LET R = 1
50 IF INKEY$ = " " THEN LET X = 2
60 NEXT X
70 PRINT "O VALOR DE R E  ";R
```

Neste caso, o corpo principal (aumento do valor da variável R) tem de ser executado, pelo menos uma vez, já que o teste para desvio do loop (IF INKEY$ = " ") só se apresenta após a instrução para aumento (LET R = R + 1).

Outra estrutura de controle, não essencial, mas útil, é a geralmente denominada CASE. Em BASIC a estrutura CASE realiza-se mediante a instrução ON-GOTO ou a instrução ON-GOSUB. Eis como funciona: ON-GOTO é uma instrução de desvios múltiplos que incorpora vários testes condicionais IF-THEN em uma única instrução. Observe um fragmento de programa que converte os números de 1 a 7 nas palavras que representam os sete dias da semana:

```
1050 IF D = 1 THEN GOTO 2020
1060 IF D = 2 THEN GOTO 2040
1070 IF D = 3 THEN GOTO 2060
1080 IF D = 4 THEN GOTO 2080
1090 IF D = 5 THEN GOTO 3000
2000 IF D = 6 THEN GOTO 3020
2010 IF D = 7 THEN GOTO 3040
2020 PRINT "SEGUNDA-FEIRA"
2030 GOTO *END*
2040 PRINT "TERCA-FEIRA"
2050 GOTO *END*
2060 PRINT "QUARTA-FEIRA"
2070 GOTO *END*
2080 PRINT "QUINTA-FEIRA"
2090 GOTO *END*
3000 PRINT "SEXTA-FEIRA"
3010 GOTO *END*
3020 PRINT "SABADO"
3030 GOTO *END*
3040 PRINT "DOMINGO"
3050 GOTO *END*
```

DECIMAL	BINÁRIO	CARACTERES
32	0 0 1 0 0 0 0 0	(espaço)
33	0 0 1 0 0 0 0 1	!
34	0 0 1 0 0 0 1 0	"
35	0 0 1 0 0 0 1 1	#
36	0 0 1 0 0 1 0 0	$
37	0 0 1 0 0 1 0 1	%
38	0 0 1 0 0 1 1 0	&
39	0 0 1 0 0 1 1 1	'
40	0 0 1 0 1 0 0 0	(
41	0 0 1 0 1 0 0 1	)
42	0 0 1 0 1 0 1 0	*
43	0 0 1 0 1 0 1 1	+
44	0 0 1 0 1 1 0 0	,
45	0 0 1 0 1 1 0 1	-
46	0 0 1 0 1 1 1 0	.
47	0 0 1 0 1 1 1 1	/
48	0 0 1 1 0 0 0 0	0
49	0 0 1 1 0 0 0 1	1
50	0 0 1 1 0 0 1 0	2
51	0 0 1 1 0 0 1 1	3
52	0 0 1 1 0 1 0 0	4
53	0 0 1 1 0 1 0 1	5
54	0 0 1 1 0 1 1 0	6
55	0 0 1 1 0 1 1 1	7
56	0 0 1 1 1 0 0 0	8
57	0 0 1 1 1 0 0 1	9
58	0 0 1 1 1 0 1 0	:
59	0 0 1 1 1 0 1 1	;
60	0 0 1 1 1 1 0 0	<
61	0 0 1 1 1 1 0 1	=
62	0 0 1 1 1 1 1 0	>
63	0 0 1 1 1 1 1 1	?
64	0 1 0 0 0 0 0 0	@
65	0 1 0 0 0 0 0 1	A
66	0 1 0 0 0 0 1 0	B
67	0 1 0 0 0 0 1 1	C
68	0 1 0 0 0 1 0 0	D
69	0 1 0 0 0 1 0 1	E
70	0 1 0 0 0 1 1 0	F
71	0 1 0 0 0 1 1 1	G
72	0 1 0 0 1 0 0 0	H
73	0 1 0 0 1 0 0 1	I
74	0 1 0 0 1 0 1 0	J
75	0 1 0 0 1 0 1 1	K
76	0 1 0 0 1 1 0 0	L
77	0 1 0 0 1 1 0 1	M
78	0 1 0 0 1 1 1 0	N
79	0 1 0 0 1 1 1 1	O
80	0 1 0 1 0 0 0 0	P
81	0 1 0 1 0 0 0 1	Q
82	0 1 0 1 0 0 1 0	R
83	0 1 0 1 0 0 1 1	S
84	0 1 0 1 0 1 0 0	T
85	0 1 0 1 0 1 0 1	U
86	0 1 0 1 0 1 1 0	V
87	0 1 0 1 0 1 1 1	W
88	0 1 0 1 1 0 0 0	X
89	0 1 0 1 1 0 0 1	Y
90	0 1 0 1 1 0 1 0	Z
91	0 1 0 1 1 0 1 1	[
92	0 1 0 1 1 1 0 0	\
93	0 1 0 1 1 1 0 1	]
94	0 1 0 1 1 1 1 0	^
95	0 1 0 1 1 1 1 1	_
96	0 1 1 0 0 0 0 0	`
97	0 1 1 0 0 0 0 1	a
98	0 1 1 0 0 0 1 0	b
99	0 1 1 0 0 0 1 1	c
100	0 1 1 0 0 1 0 0	d
101	0 1 1 0 0 1 0 1	e
102	0 1 1 0 0 1 1 0	f
103	0 1 1 0 0 1 1 1	g
104	0 1 1 0 1 0 0 0	h
105	0 1 1 0 1 0 0 1	i
106	0 1 1 0 1 0 1 0	j
107	0 1 1 0 1 0 1 1	k
108	0 1 1 0 1 1 0 0	l
109	0 1 1 0 1 1 0 1	m
110	0 1 1 0 1 1 1 0	n
111	0 1 1 0 1 1 1 1	o
112	0 1 1 1 0 0 0 0	p
113	0 1 1 1 0 0 0 1	q
114	0 1 1 1 0 0 1 0	r
115	0 1 1 1 0 0 1 1	s
116	0 1 1 1 0 1 0 0	t
117	0 1 1 1 0 1 0 1	u
118	0 1 1 1 0 1 1 0	v
119	0 1 1 1 0 1 1 1	w
120	0 1 1 1 1 0 0 0	x
121	0 1 1 1 1 0 0 1	y
122	0 1 1 1 1 0 1 0	z
123	0 1 1 1 1 0 1 1	{
124	0 1 1 1 1 1 0 0	|
125	0 1 1 1 1 1 0 1	}
126	0 1 1 1 1 1 1 0	~

ASCII
Eis aqui uma lista completa dos valores ASCII entre 32 e 126, seus equivalentes binários e os caracteres que representam. O significado dos valores fora destes limites varia de uma máquina para outra.

Um modo mais conciso de atingir o mesmo objetivo em BASIC é usar a instrução ON-GOTO, da seguinte forma:

```
1050 ON D GOTO 2020,2040,2060,2080
     3000,3020,3040
```

A instrução ON-GOSUB opera do mesmo modo, com exceção de que o valor da variável determina para qual sub-rotina o desvio é dirigido. Aqui apresentamos uma pequena modificação do programa de lançamento de dados (ver p. 174), usando a instrução ON-GOSUB, que seleciona os gráficos apropriados para os dados escolhidos pela função RND:

```
390 REM SUB-ROTINA ESCOLHER
400 REM UTILIZANDO A INSTRUCAO ON-GOSUB
410 ON D GOSUB 530,600,670,740,810,880
470 RETURN
```

Embora a versão de BASIC de seu equipamento possua, provavelmente, muitas instruções e funções que ainda não apresentamos, a maioria delas serão extensões da linguagem BASIC "padrão". Muitas estarão vinculadas a finalidades gráficas incorporadas ao hardware — instruções como PAINT, PAPER, INK, BEEP e CIRCLE. Estas instruções, de modo geral, são "específicas da máquina" e, por isso, não as incluímos em nosso curso, mas delas daremos detalhes em artigos futuros.

Antes de encerrar a parte fundamental de nosso curso de BASIC, há alguns outros pontos a considerar: um exame do conjunto de caracteres ASCII, juntamente com as funções para auxiliar a manipulação de caracteres, e um método de definição de novas funções (ou funções não incluídas na versão de BASIC de seu computador).

Elaboraram-se vários métodos de representação das letras do alfabeto e de outros caracteres, como números e sinais de pontuação em forma digital, no decorrer dos anos. Um dos primeiros foi o código Morse, que utiliza combinações de pontos e traços para representar caracteres. Para o computador, o código Morse tem a desvantagem de utilizar diferentes quantidades de bits para letras diferentes — entre um e seis pontos e traços para cada caractere. Outras tentativas de formar um código de caracteres mais regular e sistemático (por exemplo, o código Baudot, que utiliza 5 bits para representar até 32 caracteres) foram superadas, e o sistema quase universal agora em uso é o código ASCII (American Standard Code for Information Interchange).

O código ASCII utiliza 1 byte para representar os 94 caracteres imprimíveis, o "espaço" e uma série de "caracteres" de controle. 8 bits podem fornecer 256 combinações diferentes (2^8), mas isto é muito mais do que o necessário para representar os caracteres de uma máquina de escrever comum ou de um teclado de computador; desse modo, apenas 7 desses bits são usados, permitindo 128 combinações diferentes. (O oitavo bit, se utilizado, especifica um conjunto alternativo de caracteres gráficos ou de línguas estrangeiras.) No quadro da página anterior apresentamos os códigos ASCII, em decimal e binário, para o conjunto standard de caracteres.

Como você pode verificar no quadro, o código ASCII para a letra A é 65 e, para a letra B, 66. Os códigos para as minúsculas a e b são 97 e 98. Cada letra minúscula tem um valor, em código ASCII, 32 números acima de seu correspondente em maiúscula. Essa diferença constante torna fácil a conversão de minúsculas para maiúsculas e vice-versa. Para realizar esta conversão, necessitaremos de outras duas funções não utilizadas até agora no curso de programação em linguagem BASIC — as funções ASC e CHR$.

A função ASC toma o caractere imprimível e o transforma em código ASCII; desse modo, a instrução PRINT ASC("A") imprime o número 65 na tela; a instrução PRINT ASC("b") imprime o número 98.

A função CHR$ faz o inverso: toma um número, supondo-o como código ASCII e o transforma no caractere que ele representa. Desse modo, a instrução PRINT CHR$(65) imprime a letra A, enquanto a instrução PRINT CHR$(98) imprime a letra b. As funções CHR$ e ASC são amplamente empregadas com as instruções LEFT$, RIGHT$ e MID$ em programas que fazem uso freqüente de séries de caracteres. Eis aqui um programa que recebe um caractere do teclado, verifica se está em maiúscula e, se não estiver, converte-o em maiúscula:

```
10 REM CONVERSOR DE MINUSCULAS
   PARA MAIUSCULAS
20 PRINT "FORNECER UM CARACTERE"
30 INPUT C$
40 LET C = ASC(C$)
50 IF C > 90 THEN LET C = C-32
60 PRINT CHR$(C)
```

Veremos outros exemplos deste tipo de manipulação de séries em próximas explanações deste curso.

Examinemos agora funções que talvez não existam na versão BASIC de seu microcomputador. Quase todas as versões desta linguagem permitem ao programador criar novas funções e isto é quase tão fácil como utilizar funções incorporadas. A instrução DEF indica, em linguagem BASIC, que uma nova função está sendo definida. Eis aqui como definir uma função para cálculo do volume de uma esfera (a fórmula é $V = \frac{4}{3}\pi r^3$, onde r é o raio da esfera e π (pi) a constante aproximadamente igual a 3,1416):

```
10 REM FUNCAO PARA CALCULO DO VOLUME DE
   UMA ESFERA
20 DEF FNV(X) = 4 * 3,1416 * X * X * X/3
30 PRINT "FORNECER O RAIO DA ESFERA"
40 INPUT R
50 PRINT "O VOLUME DA ESFERA DE RAIO ";
   R;" E"
60 PRINT FNV(R)
70 END
```

Este modo de definir uma função é bastante simples, mas vejamos a linha detalhadamente:

```
DEFine o identificador de função
      ↓   ↓
20 DEF FNV(X) = 4* 3,1416 * X * X * X/3
        ↑   ↑
FuNção variável fictícia
```

Quando a função é definida, as letras FN são seguidas de uma letra identificadora — V, no caso da função acima —, que deve ser seguida de uma "variável fictícia". Esta deve também ser usada na defini-

ção da função à direita do sinal de igualdade. Utilizada a função em um programa, qualquer variável numérica pode substituir a variável fictícia da definição.

Em um ponto posterior do programa acima, seria igualmente possível empregar a função "volume de uma esfera" da seguinte forma:

```
999 LET A = 66
1000 LET B = FNV(A)
1010 PRINT B
1020 LET C = 5
1030 LET D = B + FNV(C)
1040 PRINT D
1050 LET G = FNV(16)
1060 PRINT G
```

Algumas versões de BASIC permitem a utilização de múltiplas variáveis na função definida. Desse modo, uma função para calcular a média aritmética de dois números poderia ser escrita assim:

```
100 DEF FNA(B,C) = (B+C)/2
110 INPUT "FORNECER DOIS NUMEROS";B,C
120 LET A = FNA(B,C): REM A FUNCAO "MEDIA"
130 PRINT "A MEDIA DE "; B; " E "; C; " E "; A
```

Observe que a linha 110, acima, combina o equivalente a duas instruções separadas em uma única. A maioria das versões de BASIC imprime automaticamente as palavras que aparecem entre aspas após a instrução INPUT; assim, esta linha equivale a:

```
110 PRINT "FORNECER DOIS NUMEROS"
115 INPUT A,B
```

A linha 120 também consegue o equivalente a duas instruções em uma linha, usando como separador dois pontos (:). Instruções que pertenceriam a linhas separadas podem ser escritas em uma única, desde que cada instrução "independente" seja separada da anterior pelo sinal de dois pontos. Isto economiza espaço em programas extensos, mas seu uso deve ser limitado, pois torna mais difícil a leitura e aumenta a probabilidade de erros.

Respostas aos exercícios da p. 197

Erros

Resultará em uma mensagem "OUT OF DATA ERROR", pois deveria haver doze valores na instrução DATA. Em segundo lugar, surgirá um erro na linha 100, quando o programa tentar endereçar um elemento A(4,1). A linha 100 deveria ser:

```
100 PRINT A(X, Y)
```

Atribuição de valores

Abaixo, um programa que executará o solicitado.
Seu programa poderá estar desenvolvido de modo diferente.

```
10 DIM A(8,13)
20 FOR R=1 TO 7
30 FOR C=1 TO 12
40 READ A(R,C)
50 NEXT C
60 NEXT R
70 REM SOMAR TOTAIS
80 GOSUB 300
90 REM IMPRIMIR DADOS SOLICITADOS
100 GOSUB 200
110 PRINT "OUTROS DADOS?"
120 PRINT "S OU N"
130 INPUT A$
140 IF A$ = "N" THEN GOTO 160
150 GOTO 100
160 END
200 PRINT "QUAL MES?"
210 PRINT "1-PARA JANEIRO,"
220 PRINT "13-PARA TOTAL, ETC"
230 INPUT M
240 PRINT "QUAL DESPESA?"
250 PRINT "1-PARA GASOLINA"
260 PRINT "8-PARA TOTAL, ETC"
270 INPUT X
280 PRINT "O VALOR E ";A(X,M)
290 RETURN
300 FOR R=1 TO 7
310 LET T=0
320 FOR C=1 TO 12
330 LET T=T+A(R,C)
340 NEXT C
350 LET A(R,13)=T
360 NEXT R
370 FOR C=1 TO 13
380 LET T=0
390 FOR R=1 TO 7
400 LET T=T+A(R,C)
410 NEXT R
420 LET A(8,C)=T
430 NEXT C
440 RETURN
500 REM AQUI SEGUEM SEUS DADOS
510 REM OITENTA E QUATRO VALORES
520 REM "DADOS 45000, 50000" ETC
```

A propósito...

Os computadores tipo TK83, CP 200 e compatíveis possuem uma versão não standard do código ASCII, que não permite caracteres em minúsculas; assim, o programa de conversão de minúsculas em maiúsculas não executará essa operação; entretanto, experimente-o e procure em seu manual mais informações sobre o conjunto de caracteres destes equipamentos.

Nos micros VIC-20, Commodore 64 e Apple você não pode escrever mais de uma variável dentro do parênteses.

Nos micros tipo Sinclair (TK83, TK85, CP 200, Ringo), substituir:

ASC(A$) por CODE A$

e substituir:

CHR$(65) por CHR$ 65

Se o argumento for uma expressão, deverá ficar entre parênteses. Mas argumento simples — como A$ e 65 — não precisam de parênteses.

Estas instruções não existem nos micros compatíveis com Sinclair (TKs, CP 200, Ringo).

Em alguns computadores, não há ";" após a mensagem. Neste caso, veja o manual de seu micro, e troque por "," ou mesmo elimine o ";".

Esta instrução não existe em micros do tipo Sinclair (TKs, CP 200, Ringo) e compatíveis TRS-80 (CP 500, Digitus etc.). No entanto, pode-se contornar o problema com instruções que atribuam a uma função os valores a serem calculados.

Em alguns computadores convém alterar IF INKEY$ = " " THEN LET X=2 por IF INKEY$ = " " THEN LET X=1. Nos micros tipo VIC-20 e Commodore 64, por exemplo, deve-se alterar IF INKEY$ = " " THEN LET X=2 por GET A$: IF A$ = " " THEN LET X=1.

Soluções reais

Utilizando todas as técnicas de programação BASIC que aprendemos até aqui, iniciamos agora os primeiros estudos para desenvolver um programa de banco de dados.

Agora que já examinamos muitos dos aspectos fundamentais da linguagem BASIC, é tempo de utilizar o que aprendemos, desenvolvendo um programa verdadeiro. Evidentemente, todos os programas que vimos até agora são "verdadeiros", no sentido de que cumprem uma determinada tarefa, mas constituem apenas ilustrações ou exemplos de como funcionam os vários elementos da linguagem BASIC, e não tipos de programa que você poderá querer utilizar diariamente. Eles ilustram como as "engrenagens" do BASIC são ajustadas de modo a formar um mecanismo simples. Agora passaremos a construir todo o mecanismo, ou seja, a desenvolver um programa completo.

Uma pergunta que o não-usuário de computador faz ao usuário é a seguinte: "Em que o computador pode ser realmente útil?" A questão não é tão simplista quanto parece. A resposta a uma dona-de-casa, por exemplo, tende a ser do tipo: "Bem, você pode colocar suas receitas no computador"; ou "Você pode fazer uma agenda telefônica ou de endereços computadorizada". Esse tipo de resposta não é satisfatório porque geralmente o interlocutor faz a seguinte objeção: "Eu consulto meu livro de receitas quando quero cozinhar e procuro em minha agenda o endereço, sem me dar ao trabalho de escrever um programa que faça isso". Por conseguinte, precisamos reconhecer quando um problema se torna merecedor de uma solução pelo computador, em vez de lhe darmos uma solução convencional. Vejamos o que ocorre no exemplo específico de uma lista de endereços computadorizada.

Agendas comuns de endereços trazem um índice alfabético que possibilita ao usuário encontrar um determinado nome. Os nomes vão sendo acrescentados à medida do necessário e não se pode obedecer à estrita ordem alfabética. A primeira anotação na letra D talvez seja Dimas. Posteriormente, você acrescenta Dulce e Daniela. Embora não colocados em ordem alfabética, estão agrupados na letra D; assim, é fácil encontrar determinado nome iniciado por essa letra. Nem é preciso dizer que, se não for utilizado um índice, será muito trabalhoso tentar localizar o nome desejado.

As outras anotações num livrinho ou caderneta são o endereço e o número do telefone, e às vezes alguns dados pessoais. Entretanto, uma agenda convencional não nos daria uma lista separada de todas as pessoas com as quais nos interessa manter contato em certa cidade ou que podem atender em determinado número telefônico.

É verdade que isso talvez não represente um sério contratempo, mas, se você dirigir uma pequena empresa que atende pedidos pelo correio, disporá de um recurso valioso se tiver informação específica sobre as pessoas que constam de sua lista de endereçamento e correspondência. Por exemplo, se a empresa lançar uma nova coleção de roupas infantis e enviar anúncio a seus clientes, poderá economizar em despesas postais, porque provavelmente não será interessante remeter a publicidade a clientes que não têm filhos. Isso serve apenas para exemplificar detalhes que devem ser levados em conta ao decidir se um problema merece ser tratado por um computador ou ter uma solução convencional.

Sendo o recurso ao computador a solução conveniente, a nova questão é saber se existe ou não para ele software à venda. Observando os anúncios de revistas de computação talvez você conclua que todas as possibilidades em programação já foram cobertas pelos programadores. Entretanto, um exame mais atento poderá mostrar que o programa encontrado não realiza exatamente o que você pretende, ou não é compatível com o seu computador, ou, ainda, é muito caro. O preço dos programas, em geral, reflete o custo do trabalho exigido em seu desenvolvimento. Um pacote processador de palavras pode parecer caro à primeira vista, mas escrever seu próprio programa representará um custo ainda maior, se lhe tomar seis meses de trabalho intensivo.

Um aspecto positivo do desenvolvimento de seu próprio software é que ele vai executar exatamente aquilo que você deseja. Outro fator de relevo está na absoluta satisfação de escrever com êxito um programa de importância para você mesmo.

A elaboração de um programa abrange vários estágios. O primeiro é a compreensão completa do assunto, o que requer a colocação clara do problema — o estágio da Definição do Problema.

O segundo consiste em encontrar um método adequado para a solução. Isto envolve a descrição da forma esperada de entrada e de saída, como uma "descrição de primeiro nível" do problema. Os problemas e as soluções devem ser apresentados nos termos mais amplos possíveis, os quais serão gradualmente elaborados até atingir um estágio em que se possa codificá-los em uma determinada linguagem.

O terceiro estágio é a própria codificação, quando usaremos o BASIC como a nossa linguagem de alto nível, embora possamos utilizar qualquer outra linguagem. Até o estágio final da codificação em BASIC, usaremos uma pseudolinguagem, intermediária entre a linguagem livre e flexível com que cotidianamente nos exprimimos e as estruturas rígidas da linguagem real do computador, como o BASIC.

O método de programação que acabamos de expor é em geral chamado de programação top-down. Ela opera do nível mais elevado — uma apresentação genérica dos objetivos mais globais, passando

por vários níveis de elaboração — até atingir o nível de elaboração detalhada do programa necessário para o início da codificação em linguagem de alto nível. Procuraremos, também, ater-nos aos princípios da chamada "programação estruturada". Estes princípios ficarão claros no decurso deste projeto.

Os estágios incluídos no desenvolvimento do programa podem ser assim resumidos:

1. Definição clara do problema.
2. A forma da entrada e da saída (descrição em primeiro nível).
2.1 Elaboração (descrição em segundo nível).
2.2 Elaboração complementar (descrição de terceiro a enésimo nível).
3. Codificação em linguagem de alto nível.

Antes de iniciar a elaboração de um software de porte, há necessidade de apresentar o problema com clareza — o que não é um trabalho insignificante. Examinemos algumas idéias possíveis para a nossa relação de endereços computadorizada.

Primeiramente, começaremos com uma lista dos aspectos desejados; depois, decidiremos quais deles podem ser realizados com um trabalho moderado de programação. Desejamos, portanto:

1. Encontrar o endereço, número telefônico e anotações, por meio da digitação de um nome.
2. Obter uma lista de nomes, endereços e números de telefone, com a entrada de apenas parte do nome (talvez o sobrenome ou o primeiro nome).
3. Encontrar uma lista de nomes, com endereço e telefone, de determinada área ou cidade.
4. Obter uma relação de todos os nomes que comecem por determinada letra.
5. Conseguir a relação completa de todos os nomes, ordenados alfabeticamente.
6. Acrescentar novos itens, quando necessário.
7. Alterar itens, quando quiser.
8. Eliminar itens, quando desejar.

Vamos supor que o programa da relação de endereços tenha sido escrito. Que forma deveriam tomar as entradas e saídas? Como você gostaria, do ponto de vista de usuário, que o programa funcionasse? Em termos gerais, os programas são dirigidos por menu ou dirigidos por comandos, ou por uma combinação de ambos. No programa dirigido por menu, o usuário recebe sugestões de uma lista de opções (menu), em cada ponto onde uma decisão tiver de ser tomada. Geralmente, a escolha é feita pressionando-se uma única tecla. Em programa dirigido por comandos, o usuário deve digitar palavras ou expressões específicas, sem necessidade de receber sugestões. Alguns programas combinam ambas as técnicas. A vantagem de um programa dirigido por menu é que este será fácil para o iniciante, não familiarizado com seu uso — o que torna o programa um "amigo do usuário" (user friendly). O programa dirigido por comandos apresenta maior rapidez, no caso de usuários mais experientes. Vamos optar por um método dirigido por menu, embora você possa preferir realizar o programa usando rotinas por comandos — a escolha é sua.

Uma vez que o programa se concentrará em uma lista de nomes, a primeira coisa a considerar é a forma que esses nomes deverão assumir. O computador poderá compreender, por exemplo, todos os formatos que damos a seguir?

A. J. P. Pereira
Leonardo da Vinci
Clara NUNES
Liz T.
A. Barroso
o. v. filho
P Pontes
Susana
CHARLIE CHAPLIN
José da Silva

Talvez pareça um detalhe supérfluo, mas imagine o que acontecerá se você fornecer o nome sob a forma P Pontes e depois solicitar ao programa a localização de P. Pontes. A menos que você tenha previsto esta possibilidade, o computador provavelmente responderá com NOME NAO ENCONTRADO.

Há duas maneiras de evitar o problema: fornecer os dados de modo "impreciso", permitindo a entrada dos nomes sob qualquer forma com rotinas bem elaboradas que possam lidar com isso quando as buscas forem feitas; ou fazer a entrada dos nomes de forma estritamente definida. Qualquer nome que não estiver na forma convencionada resultará, neste último caso, em uma mensagem de erro, como NOME EM FORMATO NAO ACEITAVEL, ou algo semelhante. A escolha é arbitrária, mas optaremos por uma entrada de modo "impreciso" e deixaremos ao programa a tarefa de realizar a conversão dos nomes para a forma padrão.

Para uma busca em ordem alfabética, consideremos os nomes como constituídos de duas partes — o sobrenome e o restante. É relativamente fácil definir um sobrenome: qualquer série de caracteres alfabéticos em maiúsculas ou minúsculas terminada por um RETURN e precedida por um espaço (ASCII 32). Aqui se apresenta de imediato um problema: o que acontecerá se dermos entrada ao nome Susana, sem deixar um espaço antes? Provavelmente, o programa o rejeitará por apresentar um formato não previsto. Assim, será melhor mudar nossa definição.

Os nomes incluem uma ou mais partes: o sobrenome ou o sobrenome e o nome. O nome consiste em caracteres maiúsculos ou minúsculos e os sinais de ponto, apóstrofo e hífen. Será sempre iniciado por um caractere alfabético e terminado por um RETURN (ponto final para encerramento da linha não será admitido). Se houver um espaço, o último grupo de caracteres — incluindo apóstrofo e hífen — será considerado sobrenome, e outros elementos, inclusive o espaço, nome. Se não houver espaço, o nome completo será considerado sobrenome.

O sobrenome exige cuidado especial, pois em qualquer busca em ordem alfabética sempre terá precedência em relação ao prenome; desse modo, o nome Alberto Pereira será colocado após Tiago Pavan. Se o nome tiver apenas um único grupo de caracteres, como Soares, Susana, ou for um apelido, como Zeca, para os objetivos de nosso programa será considerado um sobrenome.

Em qualquer pesquisa ou colocação em ordem alfabética, que nome deverá vir primeiro: A. J. P. Pereira ou Alfredo Pereira? A escolha é arbitrária, mas a solução mais simples consistirá em desprezar

os pontos de abreviação e os espaços que precedem o último espaço, e apresentar os nomes como AJP PEREIRA e ALFREDO PEREIRA. Se fizermos assim, AJP e ALFREDO serão considerados prenomes e, desse modo, AJP será colocado antes de ALFREDO.

Parte de nosso programa admite nomes como entrada e apresenta como saída o nome, endereço e número telefônico (note que ainda não examinamos o significado dos termos "endereço" e "número telefônico"). Se fôssemos aceitar nomes fornecidos em um formato impreciso, com conversão interna para um formato padronizado, poderíamos esperar na saída a forma padronizada ou a mesma forma assumida na entrada inicial? A saída mais "amiga do usuário" seria o nome na forma inicial, mas, como veremos, isto tornará o programa complicado.

Como primeira tarefa de programação, suponha que um nome tenha sido atribuído à variável alfanumérica NOME$ e que temos outras duas variáveis, PRENOME$ e SOBRENOME$. Como atribuiremos as partes correspondentes de NOME$ a PRENOME$ e SOBRENOME$? Deixando de lado, por enquanto, o problema de manter um registro na forma inicial em que o nome foi fornecido (para que possa ser recuperado mais tarde, quando necessário). Uma apresentação simples do programa seria:

> Converter todos os caracteres em maiúsculas.
> Eliminar todos os caracteres não-alfabéticos, exceto o espaço final.
> Atribuir à variável SOBRENOME$ todos os caracteres que seguem o último espaço.
> Atribuir à variável PRENOME$ todos os caracteres que precedem o último espaço.

Antes de examinar o modo como este problema pode ser codificado em BASIC, veremos como o processo de programação top-down pode nos levar de uma apresentação muito ampla de nosso objetivo ao ponto em que a codificação em determinada linguagem de programação se torna possível. Você verificará que utilizamos não apenas nomes extensos de variáveis como SOBRENOME$, mas também palavras de comando como BEGIN, LOOP e ENDLOOP. Há estruturas criadas para auxiliar-nos a definir nosso programa; no último estágio de desenvolvimento serão substituídas por comandos equivalentes em BASIC. Entraremos em maiores detalhes quanto a esses comandos e explicaremos por que deslocamos a posição de algumas linhas no próximo fascículo do curso.

1.ª APRESENTAÇÃO DOS OBJETIVOS

INPUT (ENTRADA)
Um nome (em qualquer formato)
OUTPUT (SAÍDA)
1. Um prenome
2. Um sobrenome

1.ª ELABORAÇÃO

1. Read (ler) NOME$
2. Converter todas as letras em maiúsculas
3. Encontrar o último espaço
4. Read (ler) SOBRENOME$
5. Read (ler) PRENOME$
6. Eliminar caracteres não alfabéticos da variável PRENOME$

2.ª ELABORAÇÃO

1. Read (ler) NOME$
2. (Converter todas as letras em maiúsculas)
   BEGIN (INICIAR)
   Executar um LOOP enquanto os caracteres não examinados permanecerem na variável NOME$
      Read (ler) os caracteres de NOME$ a seguir
      IF (SE) o caractere estiver em minúscula,
         THEN (ENTÃO) convertê-lo para maiúscula
         ELSE (CASO CONTRÁRIO) nada executar
      ENDIF (ENCERRAR A SEQÜÊNCIA)
      Atribuir o caractere à variável alfanumérica auxiliar
   ENDLOOP (ENCERRAR O LOOP)
   LET NOME$ = variável alfanumérica auxiliar
   END (ENCERRAR)
3. (Encontrar o último espaço)
   BEGIN (INICIAR)
   Executar um LOOP enquanto os caracteres não examinados permanecerem na variável NOME$
      IF (SE) o caractere = " "
         THEN (ENTÃO) anotar a posição em uma variável
         ELSE (CASO CONTRÁRIO) nada fazer
      ENDIF (ENCERRAR A SEQÜÊNCIA SE-ENTÃO)
   ENDLOOP (ENCERRAR O LOOP)
   END (ENCERRAR)
4. (Ler SOBRENOME$)
   BEGIN (INICIAR)
   Atribuir a SOBRENOME$ os caracteres à direita do último espaço em NOME$
   END (ENCERRAR)
5. (Ler PRENOME$)
   BEGIN (INICIAR)
   Executar um LOOP enquanto os caracteres não examinados permanecerem em NOME$ até o último espaço
      SCAN (EXAMINAR) os caracteres
      IF (SE) o caractere não corresponder a uma letra do alfabeto
         THEN (ENTÃO) nada fazer
         ELSE (CASO CONTRÁRIO) atribuir o caractere a PRENOME$
      ENDIF (ENCERRAR A SEQÜÊNCIA SE-ENTÃO)
   ENDLOOP (ENCERRAR O LOOP)
   END (ENCERRAR)
6. (Eliminar caracteres não alfabéticos de PRENOME$)
   (Isto já foi realizado no item 5)

Este segundo nível de elaboração está agora muito próximo do estágio em que poderá ser codificado em linguagem de programação. Desenvolveremos o segundo item (conversão de letras em maiúsculas) em um terceiro nível de elaboração e em seguida o codificaremos em BASIC. Encontramos, anteriormente, um algoritmo para fazer isso (ver p. 212).

3.ª ELABORAÇÃO

2. (Converter todas as letras em maiúsculas)
   BEGIN (INICIAR)
   READ (LER) NOME$
   LOOP
   FOR = 1 TO comprimento da série de caracteres
      READ (LER) caractere L
      IF (SE) o caractere estiver em minúscula
         THEN (ENTÃO) subtrair 32 do valor ASCII do caractere

```
        ELSE (CASO CONTRÁRIO) nada razer
      ENDIF (ENCERRAR SEQÜÊNCIA SE-ENTÃO)
      LET|SÉRIE TEMPORÁRIA$ = SÉRIE
      TEMPORÁRIA$ + caractere
  ENDLOOP (ENCERRAR O LOOP)
  LET NOME$ = SÉRIE TEMPORÁRIA$
  END (ENCERRAR)
```

Este fragmento de programa em pseudolinguagem está bastante próximo da linguagem de programação, e pode ser codificado. Nossa versão de BASIC da Microsoft não admite que nomes de variáveis apareçam por inteiro, por isso usaremos abreviaturas. Assim, NOME$ passa a ser N$.

```
1000 REM SUB-ROTINA PARA CONVERSAO EM
     MAIUSCULAS
1010 INPUT "ENTRAR NOME"; N$: REM APENAS
     PARA TESTE
1020 LET P$ = "  ": REM ASSEGURAR-SE DE QUE A
     SERIE DE CARACTERES NAO ESTA
     PREENCHIDA
1030 FOR L = 1 TO LEN(N$): REM INDICE DO LOOP
1040 LET T$ = MID$(N$,L,1): REM OBTEM
     CARACTERE
1050 LET T = ASC(T$): REM DETERMINA O VALOR
     ASCII DO CARACTERE
1060 IF T >= 97 THEN LET T = T - 32: REM
     CONVERTER EM MAIUSCULAS
1070 LET T$ = CHR$(T)
1080 LET P$ = P$ + T$: REM P$ E SERIE
     TEMPORARIA
1090 NEXT L: REM ENCERRAMENTO DO LOOP
1100 LET N$ = P$: REM N$ ESTA AGORA TODA EM
     MAIUSCULAS
2000 PRINT N$: REM APENAS PARA TESTE
2010 END: REM APENAS PARA TESTE. SUBSTITUIR
2020 REM "RETORNAR" PARA O PROGRAMA
     PRINCIPAL
```

Este fragmento de programa seria normalmente utilizado como uma sub-rotina a ser chamada de um programa principal. Nós o escrevemos, para finalidade de teste, com uma instrução INPUT, uma instrução PRINT, muitas instruções REM e uma instrução END, as quais deverão ser eliminadas antes que a sub-rotina seja incorporada em um programa completo.

A propósito...

Apesar de não se utilizar letras minúsculas nos equipamentos compatíveis com o Sinclair (TK83 e TK85, CP 200, Ringo, etc.), o programa em questão nestes equipamentos seria:

```
1000 PRINT "FORNECER O NOME"
1010 INPUT N$
1020 LET P$ = "  "
1030 FOR L=1 TO LEN N$
1040 LET T$=N$(L)
1050 LET T=CODE T$
1060 IF T >= 97 THEN LET T = T-32
1070 LET T$ = CHR$ T
1080 LET P$ = P$ + T$
1090 NEXT L
2000 LET N$=P$
2010 PRINT N$
2020 REM RETORNAR PARA O
     PROGRAMA REAL
```

Exercícios

■ Elaborar todos os estágios acima até o ponto adequado à conversão para um programa em BASIC. A "pseudolinguagem" que você utiliza não precisa ser igual à nossa, mas será interessante seguir a convenção de usar letras maiúsculas para palavras-chaves que provavelmente corresponderão às palavras de instrução na linguagem final (por exemplo, LOOP, IF, LET etc.). Use letras minúsculas em operações que precisarão ser apresentadas mais explicitamente, quando forem codificadas. Estas poderão ficar em linguagem comum.

■ Estando os programas desenvolvidos até um nível satisfatório de elaboração, convertê-los em módulos de programas (sub-rotinas) em BASIC. Testá-los individualmente, usando entradas fictícias e imprimir as instruções que puderem ser eliminadas depois, se funcionarem adequadamente.

Exercícios de revisão

Estes exercícios trazem exemplos da maioria das instruções e funções usadas em BASIC. Não há questões muito difíceis nem aspectos não vistos anteriormente. Se fizer todos ou a maioria destes exercícios sem dificuldade, considere-se a caminho de tornar-se um bom programador em BASIC.

1. Escrever um programa que admita uma entrada de dois números pelo teclado, somar e imprimir o resultado.

2. Atribuir duas palavras (séries de caracteres) a duas variáveis alfanuméricas e a seguir criar uma terceira variável alfanumérica que concatene (unifique) as duas palavras iniciais. Imprimir a terceira variável.

3. Escrever um programa que possibilite digitar qualquer palavra e que imprima a seguir a extensão da série na mensagem: A PALAVRA QUE VOCE DIGITOU TEM * CARACTERES (* representa o número).

4. Escrever um programa que admita um único caractere fornecido pelo teclado e que a seguir informe o valor ASCII do caractere (em decimal).

5. Escrever um programa que apresente a mensagem DIGITE UMA PALAVRA e em seguida responda A ULTIMA LETRA DA PALAVRA FOI * (* representa a letra).

6. Escrever um programa que lhe sugira DIGITE O NOME DE UMA PESSOA e depois responda O ESPACO CORRESPONDE AO *.º CARACTERE.

7. Alterar o programa acima de modo a imprimir QUARTO ou QUINTO ao invés de 4.º ou 5.º quando o espaço se apresentar nestas posições.

8. Desenvolver um programa que indique ao usuário DIGITAR UMA SENTENCA e depois responda com a mensagem A SENTENCA QUE VOCE DIGITOU TEM * PALAVRAS (supondo que o número de palavras será uma unidade maior que o número de espaços na sentença).

9. Teste seu computador a fim de verificar se os caracteres (ou gráficos especiais) foram atribuídos aos valores ASCII de 128 a 255 (utilizar um loop e a função CHR$(X)).

Campos e registros

Continuando o projeto de programação para desenvolvimento de uma agenda de endereços computadorizada, veremos agora como um arquivo pode ser subdividido em registros e campos.

Na parte anterior do curso de programação em linguagem BASIC, deixamos como tarefa a elaboração dos elementos do exercício de programação através de um ou mais níveis de "pseudolinguagem", até o ponto em que os exemplos pudessem ser codificados em BASIC. Iniciaremos hoje pela revisão desse exercício e pela sugestão de algumas soluções possíveis. A primeira "Apresentação dos objetivos", por exemplo, foi:

ENTRADA (INPUT)
Um nome (em qualquer formato)
SAÍDA (OUTPUT)
1. Um prenome
2. Um sobrenome

Em nosso primeiro nível de elaboração, verificamos que estes objetivos poderiam ser subdivididos em seis estágios (posteriormente vimos que o último deles podia ser eliminado). Eram os seguintes:

1. Ler o nome (*LER*)
2. Converter todas as letras em maiúsculas (*CONVERTER*)
3. Encontrar o último espaço (*ESPAÇO*)
4. Ler o sobrenome (*LERSOBRENOME*)
5. Ler o prenome (*LERPRENOME*)
6. Eliminar os caracteres não alfabéticos do prenome

Estamos tratando todos esses procedimentos como sub-rotinas e os nomes a elas atribuídos são fornecidos entre parênteses. Infelizmente, a maioria das versões de BASIC não permite a chamada de sub-rotinas por nome e será necessário, quando o programa chegar ao estágio final de desenvolvimento, inserir números de linha após as instruções GOSUB correspondentes. Durante a fase de desenvolvimento, todavia, é muito mais fácil fazer referência a sub-rotinas por meio de nomes. Esses nomes poderão depois ser incorporados nas instruções REM. Estamos indicando essa utilização de sub-rotinas com nomes colocando-os entre asteriscos. Em linguagens que podem chamar sub-rotinas pelos nomes (como a PASCAL), esse tipo de sub-rotina é denominado "procedimento".

Apesar de a linguagem BASIC não poder operar com procedimentos, é interessante você supor que isso é possível, enquanto a programação ainda estiver no estágio de pseudolinguagem. De modo semelhante, sua versão de BASIC pode não ser adequada para lidar com nomes longos de variáveis, como CONTAR ou NOMEDARUA$, mas em nível de pseudolinguagem é mais conveniente designá-los desse modo. Tente fazer com que eles sejam descritivos. Fica muito mais claro chamar uma variável auxiliar, para uma série de caracteres, de VARIAVELAUX$ do que chamá-la de VX$. Felizmente, muitas das versões de BASIC agora permitem nomes mais longos de variáveis.

Também desenvolvemos o segundo dos estágios (converter todas as letras em maiúsculas) através de um segundo e terceiro níveis de elaboração e criamos um pequeno programa para executar esta tarefa. Agora, tentaremos o mesmo com relação aos outros estágios:

2.ª ELABORAÇÃO
3. (Encontrar o último espaço)
INICIAR
Executar um LOOP enquanto permanecerem caracteres não examinados na variável NOME$
  IF (SE) o caractere = " "
    THEN (ENTÃO) registrar a posição em uma variável
    ELSE (CASO CONTRÁRIO) nada executar
  ENDIF (ENCERRAR)
ENDLOOP (ENCERRAR O LOOP)
END (ENCERRAR)

3.ª ELABORAÇÃO
3. (Encontrar o último espaço)
INICIAR
READ (LER) NOMECOMPLETO$
Executar um LOOP (executado enquanto permanecerem caracteres não examinados)
  FOR L=1 para a extensão de NOMECOMPLETO$
  READ (LER) o caractere proveniente de NOMECOMPLETO$
  IF (SE) o caractere = " "
    THEN LET (ENTÃO) CONTAR = posição do caractere
    ELSE (CASO CONTRÁRIO) nada executar
  ENDIF (ENCERRAR)
ENDLOOP (ENCERRAR O LOOP)
END (ENCERRAR)

Estamos agora em condição de codificar, da pseudolinguagem para a linguagem de programação:

```
10 INPUT "FORNECER NOME COMPLETO ";
   NOMECOMPLETO$
20 FOR L=1 TO LEN (NOMECOMPLETO$)
30 LET CARACT$ = MID$
   (NOMECOMPLETO$,L,1)
40 IF CARACT$ = " " THEN LET CONTAR = L
50 NEXT L
60 PRINT "O ULTIMO ESPACO ESTA NA
   POSICAO ";CONTAR
70 END
```

Observe que a linha 10 é uma entrada fictícia para testar a rotina; a linha 60 é uma saída fictícia, também para teste; e a instrução da linha 70 deverá ser

alterada para RETURN, quando a rotina for utilizada como sub-rotina.

Vamos tentar agora o mesmo procedimento no estágio quatro:

```
2.ª ELABORAÇÃO
4. (Ler sobrenome)
INICIAR
Atribuir a SOBRENOME$ os caracteres à direita do
último espaço
END (ENCERRAR)

3.ª ELABORAÇÃO
4. (Ler sobrenome)
INICIAR
   READ (LER) NOMECOMPLETO$
   Localizar o último espaço (chamar a sub-rotina
   *ESPAÇO*)
   Executar o LOOP enquanto os caracteres
   permanecerem na série, após o espaço
      READ (LER) os caracteres e acrescentá-los a
      SOBRENOME$
   ENDLOOP (ENCERRAR O LOOP)
END (ENCERRAR)
```

Antes de iniciar a codificação em BASIC, você deve observar alguns enganos possíveis. Ao localizar o último espaço, na última elaboração acima apresentada, a pseudolinguagem exige a utilização da sub-rotina *ESPAÇO*, mas não será possível escrevê-la em BASIC e testá-la, se a sub-rotina *ESPAÇO* já não tiver sido escrita. De modo geral, não é interessante codificar cada módulo em BASIC (ou qualquer outra linguagem de alto nível) antes de o programa ser desenvolvido em pseudolinguagem. Todavia, se você deseja testar um módulo, talvez seja necessário escrever alguns valores fictícios de variáveis, bem como algumas entradas e saídas fictícias. No exemplo acima, CONTAR é a variável que contém o número da posição do último nome, na variável NOMECOMPLETO$. Ao testar, podemos trapacear um pouco com a suposição de que a rotina funciona de modo adequado:

```
10 LET NOMECOMPLETO$ = "TOM JOBIM"
20 LET CONTAR = 4
30 FOR L = CONTAR + 1 TO LEN
   (NOMECOMPLETO$)
40 LET SOBRENOME$ = SOBRENOME$ + MID$
   (NOMECOMPLETO$,L,1)
50 NEXT L
60 PRINT "O SOBRENOME E "; SOBRENOME$
70 END
```

Segue-se agora o processo para encontrar o prenome (estágio cinco). Lembre-se de que definimos o prenome como uma concatenação de todos os caracteres alfabéticos, até o último espaço do nome. Sinais de ponto, apóstrofos, espaços etc. tiveram de ser eliminados.

```
2.ª ELABORAÇÃO
5. (Ler prenome)
INICIAR
Executar o LOOP até o último espaço, enquanto
os caracteres permanecerem em
NOMECOMPLETO$
   Examinar os caracteres
      IF (SE) o caractere não for uma letra
         THEN (ENTÃO) nada executar
         ELSE (CASO CONTRÁRIO) acrescentar
         o caractere a PRENOME$
      ENDIF (ENCERRAR)
   ENDLOOP (ENCERRAR O LOOP)
END (ENCERRAR)

3.ª ELABORAÇÃO
5. (Ler prenome)
INICIAR
Executar o LOOP até CONTAR, enquanto
permanecerem os caracteres
   LET CARACTAUX$ = elésimo caractere na série
   IF (SE) CARACTAUX$ não for uma letra
      THEN (ENTÃO) nada executar
      ELSE (CASO CONTRÁRIO) LET PRENOME$ =
      PRENOME$ + CARACTAUX$
   ENDIF (ENCERRAR)
ENDLOOP (ENCERRAR O LOOP)
```

Agora temos condição de codificar em BASIC, mas, como estágio intermediário, utilizaremos instruções BASIC, não numeradas, em um formato estruturado, de modo que você possa comparar a estrutura com o estágio acima:

```
CODIFICAÇÃO
5. (Ler prenome)
REM INICIAR
REM LOOP
   FOR L = 1 TO CONTAR - 1
      LET CARACTAUX$ = MID$
      (NOMECOMPLETO$,L,1)
      LET CARACT = ASC(CARACTAUX$)
      IF CARACT>64 THEN PRENOME$ =
         PRENOME$ + CHR$(CARACT)
      REM ENDIF
   NEXT L: REM ENDLOOP
REM END
```

Em linguagem BASIC, isto seria:

```
10 FOR L = 1 TO CONTAR - 1
20 LET CARACTAUX$ =
   MID$(NOMECOMPLETO$,L,1)
30 LET CARACT = ASC(CARACTAUX$)
40 IF CARACT>64 THEN PRENOME$ = PRENOME$
   + CHR$(CARACT)
50 NEXT L
60 END
```

Entretanto, do modo como se apresenta, o programa não funcionará. Há três problemas com ele: a variável CONTAR precisa receber um valor; não está prevista a entrada dos nomes (atribuição de uma série de caracteres à variável NOMECOMPLETO$); e não há "saída" na forma de uma instrução para impressão, de modo que o usuário possa verificar se o programa funcionou de forma adequada.

Se essa rotina fosse parte de uma sub-rotina, os parâmetros a ela transferidos (a entrada) e os parâmetros dela provenientes (a saída) teriam de ser manipulados em outra parte do programa. Esta é uma questão importante: o fluxo dos dados no interior do programa deve ser sempre avaliado com cuidado antes do início da codificação em BASIC. Isso é especialmente importante quando utilizamos variáveis (CONTAR, por exemplo) e o mesmo nome de variá-

vel é usado em diversas partes do programa. Não adianta chamar uma sub-rotina que utiliza uma variável como CONTAR, se a sub-rotina não possui meios de conhecer o valor que ela supostamente possui. Se uma sub-rotina atribui um valor inicial à variável CONTAR, esse valor permanecerá o mesmo, a menos que um novo valor seja atribuído posteriormente — talvez em outra sub-rotina. Este é o motivo por que não é boa prática de programação saltar do meio de um loop, uma vez que o valor da variável no loop será desconhecido. Examine as conseqüências de deixar esses dois fragmentos de programa como partes de diferentes sub-rotinas em um programa:

Parte da sub-rotina X

```
FOR L = 1 TO LEN(PALAVRA$)
LET CARACT$ = MID$(PALAVRA$,L,1)
IF CARACT$ = " . " THEN GOTO 1550
NEXT L
```

Parte da sub-rotina Y

```
FOR Q=1 TO LIMITE
LET A(L) = P(Q)
NEXT Q
```

Esta parte da sub-rotina Y lê valores em uma tabela indexada, na qual o índice é a variável L. Se a sub-rotina Y for chamada após a sub-rotina X e se a condição de teste na sub-rotina X tiver sido obtida (que um dos caracteres seja um " . "), o valor da variável L será completamente imprevisível; assim, não saberemos a que elementos da tabela os valores estarão sendo atribuídos na sub-rotina Y. Além do erro de desviar para fora do loop, esta sub-rotina também utiliza uma instrução GOTO, procedimento que também é desaconselhável. Instruções GOTO dificultam muitas vezes a compreensão do programa e devem ser utilizadas com cuidado.

A fim de evitar confusões, um bom procedimento é fazer uma lista de todas as variáveis, nos estágios de pseudolinguagem, acompanhada de observações que informem para que estão sendo utilizadas. Algumas linguagens (mas não o BASIC) permitem a apresentação das variáveis como "locais" ou "globais", isto é, possuem valores que são utilizados em apenas uma parte do programa (locais) ou através de todo o programa (globais). Muitas variáveis, como as utilizadas nos loops (por exemplo, o L, na instrução LET L = 1 TO 10), são quase sempre locais; desse modo, é geralmente conveniente dar um valor inicial à variável, antes de sua utilização (por exemplo, LET L = 0). Algumas linguagens, como PASCAL, exigem isto; e, embora o BASIC sempre assuma que o valor inicial da variável é 0 (a menos que um outro seja especificado), ainda assim é recomendável fornecer o valor inicial.

Até aqui formulamos uma definição razoável de um nome, para as finalidades de nossa agenda de endereços computadorizada, e desenvolvemos algumas rotinas que podem lidar com nomes de vários modos, e que serão utilizadas em nosso programa completo. Agora vamos novamente nos afastar dos detalhes de codificação do programa e examinar a estrutura dos "registros" em nosso "arquivo" da agenda de endereços.

Os termos "registro", "arquivo" e "campo" têm significado relativamente específico na área de computação. *Arquivos* são conjuntos completos de dados relacionados. Em sistemas de computação, correspondem aos itens identificáveis armazenados em disco flexível (disquete) ou em fita cassete, com um nome determinado. Podemos considerar toda a nossa agenda de endereços como um arquivo e a denominaremos AGEEND.

No interior dos arquivos, temos *registros*, que são também conjuntos de dados relacionados. Se imaginarmos nossa agenda de endereços como uma caixa de fichário, o arquivo será a caixa repleta de cartões e os registros serão os cartões individuais — cada qual com seu próprio nome, endereço completo e telefone.

Em cada registro temos *campos*. Os campos podem ser considerados como uma ou mais linhas de dados relacionados, no interior do registro. Cada um dos registros em nosso arquivo AGEEND terá os seguintes campos: NOME, ENDEREÇO e TELEFONE. Um exemplo de registro é o seguinte:

```
Pedro Lameira
Rua Antonio Celso 73
São Paulo
SP
011-5402588
```

Neste registro há três campos: o campo do nome, que inclui as letras do alfabeto (e, talvez, o apóstrofo, como Pedro d'Ávila); o campo do endereço, que inclui alguns números e muitas letras; e o campo do telefone, que inclui apenas números, desprezando-se os hifens, como neste exemplo: 011-258-1191). Antes de começar a escrever um programa que lide, de maneira flexível, com informação complexa como esta, precisamos determinar o modo de representar os dados no interior do computador. Uma maneira pode ser considerar toda a informação no interior de um registro como sendo simplesmente uma longa série de caracteres. O problema com este tipo de procedimento é que a obtenção de dados específicos é extremamente difícil. Vamos admitir que a seguinte entrada é apenas uma longa série de caracteres:

```
PERCIVAL R. BURTON
1056 AVENUE OF THE AMERICAS
RIO DEL MONTENEGRO
CALIFORNIA
U.S.A.
(415) 884 5100
```

Se percorrermos os registros para encontrar o número telefônico de PERCIVAL R. BURTON, será seguro supor que os 14 últimos caracteres do registro representam o número? O que acontecerá se incluirmos o código de discagem direta internacional, da seguinte forma: 0101 (415) 884 5100? O número teria então um total de 19 caracteres. Para superar esta dificuldade, é atribuído ao número telefônico um campo separado, e o programa nos fornecerá todos os caracteres (ou números) nesse campo, quando for solicitado.

A dificuldade com este tipo de procedimento está na necessidade de algum meio para relacionar os vários campos separados, de modo que a referência a um campo (o campo do nome, por exemplo) possa igualmente nos fornecer os outros campos do registro. Um meio de resolver isso consiste em possuir

um campo complementar vinculado ao registro apenas para finalidades de indexação. Se um registro for, por exemplo, o 15.º do arquivo, o campo para o índice deverá conter o número 15. Isto então poderia ser utilizado para indicar os elementos em uma série de tabelas. Para ilustrá-lo, suponha que um registro se apresente da seguinte forma:

Jaime Fonseca	campo do NOME
Rua Feliciano 59	campo do ENDEREÇO
Sorocaba	campo da CIDADE
SP	campo do ESTADO
0152322303	campo do TELEFONE
015	campo do ÍNDICE

Se soubermos o nome da pessoa e desejarmos o número de seu telefone, teremos apenas de percorrer os elementos da tabela, mantendo os nomes, até que seja encontrado o correspondente. Procuraremos, a seguir, aquele elemento da tabela no qual está o nome — neste caso, o número 15. Assim, tudo que nos resta fazer será encontrar o 15.º elemento da tabela, TELEFONE, para obter o número correto do telefone.

Se tivermos amigos na região Nordeste do país, poderemos querer que o programa detecte cada ocorrência do nome "Recife" no campo CIDADE. O programa examina os campos CIDADE e registra a posição de cada ocorrência do nome Recife. Assim, para a impressão dos nomes e endereços desses amigos, será necessário unicamente recuperar todos os elementos que têm o mesmo número em todas as tabelas para cada registro "Recife". A utilização deste procedimento elimina a necessidade de pesquisar o campo ÍNDICE, e a técnica apresenta a vantagem de ser uma operação relativamente simples.

Na próxima parte do nosso curso, veremos alguns problemas vinculados à pesquisa de listas para encontrar itens específicos.

Exercícios

■ Admitir que os registros com os seguintes campos serão adequados à nossa agenda de endereços computadorizada:

campo do NOME
campo do ENDEREÇO
campo da CIDADE
campo do ESTADO
campo do TELEFONE

Supor que uma das opções oferecidas por um menu na agenda de endereços é a seguinte:

5. CRIAR UMA NOVA ENTRADA

Você digita o número 5 e o programa desvia para a parte em que novos registros são criados (você pode admitir que ainda não há entradas na agenda de endereços). Uma vez que o programa é completamente dirigido por menu, você estará sempre sendo advertido para as entradas previstas — com indicações como FORNECER O NOME, FORNECER A RUA e assim sucessivamente. Eis aqui uma lista dos resultados esperados.

1. Um elemento em uma tabela para o nome
2. Um elemento em uma tabela para o endereço
3. Um elemento em uma tabela para a cidade
4. Um elemento em uma tabela para o estado
5. Um elemento em uma tabela para o telefone

Sua tarefa é desenvolver estes procedimentos, por um processo de programação top-down e usando uma pseudolinguagem, até o ponto em que a conversão direta para o BASIC se torne possível. A pseudolinguagem pode seguir as regras que você estabelecer; sugerimos apenas o emprego de letras maiúsculas para palavras-chaves, como IF, LOOP etc., e de letras minúsculas para descrições em linguagem comum das operações a serem efetuadas.

A propósito...

Apresentaremos, a partir de agora, uma relação dos principais comandos, instruções e funções do BASIC. Serão descritas instruções de quatro versões dessa linguagem: o BASIC da Microsoft, muito utilizado em microcomputadores comerciais; a versão Applesoft, encontrada nos micros compatíveis com o Apple II (Micro Engenho, Unitron, TK2000, Elppa etc.); a versão TRS-80, disponível em micros como CP 300, CP 500, DGT 1000 e outros; e a versão Sinclair ZX81, compatível com as linguagens dos micros TK83, TK85, CP 200, Ringo e outros.

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
ABS(x)	Dá o valor absoluto de x	+	+	+	+
ACS(x)	Dá o arco-cosseno em radianos de x				+
APPEND "arquivo"	Abre um arquivo e posiciona o pointer no final do mesmo		+		
ASC(x$)	Dá o código ASCII do primeiro caractere de x$	+	+	+	
ASN(x)	Dá o arco-seno em radianos de x				+
ATN(x)	Dá o arctangente (em radianos) de x	+	+	+	+
AUTO número, incremento	Gera número de linhas automaticamente	+		+	
BEEP	Executa um bip no alto-falante	+			
BLOAD "arquivo", saída	Carrega dados binários na memória	+	+		
BSAVE "arquivo", saída	Grava dados binários	+	+		
CALL variável	Chama um programa em linguagem de máquina	+			
CDBL(x)	Converte x para precisão dupla	+		+	
CHAIN "arquivo"	Chama um programa e lhe transfere variáveis	+			

Novas entradas

Para inserir um novo item em uma matriz é necessário primeiro encontrar um espaço em branco. A busca binária é um modo eficiente de consegui-lo.

Vimos na parte anterior como um arquivo de dados é constituído de registros, que são divididos em campos, podendo cada um deles ter acesso a outros, através de um campo de indexação. Vejamos agora algumas técnicas de pesquisa disponíveis.

Não é difícil a criação de registros para nossa agenda de endereços. Suponhamos que exista uma variável alfanumérica para cada um dos campos no registro. Estes podem ser denominados NOMECOMP$ (para referência a "nome completo"), RUA$, CIDADE$ e TEL$ (posteriormente explicaremos nossa utilização aqui de variáveis alfanuméricas, em vez de variáveis numéricas para o campo do número telefônico). Na lista das oito funções necessárias ao programa da agenda de endereços, a de número seis possibilitava a inclusão de novas entradas. Se essas oito opções forem apresentadas na tela, no início do processamento do programa, a escolha da função de número seis levará você a uma rotina de entrada semelhante à apresentada como exercício.

Admitamos que haja uma série de entradas na agenda de endereços, mas você não pode se lembrar de quantas. É fundamental que as novas entradas não se sobreponham às já existentes; desse modo, uma das tarefas do programa poderá ser a de buscar os elementos em uma das matrizes, a fim de encontrar o primeiro deles que não contenha dados.

A busca em uma matriz para verificar se um elemento está "ocupado" não é difícil. Como as variáveis numéricas, também as variáveis alfanuméricas podem ser comparadas, na linguagem BASIC. A instrução IF A$ = "RESIDÊNCIA" THEN... é tão válida como IF A = 61 THEN..., pelo menos na maioria das versões do BASIC. Se qualquer das matrizes em nossa agenda de endereços já tiver uma entrada, esta consistirá em pelo menos um caractere alfanumérico. Os elementos "vazios" não conterão caracteres alfanuméricos; desse modo, tudo que necessitamos fazer é examinar os elementos, desde o primeiro, até encontrarmos um que não contenha caracteres.

Se houver matrizes para nome, rua, cidade e telefone, teremos quatro tabelas, com um elemento em cada, para cada campo do registro. Uma vez que todos esses campos "estão em correspondência", o 15.º registro terá seu item "nome" no 15.º elemento da matriz "nome"; o item "rua", no 15.º elemento da matriz "rua"; o item "cidade", no 15.º elemento da matriz "cidade"; e o item "telefone", no 15.º elemento da matriz "telefone". Desse modo, necessitamos apenas examinar uma dessas matrizes para encontrar um elemento vazio; não precisamos verificar todas as matrizes.

Se a variável POSIÇÃO representar o número do primeiro elemento não ocupado em qualquer uma das matrizes, o programa para localizar POSIÇÃO (supondo-se que ela não seja ainda conhecida) poderá ser simplesmente o seguinte:

```
PROCEDIMENTO (encontrar o elemento não ocupado)
INICIAR
  LOOP
    REPEAT UNTIL (REPETIR ATÉ) que o elemento
    não ocupado seja localizado
    READ (LER) a matriz (POSIÇÃO)
    POSIÇÃO = POSIÇÃO + 1
    IF (SE) a matriz (POSIÇÃO) = " "
      THEN (ENTÃO) registrar a POSIÇÃO
      ELSE (CASO CONTRÁRIO) nada executar
    ENDIF
  ENDLOOP
END
```

Em BASIC, isto seria simplesmente o seguinte:

```
1000 FOR L = 0 TO 1 STEP 0
1010 LET POSICAO = POSICAO + 1
1020 IF NOMECOMP$ (POSICAO) = " " THEN LET
  L = X
1030 NEXT L
1040 REM resto do programa
```

Observe que o valor da variável X na linha 1020 corresponde ao valor exigido para o encerramento do loop FOR-NEXT e este valor varia de acordo com a máquina utilizada (os compatíveis Sinclair exigem que as variáveis sejam predefinidas). É igualmente importante notar que este é um fragmento de programa que supõe que a variável 1 NOMECOMP$( ) está DIMensionada e que foi dado um valor inicial à variável POSIÇÃO. Para processar este fragmento independente, você deve DIMensionar a variável NOMECOMP$( ) e dar um valor inicial às variáveis POSIÇÃO e X, em algum ponto, antes da linha 1000.

Embora já tenhamos utilizado a técnica FOR X = 0 TO 1 STEP 0, esta é uma boa ocasião para examinar mais detalhadamente seu modo de operar. Geralmente, um loop FOR-NEXT na linguagem BASIC "sabe" com antecedência quantas vezes o fragmento do programa deverá repetir-se. Se você quiser repetir alguma coisa 30 vezes, a forma FOR X = 1 TO 30 será muito conveniente. Todavia, desta vez, estamos simulando um loop REPEAT...UNTIL. Apesar de as versões comuns do BASIC não terem procedimento REPEAT...UNTIL, é bastante fácil simulá-lo usando a seqüência FOR-NEXT. Enquanto o teste da linha 1020 for negativo, a variável L (o contador do loop FOR-NEXT) permanecerá com o valor 0, sendo-lhe acrescentado o valor 0 a cada interação (repetição do loop), enquanto a linha 1010 fará com que a variável POSIÇÃO seja aumentada em uma unidade a cada interação. Quando o teste da linha 1020 apresentar o resultado "verdadeiro" (isto é, quando for encontrado um elemento vazio da variável

NOMECOMP$( ), a variável L assumirá o valor X e o loop FOR-NEXT será encerrado na linha 1030. Isso faz com que a variável POSIÇÃO indique o primeiro elemento livre na variável NOMECOMP$( ).

POSIÇÃO é uma variável cujo valor provavelmente precisará ser determinado no início, toda vez que o programa da agenda de endereços for utilizado, e que também necessitará ser atualizada diversas vezes durante a utilização do programa. Assim, ela será uma das nossas variáveis "globais" e o estabelecimento de seu valor deverá ser parte de uma rotina de "inicialização". Isto pode ser feito toda vez que o programa for processado, ou um "flag" pode ser criado para indicar se o valor da variável POSIÇÃO foi alterado ou não desde o último processamento do programa. Esta última abordagem não é difícil; porém, neste estágio, cria uma complexidade desnecessária. Para deixar simples o procedimento, nossa primeira tarefa será encontrar o valor da variável POSIÇÃO, sempre que o programa for processado.

Vamos revisar os procedimentos que desejamos que a agenda de endereços computadorizada execute, examinando também se é possível chegar a uma estratégia global de programa. Desta vez, seremos um pouco rigorosos, supondo que cada uma das atividades será tratada como uma sub-rotina separada (cujo nome virá indicado entre asteriscos).

1. Encontrar um registro (pelo nome)	*ENCREG*
2. Encontrar nomes (a partir de nomes incompletos)	*ENCNOMES*
3. Encontrar registro (a partir da cidade)	*ENCNOMCID*
4. Encontrar registros (a partir de iniciais)	*ENCINICIAL*
5. Listar registros (todos)	*LISTREGS*
6. Acrescentar registro	*ACRESCREG*
7. Modificar registro	*MODREG*
8. Eliminar registro	*ELIMREG*
9. Sair do programa (gravar)	*SAIRPROGR*

Sabemos agora, em termos gerais, quais são as "entradas" e "saídas" do programa que desejamos; desse modo, podemos começar a pensar em termos de programa principal. Todos os detalhes podem ser executados pelo processo de programação "top-down" e codificados em várias sub-rotinas. Também sabemos que deveremos dar um valor inicial a vários elementos, inclusive o valor da variável POSIÇÃO. Sabemos igualmente que, como o programa é dirigido por menu, receberemos um conjunto de escolhas toda vez que o programa for processado. Sabemos ainda que, qualquer que seja a resposta às opções apresentadas, desejaremos que pelo menos uma delas seja executada.

Assim, o corpo do programa principal já pode assumir uma forma:

PROGRAMA PRINCIPAL
INICIAR
  ATRIBUIR UM VALOR INICIAL (procedimento)
  SAUDAÇÃO (procedimento)
  ESCOLHA (procedimento)
  EXECUÇÃO (procedimento)
ENCERRAR

O programa acima seria escrito em BASIC da seguinte forma (com os números de linha substituídos para os nomes das sub-rotinas):

```
10 REM PROGRAMA AGENDA DE ENDERECOS
20 GOSUB *INICIALIZAR*
30 GOSUB *SAUDACAO*
40 GOSUB *ESCOLHA*
50 GOSUB *EXECUCAO*
60 END
```

A sub-rotina ou procedimento *SAUDAÇÃO* apresenta, por alguns segundos, uma saudação na tela seguida pelo menu. A saudação provavelmente terá a seguinte forma:

```
*BEM-VINDO AO*
*MICROCOMPUTADOR — CURSO BASICO*
*AGENDA DE ENDERECOS COMPUTADORIZADA*
(PRESSIONAR A BARRA ESPACEJADORA, QUANDO
PRONTO PARA CONTINUAR)
```

Em resposta à solicitação para pressionar a barra espacejadora, o programa desviará para a sub-rotina *ESCOLHA* e a tela se apresentará ao usuário da seguinte forma:

```
*VOCE DESEJA*
1. ENCONTRAR UM REGISTRO (pelo nome)
2. ENCONTRAR NOMES (por trecho de um nome)
3. ENCONTRAR REGISTROS (de uma cidade)
4. ENCONTRAR REGISTROS (de uma inicial)
5. LISTAR TODOS OS REGISTROS
6. ACRESCENTAR UM REGISTRO
7. MODIFICAR UM REGISTRO
8. ELIMINAR UM REGISTRO
9. SAIR E GRAVAR
        *ESCOLHER DE 1 A 9*
        *SEGUIDO DE RETURN*
```

Neste ponto, o programa desviará para a sub-rotina correspondente, de acordo com o número fornecido. A estrutura do programa começa agora a tomar forma. Todas as alternativas, exceto a de número 9 (para SAIR e GRAVAR), necessitarão encerrar-se com uma instrução para retornar à sub-rotina *ESCOLHA*. Mas há muitos detalhes de organização interna dos dados que não examinamos. Trataremos deles posteriormente.

Vamos admitir que estamos processando o programa, que ele já possui todos os registros de que precisamos e que desejamos encontrar um registro completo, dando entrada a um único nome. Isto exige a opção 1 — ENCONTRAR UM REGISTRO (*ENCREG*). Antes de tentar elaborar esta parte do programa, examinemos alguns dos problemas vinculados às rotinas de busca.

Busca

Manuais sobre técnicas de programação tendem a tratar conjuntamente dos procedimentos de busca e ordenação (sort). Os leitores devem lembrar-se de que já falamos sobre o procedimento de ordenação em um programa elaborado para dispor nomes em ordem alfabética (ver p. 134). Tanto o procedimento de busca como o de ordenação levantam aspectos interessantes sobre como os dados são organizados — em computador ou em outro sistema de informação.

Se uma agenda de endereços comum for constituída por uma caderneta sem índice alfabético e se os itens forem sendo acrescentados à medida que se quiser incluir novos nomes e endereços, sem ordem alfabética, teremos uma estrutura de dados denominada "pilha". Pilhas são conjuntos de dados reunidos segundo a ordem de chegada. É evidente que a forma de pilhas é a menos eficiente para organizar dados. Para encontrar o endereço e o número do telefone de alguém, seria preciso procurar em toda a agenda. O mesmo, geralmente, acontece com os sistemas de computação, embora haja ocasiões em que os critérios de acesso aos dados sejam tão incomuns que o procedimento por pilhas pode mostrar-se conveniente.

Uma estrutura de dados mais organizada e de mais fácil utilização, tanto para os usuários como para o próprio computador, é obtida pela organização dos dados de acordo com um método conhecido e simples. A lista telefônica é um bom exemplo de conjunto de dados (nomes, endereços e números telefônicos) em que o campo do nome é ordenado de acordo com as regras simples da seqüência alfabética. Os números, para todos os efeitos, são ordenados aleatoriamente, mas os nomes — que são mais "significativos" —, organizados de acordo com regras fáceis de serem observadas.

Uma vez examinada a organização interna dos dados em nossa agenda de endereços computadorizada, os dados são organizados como uma pilha, armazenando um primeiro registro no elemento × da matriz nome, elemento × da tabela rua, e assim sucessivamente; e o registro seguinte, no elemento × + 1 da matriz nome, e o elemento × + 1 da matriz rua e assim por diante. A busca de um determinado item de dados — por exemplo, LUIZ SILVA — exigirá a observação do primeiro elemento da matriz nome a fim de verificar se é LUIZ SILVA; repetir esse procedimento com o segundo elemento e assim sucessivamente, até que tenhamos localizado o campo ou até a constatação de que não existe o item LUIZ SILVA.

Se os dados que desejamos encontrar já foram ordenados em uma estrutura identificável, poderemos verificar como esta ordenação simplificará a busca. Suponhamos que você tenha um banco de dados para times de futebol e que um dos campos dos registros seja a contagem de gols em uma semana determinada. Um banco de dados de alta capacidade lhe permite buscar o time ou times que marcaram 11 gols nessa semana. Eis uma tabela que apresenta as contagens de gols para cada time na semana em questão:

1,6,2,2,1,9,0,0,2,1,4,11,4,2,12,5,2,1,0,1

Evidentemente, os gols estão dispostos pela ordem de time e não da própria contagem. São vinte times e apenas um deles efetivamente marcou 11 gols nessa semana: o 12.º time da tabela. No caso de dados não estruturados como estes, o único modo de encontrar a informação que você deseja consiste em examinar o primeiro elemento e verificar se corresponde ao valor 11; se não, repetir o procedimento com o segundo elemento, e assim por diante até o encontro do elemento de valor 11 ou a constatação da inexistência desse valor na tabela.

Analisando esses dados, vemos que há um total de vinte escores que variam de 0 a 12. Esse exemplo é simples e, mesmo que tenhamos de examinar cada item, logo descobriremos que o valor 11 é o 12.º elemento apresentado na matriz. Porém, o que acontecerá se houver milhares de elementos constituindo uma grande matriz? O exame de um grande número de itens de dados não ordenados tornará o programa muito lento e inconveniente.

A solução está em ordenar primeiramente os dados, de modo que as buscas sejam feitas com rapidez. Eis aqui outra vez a matriz de contagens, agora disposta em ordem numérica:

0,0,0,1,1,1,1,1,2,2,2,2,2,4,4,5,6,9,11,12

Sabendo que o total de times é vinte, o modo mais rápido de determinar a posição do escore desejado consiste em dividir a matriz em duas partes e buscar unicamente a parte em que é provável que o número procurado se encontre. Lembre-se: a busca em grandes quantidades de dados tem a possibilidade de tomar muito mais tempo do que as operações aritméticas simples, como a divisão de um número por dois. O algoritmo para localização do escore se apresentará da seguinte forma:

```
Encontrar a matriz que contém os escores
Ler o número que desejamos encontrar
Determinar a extensão da matriz
Determinar o ponto intermediário da matriz
Executar um loop até o número ser localizado
    Se o item no ponto intermediário for igual
    ao número que estamos buscando, então o
    número foi localizado
    Se não, verificar se o número procurado é
    maior ou menor que o número no ponto
    intermediário
    Se o número buscado for maior que o
    número no ponto intermediário, então
    determinar o ponto intermediário da parte
    superior da matriz
    Se o número exigido for menor que o
    número no ponto intermediário, então
    determinar o ponto intermediário da parte
    inferior da matriz
    (Repetir esse procedimento até que o
    número seja localizado)
```

Esse procedimento pode assumir a seguinte formalização:

```
INICIAR
  Encontrar a matriz das contagens
  INPUT (FORNECER) O NÚMERO (buscado)
  Executar um LOOP até que o número seja
  localizado
   IF (SE) O NÚMERO = (ponto intermediário)
    THEN (ENTÃO) registrar a posição do ponto
    intermediário
    ELSE (CASO CONTRÁRIO)
      IF (SE) O NÚMERO > (ponto intermediário)
        THEN (ENTÃO) encontrar o ponto
        intermediário da metade superior
        ELSE (CASO CONTRÁRIO) encontrar o
        ponto intermediário da metade inferior
      ENDIF
    ENDIF
  ENDLOOP
```

```
    SE O NÚMERO tiver sido localizado
      THEN (ENTÃO) PRINT (IMPRIMIR) a posição do
      ponto intermediário
      ELSE (CASO CONTRÁRIO) PRINT (IMPRIMIR)
      "NÚMERO NÃO ENCONTRADO"
    ENDIF
  END
```

Se você refletir sobre este programa em pseudolinguagem, vai notar que ele afinal localizará o número em questão, se este se encontrar na matriz. Desenvolveremos esta pseudolinguagem até o ponto em que possamos chegar a um programa coerente. Este processo de pesquisa por subdivisões sucessivas é denominado "busca binária".

Para você experimentar, apresentamos um programa baseado em BASIC ainda que em pseudolinguagem. A finalidade do programa é criar uma matriz e ler os escores, a partir de uma instrução DATA; a tarefa seguinte é indicar a pesquisa do escore. Se este for encontrado, o programa imprimirá o elemento da matriz em que o número for encontrado.

```
10 REM PROGRAMA PARA LOCALIZAR UM
   NÚMERO EM UMA MATRIZ
20 DIM ESCORES(20)
30 FOR Z = 1TO 20
40 READ ESCORES(Z)
50 NEXT Z
60 DATA 0,0,0,1,1,1,1,1,2,2,2,2,2,4,4,5,6,9,11,12
70 LET L = 20
80 LET BTM = 1
90 LET TP = L
100 INPUT "ENTRAR ESCORE ";N
110 FOR Z = 0 TO 1 STEP 0
120 LET L = TP - BTM
130 LET MD = BTM + INT(L/2)
140 IF N = ESCORES(MD) THEN LET Z = X
150 IF N > ESCORES(MD) THEN LET BTM = MD
160 IF N < ESCORES(MD) THEN LET TP = MD
170 NEXT Z
180 PRINT "O ESCORE ESTÁ NO ELEMENTO
    NO. ";MD
190 END
```

Observe ainda que a variável X tem de receber um valor inicial, de acordo com as exigências de sua máquina particular (como os compatíveis Sinclair, por exemplo).

Se os dados presentes em um arquivo ou matriz tiverem uma distribuição normal, como acontece com as listas telefônicas, nas quais os nomes são distribuídos de acordo com a ordem alfabética, a busca binária será um método eficiente de encontrar um determinado item. Todavia, não é o mais eficiente e há mesmo algoritmos alternativos que podem encontrar dados de modo mais rápido. Tal procedimento constitui a técnica de "informação não significativa", na qual o programa realiza um cálculo aproximado da posição de entrada, refinando-o até que essa posição seja encontrada. Entretanto, tais métodos vão além dos propósitos deste curso e o método de busca binária é suficiente para o que precisamos.

Exercícios

Se você processar este programa, ele funcionará, desde que seja fornecido um escore que esteja presente na tabela. Fornecendo um escore de valor 3, por exemplo, que não se encontra na tabela, o programa não chegará a se encerrar e não será apresentada nenhuma mensagem de erro. Se você digitar o número 12, que se encontra na tabela, o programa não poderá localizá-lo. O programa também supõe que todos os números na tabela ordenada serão diferentes, mas, como você pode observar pela apresentação dos dados, vários números se apresentam mais de uma vez. O programa não detecta essa eventualidade nem registra todas as posições em que o número aparece.

Sua tarefa consiste em:

1. Analisar o programa e descobrir por que ele não pode localizar o escore de valor 12.
2. Alterar uma linha do programa para corrigir essa deficiência.
3. Determinar o motivo por que o programa não pode lidar com números que não se encontram na tabela e criar um procedimento que supere essa deficiência.

Na página 235 deste nosso curso apresentamos uma série de exercícios de revisão a fim de ajudá-lo a avaliar seu progresso no curso de programação em BASIC. Veja as soluções na página 280.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
CHR$(n)	Dá o caractere código ASCII n	+	+	+	+
CINT(x)	Converte x em um inteiro com arredondamento	+		+	
CIRCLE(x,y),r	Desenha um círculo com centro (x,y) e raio r	+			
CLEAR	Zera as variáveis do programa	+	+	+	+
CLOAD "arquivo"	Carrega um programa da fita			+	
CLOSE#arquivo	Fecha um arquivo	+	+		
CLS	Limpa a tela	+		+	+
CODE(x$)	Dá o código ASCII do primeiro caractere de x$				+
COLOR=x	Define a cor a ser usada em gráfico		+		
COMMON lista de variáveis	Passa variáveis a um programa encadeado	+			

Respostas aos exercícios

Qual foi seu desempenho nos exercícios da página 235? Eis algumas soluções possíveis, embora você possa ter encontrado métodos alternativos, com os mesmos resultados.

Na página 235, apresentamos nove problemas, com o objetivo de testar sua habilidade na utilização das instruções e funções geralmente encontradas na linguagem BASIC. Aqui estão as soluções que sugerimos.

Se você acompanhou o curso de programação em BASIC até agora, deve ter percebido que os exercícios são semelhantes a problemas vistos e resolvidos anteriormente. Suas soluções podem ser diferentes das que aqui indicamos. Quase nunca há apenas um método de resolver um problema; o seu pode ser tão bom ou melhor que o nosso.

Se achou difíceis as questões e as soluções, leia novamente a partir da página 149 e estude as soluções indicadas, à medida que for avançando. Se você resolveu bem os exercícios 2, 4, 6 e 8, compreendeu a maior parte das lições apresentadas.

```
100 REM EXERCICIO DE REVISAO N.º 1
200 INPUT "DIGITAR QUALQUER NUMERO";A
300 INPUT "DIGITAR OUTRO NUMERO";B
400 LET C = A + B
500 PRINT "A SOMA DOS NUMEROS E ";C
```

Nos computadores compatíveis Sinclair, colocar um PRINT ''mensagem'' antes do INPUT.

```
100 REM EXERCICIO DE REVISAO N.º 2
200 LET A$ = "PRIMEIRAPALAVRA,"
300 LET B$ = "SEGUNDAPALAVRA"
400 LET C$ = A$ + B$
500 PRINT C$
```

```
100 REM EXERCICIO DE REVISAO N.º 3
200 INPUT "DIGITAR QUALQUER PALAVRA";P$
300 LET C = LEN(P$)
400 PRINT "A PALAVRA QUE VOCE DIGITOU TEM ";C;
    "CARACTERES"
```

```
100 REM EXERCICIO DE REVISAO N.º 4
200 PRINT "PRESSIONAR QUALQUER TECLA"
300 FOR C = 0 TO 1 STEP 0
400 LET A$ = INKEY$
500 IF A$ < > " " THEN LET C = 2
600 NEXT C
700 PRINT "O VALOR ASCII DE " ;A$; " E " ;ASC(A$)
```

Veja a seção ''A propósito...'', nas pp. 175 e 215. Nos computadores Sinclair, substituir a linha 700 por:

```
700 PRINT "O VALOR ASCII DE " ;A$; " E " ; CODE(A$)
```

```
100 REM EXERCICIO DE REVISAO N.º 5
200 INPUT "DIGITAR QUALQUER PALAVRA";P$
300 LET U$ = RIGHT$(P$,1)
400 PRINT "O ULTIMO CARACTERE DA PALAVRA
    E ";U$
```

Veja ''A propósito...'', na p. 149.

Este exercício nos compatíveis Sinclair é resolvido assim:

```
100 REM EXERCICIO DE REVISAO N.º 5
150 PRINT "DIGITAR QUALQUER PALAVRA"
200 INPUT P$
250 LET N = LEN P$
300 LET U$ = P$(N)
400 PRINT "O ULTIMO CARACTERE DA PALAVRA
    E ";U$
```

```
100 REM EXERCICIO DE REVISAO N.º 6
200 PRINT "DIGITAR UM NOME NA FORMA:"
300 PRINT "PRIMEIRONOME SEGUNDONOME"
400 PRINT "EXEMPLO: ROBERTO PACHECO"
500 INPUT "NOME ";N$
600 LET S = 0:LET L = LEN(N$)
700 FOR P = 1 TO L
800 IF MID$(N$,P,1) = " " THEN LET S = P
900 NEXT P
950 PRINT "O ESPACO CORRESPONDE AO ";S; "0.
    CARACTERE"
```

Nos compatíveis Sinclair, colocar PRINT ''mensagem'' antes do INPUT, trocar LEN(N$) por LEN$ e veja ''A propósito...'', na p. 149, para substituir a linha 800, acima, por:

```
800 IF N$(P) = " " THEN LET S = P
```

```
100 REM EXERCICIO DE REVISAO N.º 7
200 PRINT "DIGITAR UM NOME NA FORMA:"
300 PRINT "PRIMEIRONOME SEGUNDONOME"
400 PRINT "EXEMPLO: ROBERTO PACHECO"
500 INPUT "NOME ";N$
600 LET S = 0:LET L = LEN(N$)
700 FOR P = 1 TO L
800 IF MID$(N$,P,1) = " " THEN LET S = P
900 NEXT P
920 IF S = 4 THEN LET X$ = "QUARTO"
940 IF S = S THEN LET X$ = "QUINTO"
950 PRINT "O ESPACO CORRESPONDE AO ";X$;
    "CARACTERE"
```

```
100 REM EXERCICIO DE REVISAO N.º 8
200 INPUT "DIGITAR UMA SENTENCA ";S$
300 LET C = 1
400 FOR P = 1 TO LEN(S$)
500 IF MID$(S$,P,1) = " " THEN LET C = C + 1
600 NEXT C
700 PRINT "A SENTENCA QUE VOCE DIGITOU
    TEM ";C;" PALAVRAS"
```

Veja ''A propósito...'', na p. 149. Nos computadores compatíveis Sinclair, substituir a linha 500, acima, por:

```
500 IF S$(P) = " " THEN LET C = C + 1
```

```
100 REM EXERCICIO DE REVISAO N.º 9
200 FOR C = 128 TO 255
300 PRINT "O CARACTERE NO. ";C;"=";CHR$(C)
400 REM UM PEQUENO INTERVALO AQUI
500 FOR D = 1 TO 500
600 NEXT D
700 REM ENCERRAMENTO DO INTERVALO
800 NEXT C
```

Ver as observações anteriores para os micros tipo Sinclair.

Elaboração do programa

À medida que um programa extenso é desenvolvido, sua estrutura adquire a aparência de uma árvore, com maior número de ramificações em cada estágio de refinamento.

No último fascículo, em nosso curso de programação em BASIC, analisamos alguns dos problemas envolvidos no exame de uma lista para encontrar um determinado item, partindo da premissa de que a lista já estava organizada segundo uma determinada ordem. Trata-se de uma questão que reexaminaremos com maiores detalhes no momento em que iniciarmos o desenvolvimento das rotinas de busca. Nesse meio tempo, porém, vamos desenvolver o tema da programação top-down: a produção de código para as duas segundas partes do programa principal. Esse procedimento inclui quatro chamadas para sub-rotinas ou procedimentos:

```
PROGRAMA PRINCIPAL
INICIAR
  INICIALIZAR (procedimento)
  SAUDAÇÃO (procedimento)
  ESCOLHA (procedimento)
  EXECUÇÃO (procedimento)
ENCERRAMENTO (END)
```

O primeiro procedimento, *INICIALIZAR*, exige inúmeras atividades de razoável complexidade — constituição de matrizes, incorporação de dados a elas, execução de várias verificações, e assim por diante —, de modo que deixaremos os detalhes dessas várias etapas para exame posterior. As duas partes subseqüentes do programa principal abrangem os procedimentos de SAUDAÇÃO e ESCOLHA. Ao desenvolver esses procedimentos, sugerimos uma metodologia para ajudar a evitar que se tornem desorganizadas e confusas as muitas camadas incluídas na programação top-down.

A dificuldade no desenvolvimento do programa com o refinamento top-down está na imprevisibilidade do número de estágios necessários até atingirmos o ponto em que é possível a codificação em linguagem de alto nível. Para procedimentos simples, talvez dois ou três estágios sejam suficientes, mas procedimentos mais complexos poderão exigir muitos estágios, antes de o problema ser suficientemente analisado para permitir a conversão em "código fonte" (conforme é chamado o programa em linguagem de alto nível). Isso significa que escrever um programa por esse método assemelha-se a desenhar uma árvore tombada. Conforme as ramificações aumentam (isto é, à medida que os refinamentos se tornam mais detalhados), elas passam a ocupar mais espaço na página. Por fim, torna-se impossível colocar tudo em uma única página e é este o ponto em que se corre o risco de perder a noção do que está acontecendo.

Um modo muito eficiente de organizar a documentação do programa consiste em numerar sistematicamente os estágios de seu desenvolvimento. Usamos números romanos para indicar o nível de refinamento, e números arábicos para indicar as subseções do programa. Uma folha separada de papel é empregada para cada nível de refinamento, e as páginas para cada bloco de programa ou módulo podem facilmente ser mantidas juntas. Apresentamos abaixo o sistema de numeração para nosso programa:

```
I PROGRAMA PRINCIPAL
INICIAR
  1. INICIALIZAR
  2. SAUDAÇÃO
  3. ESCOLHA
  4. EXECUÇÃO
ENCERRAR (END)
```

Conforme dissemos antes, por ora deixamos de lado o procedimento INICIALIZAR, concentrando-nos no desenvolvimento dos procedimentos SAUDAÇÃO e ESCOLHA.

```
II 2 (SAUDAÇÃO)
INICIAR
  1. Apresentar mensagem de saudação
  2. LOOP (até ser pressionada a barra de espaço)
     ENCERRAR O LOOP (ENDLOOP)
  3. Chamar rotina *ESCOLHA*
ENCERRAR (END)
```

```
III 2 (SAUDAÇÃO) 1 (apresentar mensagem)
INICIAR
  1. Limpar a tela
  2. IMPRIMIR (PRINT) mensagem de saudação
ENCERRAR (END)
```

```
III 2 (SAUDAÇÃO) 2 (LOOP para esperar a barra de espaço)
INICIAR
  1. LOOP (até ser pressionada a barra de espaço)
        SE (IF) a barra de espaço for pressionada
        ENTÃO (THEN)
     ENCERRAR O LOOP (ENDLOOP)
ENCERRAR (END)
```

```
III 2 (SAUDAÇÃO) 3 (chamar *ESCOLHA*)
INICIAR
  1. GOSUB *ESCOLHA*
ENCERRAR (END)
```

Até aqui deve estar claro que os níveis **III-2-1** e **III-2-3** estão prontos para ser codificados diretamente em BASIC, mas o nível **III-2-2** necessita de outro estágio de refinamento:

```
IV 2 (SAUDAÇÃO) 2 (LOOP)
INICIAR
  1. LOOP (até ser pressionada a barra de espaço)
        SE (IF) INKEY$ não for pressionada, ENTÃO
        (THEN) continue
     ENCERRAR O LOOP (ENDLOOP)
ENCERRAR (END)
```

Agora atingimos o ponto em que toda a codificação em BASIC para o procedimento SAUDAÇÃO pode prosseguir com pouco refinamento adicional:

IV 2 (SAUDAÇÃO) 1 (apresentar mensagem) CÓDIGO BASIC

```
REM SUB-ROTINA *SAUDAÇÃO*
PRINT
PRINT
PRINT
PRINT
PRINT TAB(14); "*BENVINDO AO*"
PRINT TAB(4); "*MICROCOMPUTADOR: CURSO
      BÁSICO*"
PRINT TAB(1); "*AGENDA DE ENDEREÇOS
      COMPUTADORIZADA*"
PRINT
PRINT "(PRESSIONAR BARRA DE ESPAÇO PARA
      CONTINUAR)"
```

V 2 (SAUDAÇÃO) 2 (LOOP para esperar a barra de espaço) CÓDIGO BASIC

```
LET L = 0
FOR L = 1 TO 1
IF INKEY$ < > " " THEN LET L = 0
NEXT L
```

IV 2 (SAUDAÇÃO) 3 (chamar *ESCOLHA*) CÓDIGO BASIC

```
GOSUB *ESCOLHA*
RETURN
```

Observe que começamos agora a inicializar as variáveis nas diferentes rotinas que escrevemos, usando instruções da forma LET I = 0. A rigor, esta precaução é desnecessária em algumas das situações em que foi aqui empregada. Contudo, representa um bom hábito a ser adquirido, caso consiga lembrar-se, e se houver suficiente capacidade de RAM. Há três motivos para isso: primeiramente, porque colocar uma lista de instruções LET no início de qualquer rotina serve como um lembrete útil das variáveis específicas que a rotina está usando. Em segundo lugar, porque você não pode ter certeza do valor que permaneceu na variável desde a última vez em que foi utilizada em uma rotina (embora isso nem sempre tenha importância). Por fim, como explicaremos mais adiante no curso, a colocação de instruções do tipo LET I = 0 na ordem correta pode acelerar a execução do programa.

Modificamos o modo de usar o loop FOR-NEXT para simulação da estrutura DO-WHILE ou REPEAT-UNTIL, em relação aos fascículos anteriores do curso. Em vez de empregar os loops FOR I = 0 TO 1 ou FOR I = 0 to 1 STEP 0, agora utilizamos o loop FOR I = 1 to 1. Esta instrução será corretamente processada em todos os microcomputadores que costumamos apresentar, enquanto outros métodos exigem as adaptações para os diversos equipamentos, apresentadas no quadro "A propósito...". A instrução FOR I = 1 TO 1-NEXT I executará o loop uma única vez. No entanto, se em algum ponto do loop a variável I receber o valor 0, o loop será executado novamente, e assim por diante. Podemos inserir uma instrução LET I = 0 como resultado de uma condição de saída que não corresponda ao valor esperado, ou podemos atribuir o valor 0 à variável I imediatamente após a instrução FOR, atribuindo-lhe o valor 1, caso o contrário ocorra. Desse modo, ambos os loops abaixo alcançam o mesmo objetivo:

```
FOR I = 1 TO 1
IF INKEY$ < > " " THEN LET I = 0
NEXT I
```

ou

```
FOR I = 1 TO 1
LET I = 0
IF INKEY$ = " " THEN LET I = 1
NEXT I
```

Para o bloco SAUDAÇÃO completo no programa principal, necessitamos apenas do código em BASIC que acabamos de apresentar. Não colocamos os números de linha porque não podemos realmente fazê-lo antes de todos os módulos do programa ficarem prontos para a codificação final. Por exemplo, não se sabe neste estágio quais são os números de linha apropriados para os comandos GOSUB. Se você quiser testar o módulo neste estágio, será necessário criar entradas e sub-rotinas fictícias. Alguns pontos precisam ser observados quanto a este fragmento de programa: o emprego da função TAB e as instruções de "limpeza de tela". A função TAB move o cursor ao longo da linha, de acordo com o número (o "argumento") especificado entre parênteses. Com os números que fornecemos, a mensagem será impressa claramente, centralizada em uma tela de 40 caracteres de largura. Se a tela de seu equipamento tiver menor capacidade ou maior (computadores maiores têm capacidade para 80 caracteres), será necessário fazer as devidas alterações nesses argumentos TAB. A instrução para limpar a tela em muitas das versões do BASIC corresponde ao comando CLS, mas a versão BASIC da Microsoft usada para desenvolver este programa não admite seu emprego. Por isso, usamos a instrução PRINT CHR$(12), uma vez que nossa máquina usa o código ASCII 12 como caractere de não-impressão para "limpeza de tela". Outros equipamentos empregam o código ASCII 24 com a mesma função.

```
10 REM PROGRAMA PRINCIPAL FICTICIO
20 PRINT CHR$(12)
30 GOSUB 100
40 END
100 REM SUB-ROTINA *SAUDACAO*
110 PRINT
120 PRINT
130 PRINT
140 PRINT
150 PRINT TAB(14);"*BENVINDO AO*"
160 PRINT TAB(4);"*MICROCOMPUTADOR: CURSO
    BASICO*"
170 PRINT TAB(1);"*AGENDA DE ENDERECOS
    COMPUTADORIZADA*"
180 PRINT
190 PRINT "(PRESSIONAR BARRA DE ESPACO PARA
    CONTINUAR)"
195 LET L = 0
200 FOR L = 1 TO 1
210 IF INKEY$ < > " " THEN LET L = 0
220 NEXT L
230 PRINT CHR$(12)
240 GOSUB 1000
250 RETURN
1000 REM SUB-ROTINA FICTICIA
1010 PRINT "SUB-ROTINA FICTICIA"
1020 RETURN
```

Agora utilizaremos exatamente a mesma abordagem para elaborar o procedimento ESCOLHA.

```
II 3 (ESCOLHA)
INICIAR
  1. IMPRIMIR (PRINT) o menu
  2. FORNECER (INPUT) ESCOLHA
  3. Chamar a sub-rotina ESCOLHA
ENCERRAR (END)

III 3 (ESCOLHA) 1 (IMPRIMIR (PRINT) o menu)
INICIAR
  1. Limpar a tela
  2. IMPRIMIR (PRINT) o menu com opções
ENCERRAR (END)

III 3 (ESCOLHA) 2 (FORNECER (INPUT) ESCOLHA)
INICIAR
  1. FORNECER (INPUT) ESCOLHA
  2. Verificar se a ESCOLHA está dentro do limite
ENCERRAR (END)

III 3 (ESCOLHA) 3 (chamar ESCOLHA)
INICIAR
  1. OPÇÃO ESCOLHIDA
     ENCERRAR A OPÇÃO
ENCERRAR (END)
```

III-3-1 (IMPRIMIR (PRINT) o menu) pode agora ser codificado em BASIC:

```
IV 3 (ESCOLHA) 1 (IMPRIMIR (PRINT) o menu) CÓDIGO BASIC
REM LIMPAR A TELA
PRINT CHR$(12): REM OU 'CLS'
PRINT
PRINT
PRINT
PRINT
PRINT "1. ENCONTRAR UM REGISTRO
      (PELO NOME)"
PRINT "2. ENCONTRAR NOMES (POR TRECHO
      DE UM NOME)"
PRINT "3. ENCONTRAR REGISTROS (DE UMA
      CIDADE)"
PRINT "4. ENCONTRAR REGISTROS (DE UMA
      INICIAL)"
PRINT "5. LISTAR TODOS OS REGISTROS"
PRINT "6. ACRESCENTAR UM REGISTRO"
PRINT "7. MODIFICAR UM REGISTRO"
PRINT "8. ELIMINAR UM REGISTRO"
PRINT "9. SAIR E GRAVAR"
```

Todavia os níveis **III-3-2 (FORNECER ESCOLHA)** e **III-3-3 (chamar ESCOLHA)** exigem refinamento complementar. Vejamos primeiro o nível subseqüente de desenvolvimento, o **III-3-2**.

A atribuição de um valor numérico à variável ESCOLHA é extremamente simples: após a indicação, um comando FORNECER ESCOLHA realizará essa tarefa. Entretanto, há apenas nove escolhas possíveis. O que acontecerá se, por engano, fornecermos o valor 0 ou 99? Uma vez que a ESCOLHA feita vai determinar que parte do programa será chamada a seguir, é preciso garantir que não haja erros. Assim, devemos executar um procedimento de "verificação de limite", que consiste em uma pequena rotina que confere se o número fornecido está no limite aceitável, antes de dar prosseguimento ao programa. Eis, a título de exemplo, uma rotina para detectar entradas errôneas:

ROTINA PARA VERIFICAR O LIMITE

```
1 REM ROTINA
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTRADA 1-9";ESCOLHA
40 IF ESCOLHA < 1 THEN LET L = 0
50 IF ESCOLHA > 9 THEN LET L = 0
60 NEXT L
70 PRINT "A ESCOLHA FOI ";ESCOLHA
80 END
```

Muitas das versões do BASIC podem simplificar esta rotina com a inclusão de um operador booleano no teste, da seguinte forma:

```
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTRAR 1-9";ESCOLHA
40 IF ESCOLHA < 1 OR ESCOLHA > 9 THEN LET L = 0
50 NEXT L
60 PRINT "A ESCOLHA FOI ";ESCOLHA
70 END
```

Essas rotinas exemplificam igualmente outro aspecto da instrução INPUT. Esta instrução faz com que o programa seja interrompido e espere a entrada de um item pelo teclado. O BASIC só reconhecerá que o número foi fornecido por inteiro se a tecla RETURN for pressionada; assim, é necessário que você se lembre de pressionar a tecla RETURN após dar entrada ao número.

Um procedimento mais interessante para o usuário consiste em fazer com que o programa continue imediatamente após o fornecimento de um número válido. Para isto, usamos a função INKEY$, e o BASIC lerá um caractere do teclado toda vez que a função INKEY$ for encontrada. O programa, no entanto, não será interrompido, passando para a unidade seguinte, sem realizar um intervalo. Assim sendo, é comum o emprego da função INKEY$ no interior dos loops. O loop para verificar o pressionamento da tecla pode ser IF INKEY$ = " " THEN..., ou seja, caso a tecla pressionada corresponda a "nada" (isto é, se nenhuma tecla for pressionada), retornar e verificar novamente. Um loop que serviria a nosso objetivo seria o seguinte:

```
LET I = 0
FOR I = 1 TO 1
LET A$ = INKEY$
IF A$ = "  " THEN LET I = 0
NEXT I
```

O único inconveniente no emprego da função INKEY$ é que ela fornece um caractere do teclado, em vez de um caractere numérico. Quando há uma estrutura em que é feita uma entre várias escolhas (um desvio multicondicional), é mais fácil, em BASIC, usar números em lugar de caracteres. É neste ponto que as funções NUM ou VAL do BASIC mostram sua utilidade. Os números apresentados em séries de caracteres são convertidos em números "reais" (isto é, valores numéricos e não códigos ASCII, que representam numerais). São utilizadas da seguinte forma:

```
LET N = VAL(A$) ou LET N = NUM(A$)
```

Mediante as funções NUM ou VAL, podemos fazer com que o programa converta as entradas em variáveis numéricas, usando a função INKEY$. Este procedimento elimina a necessidade de usarmos a tecla RETURN depois de apertarmos a tecla do número. Todavia também é aconselhável a verificação fora do limite.

O fragmento do programa, abaixo, inclui dois loops, um alojado no interior do outro. O loop interno espera o pressionamento da tecla; o externo converte a série em um número e verifica se está dentro do limite:

```
FOR L = 1 TO 1
  PRINT "ENTRAR ESCOLHA (1-9)"
    FOR I = 1 TO 1
      LET A$ = INKEY$
      IF A$ "  " THEN LET I = 0
    NEXT I
  LET ESCOLHA = VAL(A$)
    IF ESCOLHA < 1 THEN LET L = 0
    IF ESCOLHA > 9 THEN LET L = 0
NEXT L
```

Por fim, reproduzimos um programa completo em BASIC para o módulo *ESCOLHA*, inclusive a entrada fictícia e as sub-rotinas para fins de teste. Ressaltamos, ainda uma vez, que os números de linha têm apenas objetivo de teste, e deverão ser remanejados quando o programa definitivo for montado.

```
10 PRINT CHR$(12)
20 PRINT "VOCE DESEJA"
30 PRINT
40 PRINT
50 PRINT
60 PRINT "1. ENCONTRAR UM REGISTRO (PELO
        NOME)"
70 PRINT "2. ENCONTRAR NOMES (POR TRECHO
        DE UM NOME)"
80 PRINT "3. ENCONTRAR REGISTROS (DE UMA
        CIDADE)"
90 PRINT "4. ENCONTRAR REGISTROS (DE UMA
        INICIAL)"
100 PRINT "5. LISTAR TODOS OS REGISTROS"
110 PRINT "6. ACRESCENTAR UM REGISTRO"
120 PRINT "7. MODIFICAR UM REGISTRO"
130 PRINT "8. ELIMINAR UM REGISTRO"
140 PRINT "9. SAIR E GRAVAR"
150 PRINT
160 PRINT
170 LET L = 0
180 LET I = 0
190 FOR L = 1 TO 1
200 PRINT "ESCOLHER DE 1 A 9"
210 FOR I = 1 TO 1
220 LET A$ = INKEY$
230 IF A$ = "  " THEN LET I = 0
240 NEXT I
250 LET ESCOLHA = VAL(A$)
260 IF ESCOLHA < 1 THEN LET L = 0
270 IF ESCOLHA > 9 THEN LET L = 0
280 NEXT L
290 ON ESCOLHA GOSUB
    310,330,350,370,390,410,430,450,470
300 END
310 PRINT "SUB-ROTINA FICTICIA 1"
320 RETURN
330 PRINT "SUB-ROTINA FICTICIA 2"
340 RETURN
350 PRINT "SUB-ROTINA FICTICIA 3"
360 RETURN
370 PRINT "SUB-ROTINA FICTICIA 4"
380 RETURN
390 PRINT "SUB-ROTINA FICTICIA 5"
400 RETURN
410 PRINT "SUB-ROTINA FICTICIA 6"
420 RETURN
430 PRINT "SUB-ROTINA FICTICIA 7"
440 RETURN
450 PRINT "SUB-ROTINA FICTICIA 8"
460 RETURN
470 PRINT "SUB-ROTINA FICTICIA 9"
480 RETURN
```

No próximo fascículo examinaremos estruturas em arquivo e iniciaremos o refinamento do procedimento INICIALIZAR.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
CONT	Continua a execução do programa	+	+	+	+
COS(x)	Dá o co-seno de x (x em radianos)	+	+	+	+
CSAVE "arquivo"	Grava um programa em fita			+	
CSNG(x)	Converte x para precisão simples	+		+	
DATA lista de dados	Cria uma tabela de dados a ser usada pelo READ	+	+	+	
DEF FN nome(arg)=expressão	Define uma função numérica ou alfanumérica	+	+		
DEF SEG=endereço	Define um segmento da memória	+			
DEFDBL lista de variáveis	Define as variáveis com precisão dupla			+	
DEFINT lista de variáveis	Define as variáveis como sendo inteiras			+	
DEFSNG lista de variáveis	Define as variáveis com precisão simples			+	
DEFSTR lista de variáveis	Define as variáveis como sendo alfanuméricas			+	

Ampliação de arquivos

Estabelecida uma estrutura geral, continuamos nosso projeto de programação examinando a manipulação de arquivos.

A agenda de endereços computadorizada que desenvolvemos nos últimos fascículos do curso de programação em BASIC é, na verdade, um tipo de banco de dados simples e, como tal, envolve o conceito de "arquivo". Este termo é empregado de vários modos relacionados, mas ligeiramente diferentes. Começaremos por analisá-lo com mais detalhes, a fim de que possamos depois empregá-lo com maior precisão.

Em linguagem de computação, um "arquivo" pode ser visto como muito semelhante a uma estante de arquivamento. Consiste em uma coletânea de unidades de dados armazenados em conjunto. Os computadores armazenam arquivos provenientes de fitas magnéticas ou de discos. Cada "arquivo" de dados recebe um nome individual, de modo que o computador possa acessá-lo, sempre que necessário. Os dados armazenados em uma fita cassete ou em disco flexível podem ser programas ou "dados" utilizados em programas. Tomando como exemplo a agenda de endereços computadorizada, os dados necessários constituem duas partes separadas: o programa em si e os dados com que o trabalha. O programa é o conjunto de instruções que possibilitam ao computador (e ao usuário) lidar e operar com os dados.

Os dados empregados pelo programa são o conjunto de registros que contém as informações que se espera encontrar em uma agenda — nomes, endereços etc. Também incluem certos tipos de dados aos quais o usuário normalmente não tem acesso. São os dados de "preparo inicial" utilizados pelo programa como auxiliar para o processamento. Exemplos desse tipo de dados são: "flags"; as informações relacionadas ao tamanho efetivo do banco de dados (isto é, o número de registros nele contidos); se foi feita ou não alguma ordenação desde a última inserção de um novo registro; ou, possivelmente, uma indicação sobre quantas vezes determinado registro foi acessado ou impresso. O motivo por que dados como esses — e os que incluem os registros — devem ser operados independentemente do programa se tornará evidente tão logo comecemos a elaborar o programa.

Nos primeiros estágios do curso de programação em BASIC, empregamos as instruções READ e DATA como métodos para introduzir os dados no interior do programa. Esse procedimento só é conveniente se os dados não estiverem sujeitos a alterações, como o número de dias em um mês. Se os dados estiverem sujeitos a mudanças, o programa poderá indicá-lo na tela e empregam-se as instruções INPUT, INKEY$ ou outros métodos para fornecer dados ao programa. Um exemplo do uso adequado deste método de fornecimento de dados é o jogo de adivinhação de números, no qual parte do programa pode assumir a seguinte forma:

```
PRINT "PENSE EM UM NUMERO"
INPUT N
IF N < > NUMCOMP THEN...
```

No entanto, os dados do programa da agenda de endereços estão sujeitos a grande número de alterações.

Teoricamente, todos os registros podem ser armazenados no programa e fornecidos para as matrizes correspondentes, usando-se as instruções READ e DATA. Mas todos os dados do registro teriam de ser fornecidos como parte do programa. Sempre que fossem feitas alterações — nomes e endereços acrescentados ou eliminados, por exemplo — seriam necessárias modificações consideráveis. Isso exigiria, no mínimo, imprimir o programa, verificar onde as alterações deveriam ser realizadas, desenvolver novos segmentos de programa e então digitá-los. O maior problema, entretanto, estaria em que os novos segmentos não seriam módulos completos de programa que pudessem ser testados independentemente — as alterações estariam espalhadas ao acaso por todo o programa. O único modo de saber se o programa alterado funcionaria de forma conveniente seria processá-lo.

Felizmente, nenhum desses procedimentos é necessário, porque os dados podem ser armazenados de modo independente do programa. Isso é feito com a criação de arquivos de dados em fita cassete ou em disco. Esses arquivos são coletâneas de registros tratados de modo muito semelhante ao dos dados nas instruções DATA. O programa pode "abrir" um ou mais desses arquivos, extrair os dados neles incluídos (geralmente de uma matriz) e, a seguir, "fechar" o arquivo. Sendo necessária uma alteração dos dados, o programa abre um arquivo apropriado, extrai os dados, modifica-os e, em seguida, registra-os outra vez assim alterados no arquivo.

Para os equipamentos de computação que operam com discos, a localização de um arquivo específico e o registro ou a extração de dados nele contidos são procedimentos muito rápidos — a localização do arquivo leva uma fração de segundo e a extração de dados ou seu registro, no máximo alguns segundos. Sistemas de computação que operam com cassete podem ser muito mais lentos e exigir que o usuário retroceda a fita e espere que avance até o ponto correto do arquivo. Outra vantagem do emprego de discos está na possibilidade de se obter mais de um arquivo "aberto" ao mesmo tempo, ao passo que não são práticos os sistemas que operam com cassete.

Os arquivos consistem, portanto, em conjuntos de dados guardados em um meio de armazenamento

de grande capacidade disponível para emprego em um ou mais programas. Por exemplo, um programa de processamento de palavras pode necessitar de acesso ao mesmo conjunto de nomes e endereços para escrever cartas automáticas "personalizadas".

A manipulação dos arquivos ocorre de diferentes modos, de acordo com a versão de BASIC empregada. Para saber como seu computador lida com arquivos, verifique no manual que o acompanha as indicações sobre as instruções OPEN e CLOSE e experimente alguns exemplos. A descrição que apresentamos aqui é bastante genérica e preparada para dar uma visão geral do uso de arquivos.

Os arquivos podem ser seqüenciais ou aleatórios. Em um arquivo seqüencial, a informação é armazenada com a colocação da primeira unidade de dados em primeiro lugar, seguida pela próxima unidade e depois pela terceira, e assim sucessivamente. O arquivo aleatório é organizado de modo que o computador possa se dirigir diretamente à unidade de dados que se busca, sem necessidade de partir do início e passar por todos os dados até localizar a unidade procurada. O funcionamento do arquivo seqüencial é muito semelhante ao desenrolar de um filme no cinema, onde se começa do início e se acompanha a passagem dos dados ou eventos até o fim. O arquivo aleatório é mais parecido, digamos, com fazer passar uma fita de videocassete em casa, em que se pode avançar e retroceder a mesma e ver a parte desejada. Examinaremos apenas os arquivos seqüenciais, pois são mais adequados para sistemas que operam com cassete.

Suponhamos que você queira guardar um registro da temperatura média nos dias da semana:

SEGUNDA-FEIRA	13,6
TERÇA-FEIRA	9,6
QUARTA-FEIRA	11,4
QUINTA-FEIRA	10,6
SEXTA-FEIRA	11,5
SÁBADO	11,1
DOMINGO	10,9

Para simplificar, todos os dados serão tratados como dados numéricos, segunda-feira sendo o dia 1 e domingo, o dia 7. Os dados podem então ser representados da seguinte forma:

```
1.13,6,2.9,6,3.11,4,4.10,6,5.11,5,6.11,1,7.10,9
```

Serão necessários os seguintes estágios no programa para armazenamento dos dados em arquivo seqüencial:

```
ABRIR (OPEN) o arquivo
Fornecer os dados para o arquivo
FECHAR (CLOSE) o arquivo
```

Sempre que se usa a instrução OPEN, é preciso indicar se estamos fornecendo dados provenientes do computador para o arquivo (uma saída) ou dados provenientes do arquivo para o computador (uma entrada). Na linguagem BASIC da Microsoft isso é feito com o emprego das instruções OPEN "O" e OPEN "I". Um pequeno fragmento de programa para registrar os dados acima em um arquivo (em BASIC da Microsoft) seria:

```
100 OPEN "O", #1, "DAD.TEMP"
110 PRINT# 1.1.13,6,2.9,6,3.11,4,4.10,6,5.11,5,
    6.11,1,7.10,9
120 CLOSE #1
```

A palavra OPEN na linha 100 dá ao programa acesso ao arquivo. Essa instrução é seguida pelo sinal "O" para indicar que os dados provirão do programa para ser armazenado no arquivo. Esse procedimento vem seguido do sinal "#1", que informa ao computador que esse arquivo será referenciado pelo número 1 em nosso programa. Cada arquivo recebe um número arbitrário que posteriormente será empregado com as instruções INPUT# ou PRINT#, quando quisermos extrair ou fornecer dados nesse arquivo. Por fim, temos o nome do arquivo entre aspas. Nós o chamamos de DAD.TEMP para indicar que contém registros de temperaturas e que é um arquivo de dados e não um programa.

Damos abaixo um programa completo em BASIC da Microsoft para registro dos dados em um arquivo e a sua leitura e impressão subseqüentes:

```
100 OPEN "O", #1, "DAD.TEMP"
110 PRINT# 1.1.13,6,2.9,6,3.11,4,4.10,6,5.11,5,
    6.11,1,7.10,9
120 CLOSE #1
130 REM AS LINHAS 130 E 140 SAO LINHAS
    "FICTICIAS"
140 REM PARA REPRESENTAR O PROGRAMA
    INSERIDO
150 OPEN "I", #1 , "DAD.TEMP"
160 FOR X = 1 TO 7
170 INPUT #1, DIA,TEMP
180 PRINT "DIA # ";DIA,TEMP
190 NEXT X
200 CLOSE #1
210 END
```

Este programa abre um arquivo, com o número #1, e é denominado DAD.TEMP; fornece dados usando a instrução PRINT# e, a seguir, o ENCERRA (CLOSE). Mais adiante no programa, o mesmo arquivo é aberto com o emprego do número e também do nome do arquivo (o número não precisa ser o mesmo usado quando da criação do arquivo, mas o número utilizado nas instruções PRINT# ou INPUT# deve ser o mesmo dado ao nome do arquivo, quando este foi aberto). A instrução INPUT #1 na linha 170 indica que a entrada será proveniente de um arquivo com o número #1 (isto é, do arquivo DAD.TEMP) e não do teclado.

Vamos deixar, por enquanto, o exame da manipulação de arquivo e voltemos ao programa da agenda de endereços e de alguns dos elementos incluídos na subseção INICIALIZAR do programa. Primeiro, vamos ver a capacidade de memória necessária para um único registro no arquivo da agenda de endereços (o termo "arquivo" é aqui empregado em sentido específico ao banco de dados, como o conjunto de todos os registros relacionados, e não no sentido do sistema operacional, como um grupo denominado de dados, armazenado em fita ou disco).

O emprego de campos de determinada extensão consome uma parte da capacidade de memória, mas torna o procedimento de programação bem mais simples. Se reservarmos uma linha inteira para cada campo, com quarenta caracteres por linha, todos estes serão retidos em uma matriz, mesmo que a maior

parte da linha consista em espaços vazios. Em algumas versões de BASIC, todavia, quando as matrizes alfanuméricas são DIMensionadas, cada elemento pode totalizar 256 caracteres de comprimento. O dimensionamento apenas determina o número de elementos na matriz e não o tamanho deles.

Se você tem um equipamento em BASIC que pode lidar com matrizes multidimensionais, é possível empregar uma dimensão específica para cada um dos campos; porém, muitas versões de BASIC não o podem fazer. Assim, apresentaremos procedimentos alternativos. O método mais simples consiste em usar uma matriz alfanumérica separada para cada um dos campos. Mas aqui está um "truque", caso você queira matrizes multidimensionais e o BASIC de seu computador não possa manipulá-las.

O "truque" consiste em lidar com todos os elementos da matriz multidimensional como se fossem elementos de uma unidimensional. Por exemplo, uma matriz bidimensional com três linhas e cinco colunas poderá ser dimensionada da seguinte forma: DIM A(3,5) e conterá um total de quinze elementos: A(1,1) até A(3,5). Os mesmos dados poderão ser manipulados por uma matriz indexada comum, como esta: DIM A(15).

Se empregarmos uma matriz alfanumérica para cada campo, teremos de decidir sobre o modo como DIMensionar as matrizes. O mais simples é utilizar um tamanho fixo de matriz, mas isto limita a quantidade de registros que poderemos armazenar no banco de dados. Um procedimento melhor será a detecção do tamanho da matriz em função do número de registros que estão sendo utilizados. Todavia, nem todos os dialetos de BASIC admitem que o tamanho das matrizes alfanuméricas seja o que desejamos. Mesmo que admitam a presença de muitos registros no banco de dados, logo se esgotará a capacidade de memória disponível no computador. A seguir, apresentamos um programa que lhe permitirá encontrar o número máximo de elementos que seu computador poderá admitir. Muitas versões de BASIC, entretanto, admitirão tantos elementos na matriz quantos você desejar, até o ponto em que toda a capacidade de memória disponível tiver sido consumida. Toda vez que o programa apresentar a pergunta "QUAL O TAMANHO DA MATRIZ?", você deverá fornecer um valor maior, até finalmente obter uma mensagem de erro. A instrução CLEAR, na linha 100, tem como resultado a eliminação de uma matriz no final de cada passagem. Sem essa instrução, você receberá uma mensagem de erro na linha 30, ao tentar redimensionar a matriz.

```
10 READ D$
20 INPUT "QUAL O TAMANHO DA MATRIZ";A
30 DIM N$(A)
40 FOR L = 1 TO A
50 LET N$(L) = D$
60 NEXT L
70 FOR L = 1 TO A
80 PRINT L, N$(L)
90 NEXT L
100 CLEAR
110 GOTO 10
120 DATA "MICROCOMPUTADOR — CURSO
    BASICO"
130 END
```

Mesmo que cada elemento admita apenas quarenta caracteres, com cinco campos por registro, havendo 256 elementos reservados para cada matriz, o total de capacidade de memória para manter todos os dados dentro da memória principal se torna enorme. Se 1 byte é necessário para o armazenamento de cada caractere, precisamos de 51.200 bytes (5 x 40 x 256 bytes) só para armazenar os dados. Evidentemente, não é prático empregar tanta capacidade de memória principal para os dados, e é por isso que se usam arquivos separados de dados.

Infelizmente, como já mencionamos, a manipulação de rotinas de arquivo pode ser de execução um pouco difícil. Se quisermos evitar o uso de arquivos externos, a única alternativa é pôr os dados na instrução DATA, de modo que estejam sempre presentes no programa. Sem considerar o desgaste que impõe à capacidade de memória do computador, esse procedimento torna a modificação dos dados extremamente cansativa e aumenta a probabilidade de erro. Assim, é preferível usar arquivos de dados externos. Entretanto, se você experimentar alguns programas curtos para registrar dados para arquivos externos ou deles provenientes, o processo será mais claro e fácil de compreender.

Por exemplo, em equipamentos compatíveis com Apple, usando-se discos, podemos recorrer ao comando OPEN NOME ARQUIVO, S6, D2 para abrir um arquivo seqüencial chamado NOMEARQUIVO no disco (Slot 6) do drive 2 (D2). Num programa, o comando OPEN deve estar dentro de uma instrução PRINT e ser precedido pelo caractere CTRL-D. Para fechar o arquivo, o comando correspondente é o CLOSE NOMEARQUIVO. Para gravar os dados no arquivo seqüencial, usa-se o comando WRITE NOMEARQUIVO, e desta forma todos os demais dados das instruções PRINT seguintes serão gravados no disco, em vez de dar saída na tela. O comando WRITE deve estar igualmente dentro de um PRINT e precedido por CTRL-D. Assim, um pequeno exemplo de criação de um arquivo seqüencial em disco e gravação de dados num equipamento compatível com Apple seria:

```
100 D$ = " "
110 REM CTRL- D
120 PRINT D$; "OPEN ARQUIVO1, S6, D1"
130 PRINT D$; "WRITE ARQUIVO1"
140 FOR I = 1 TO 50
150 PRINT I
160 NEXT I
170 PRINT D$; "CLOSE"
180 END
```

Neste exemplo, serão gravados em disco no arquivo seqüencial ARQUIVO1 os números 1, 2, 3... 50. Para se recuperar do disco este arquivo, usa-se o comando READ NOMEARQUIVO, que deve ser utilizado da mesma forma que o WRITE, ou seja, dentro de uma declaração PRINT e precedido de um CTRL-D.

Nos equipamentos compatíveis com o Sinclair, particularmente no TK85, pode-se utilizar rotinas embutidas em linguagem de máquina para fazer gravação e recuperação de dados. Para tanto, cria-se um buffer, que servirá de área intermediária entre o programa e a fita cassete. Depois, utilizam-se as funções USR 8288 para gravar em fita uma matriz alfanumérica e USR 8305 para recuperar a informação

da fita. Um pequeno exemplo de gravação e leitura de um arquivo seqüencial em fita cassete no TK85 segue abaixo:

```
100 DIM C$(200)
110 C$ = "DADOS QUE SERAO GRAVADOS"
120 LET Z = 26
130 LET Z$ = "C"
140 LET CONTROLE = USR 8288
150 DIM T$(100)
160 LET Z$ = "T"
170 LET CONTROLE = USR 8305
180 PRINT T$
```

No exemplo acima, C$ é a variável alfanumérica de entrada e T$ é a variável de recuperação dos dados da fita. A variável Z$ indica o endereçamento no buffer ("C" para gravação e "T" para recuperação no nosso caso).

Finalmente, para equipamentos da linha TRS-80 (CP 500, Digitus etc.), a linguagem para gravação e recuperação de dados é muito próxima do exemplo inicial dado no BASIC da Microsoft. Para detalhes de programação e de operação com fita e disco, deve-se consultar o manual do fabricante do equipamento.

Até aqui, vimos como se pode transferir dados, através de matrizes, para arquivo em fita (ou disco) e vice-versa. No próximo passo veremos com detalhes o processo de INICIALIZAÇÃO, observando exatamente quantas matrizes serão necessárias, quantos elementos cada uma vai precisar e em que pontos do programa deve ser feita a transferência de dados, para elas e a partir delas.

Exercício

Escrever um programa com os seguintes elementos:

```
INICIAR
  INICIALIZAR
  FORNECER OS DADOS
  ESCOLHER
    Gravar os dados
    Carregar os dados
    Sair do programa
ENCERRAR
```

O procedimento INICIALIZAR introduz quaisquer variáveis e matrizes necessárias ao programa. Os dados abrangerão, digamos, quinze nomes, fornecidos através do teclado, com indicações na tela. O procedimento ESCOLHER fornecerá ao usuário um menu perguntando-lhe se "deseja GRAVAR DADOS (SAVE DATA)?", "CARREGAR DADOS (LOAD DATA)?" ou "SAIR DO PROGRAMA (EXIT PROGRAM)?" Veja se pode criar uma flag na parte "SAIR DO PROGRAMA", que automaticamente grave os dados se, e somente se, não tiver sido feita anteriormente uma instrução SAVE.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
DEL linha1, linha2	Elimina as linhas entre linha1 e linha2 inclusive		+		
DELETE "arquivo"	Elimina um arquivo em disco		+		
DELETE linha1-linha2	Elimina linhas específicas do programa	+		+	
DIM lista de var(index)	Define dimensão das matrizes e reserva espaço de memória	+	+	+	+
DRAW x	Desenha uma figura de comando equivalente a x		+		
EDIT linha	Exibe uma linha do programa para edição	+		+	
END	Pára o programa e fecha todos os arquivos	+	+	+	
EOF(a)	Indica uma condição de fim de arquivo em a	+			
ERASE matrizes	Elimina matrizes do programa	+			
ERL	Retorna a linha onde ocorreu o último erro	+		+	
ERR	Retorna o código do último erro	+		+	
ERROR n	Simula um erro de código n	+		+	
EXEC "arquivo"	Executa um arquivo de texto em disco como se digitado		+		
EXP(x)	Eleva e à potência x	+	+	+	+
FILES "arquivo"	Lista os arquivos do disquete c/ arquivo	+			
FIX(x)	Trunca x para um inteiro	+			
FLASH	Provoca um piscar na tela a qualquer mensagem	+	+		
FN variável (expressão)	Chama uma função definida anteriormente		+		
FOR variável=x TO y STEP z	Repete linhas de programa. O NEXT encerra o loop	+	+		
FRE(x$)	Dá o espaço livre de memória para o usuário	+			

Trocando de lugar

Após examinar meios para inserção de novos registros, passamos agora aos métodos para recuperá-los. Como previsto, o problema inicial consiste em determinar a combinação exata.

Encerramos nosso último fascículo com um exercício para desenvolvimento de um programa do tipo "banco de dados", que admitia o fornecimento de dados. Vamos observar alguns dos procedimentos incluídos no fornecimento de um novo registro, em continuação ao nosso exame sobre procedimentos exigidos pelo estágio INICIALIZAR de nosso programa principal. Primeiramente, suponhamos a existência dos seguintes campos, com suas matrizes correspondentes:

CAMPO	MATRIZ
1 campo do NOME	CAMPNOM$
2 campo do NOME MODIFICADO	CAMPMOD$
3 campo da RUA	CAMPRUA$
4 campo da CIDADE	CAMPCID$
5 campo do ESTADO	CAMPEST$
6 campo do NÚMERO TELEFÔNICO	CAMPTEL$
7 campo do ÍNDICE	CAMPIND$

É relativamente fácil entender o sentido da maior parte desses campos, com exceção, talvez, dos campos 2 e 7. Examinemos primeiro o campo no NOME MODIFICADO. Quando inicialmente analisamos o problema do formato dos dados para o nome, discutimos quanto a fazer com que o formato especificado de modo preciso (rígido) ou de modo vago (sem padrão), tendo optado por esta última alternativa. Uma vez que os nomes podem ser fornecidos de forma muito variada, o formato rígido tornará complexas as rotinas de busca e de ordenação. Para solucionar este problema, optamos pela conversão de todos os nomes para um formato padrão: todas as letras convertidas para maiúsculas, todos os caracteres não alfabéticos (como os espaços, sinais de ponto, apóstrofos etc.) eliminados, e um único espaço entre o nome (se houver) e o sobrenome.

A necessidade de uma padronização de nomes como essa deve-se ao fato de que as rotinas de busca e ordenação precisam de um método de comparação entre os elementos semelhantes. Por outro lado, ao recuperarmos nomes e endereços de um banco de dados, vamos querer a apresentação dos dados sob a forma inicialmente fornecida. Há dois métodos de lidar com o problema: ou cada campo de nome é convertido para uma forma padrão, apenas quando os procedimentos de ordenação e de busca estão se realizando, ou, então, o campo do nome pode ser convertido para uma forma padrão e armazenado como um campo separado, de modo que as rotinas de busca e ordenação possam ter acesso imediato aos nomes padronizados.

Ambos os métodos apresentam vantagens e desvantagens. A conversão temporária dos campos de nomes requisitados por outras rotinas economiza espaço de memória, pois é necessário armazenamento de menos dados no arquivo. Por outro lado, este procedimento consome muito tempo. Entretanto, se reservarmos um campo separado para a forma padronizada do nome, a conversão deverá ser executada uma única vez no registro. E, embora haja consumo de memória adicional, os procedimentos de busca e ordenação serão executados de modo mais rápido.

Outro campo que pode ocasionar confusão é o campo do ÍNDICE, que é incluído como um campo de reserva para possibilitar futura ampliação ou alteração do banco de dados, sem necessidade de reestruturação complementar do programa. Sua inclusão introduz a questão da "ligação" — termo que significa a determinação das relações de dados e de processamento. Todos os campos ou elementos em cada um dos registros são ligados porque possuem o mesmo índice (o mesmo elemento numérico ou índice em suas respectivas matrizes), e porque todos os campos de um registro serão armazenados juntos em um arquivo. Isto pode tornar difícil o procedimento de acréscimos de novos tipos ou relações de dados em estágios avançados, com a possível exigência de reorganização completa da estrutura do arquivo e uma reestruturação mais detalhada do programa. A incorporação do campo ÍNDICE neste estágio tornará muito mais simples as alterações futuras do programa.

Antes de tentar acrescentar novos registros ao banco de dados, faremos algumas suposições quanto à estrutura dos arquivos. Primeiro, limitaremos o número de registros a um total de 50, embora este número seja muito reduzido para uma agenda de endereços prática; posteriormente examinaremos como lidar com grandes quantidades de dados. Vamos supor também que todos os dados já foram transferidos — como parte do procedimento INICIALIZAR — para matrizes.

O acréscimo de um novo registro é de execução mais fácil no fim do arquivo (isto é, no primeiro elemento vazio de cada matriz). A possibilidade de que o novo registro esteja fora de ordem com relação aos demais é grande, porém este é um problema que poderá ser examinado posteriormente. Assim, a primeira coisa a fazer será determinar o tamanho da matriz. Uma vez que esta unidade de dados provavelmente será útil em muitas partes do programa, o melhor lugar para fazê-lo é o procedimento INICIALIZAR. Trata-se de um exemplo claro da necessidade de variáveis globais (isto é, variáveis que podem ser utilizadas em qualquer parte do programa). Nós a denominaremos TAMANHO. Outra variável global que provavelmente será útil consiste no índice do registro em uso corrente. Uma vez que nenhum registro estará em uso corrente, quando o programa for processado pela primeira vez, a atribuição de um

valor inicial à variável CORR deverá aguardar até que o programa realize algum procedimento com os dados. Todavia, a variável CORR pode ser inicializada em 0 no procedimento de INICIALIZAÇÃO. A inicialização de variáveis em 0 não é estritamente necessária na linguagem BASIC, pois se realiza de forma automática. Entretanto, é uma boa prática e deve ser sempre realizada em variáveis locais, a fim de evitar "efeitos colaterais" resultantes do emprego da mesma variável em outro ponto do programa.

Quando o programa for processado pela primeira vez, vários tipos de inicialização terão lugar e os dados serão fornecidos pelo disco ou fita e transferidos para as variáveis alfanuméricas. O menu ESCOLHA será então apresentado. Se o usuário escolher a opção 6 (para acrescentar um registro ao arquivo), o valor que a variável ESCOLHA apresentará será 6, o que chamará a sub-rotina ACREREG. Esta pressupõe que a variável TAMANHO já posssui um valor que lhe foi atribuído e pode, desse modo, iniciar a solicitação de entradas (obs.: também supõe que o procedimento INICIALIZAR já DIMensionou corretamente as matrizes necessárias).

O acréscimo de novos registros significa ainda que o arquivo está agora, pelo menos potencialmente, fora da ordem. Uma vez que a ordenação pode exigir um certo tempo, talvez não seja necessário ordenar os registros após a realização de cada acréscimo — essa é uma decisão que deixaremos de lado por enquanto. Em vez disso, estabeleceremos uma flag que indicará que o novo registro foi acrescentado.

Agora estamos em condição de começar a fazer uma lista experimental das possíveis matrizes, variáveis e flags que podem ser necessárias ao programa.

TABELAS

CAMPNOM$	(campo do nome)
CAMPMOD$	(campo do nome modificado)
CAMPRUA$	(campo da rua)
CAMPCID$	(campo da cidade)
CAMPEST$	(campo do Estado)
CAMPTEL$	(campo do número telefônico)
CAMPIND$	(campo do índice)

VARIÁVEIS

TAMANHO	(tamanho atual do arquivo)
CORR	(índice do registro corrente)

FLAGS

RACR	(novo registro acrescentado)
ORDEM	(ordenado depois da modificação do registro)
GRAV	(gravação realizada após a modificação do registro)
RMOD	(modificação realizada depois da última gravação)

É provável que no decorrer do desenvolvimento do programa mais algumas matrizes sejam necessárias. Seguramente haverá necessidade de mais variáveis. Embora seja evidente que outras flags terão de ser incluídas, é possível que as quatro acima apresentadas não venham todas a ser requisitadas. Não será preciso gravar nem ordenar o arquivo (supondo-se que já esteja gravado e ordenado), a menos que alguma alteração tenha se realizado; assim, a flag RMOD será provavelmente a única necessária. Porém, se decidirmos utilizar as quatro flags, o subprograma de INICIALIZAÇÃO deverá atribuir a todas elas seu valor correspondente. Como treino complementar da elaboração de programas top-down, observamos a facilidade com que se pode codificar a variável *ACREREG*.

I 4(EXECUTAR)6(ACREREG)

```
INICIAR
   Localizar o tamanho atual do arquivo
   Indicar para entradas
   Atribuir as entradas no final das matrizes
   Acertar a flag RMOD
ENCERRAR
```

II 4(EXECUTAR)6(ACREREG)

```
INICIAR
   (o tamanho do arquivo é TAMANHO)
   (indicação para entradas)
   Limpar a tela
   Imprimir mensagem indicando para a primeira
   matriz (TAMANHO)
   Fornecer os dados para a matriz (TAMANHO)
   (indicar e dar entrada para todas as matrizes)
   Atribuir o valor 1 a RMOD
ENCERRAR
```

Todo este processo é direto e não exige loops ou outras estruturas complexas. O passo seguinte poderá ser a codificação direta para a linguagem BASIC. O único ponto a observar consiste na variável TAMANHO, que deve ser estabelecida durante a execução do procedimento INICIALIZAR e não precisa ser codificada como parte desta unidade.

III 4(EXECUTAR)6(ACREREG)CÓDIGO BASIC

```
CLS: REM OU USAR PRINT CHR$(24) ETC. PARA
   LIMPAR A TELA
INPUT "FORNECER O
   NOME";CAMPNOM$(TAMANHO)
INPUT "FORNECER A
   RUA";CAMPRUA$(TAMANHO)
INPUT "FORNECER A
   CIDADE";CAMPCID$(TAMANHO)
INPUT "FORNECER O
   ESTADO";CAMPEST$(TAMANHO)
INPUT "FORNECER NÚMERO TELEFÔNICO";
   CAMPTEL$(TAMANHO)
LET RMOD=1
LET CAMPIND$=STR$(TAMANHO)
GOSUB *NOMEMOD*
RETURN
```

Na antepenúltima linha, há a atribuição do campo CAMPIND$ ao valor da variável TAMANHO (convertida em uma variável alfanumérica pela função STR$), de modo a funcionar como índice em estágios posteriores. A sub-rotina *NOMEMOD*, chamada logo após o encerramento do programa, corresponde ao programa descrito pormenorizadamente na página 254. Algumas leves alterações serão necessárias nesse programa, porém apenas como detalhamento. Esta sub-rotina tem como função tomar a entrada do nome na ordem habitual (sem padrão definido) e colocá-la na forma padrão. A saída desta sub-rotina será um elemento (TAMANHO) na matriz denominada CAMPMOD$. Todos os procedimentos de busca de nomes e de ordenação podem agora ser realizados entre os elementos na tabela

CAMPMOD$ e, uma vez que o elemento terá o mesmo índice que os outros campos no registro, será fácil apresentar o nome e o endereço do modo como foram inicialmente fornecidos. Em outras palavras, a busca será realizada na tabela CAMPMOD$, porém a apresentação provirá da variável CAMPNOM$.

Isto é quase tudo o que se exige para o acréscimo de novos registros ao arquivo, embora não tenhamos feito concessões a erros de verificação nem previsões para o caso de não haver espaços disponíveis na matriz. Uma vez que nossos programas são desenvolvidos de forma modular, alterações e elaborações como essas poderão ser efetuadas posteriormente, sem a necessidade de reestruturação de todo o programa.

Os subprogramas REGMOD e REGELIM (respectivamente, para alterar e eliminar registros) são muito semelhantes ao programa ACREREG, exceto pelo fato de termos de localizar o registro que desejamos alterar para poderem ser executados. Por conseguinte, esses dois subprogramas deverão iniciar pela chamada do subprograma ENCREG, que segue uma rotina de busca semelhante à descrita na página 273. Aqui, a principal diferença está em que (muito provavelmente) não haverá dois itens de dados idênticos, pois poucas são as pessoas que possuem o mesmo nome.

Há dois modos de realizar uma rotina de busca. O primeiro deles consiste em examinar uma lista não ordenada, o que torna a busca mais lenta do que seria necessário. Na pior das hipóteses, a rotina deverá examinar todos os itens antes de localizar o item buscado. Entretanto, a busca em uma lista não ordenada tem a vantagem de que as rotinas de busca não são necessárias a cada vez que um novo registro é acrescentado, eliminado ou modificado.

Se houver algum tipo de ordenação dos dados — numérica ou alfabética, por exemplo —, o programa deverá examinar apenas uma pequena parte dos itens na lista. Quanto maior a lista, mais eficiente se tornará a busca binária, em comparação com a busca através de uma pilha não ordenada. De fato, se no arquivo houver dados suficientes que o garantam, a ordenação dos registros após uma alteração pode ser acelerada pela realização de uma busca preliminar para localizar a primeira e a última ocorrência, na matriz, da letra inicial do sobrenome no registro em questão.

Outro modo de acelerar a rotina de ordenação pode ser a manutenção de uma tabela de consulta das posições na matriz em que pela primeira vez cada letra do alfabeto aparece. Todavia, essa tabela deverá ser cuidadosamente mantida (atualizada) sempre que houver alterações nos dados.

O problema da busca e da ordenação é uma das áreas mais amplas da programação, e muitos livros foram escritos sobre a questão. Não tentaremos encontrar a solução definitiva para o programa da agenda de endereços, uma vez que isso depende de uma série de fatores, inclusive do número de registros no arquivo e da disponibilidade ou não de unidades de discos.

Apresentamos, agora, programas em pseudolinguagem para um exame dos elementos da tabela CAMPMOD$. A variável alfanumérica CHAVE$ é a chave para a busca. Neste contexto, o termo "chave" corresponde ao grupo de identificação de caracteres utilizados na especificação do registro (ou registros) exigido.

```
Indicar para a busca do nome
LETCHAVE$=nome (a ser buscado)
LET BTM=1
LET BUSCA=0
LET TOP=TAMANHO
executar um LOOP enquanto
   (BTM<=TOP)E(BUSCA=0)
   LET MID=INT((BTM+TOP)/2)
   IF CHAVE$=CAMPMOD$(MID)
      THEN
         PRINT CAMPNOM$(MID)
         PRINT CAMPRUA$(MID)
         PRINT CAMPCID$(MID)
         PRINT CAMPEST$(MID)
         PRINT CAMPTEL$(MID)
         LET BUSCA=1
      ELSE
         IF CHAVE$>CAMPMOD$(MID)
                  THEN LET BTM=MID+1
                  ELSE LET TOP=MID-1
                    ENDIF
      ENDIF
   ENDLOOP
   IF BUSCA=0 THEN PRINT "REGISTRO NÃO
      ENCONTRADO"
   END
```

Esta parte de pseudolinguagem segue basicamente o programa empregado na busca das contagens dos jogos de futebol da página 275, porém você perceberá que não há uma saída conveniente se o registro não puder ser encontrado (a última instrução PRINT), o que será realizado apenas se o loop não puder localizar uma correspondência exata entre as variáveis CHAVE$ e CAMPMOD$(MID).

Infelizmente, é pouco provável o encontro de uma combinação perfeita, mesmo que o nome e o número telefônico que você deseja estejam no banco de dados. Isto se dá porque a instrução IF CHAVE$=CAMPMOD$ é totalmente invariável e não pode admitir a mínima diferença entre a série de caracteres fornecida pelo usuário na resposta à indicação e a série de caracteres armazenada na variável CAMPMOD$(MID). Em agendas de endereços comuns, examinamos a página com o olhar e isso nos capacita a aceitar os vários tipos de pequenas diferenças na apresentação efetiva do registro e aquilo que estamos buscando. O computador, porém, não pode realizar isso.

Entretanto, há recursos para superar o problema, embora todos exijam esforço de programação e um pouco mais de tempo para seu processamento. A primeira elaboração consistirá em verificar inicialmente só o sobrenome, e por este motivo é conveniente que o nome armazenado na variável CAMPMOD$ se apresente sob a forma de SOBRENOME (espaço) PRENOME. Desenvolvemos uma rotina para inverter a ordem de nomes em um fascículo anterior do curso de programação em linguagem BASIC e ela pode ser incorporada como uma sub-rotina no interior de uma rotina ACREREG, quando o campo CAMPMOD$ for criado.

Tendo conseguido localizar a primeira ocorrência

do sobrenome procurado, a rotina ENCREG deverá conferir a parte prenome desse elemento, de modo a verificar se é idêntico ao nome fornecido (CHAVE$). Se for, não haverá problema — o registro terá sido localizado. Entretanto, se não for, o problema começa a se complicar e temos de planejar nossa estratégia com cuidado. Poderíamos, por exemplo, examinar todos os prenomes e, se não encontrarmos uma correspondência exata, começaremos a procurar uma correspondência aproximativa. A dificuldade aqui é a seguinte: o que precisamente constitui uma correspondência exata?

Em vez de uma mensagem "REGISTRO NÃO ENCONTRADO", como no programa acima, poderá ser melhor uma mensagem como "CORRESPONDÊNCIA EXATA NÃO ENCONTRADA, DEVE SER TENTADA CORRESPONDÊNCIA APROXIMATIVA? (S/N)" O que significam as palavras ''correspondência aproximativa''? Beto pode ser considerado uma correspondência aproximativa a Roberto? E quanto a Robert? Ambos apresentam entradas possíveis para o programa ENCREG. Vamos tentar definir o que queremos dizer com correspondência aproximativa e começar a desenvolver um programa em linguagem BASIC para determinar a correspondência mais aproximativa de uma série fornecida.

Suponhamos que a série na memória corresponda a ROBERTO. Qual das duas correspondências é a mais aproximativa: ROB ou RBRT? A segunda apresenta quatro das sete letras, enquanto a primeira, apenas três entre sete. Por outro lado, a primeira apresenta três letras na seqüência correta, enquanto a segunda, apenas duas.

A escolha é muito arbitrária. Daremos prioridade a uma correspondência exata entre a variável CHAVE$ e uma parte da variável do nome na memória. Se não for possível encontrar correspondência exata com uma parte da variável, o programa tentará obter o número maior de letras em comum. Eis aqui o programa apresentado em termos de entrada e saída:

ENTRADA
Uma série de caracteres
SAÍDA
A correspondência mais aproximada à série fornecida

O programa seguinte, em pseudolinguagem, próxima à linguagem BASIC, examinará as variáveis alfanuméricas em uma matriz, bem como as primeiras ''n'' letras em cada uma, onde ''n'' corresponde ao número de letras na chave (CHAVE$). Se não houver correspondência, será apresentada uma mensagem nesse sentido.

```
DIM MATRIZ$(4)
FOR L = 1 TC 4
READ MATRIZ$(L)
NEXT L
DATA "ROBERTO", "RICARDO", "ROBIANA",
   "ROBERTA"
LET CHAVE$ = "RON"
LET LCHAVE = LEN(CHAVE$)
LET BUSCA = 0
LOOP PARA ÍNDICE = 1 TO 4
   IF CHAVE$ = LEFT$(MATRIZ$(ÍNDICE),
   LCHAVE)
      THEN PRINT "A CORRESPONDÊNCIA E ";
         MATRIZ$(ÍNDICE)
         LET BUSCA = ÍNDICE
   ENDIF
ENDLOOP
IF BUSCA = 0
      THEN PRINT CHAVE$; "NÃO CORRESPONDE
      A NENHUM DOS"
         PRINT LCHAVE; "PRIMEIROS CARACTERES"
```

Após isso, o programa poderá prosseguir examinando os grupos de caracteres de comprimento LCHAVE, a começar pelo segundo caractere em cada matriz. Se nenhum desses grupos corresponder, pode-se buscar grupos que se iniciam com o terceiro caractere, e assim por diante. Finalmente, se não houver correspondência entre quaisquer dos grupos de três caracteres na série, o programa tentará determinar qual das séries apresenta o maior número de letras em comum com a variável CHAVE$. Esta tarefa é deixada ao leitor como exercício.

Podemos, de fato, escrever muitas páginas sobre a questão da ordenação ''embaralhada'' e a das diversas técnicas empregadas em pacotes de bancos de dados à venda no mercado. A maioria apresenta o recurso de buscar nos primeiros poucos caracteres do campo, como o código que acabamos de desenvolver. Outros recuperarão um registro se a seqüência especificada de caracteres se apresentar em algum local na tela, ou na verdade em qualquer ponto do registro. O recurso do ''wildcard'' é especialmente útil: a especificação J?N identificará JONAS ou JANE, porém não JOANA. A forma mais sofisticada de correspondência pouco definida funciona foneticamente, de modo que o fornecimento do nome MORAIS também identificará o nome MORAES.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
GET x$	Provoca a parada do programa e aguarda por x$	+	+		
GET"arquivo",número	Lê um registro de um arquivo randômico	+			
GET(x1,y1)-(x2,y2),matriz	Lê informações gráficas da tela	+			
GOSUB linha	Chama uma sub-rotina da linha especificada	+	+	+	+
GOTO linha	Conecta o programa à linha especificada	+	+	+	+
GR	Comando para gráficos em baixa resolução (40x40)	+	+		

Montagem de programas

Podemos agora unir os subprogramas que vão processar nossa agenda de endereços computadorizada. Examinaremos também métodos para tornar o programa mais acessível ao usuário.

Embora muitos detalhes do programa da agenda de endereços ainda tenham de ser estudados, a estrutura geral já está se tornando clara. Neste ponto do desenvolvimento de programas de qualquer tamanho, é conveniente desenhar um diagrama de blocos e avaliar o fluxo de atividades no programa.

Este é também o momento em que o programador examina a "interface humana" e os aspectos da "imagem do usuário" do programa. Esses conceitos e práticas, embora muito importantes, quase nunca recebem a atenção devida, mesmo no caso de software profissional. A "interface humana", definida em termos simples, corresponde à "ergonomia" do software, ou seja, sua facilidade de uso. A "imagem do usuário" vincula-se ao modo como ele percebe o programa em uso. Examinaremos esses conceitos com relação a nosso programa, da forma como foi desenvolvido até agora, e determinaremos até onde se pode complementá-los.

A lista abaixo apresenta os principais blocos do programa já examinados. Como convenção, e apenas para manter certa ordem no procedimento de verificação, empregaremos nomes com seis caracteres para procedimentos ou sub-rotinas, com oito caracteres (inclusive o caractere $) para matrizes alfanuméricas, com quatro caracteres para variáveis numéricas simples, e com cinco caracteres (inclusive $) para variáveis alfanuméricas simples de uso geral. As variáveis locais (nos loops, por exemplo) terão letras isoladas.

BLOCOS DE PROGRAMA PRINCIPAL

INICIL	CRIMAT	(cria matrizes e inicializa variáveis)
	LERARQ	(lê arquivos em fita ou disco)
	ESTFLG	(estabelece flags e altera variáveis)
SAUDAR		(imprime mensagem de saudação)
ESCLHA	MENESC	(imprime menu de escolhas)
	ATRESC	(atribui a escolha à variável ESCL)
EXECUT	ENCREG	(encontra e imprime um registro)
	ENCNOM	(encontra nomes a partir de nomes incompletos)
	ENCCID	(encontra o registro de determinada cidade)
	ENCINI	(encontra nomes a partir de iniciais)
	LISREG	(faz a listagem de todos os registros)
	ACRREG	(acrescenta novos registros)
	MODREG	(modifica registros já existentes)
	ELMREG	(elimina registros)
	SAIPRG	(grava arquivos e sai do programa)

Cada bloco do programa principal na segunda coluna precisa ser subdividido em duas unidades e estas necessitam de elaboração complementar até que se atinja o nível de detalhe suficiente para escrever o próprio código em BASIC. Os processos incluídos nessa forma de "elaboração por estágios" foram exemplificados, no caso de muitos dos blocos, em unidades anteriores deste curso de programação em BASIC.

Admitamos que a maioria dos módulos do programa ou todos eles foram elaborados, codificados em BASIC e testados individualmente. É preciso, agora, unificá-los para formar um programa completo. O melhor processo consiste em gravar cada módulo em fita ou disco, atribuindo-lhe o mesmo nome de arquivo utilizado nas notas de desenvolvimento do programa. Dessa forma, o bloco ACRREG pode ser escrito e testado até onde for possível e então gravado sob o nome de arquivo ACRREG. Em geral, quando o programa é carregado em fita ou disco, usamos o comando LOAD, seguido do nome de arquivo, como acontece na instrução LOAD "ACRREG". No entanto, tal procedimento resulta na limpeza total dos conteúdos da memória; assim, se carregarmos o bloco ACRREG e logo após o bloco MODREG, todo o programa ACRREG será apagado.

Em boa parte, porém, o problema pode ser solucionado. O comando MERGE carrega programas procedentes de fitas ou discos sem apagar programas já armazenados na memória. Mas há uma condição importante. Se quaisquer dos números de linha no programa que recebeu a instrução MERGE coincidirem com os já presentes na memória, os novos números de linha serão sobrepostos aos antigos, gerando caos. Versões do BASIC com o comando RENUM podem contornar esse inconveniente pela renumeração das linhas no módulo de programa, antes de realizar seu armazenamento. Assim, quando elas se unificarem, não haverá conflito.

Muitas versões do BASIC para computadores pessoais não possuem o comando RENUM, tornando-se necessário um planejamento cuidadoso dos números de linha desde o início. Quando se tiver elaborado

um mapa completo de todos os módulos do programa principal (como fizemos parcialmente na lista), pode-se atribuir números de linha de início para cada bloco. Partes do programa passíveis de grandes modificações — como o programa principal ou as unidades de manipulação de arquivos — devem ser numeradas a intervalos de 50 ou mesmo 100 unidades. Assim, haverá espaço para eventuais inclusões. Os intervalos dos números de linha em módulos de programa menos propensos a alterações, como a rotina SAUDAR, podem ser de 10 unidades. A inclusão, no programa, de numerosas instruções REM em branco facilita sua leitura e possibilita o posterior acréscimo de instruções ou de chamadas de sub-rotinas adicionais. Se o BASIC de seu micro não possuir o comando MERGE, digite os vários módulos à medida que se desenvolvem e grave-os juntos.

Na lista, os blocos de programa foram unificados como um "processamento experimental", para ilustrar os erros em que se pode incidir ao fazer a abordagem do tipo "experimente e veja" que o BASIC encoraja. Nosso programa não funcionaria a contento porque não se elaborou o fluxo de controle do programa com atenção suficiente. Seria trabalho inútil digitar todo um programa no computador só para verificar que não funciona; mas, se você gravou as rotinas de unidades anteriores do curso, e se o BASIC de seu micro posssui o comando RENUM, pode experimentar a renumeração e a seguir unificá-la para produzir uma listagem semelhante.

O primeiro bloco do programa principal é INICIL, com a finalidade de inicializar variáveis, dimensionar matrizes, ler arquivos, atribuir dados às matrizes, estabelecer flags etc.

A sub-rotina *INICIL* está subdividida na sub-rotina *CRIMAT* (para criar matrizes); na sub-rotina *LERARQ* (para ler arquivos e atribuir os dados às matrizes correspondentes) e na sub-rotina *ESTFLG* (para estabelecer flags etc.).

O programa seguirá então para *SAUDAR*, uma sub-rotina que imprime mensagem de saudação na tela. A última parte faz com que o programa se interrompa até que o usuário pressione a barra de espaço para seu prosseguimento.

O programa então continua até a rotina *ESCLHA*, subdividida em duas partes: uma apresenta repertório (menu) de opções oferecidas pelo programa da agenda de endereços; a segunda recebe a escolha fornecida por meio do teclado e atribui o valor (numérico) a uma variável denominada ESCL.

O valor dessa variável é utilizado pelo bloco seguinte do programa, EXECUT, para selecionar um entre nove outros blocos de programa. Todos eles, exceto o SAIPRG, deverão retornar à sub-rotina *ESCLHA* após sua execução, de forma que o usuário terá oportunidade de fazer outra escolha. Tal procedimento não será necessário se optar pelo bloco 9 (SAIPRG), pois este tem como finalidade o encerramento da operação do programa.

Do modo como se apresenta, o principal problema corresponde à incorreção do fluxo de controle. A sub-rotina INICIL exige a leitura de um arquivo da memória de armazenamento em massa, exista ou não esse arquivo. Se o programa estiver sendo processado pela primeira vez, não haverá dados já fornecidos — e nem arquivos de dados na fita ou no disco. Qualquer tentativa de abrir e ler um arquivo não existente resultará numa mensagem de erro e o programa não funcionará.

Para contornar essa dificuldade, chama-se a rotina *LERARQ* por meio de apenas um dos módulos EXECUT, e depois uma única vez, sempre que o programa for processado. Isso indica que deve haver uma flag ARQV com o valor inicial 0, que assumirá o valor 1, após a leitura do arquivo. Tendo assumido o valor 1, a flag não admitirá outrás tentativas de leitura do arquivo. Desse modo, a sub-rotina ACRREG sempre examina as matrizes para localizar o primeiro elemento vazio e aí registra os dados. É bem provável que esse registro não esteja na seqüência adequada de ordenação; assim, deve haver uma flag RMOD que assume o valor 1 durante a execução. A flag RMOD também deve assumir o valor 1, se as rotinas MODREG ou ELMREG forem executadas. Você pode desenvolver o código relevante para obter esse resultado, ou, se só quiser processar o programa, alterar a linha 1310 para RETURN.

Procedimentos como a inclusão, a eliminação ou a modificação de registros indicam que a seqüência deles está fora de ordem; assim, qualquer módulo (ENCREG, por exemplo) deve primeiro examinar a flag RMOD para verificar se ocorreram alterações. Em caso afirmativo, pode-se insistir numa ordenação, antes de efetuar uma busca. Ou, então, admitir um procedimento ineficiente de busca no arquivo. A sub-rotina SAIPRG examina de modo automático a flag RMOD e chama a rotina de ordenação, se esta receber uma atribuição (do valor 1) antes de gravar os dados no arquivo em fita ou disco.

Os aspectos da "interface humana" dos programas mencionados subdividem-se nestas categorias:

Interface do usuário
Imagem do usuário
Recuperação de erros
Segurança
Adaptabilidade

A interface do usuário corresponde ao modo como este se comunica com o programa. Optamos pelo uso de menus em todas as ocasiões (em vez de comandos). Muitas pessoas preferem comandos, mas a questão importante é que, não importa a forma de comunicação empregada, ela tem de ser coerente. Não convém que comandos semelhantes realizem tarefas diferentes em partes diferentes do programa. Se isso ocorrer, é necessário que o usuário leia com atenção cada menu, antes de realizar a escolha, e evite "reflexos".

Do modo como se apresenta, nosso programa mostra-se frágil no seguinte aspecto: o pressionamento da barra de espaço encerra a mensagem de saudação; o menu de opções é automaticamente encerrado pela digitação de uma das teclas numéricas de 1 a 9; e o fornecimento de dados na sub-rotina ACRREG encerra-se (em cada campo) com o pressionar da tecla RETURN. Admite-se uma incoerência como essa em programas de "amadores", mas ela é inaceitável no caso de software profissional.

A "imagem do usuário" relaciona-se ao modo como ele percebe o funcionamento do programa e sofre a influência da qualidade da interface do usuário. O operador não tem ciência da maior parte das

operações que se processam no interior do computador. O único modo pelo qual pode avaliar o que se passa no programa é proporcionado pela entrada visual recebida na tela em resposta às entradas de dados fornecidos pelo teclado. A "imagem do usuário" que esperamos de nosso programa da agenda de endereços computadorizada corresponde à de um livrinho de endereços fisicamente real. De modo semelhante, a "imagem do usuário" de um programa processador de palavras é a de uma folha de "papel" (na tela), na qual digitamos. Nesse caso, teoricamente, os tipos em negrito aparecem em negrito na tela, os tipos sublinhados mostram-se sublinhados e os tipos justificados (dispostos de modo que formem a margem direita em linha reta) surgem desse modo na tela.

A "imagem do usuário" quase nunca se mostra perfeita — nenhum livrinho de endereços real aguarda o usuário "PRESSIONAR 1" para localizar um nome. No entanto, a boa imagem do usuário implica a elaboração bem feita de esquemas na tela e um padrão coerente de operações. As indicações devem aparecer sempre na mesma posição na tela (alguns programas processadores de palavras, por exemplo, contêm algumas indicações na linha superior da tela e outras na inferior, de modo aparentemente aleatório). Programas que correspondem à boa imagem do usuário também podem informá-lo, a qualquer momento, em que posição do programa ele se encontra. Se você está no modo ACRREG, precisa de uma imagem sempre visível que lhe indique isso. Se você acaba de fornecer um campo (para acrescentar novo registro), convém que exista uma mensagem dizendo, por exemplo, PRESSIONAR A TECLA DE RETURN, SE A ENTRADA ESTIVER CORRETA, CASO CONTRÁRIO, PRESSIONAR ESCAPE. (Isso nos remete à importante questão da recuperação e indicação de erros, discutida mais adiante.)

Teoricamente, toda a formatação deve aparecer na tela de modo que, por exemplo, o registro apresentado tenha o mesmo formato do registro da impressora. Muitas unidades de software à venda incluem "menus auxiliares" que indicam o procedimento a ser adotado a seguir, caso você não se sinta seguro.

A "imagem do usuário" de um programa melhora quando é concreta — um pedaço de papel ou um cartão de fichário, por exemplo — e não abstrata, como "subarquivos", "memórias intermediárias" (buffers) etc. Muitos programas de bancos de dados à venda apresentam problemas com relação a esse fator; o usuário deve ter em mente que certas unidades de dados estão em subarquivos ou em campos auxiliares temporários, o que tende a dificultar o uso de programas.

A recuperação de erros é outro item importante. O que acontecerá, por exemplo, se você fornecer o nome de alguém, mas perceber, logo depois, que cometeu um erro de digitação? Você deve prosseguir e chamar a sub-rotina MODREG para corrigi-lo? Ou o programa lhe dá a possibilidade de "resolver" antes de seguir adiante? A maioria das versões do BASIC indica a existência de erro na entrada do programa, no momento em que ele é cometido ou quando se processa o programa. Mas isso não faz parte da "interface do usuário". O BASIC inclui uma série de mensagens que fazem uma segunda indicação ao usuário para o fornecimento dos dados corretos, quando se realiza uma entrada incorreta (por exemplo, a indicação REDO quando é fornecido um dado inadequado à instrução INPUT).

A manipulação de erros apresenta dois aspectos — o relato de erros e sua recuperação. Um programa muito conhecido de processamento de palavras tem boa capacidade de relato de erros, mas pouca de recuperação; se você criar um documento extenso e tentar gravá-lo em disco que já esteja repleto, o programa fornece uma mensagem muito útil: DISK SPACE EXHAUSTED (CAPACIDADE DO DISCO ESGOTADA). Mas isso não basta para a recuperação do erro; não será possível a formatação de novo disco, sem a destruição do texto que você levou tanto tempo para digitar.

Qualquer operação realizada pelo usuário que resulte em perda de dados (MODREG, por exemplo) deve ser examinada antes da execução. Convém que existam perguntas do tipo ESTE PROCEDIMENTO FARÁ DESTRUIR O REGISTRO, TEM CERTEZA DE QUERÊ-LO? (S/N). No programa processador de palavras uma mensagem equivalente seria: O COMANDO "SAVE" NÃO MANTERÁ CÓPIA DO DOCUMENTO ANTERIOR. ESTÁ OK DESSE MODO? (S/N).

Leve em conta a manipulação de erros (detecção e relato) na elaboração do programa, sempre que houver possibilidade de fornecimento incorreto de dados; escolha errada do menu; digitação de comandos impróprios; e dados passíveis de mudança ou gravação, sobretudo se o procedimento de gravação implicar registros sobrepostos a dados anteriores.

Dê atenção à segurança: o que acontecerá ao programa ou aos dados se ocorrer um erro irremediável (como a interrupção do fornecimento de energia elétrica)? O programador precisa considerar o limite admissível de perda dos dados e criar métodos que possibilitem recuperá-los em grande parte ou utilizar os restantes. Um programa processador de palavras mais sofisticado inclui outro, chamado RECUPERAÇÃO, de modo que, se ocorrer uma falha irremediável (como a interrupção do fornecimento de energia ou o desligamento do computador antes da gravação do documento), quase nada se perde. Tais técnicas avançadas de programação estão além dos objetivos deste curso. De todo modo, a questão consiste em tornar seguros os programas pela previsão de eventuais falhas graves (com as quais se possa lidar), e desenvolver rotinas para dar conta delas.

A flexibilidade — a rapidez com que o programa pode se adaptar às necessidades do usuário — também é importante. Já tratamos desta questão algumas vezes. Em nível mais simples, consiste no empenho de sempre deixar bastante espaço entre os números de linha (em BASIC) e incorporar instruções REM suficientes para posterior preenchimento com instruções GOSUB, se necessário. Ao elaborar matrizes, pelo menos uma delas deve ser redundante, de modo a permitir ampliação futura. Uma regra básica da técnica de desenvolvimento de programas diz que não se prevêem as necessidades futuras. Mas não resta dúvida de que um bom programa sempre pode ser aperfeiçoado — e seu aprimoramento provavelmente significará a inclusão de novas instruções.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
HCOLOR=expressão	Fornece a cor para desenhos em gráficos de alta resolução		+		
HEX$(n)	Converte n em hexadecimal	+			
HGR	Comando para gráficos de alta resolução	+	+		
HIMEM	Gera limite superior da memória para programa BASIC		+		
HLIN x,y AT z	Traça uma linha a partir de x para y na posição z da tela	+	+		
HOME	Limpa a tela	+	+		
HPLOT x,y	Desenha ponto ou linha em gráfico de alta resolução	+	+		
HTAB y	Posiciona o cursor numa coluna da linha atual		+		
IF expr THEN c11 ELSE c12	Executa c11 se expr for verdadeira. Se não, executa c12	+	+	+	
IF expr THEN instrução	Se expr for verdadeira, então expressão será executada				+
IN#slot	Indica o número do slot para as próximas entradas		+		
INIT "nomearquivo"	Inicializa um disco com o arquivo da memória		+		
INKEY$	Lê um caractere do teclado	+		+	+
INP (porta)	Fornece o valor atual da porta especificada			+	
INPUT variável	Pára e aguarda uma entrada de dados				+
INPUT"mensagem"; lista de var	Lê variáveis a partir do teclado	+	+	+	
INPUT#-1,lista de variáveis	Carrega dados provenientes de uma fita cassete			+	
INPUT"arquivo",lista de variáveis	Lê dados de um arquivo	+			
INPUT$(n,"arquivo")	Lê n caracteres de um arquivo	+			
INSTR(n,x$,y$)	Procura a 1.ª ocorrência de y$ em x$ e fornece a posição	+			

```
10 REM "PRGPRN"
20 REM *INICIL*
30 GOSUB 1000
40 REM *SAUDAR*
50 GOSUB 3000
60 REM *ESCLHA*
70 GOSUB 3500
80 REM *EXECUT*
90 GOSUB 4000
100 END
1000 REM SUB-ROTINA *INICIL*
1010 GOSUB 1100: REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1020 GOSUB 1300: REM SUB-ROTINA *LERARQ* (LE ARQUIVOS)
1030 GOSUB 1380: REM SUB-ROTINA *ESTFLG* (ESTABELECE FLAGS)
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1110 DIM CAMPNOM$(50)
1120 DIM CAMPMOD$(50)
1130 DIM CAMPRUA$(50)
1140 DIM CAMPCID$(50)
1150 DIM CAMPEST$(50)
1160 DIM CAMPTEL$(50)
1170 DIM CAMPIND$(50)
1180 REM
1190 REM
1200 REM
1210 LET TAMA = 0
1220 LET RMOD = 0
1230 LET GRAV = 0
1240 LET CORR = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 RETURN
1300 REM SUB-ROTINA *LERARQ*
1310 ON ERROR GOTO 1370
1320 OPEN "I", #1, "AGEN.DAD"
1330 FOR L = 1 TO 50
1340 INPUT# CAMPNOM$(L), CAMPRUA$(L), CAMPCID$(L),
     CAMPEST$(L), CAMPTEL$(L)
1350 NEXT L
1360 CLOSE #1
1370 RETURN
1380 REM ROTINA FICTICIA *ESTFLG*
1390 RETURN
3000 REM SUB-ROTINA *SAUDAR*
3010 PRINT
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT TAB(14); "*BENVINDO AO*"
3060 PRINT TAB(4); "*MICROCOMPUTADOR — CURSO BASICO*"
3070 PRINT TAB(1); "*AGENDA DE ENDERECOS COMPUTADORIZADA*"
3080 PRINT
3090 PRINT " (PRESSIONAR BARRA DE ESPACO PARA CONTINUAR) "
3100 FOR L = 1 TO 1
3110 IF INKEY$ <> " " THEN L = 0
3120 NEXT L
3130 PRINT CHR$(12)
3140 RETURN
3500 REM SUB-ROTINA *ESCLHA*
3510 REM
3520 REM "MENESC"
3530 PRINT CHR$(12)
3540 PRINT "ESCOLHER UM DOS SEGUINTES:"
3550 PRINT
3560 PRINT
3570 PRINT
3580 PRINT "1. ENCONTRAR UM REGISTRO (PELO NOME)"
3590 PRINT "2. ENCONTRAR NOMES (POR TRECHO DE UM NOME)"
3600 PRINT "3. ENCONTRAR REGISTROS (DE UMA CIDADE)"
3610 PRINT "4. ENCONTRAR REGISTROS (DE UMA INICIAL)"
3620 PRINT "5. LISTAR TODOS OS REGISTROS"
3630 PRINT "6. ACRESCENTAR UM REGISTRO"
3640 PRINT "7. MODIFICAR UM REGISTRO"
3650 PRINT "8. ELIMINAR UM REGISTRO"
3660 PRINT "9. SAIR E GRAVAR"
3670 PRINT
3680 PRINT
3690 REM "ATRESC"
3700 REM
3710 LET L = 0
3720 LET I = 0
3730 FOR L = 1 TO 1
3740 PRINT "ESCOLHER DE 1 A 9"
3750 FOR I = 1 TO 1
3760 LET A$ = INKEY$
3770 IF A$ = " " THEN I = 0
3780 NEXT I
3790 LET ESCL = VAL(A$)
3800 IF ESCL <1 THEN L = 0
3810 IF ESCL >9 THEN L = 0
3820 NEXT L
3830 RETURN
4000 REM SUB-ROTINA *EXECUT*
4010 ON ESCL GOSUB 310,330,350,370,390,410,430,460,470
4020 RETURN
9000 REM SUB-ROTINA *ACRREG*
9010 PRINT CHR$(12)
9020 INPUT "FORNECER O NOME" ; CAMPNOM$(TAMA)
9030 INPUT "FORNECER A RUA" ; CAMPRUA$(TAMA)
9040 INPUT "FORNECER A CIDADE" ; CAMPCID$(TAMA)
9050 INPUT "FORNECER O ESTADO" ; CAMPEST$(TAMA)
9060 INPUT "FORNECER O NUMERO TELEFONICO" ; CAMPTEL$(TAMA)
9070 LET RMOD = 1
9080 LET CAMPIND$(TAMA) = STR$(TAMA)
9090 GOSUB *NOMMOD*
9100 RETURN
```

Valores fictícios

Para utilizar um arquivo é necessário criar primeiro uma estrutura e depois preenchê-la com dados.

No fascículo anterior, deixamos ao leitor um dilema para ser resolvido: como fazer o programa, que é processado pela primeira vez, consultar um arquivo que não existe (em fita ou disco)? A atividade inicial que provavelmente tentaremos fazê-lo executar será a consulta ao arquivo de dados e a atribuição desses dados a matrizes ou a variáveis. Ainda assim, se insistirmos nesse procedimento, sempre que o programa for processado, teremos de ser muito cuidadosos com a programação, para evitar a perda de todos os dados arquivados. Como vimos da última vez, a tentativa de abrir um arquivo não-existente simplesmente não funcionará ou então fará com que o programa "engasgue" (pare de funcionar).

Mas existe uma solução bastante simples para o dilema: muitos dos pacotes de software à venda incluem um programa de "instalação" ou de "preparação" que deve ser processado antes que o programa propriamente dito seja usado; e será esta a abordagem que adotaremos. Tais programas, em geral, permitem ao usuário algumas pequenas "adaptações" (como decidir se a impressora a ser usada deverá ser do tipo Epson ou Brother, paralela ou serial etc.), mas também criam arquivos de dados que posteriormente serão utilizados pelo programa principal. Lembre-se de que, ao contrário de arquivos de programas, os de dados podem ser acessados por qualquer programa (ver p. 316).

Para resolver nosso problema e permitir a execução da sub-rotina *LERARQ* (que lê o arquivo e atribui dados às matrizes), podemos escrever um programa de preparação que simplesmente abre um arquivo e nele registra valores fictícios. Escolhemos um valor que pode ser depois identificado pelo programa efetivo como não sendo um registro válido da agenda de endereços. Um valor conveniente seria o conjunto de caracteres @PRIMEIRO, pois não há possibilidade de que algum nome ou endereço comece assim, por mais estranha que seja sua origem. A sub-rotina *LERARQ* deverá ser um pouco alterada, de modo que, quando for aberta e a leitura executada, ela verifique esse valor antes de prosseguir. Se seu computador não dispuser do símbolo @, você deverá substituí-lo por um sinal "!" ou algum outro caractere que componha um nome mas não tenha possibilidade de vir a constar de sua agenda de endereços. A seguir, apresentamos um programa de preparação.

```
10 REM ESTE PROGRAMA CRIA UM ARQUIVO DE
   DADOS
20 REM PARA USO PELO PROGRAMA DA AGENDA
   DE ENDERECOS
30 REM REGISTRA UM ARQUIVO FICTICIO QUE
   PODE
40 REM SER USADO PELA SUB-ROTINA *LERARQ*
50 REM
60 REM
70 OPEN "O", #1, "AGEN.DAD"
80 PRINT #1, "@PRIMEIRO"
90 CLOSE #1
100 END
```

Conforme já mencionamos neste curso, os pormenores de leitura e desenvolvimento de arquivos diferem muito de uma versão para outra da linguagem BASIC, mas o princípio é quase sempre o mesmo. Primeiro, o arquivo deve ser declarado aberto antes de poder ser utilizado. A seguir, é estabelecida a direção do fluxo de dados, podendo ser de entrada (IN) ou de saída (OUT). Depois, atribui-se um número de "canal" ao arquivo. Isso permite a abertura e utilização de mais de um arquivo ao mesmo tempo (por enquanto, utilizaremos um único arquivo). Por fim, deve ser declarado o nome do arquivo escolhido.

A linha 70 do programa anterior está no BASIC da Microsoft e se assemelha, em princípio, às instruções OPEN utilizadas pela maioria das versões dessa linguagem. OPEN informa que vai ser aberto um arquivo e o dígito "O" indica que dados serão fornecidos a partir dele. O #1, por sua vez, mostra que o número 1 está sendo atribuído ao arquivo para essa operação; uma forma diferente de numeração de arquivo poderá ser empregada depois, se necessário. O nome dado ao arquivo é "AGEN.DAD".

A linha 80 simplesmente desenvolve um registro individual para o arquivo. A sintaxe do registro de dados no arquivo é (na maioria das linguagens BASIC) a mesma utilizada para impressão (PRINT), exceto que a instrução PRINT deve ser seguida pelo número de arquivo (#1, neste caso).

A linha 90 encerra o arquivo. Este poderia ficar aberto durante o tempo necessário no programa. Mas isso não é recomendável. Arquivos "abertos" são muito vulneráveis e devem ser fechados logo que possível no interior do programa, para proteger os dados neles contidos. Se, digamos, você desligar acidentalmente o computador enquanto o arquivo estiver aberto, talvez, quando vier a lê-lo, constate que os dados se perderam.

Há uma pequena confusão sobre o modo como os termos registro e arquivo são empregados em computadores, e essa confusão se acentua quando falamos de banco de dados e de arquivo de dados. Nos bancos, o arquivo consiste em um conjunto completo de dados relacionados. Por analogia com os tradicionais arquivos de escritório, o arquivo pode ser comparado a uma gaveta com a denominação PESSOAL. Ele abrange, por exemplo, um registro (um cartão em uma pasta) para cada pessoa da empresa. Cada um desses registros conterá uma série de campos idênticos, com as informações de NOME, SEXO, IDADE, SALÁRIO, TEMPO DE SERVIÇO etc.

Se o arquivo PESSOAL for computadorizado, todos os dados serão estruturados da mesma forma — um arquivo incluindo vários registros, cada registro com vários campos —, exatamente como nossa agenda de endereços computadorizada.

Entretanto, o arquivo seqüencial em fita cassete ou em disco não leva em conta o modo como os dados nele contidos são utilizados ou organizados pelo programa. Os arquivos de dados contêm apenas uma série de itens. E cada qual é um registro. Ou seja, o registro isolado no arquivo de dados normalmente não corresponde a um registro no sentido do banco de dados.

Cabe ao programa ler os registros do arquivo e atribuí-los às variáveis ou às matrizes, e estas devem ser organizadas de modo que formem um registro conceitual com um conjunto limitado de dados relacionados. Não há relação um-a-um, ou seja, correspondência entre os registros no arquivo de dados e os registros de um banco de dados.

Depois de processado, o programa de preparação não será mais necessário. Se for novamente rodado, destruirá todos os dados "válidos" que você houver fornecido ao banco de dados da agenda de endereços. Veremos por que isso acontece quando examinarmos o programa *LERARQ* já modificado.

O programa, durante a operação, não "sabe" se há ou não dados válidos no arquivo. A primeira coisa que a sub-rotina *LERARQ* efetua é abrir o arquivo "AGEN.DAD" e ler o primeiro registro (ou item de dados). Este não é lido em um elemento da matriz, como se poderia esperar, mas, sim, em uma variável alfanumérica especial a que chamamos TEST$. Antes da leitura de qualquer outro registro, esta variável é testada para verificar se contém o conjunto @PRIMEIRO. Se contiver, o programa sabe que não há dados válidos no arquivo e, desse modo, não mais precisa tentar lê-los e atribuí-los às matrizes. Conseqüentemente, o arquivo pode ser fechado e o resto do programa continua. Como não há dados válidos no arquivo, o usuário nada pode fazer até que dê entrada a um registro e o valor da variável TEST$ também possa ser usado, para permitir que o programa alcance a sub-rotina *ACRREG* e assim ao menos um registro válido seja acrescentado antes que se passe a outro procedimento.

Se, no entanto, o valor da variável TEST$ não for @PRIMEIRO, o programa pode presumir que há dados válidos no arquivo e começar a atribuí-los às matrizes correspondentes. Eis a sub-rotina *LERARQ* modificada:

```
1400 REM SUB-ROTINA "LERARQ"
1410 OPEN "I", #1, "AGEN.DAD"
1420 INPUT #1 , TEST$
1430 IF TEST$ = "@PRIMEIRO" THEN GOTO 1540: REM CLOSE E RETURN
1440 LET CAMPNOMS(1) = TEST$
1450 INPUT #1 , CAMPMODS(1) , CAMPRUAS(1) , CAMPCIDS(1) ,
     CAMPESTS(1) , CAMPTEL$(1)
1460 INPUT #1 , CAMPINDS(1)
1470 LET TAMA = 2
1480 FOR L = 2 TO 50
1490 INPUT #1 , CAMPNOMS(L) , CAMPMODS(L) ,
     CAMPRUAS(L) , CAMPCIDS(L) , CAMPESTS(L)
1500 INPUT #1 , CAMPTEL$(L) , CAMPINDS(L)
1510 REM ESPAÇO PARA CHAMAR SUB-ROTINA "TAMARQ"
1520 REM
1530 NEXT L
1540 CLOSE #1
1550 RETURN
```

A linha 1420 atribui um único registro proveniente do arquivo "AGEN.DAD" à variável TEST$. A linha seguinte verifica se seu valor é @PRIMEIRO. Se for, é utilizada uma instrução GOSUB para saltar até a linha que encerra o arquivo (linha 1540) e, a seguir, a sub-rotina retorna ao programa que a chamou. Não são feitas outras tentativas de leitura. Supondo que não haja dados válidos no arquivo, o controle do programa retornará à sub-rotina *INICIL*, que então chama a sub-rotina *ESTFLG*. Esta, no momento, apenas atribui o valor 1 à variável TAMA se a variável TEST$ = @PRIMEIRO. As instruções para a sub-rotina *ESTFLG* são dadas abaixo. Observe que há várias instruções REM que criam espaço para a inclusão de outras flags, se o quisermos depois.

```
1600 REM SUB-ROTINA "ESTFLG"
1610 REM ESTABELECE FLAGS APÓS "LERARQ"
1620 REM
1630 REM
1640 IF TEST$ = "@PRIMEIRO" THEN LET TAMA = 1
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
```

A sub-rotina *ESTFLG* retorna à sub-rotina *INICIL*, que, por sua vez, volta ao programa principal. A rotina *PRGPRN* chama a seguir a sub-rotina *SAUDAR*, que apresenta a mensagem de saudação na versão já publicada deste programa.

A rotina chamada a seguir pelo programa principal é *ESCLHA*. Uma pequena modificação da sub-rotina *ESCLHA* na página 357 determinará o modo de levar o usuário a acrescentar um registro, se o programa estiver sendo processado pela primeira vez.

```
3500 REM SUB-ROTINA "ESCLHA"
3510 REM
3520 IF TEST$ = "@PRIMEIRO" THEN GOSUB 3860
3530 IF TEST$ = "@PRIMEIRO" THEN RETURN
3540 REM "MENESC"
3550 PRINT CHR$(12)
3560 PRINT "ESCOLHER UM DOS SEGUINTES."
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. ENCONTRAR UM REGISTRO (PELO NOME)"
3610 PRINT "2. ENCONTRAR NOMES (POR TRECHO DE UM NOME)"
3620 PRINT "3. ENCONTRAR REGISTROS (DE UMA CIDADE)"
3630 PRINT "4. ENCONTRAR REGISTROS (DE UMA INICIAL)"
3640 PRINT "5. LISTAR TODOS OS REGISTROS"
3650 PRINT "6. ACRESCENTAR UM REGISTRO"
3660 PRINT "7. MODIFICAR UM REGISTRO"
3670 PRINT "8. ELIMINAR UM REGISTRO"
3680 PRINT "9. SAIR E GRAVAR"
3690 PRINT
3700 PRINT
3710 REM "ATRESC"
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 1 TO 1
3760 PRINT "ESCOLHER DE 1 A 9 "
3770 FOR I = 1 TO 1
3780 LET AS = INKEY$
3790 IF AS = "" THEN I = 0
3800 NEXT I
3810 LET ESCL = VAL(AS)
3820 IF ESCL < 1 THEN L = 0
3830 IF ESCL > 9 THEN L = 0
3840 NEXT L
3850 RETURN
```

Duas linhas foram acrescentadas. A primeira testa a variável TEST$, que ainda contém o valor lido na rotina *LERARQ*. Se for @PRIMEIRO, saberemos que não há dados válidos no arquivo e, desse modo, a única alternativa adequada é a rotina *ACRREG*, de número 6. Se o teste passar, o controle é transferido à rotina *PRIMEI*, que apresenta uma mensagem adequada e atribui o valor 6 à variável ESCL. Quando a sub-rotina retorna à linha 3530, a variável TEST$ é

novamente testada (ela está destinada a passar) e a sub-rotina retorna ao programa principal, saltando o resto da sub-rotina *ESCLHA*, uma vez que isto é inadequado.

Você talvez tenha estranhado a necessidade de TEST$ ser testada duas vezes. Isso ocorre para evitar que a sub-rotina retorne ao ponto errado do programa. Sem a linha 3530, o programa prosseguiria pelo resto da rotina *ESCLHA*, apresentando o menu de escolha, ainda que desnecessário. Ela também evita o emprego de instruções GOTO, embora a instrução IF TEST$ = "@PRIMEIRO" THEN GOTO 3850 funcionasse igualmente bem. As instruções GOTO tornam o programa confuso e difícil de acompanhar (os programas que fazem uso excessivo de instruções GOTO são apelidados de "codificação espaguete").

Antes de examinar a sub-rotina *PRIMEI*, remeteremos o leitor à rotina *LERARQ* e à instrução GOTO na linha 1430. Já que advertimos incisivamente o leitor contra o emprego de instruções GOTO, por que usamos uma delas aqui? Teria sido fácil encerrar o arquivo e retornar através do teste do valor da variável TEST$ em duas linhas separadas. Na verdade, utilizamos GOTO aqui para exemplificar uma das poucas circunstâncias em que seu emprego é admissível: no interior de segmentos de programas muito curtos e identificáveis. Sua função é evidente (e se torna ainda mais pelo comentário REM). As instruções GOTO não devem nunca ser usadas para sair de um loop (isto pode deixar o valor das variáveis em estado imprevisível), muito menos de uma sub-rotina (isto tornaria confusa a instrução RETURN, a menos que uma correspondência de retorno seja empregada na rotina), nem utilizada para saltar até pontos distantes do programa (isto faz com que o programa se torne impossível de ser acompanhado).

A sub-rotina *PRIMEI* é simples e direta: a tela é limpa e uma mensagem informa ao usuário que ele deve fornecer um registro. A linha 3870 atribui o valor 6 à variável ESCL, de modo que, quando o controle retornar à sub-rotina *EXECUT*, a rotina *ACRREG* será executada automaticamente. A codificação para a sub-rotina *PRIMVZ* é a seguinte:

```
3860 REM SUB-ROTINA "PRIMVZ" (DA MENSAGEM)
3870 LET ESCL = 6
3880 PRINT CHR$(12); REM LIMPA TELA
3890 PRINT
3900 PRINT TAB(8); "NAO EXISTEM REGISTROS NO"
3910 PRINT TAB(8); "ARQUIVO. VOCE DEVERA COMECAR"
3920 PRINT TAB(8); "ACRESCENTANDO UM REGISTRO"
3930 PRINT
3940 PRINT "(PRESSIONAR A BARRA DE ESPACO PARA CONTINUAR)"
3950 FOR B = 1 TO 1
3960 IF INKEY$ <> " " THEN B = 0
3970 NEXT B
3980 PRINT CHR$(12); REM LIMPA TELA
3990 RETURN
```

A sub-rotina *ACRREG*, apresentada na página 379, tem duas pequenas mas importantes alterações em relação à versão antes apresentada. Após fornecermos os campos como elementos das várias matrizes alfanuméricas, a variável TAMA é aumentada e atribui-se à variável TEST$ um conjunto nulo (ver as linhas 10120 e 10090). TAMA tem importância porque, utilizada em várias partes do programa, permite que este identifique os registros que estão sendo operados. Essa variável recebeu inicialmente o valor 0 como parte da sub-rotina *CRIMAT*. Mais adiante, na sub-rotina *ESTFLG* ela recebe o valor 1 se TEST$ = "@PRIMEIRO". Isso ocorre de tal modo que, quando a sub-rotina *ACRREG* é executada pela primeira vez, as instruções INPUT colocam os dados no primeiro elemento de cada matriz. Em outras palavras, INPUT "FORNECER O NOME"; CAMPNOM$(TAMA) corresponde a INPUT "FORNECER O NOME";CAMPNOM$(1).

A linha 10120 aumenta o valor da variável TAMA que passa agora a ser 2. Se a rotina *ACRREG* for executada novamente, os dados serão fornecidos no segundo elemento de cada matriz. Finalmente, a rotina *ACRREG* atribui " " à variável TEST$, na linha 10090. Faz-se isso porque agora foi introduzido um registro (embora ainda não armazenado em fita ou arquivo de dados em disco). Se a sub-rotina *ESCLHA* for novamente executada — e deve sê-lo para gravar os dados e sair do programa —, não vamos querer ser forçados a acrescentar outros registros. Se a variável TEST$ não for eliminada, o programa ficará bloqueado em um loop sem fim, e o único modo de sair será zerar ou desligar o computador, perdendo-se todos os dados.

A atribuição de uma série nula a TEST$ fará com que falhem os testes das linhas 3520 e 3530 de *ESCLHA*, permitindo a apresentação do menu de opções. O que acontecerá então à variável TAMA vai depender da rotina em execução. Até aqui apenas garantimos que TAMA=1, se não houver dados válidos no arquivo, e que isso sofre um acréscimo de uma unidade, toda vez que se adicionar um novo registro. Porém, o que acontecerá se houver uma série de registros válidos no arquivo? Para responder à questão teremos de examinar a sub-rotina *LERARQ* novamente.

A linha 1420 lê o primeiro item de dados na variável TEST$. Se não for @PRIMEIRO, fica suposto que o item de dados é válido. Os registros no arquivo sempre estão na mesma ordem, a saber: CAMPNOM, CAMPMOD, CAMPRUA, CAMPCID, CAMPEST, CAMPTEL, CAMPIND, CAMPNOM, CAMPMOD etc. Se o primeiro registro lido for um dado válido, deve pertencer ao primeiro elemento da matriz CAMPNOM$; assim, a linha 1440 transfere esse dado da variável TEST$ para a CAMPNOM$(1). As duas linhas seguintes completam os primeiros elementos nas outras cinco matrizes. Sabemos agora que temos pelo menos um registro (banco de dados) completo; assim, a variável TAMA recebe o valor 2, uma unidade maior que o número de registros válidos fornecidos às matrizes. Caso contrário, a sub-rotina *ACRREG* registraria os novos dados em elementos que já contêm dados válidos.

A seguir, um loop de 2 a 50 fornece os dados para todas as seis matrizes, aumentando o índice L em cada passagem. Já decidimos limitar nosso programa a operar com arquivos contendo cinqüenta nomes e endereços; as instruções DIM, na sub-rotina *CRIMAT*, reservaram espaço para isso. Entretanto, quando começar a usar o programa, é improvável que você disponha de arquivo completo com cinqüenta entradas. Assim, vamos precisar de uma rotina no programa que possa detectar esse fato, adequar a variável TAMA e interromper o loop de leitura.

Por esse motivo, incluímos a linha 1510, que pos-

sibilita chamar uma sub-rotina *TAMARQ*, com a qual trabalharemos mais adiante no curso. Há três modos de lidar com esse problema. Primeiramente, quando transferimos os dados para a fita, podemos fazer com que o primeiro registro inscrito seja a variável TAMA. A sub-rotina *LERARQ* pode então ser alterada para fornecer os dados, em primeiro lugar, a TAMA e, a seguir, criar um loop na forma FOR L = 1 TO TAMA, para colocar os dados no registro. O segundo (e preferível) método (uma vez que não contradiz nosso teste anterior para a flag |@|PRIMEIRO, na linha 1430) consiste em estabelecer um procedimento a ser executado após todos os registros terem sido transferidos — aí uma flag especial (na forma @|ENCERRAR, por exemplo) pode ser incluída no final. Torna-se então viável a inserção de um teste na sub-rotina *LERARQ* para interromper o loop ao encontrar a flag @ENCERRAR.

O terceiro método consiste em usar a função EOF (END OF FILE, fim do arquivo) existente em alguns computadores, a qual na verdade é uma versão automatizada do segundo método. Esses computadores possuem uma flag EOF na maioria das vezes com o valor 0, isto é, FALSO, mas assumindo novamente valor (quase sempre valor 1, que representa VERDADEIRO), quando o fim do arquivo é atingido. Alguns equipamentos em BASIC permitem testar a flag EOF como uma variável dessa linguagem; nesse caso, o problema será resolvido através de uma construção, com a seguinte forma:

```
ENQUANTO (WHILE) NÃO EOF(N) (N é o número
do arquivo)
  REALIZAR (DO)
    ENTRAR (INPUT #N, dados para ler)
ENCERRAR O LOOP WHILE-DO
```

Em outras máquinas, a flag EOF é representada como um único bit que deve ser acessado por meio da instrução PEEK. Para saber se seu micro possui a função EOF, você precisará consultar o manual de instruções. Não usaremos essa instrução em nosso programa porque difere muito de uma máquina para outra. Como exercício, porém, talvez o leitor queira experimentar uma alteração da sub-rotina *LERARQ* para os três métodos possíveis de lidar com arquivos de menos de cinqüenta itens.

Geralmente, é muito mais fácil desenvolver programas que lidam com arquivos de extensão fixa, mas resolver o problema dos de extensão variável e dinâmica nos ajudará a alterar depois o programa, ampliando-o para mais de cinqüenta itens.

```
4000 REM SUB-ROTINA *EXECUT*
4010 REM
4019 IF ESCL = 6 THEN GOSUB 10000: REM CONSULTAR NOTA DE RODAPE
4020 REM COMO USUAL "ON ESCL GOSUB etc." — CONSULTAR A NOTA DE RODAPE
4030 REM
4040 REM 1 = *ENCREG*
4050 REM 2 = *ENCNOM*
4060 REM 3 = *ENCCID*
4070 REM 4 = *ENCINI*
4080 REM 5 = *LISREG*
4090 REM 6 = *ACRREG*
4100 REM 7 = *MODREG*
4110 REM 8 = *ELMREG*
4120 REM 9 = *SAIPRG*
4130 REM
4140 RETURN
```

A rotina *EXECUT* não teria, em condições normais, a linha 4019 (daí atribuir-lhe um número ímpar) e a linha 4020 seria:

```
ON ESCL GOSUB número,número,número etc.
```

ou, então, uma série como a seguinte:

```
IF ESCL = 1 THEN GOSUB número
IF ESCL = 2 THEN GOSUB número etc.
```

A linha 4019 foi incluída para que o programa funcione mesmo sem a prévia codificação das demais sub-rotinas *EXECUT*.

```
10 REM "PRGPRN"
20 REM *INICIL*
30 GOSUB 1000
40 REM *SAUDAR*
50 GOSUB 3000
60 REM *ESCLHA*
70 GOSUB 3500
80 REM *EXECUT*
90 GOSUB 4000
100 END
1000 REM SUB-ROTINA *INICIL*
1010 GOSUB 1100: REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1020 GOSUB 1400: REM SUB-ROTINA *LERARQ* (LE ARQUIVOS)
1030 GOSUB 1600: REM SUB-ROTINA *ESTFLG* (ESTABELECE FLAGS)
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1110 DIM CAMPNOM$(50)
1120 DIM CAMPMOD$(50)
1130 DIM CAMPRUA$(50)
1140 DIM CAMPCID$(50)
1150 DIM CAMPEST$(50)
1160 DIM CAMPTEL$(50)
1170 DIM CAMPIND$(50)
1180 REM
1190 REM
1200 REM
1210 LET TAMA = 0
1220 LET RMOD = 0
1230 LET GRAV = 0
1240 LET CORR = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN
10000 REM SUB-ROTINA *ACRREG*
10010 PRINT CHR$(12): REM LIMPA TELA
10020 INPUT "FORNECER O NOME" ; CAMPNOM$(TAMA)
10030 INPUT "FORNECER A RUA" ; CAMPRUA$(TAMA)
10040 INPUT "FORNECER A CIDADE" ; CAMPCID$(TAMA)
10050 INPUT "FORNECER O ESTADO" ; CAMPEST$(TAMA)
10060 INPUT "FORNECER O NUMERO TELEFONICO"; CAMPTEL$(TAMA)
10070 LET RMOD = 1: REM FLAG DE REGISTRO MODIFICADO
10080 LET CAMPIND$(TAMA) = STR$(TAMA)
10090 LET TEST$ = " "
10100 REM INSERIR CHAMADA PARA *MODNOM* AQUI
10110 LET ESCL = 0
10120 LET TAMA = TAMA + 1
10130 REM
10140 REM
10150 RETURN
```

Tempo e movimento

Embora demorada, a tarefa de ordenar matrizes em BASIC permite ganhar tempo na localização de registros específicos.

Até aqui desenvolvemos a maioria dos códigos necessários para criar entradas no "banco de dados" da agenda de endereços. Não examinamos, porém, a programação para gravar essas entradas em fita ou disco. E falta-nos ainda uma rotina adequada para a criação do campo CAMPMOD$, conforme especificação anterior. O programa completo para tanto é apresentado neste fascículo.

Primeiramente, convertemos os caracteres nas linhas de 10250 a 10330 em maiúsculas. As linhas de 10350 a 10370 efetuam então a contagem dos caracteres da variável alfanumérica e examinam se cada um deles corresponde a um espaço. O último espaço encontrado atribui à variável S o valor correspondente a sua posição na variável alfanumérica.

As linhas de 10400 a 10420 transferem um a um os caracteres alfanuméricos maiúsculos para a variável CNOM$, até atingirmos o último espaço. Isso, se eles tiverem o valor ASCII superior a 64. Todo caractere que não passar pelo teste é ignorado; assim, esse procedimento elimina os sinais de ponto final (ASCII 46), de apóstrofo (ASCII 39), de espaço (ASCII 32) e outros. As linhas de 10450 a 10470 efetuam a mesma operação com os caracteres após o espaço final, transferindo-os para SNOM$ (sobrenome).

Se N$ contiver uma única palavra — por exemplo, TREVANIAN —, S receberá o valor 0 e todos os caracteres serão transferidos para SNOM$. A variável empregada no prenome foi denominada CNOM$, em vez de FNOM$. Isso porque as variáveis que começam com as letras "FN" causam confusão em muitos dos dialetos do BASIC, levando-os a identificar uma chamada com uma função definida pelo usuário (FN).

As linhas 10490 e 10500 são necessárias para atribuição de valores nulos a variáveis alfanuméricas nessa rotina, antes de serem usadas de novo. Esse é um aspecto a ser considerado sempre que as estruturas do tipo LET X$=X$+Y$ forem empregadas. Deixando de "limpar" as variáveis, isso resultará no acúmulo progressivo de caracteres indesejados em cada nova utilização. Observe que ESCL recebe o valor 0 na rotina ACRREG, porque só queremos garantir que o usuário acrescente um registro, se não houver nenhum no arquivo (na primeira vez em que o programa é processado).

Agora que temos um meio de incluir quantos registros quisermos no arquivo, necessitamos de um método para gravá-lo em fita ou disco. O sistema mais simples seria escrever todos os registros no arquivo de dados (AGEN.DAD, na versão deste programa), seguindo a ordem em que acaso se encontrem. O principal inconveniente desse tipo de procedimento surge quando temos de localizar um registro específico. Se não pudermos ter certeza de que todos estão ordenados de alguma forma, o único método viável será voltar ao começo, examinando cada registro por vez, para verificar se combina com a "chave" da busca. Se, por acaso, o registro procurado for o último fornecido ao banco de dados, todos os demais terão de ser checados antes. Se o último registro fornecido for Ricardo Bergamo (isto é, CAMPMOD$(TAMA−1) = "BERGAMO RICARDO"), a rotina de busca deve prever o registro como estando em algum ponto próximo ao início do arquivo — caso os registros estejam ordenados. Mas tanto a ordenação como a busca são procedimentos demorados. Precisa-se, portanto, determinar qual dos dois tem prioridade. Adotamos o princípio de que uma agenda de endereços é consultada com muito mais freqüência do que recebe novos dados (ou alguma forma de modificação). Assim, para perder menos tempo, preferimos ordenar os registros antes de armazená-los no arquivo de dados, após a utilização do programa.

Tendo isso em mente, uma variável denominada RMOD é criada para ser usada como flag. Ela pode ter um dos dois valores: 0 ou 1. De início, recebe o valor 0, que indica a não modificação de qualquer registro durante a presente execução do programa. Qualquer operação que altere o arquivo — como o acréscimo de novos registros — atribui o valor 1 a RMOD. As operações que "precisam saber" se o arquivo foi modificado examinarão o valor de RMOD antes de prosseguir. Por exemplo, SAIPRG, a rotina que grava o arquivo e desvia o programa, verifica RMOD na linha 11050. Se RMOD = 0, a gravação e o ordenamento não são necessários, pois o arquivo de dados em fita ou disco deve estar em ordem e em forma não modificada. Outras rotinas, como as que procuram um registro específico, também terão de checar RMOD. Se RMOD for 0, a busca (ou outra operação) pode prosseguir. Se RMOD for 1, a rotina de ordenação terá de ser chamada antes. Quando o arquivo inteiro estiver ordenado, atribui-se novamente o valor 0 a RMOD.

Nossa rotina de ordenação, chamada *CLAREG* na listagem do programa, recoloca RMOD em 0 na linha 11320, após todos os registros terem sido ordenados. Antes de passar ao exame de *SAIPRG* (a rotina que grava o arquivo em fita ou disco e a seguir encerra o programa), convém discorrer um pouco sobre a *CLAREG*. Trata-se de uma forma de técnica de ordenação denominada "bubble sort" (classificação bolha) (ver p. 286). Entre os muitos modos de ordenar dados, este é um dos mais simples e rápidos. Há bons motivos para a adoção de rotinas mais eficientes, mas ordenações muito elaboradas são de compreensão difícil. Portanto, só examinando a quantidade de itens a ser ordenada se pode tomar uma decisão a respeito. A "complexidade de

tempo'' da nossa bubble sort é n^2. Ou seja, o tempo exigido para a ordenação de dados aumenta na proporção do quadrado do número de itens que devem ser ordenados. Se dois itens levarem 4 milissegundos para ordenação, quatro levarão 16 milissegundos, cinqüenta precisarão de 2 segundos e meio, e mil vão requerer mais de 16 minutos. Uma demora de 2 ou 3 segundos é admissível no uso de programas como o nosso, mas não uma de 15 minutos.

A maneira pela qual desenvolvemos este programa possibilita um máximo de cinqüenta registros, o que não implica perda de tempo durante as ordenações. Contudo, mais adiante no curso, esboçaremos algumas técnicas empregadas para criar arquivos dinâmicos, que podem ser ampliados para praticamente qualquer tamanho. Se você tentar uma modificação desse tipo no programa, um dos primeiros problemas a enfrentar será o das rotinas mais elaboradas de ordenação.

Os itens de dados que estamos ordenando são as séries de caracteres em CAMPMOD$(L) e CAMPMOD$(L+1). Os registros só ocorrem se CAMPMOD$(L) for maior que CAMPMOD$(L+1), e o campo do índice (não usado no momento) é atualizado nas linhas 11490 e 11570. Toda vez que dois registros forem permutados, a variável S (para indicar que uma permuta ocorreu) recebe o valor 1. Ao atingir a linha 11290, a rotina de busca checa o valor de S e retrocede para comparar todos os registros outra vez. Quando eles estiverem ordenados, o valor de S será 0 e a rotina vai se encerrar após a recolocação do valor de RMOD em 0.

A rotina SAIPRG (referida como *SAIPRG* na listagem do programa) começa na linha 11000. Ela primeiro examina os registros para verificar se algum deles foi modificado durante a presente execução do programa (linha 11050 IF RMOD = 0 THEN RETURN). Se não houve alteração, não é preciso regravar o arquivo. Assim, a rotina retorna ao programa principal. Isso nos traz de volta à linha 100, que checa o valor de ESCL. Se ESCL tem o valor 9 (como teria se *SAIPRG* fosse executada), o programa principal segue para a instrução END na linha 110.

No caso de o programa verificar que RMOD vale 1, na linha 11050, isso quer dizer que pelo menos um registro foi modificado, e há a possibilidade de os registros não mais estarem em ordem. Sendo assim, a rotina *SAIPRG* chama a sub-rotina de ordenação (linha 11070) e a seguir, após todos os registros terem sido ordenados, grava-os em fita ou disco.

A rotina para gravar (*GRAREG*) é chamada na linha 11090 e começa na linha 12000. Na listagem principal, ela se desenvolve em BASIC Microsoft e, portanto, os detalhes para manipulação do arquivo variam de uma versão para outra dessa linguagem. A linha 12030 abre o arquivo de dados AGEN.DAD e atribui o número de canal #1 à operação. A linha 12050 determina os limites do loop que conta todos os registros do arquivo. O limite superior é TAMA−1, e não TAMA, porque esta última variável sempre tem um valor que é uma unidade maior do que o número de registros válidos no arquivo. Assim, ao ser acrescentado um novo registro, este não se sobrepõe a um já existente.

O formato das linhas 12060 e 12070 merece atenção. Cada campo é separado por uma ",", também enviada ao arquivo. Essa vírgula é necessária à maioria das versões do BASIC porque INPUT # e PRINT# operam do mesmo modo que as instruções normais INPUT e PRINT. Observe a instrução INPUT X,Y,Z. Ela aguardaria uma entrada do teclado como 10,12,15<CR>, que atribuiria 10, 12 e 15 a X, Y e Z, respectivamente. Sem as vírgulas, a instrução INPUT não poderia identificar onde cada item de dados termina e atribuiria todos os dados à primeira variável. De modo semelhante, a instrução INPUT# (na maioria das versões do BASIC) não poderia dizer onde cada registro do arquivo termina e tentaria preencher cada variável alfanumérica com tantos dados quantos coubessem. Uma vez que, na maioria das versões do BASIC, as variáveis alfanuméricas comportam até 255 caracteres, os dados seriam atribuídos muito antes que o loop FOR L = 1 TO TAMA−1 se encerrasse. Isso resultaria numa mensagem de erro INPUT PAST END (indicando que uma instrução INPUT foi emitida após ter sido atingido o limite da capacidade de dados), e as variáveis alfanuméricas (como NAMFLD$(x)) acabariam por conter muito mais dados do que sua capacidade.

Uma vez armazenados todos os registros no arquivo, de L = 1 TO TAMA−1, *GRAREG* retorna à linha 90 no programa principal. A linha 100 examina o valor de ESCL para checar se a última operação foi *SAIPRG* ou não. No caso de ter sido 9 (gravar e desviar), o programa segue para a instrução END na linha 110. Se ESCL tiver qualquer outro valor, o programa retorna a *ESCLHA* e possibilita outra opção ao usuário.

Como observação final, mencionamos a rotina *TAMARQ* que começa na linha 12500. Ela é oferecida como possível alternativa à instrução da linha 1520. Do modo como está apresentado, o programa depende da presença de uma função de encerramento do arquivo: IF EOF(1) = −1 THEN LET L = 50. Todos os BASIC têm alguma forma de indicar que o fim de um arquivo foi atingido, por meio de função especial, como EOF(x), ou pelo acesso a uma posição especial da memória, mediante o comando PEEK. A rotina *TAMARQ*, na linha 12500, é apresentada como sugestão, caso não exista a função EOF no equipamento e a linha 1520 tenha de ser substituída por GOSUB 12500.

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
INT(x)	Fornece o maior inteiro que é menor ou igual a x	+	+	+	+
INVERSE	Passa o monitor para a forma de vídeo inverso		+		
KILL "arquivo"	Elimina um arquivo do disquete	+			

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
LEFT$(x$,n)	Fornece o enésimo caractere mais à esquerda de x$	+	+	+	
LEN (x$)	Fornece o número de caracteres de x$	+	+	+	+
LET variável=expressão	Aloca o valor de expressão à variável	+	+	+	+
LINE INPUT"mensagem";var$	Lê uma linha inteira a partir do teclado e aloca a var$	+			
LINE INPUT#arquivo,var$	Lê uma linha inteira de um arquivo e aloca a var$	+			
LINE (x1,y1)-(x2,y2)	Desenha uma linha na tela entre os dois pontos	+			
LIST linha	Lista na tela a partir da linha especificada				+
LIST linha1-linha2	Lista as linhas especificadas no vídeo	+	+	+	
LLIST linha	Lista o programa na impressora a partir da linha				+
LLIST linha1-linha2	Lista as linhas especificadas na impressora	+		+	
LN x	Fornece o logaritmo natural de x				+
LOAD "arquivo"	Carrega um arquivo de programa	+	+		+
LOC (a)	Fornece a posição atual no arquivo	+			
LOCATE linha,coluna	Posiciona o cursor no ponto indicado da tela	+			
LOCK "nomearquivo"	Protege um arquivo em disco		+		
LOG(x)	Fornece o logaritmo natural de x	+	+	+	
LOMEM	Gera limite inferior da memória para programa BASIC		+		
LPOS (n)	Fornece a posição atual da cabeça da impressora no buffer	+			
LPRINT lista de expressões	Lista um conjunto de variáveis na impressora	+		+	+
MEM	Fornece o total de bytes disponíveis na memória			+	
MERGE "arquivo"	Intercala um programa gravado com o da memória	+			
MID$(v$,n,m)=y$	Aloca na vary$ parte da varyv$, iniciando em n m carac	+		+	
MID$(x$,n,m)	Fornece m caracteres da variável x$ a partir da posição n	+	+		
NAME "arquivo" AS "nome"	Muda o nome do arquivo	+			
NEW	Elimina o programa atual e variáveis	+	+	+	+
NEXT variável	Encerra um loop FOR-NEXT	+	+	+	+
NO TRACE	Desliga o comando TRACE		+		
NORMAL	Desliga os modos FLASH e INVERSE de vídeo		+		
ON ERROR GOTO linha	Em caso de erro, pula para a linha indicada	+		+	
ON expressão GOSUB linhas	Passa a execução para a linha correspondente a INT(expr)	+	+	+	
ON expressão GOTO linhas	Passa a execução para a linha correspondente a INT(expr)	+	+	+	
ONNER GOTO linha	Em caso de erro, pula para a linha indicada		+		
OPEN "nomearquivo"	Abre um arquivo de texto		+		

```
10 REM "PRGPRN"
20 REM "INICIL"
30 GOSUB 1000
40 REM "SAUDAR"
50 GOSUB 3000
60 REM "ESCLHA"
70 GOSUB 3500
80 REM "EXECUT"
90 GOSUB 4000
100 IF ESCL <> 9 THEN 60
110 END
1000 REM SUB-ROTINA "INICIL"
1010 GOSUB 1100: REM SUB-ROTINA "CRIMAT" (CRIA MATRIZES)
1020 GOSUB 1400: REM SUB-ROTINA "LERARQ" (LE ARQUIVOS)
1030 GOSUB 1600: REM SUB-ROTINA "ESTFLG" (ESTABELECE FLAGS)
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM SUB-ROTINA "CRIMAT" (CRIA MATRIZES)
1110 DIM CAMPNOM$(50)
1120 DIM CAMPMOD$(50)
1130 DIM CAMPRUA$(50)
1140 DIM CAMPCID$(50)
1150 DIM CAMPEST$(50)
1160 DIM CAMPTEL$(50)
1170 DIM CAMPIND$(50)
1180 REM
1190 REM
1200 REM
1210 LET TAMA=0
1220 LET RMOD=0
1230 LET GRAV=0
1240 LET CORR=0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN
1400 REM SUB-ROTINA "LERARQ"
1410 OPEN "I", #1, "AGEN.DAD"
1420 INPUT #1, TEST$
1430 IF TEST$="@ PRIMEIRO" THEN GOTO 1540: REM CLOSE E
     RETURN
1440 LET CAMPNOM$(1)=TEST$
```

```
1450 INPUT #1,
     CAMPMOD$(1) ,CAMPRUA$(1) ,CAMPCID$(1) ,CAMPEST$(1) ,
     CAMPTEL$(1)
1460 INPUT #1, CAMPIND$(1)
1470 LET TAMA=2
1480 FOR L=2 TO 50
1490 INPUT #1,
     CAMPNOM$(L) ,CAMPMOD$(L) ,CAMPRUA$(L) ,CAMPCID$(L) ,
     CAMPEST$(L)
1500 INPUT #1, CAMPTEL$(L) ,CAMPIND$(L)
1510 LET TAMA=TAMA+1
1520 IF EOF(1)=-1 THEN LET L=50
1530 NEXT L
1540 CLOSE #1
1550 RETURN
1600 REM SUB-ROTINA *ESTFLG*
1610 REM ESTABELECE FLAGS APOS *LERARQ*
1620 REM
1630 REM
1640 IF TEST$="@ PRIMEIRO" THEN LET TAMA=1
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
3000 REM SUB-ROTINA *SAUDAR*
3010 PRINT CHR$(12): REM LIMPA TELA
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT
3060 PRINT TAB(14); "*BENVINDO AO*"
3070 PRINT TAB(4); "*MICROCOMPUTADOR -- CURSO BASICO*"
3080 PRINT TAB(1); "*AGENDA DE ENDERECOS COMPUTADORIZADA*"
3090 PRINT
3100 PRINT "(PRESSIONAR A BARRA DE ESPACO PARA CONTINUAR)"
3110 FOR L=1 TO 1
3120 IF INKEY$ <> " " THEN L=0
3130 NEXT L
3140 PRINT CHR$(12)
3150 RETURN
3500 REM SUB-ROTINA *ESCLHA*
3510 REM
3520 IF TEST$="@ PRIMEIRO" THEN GOSUB 3860: REM SUB-ROTINA
     *PRIMEI*
3530 IF TEST$="@ PRIMEIRO" THEN RETURN
3540 REM *MENESC*
3550 PRINT CHR$(12)
3560 PRINT "ESCOLHER UM DOS SEGUINTES:"
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. ENCONTRAR UM REGISTRO (PELO NOME)"
3610 PRINT "2. ENCONTRAR NOMES (POR TRECHO DE UM NOME)"
3620 PRINT "3. ENCONTRAR REGISTROS (DE UMA CIDADE)"
3630 PRINT "4. ENCONTRAR REGISTROS (DE UMA INICIAL)"
3640 PRINT "5. LISTAR TODOS OS REGISTROS"
3650 PRINT "6. ACRESCENTAR UM REGISTRO"
3660 PRINT "7. MODIFICAR UM REGISTRO"
3670 PRINT "8. ELIMINAR UM REGISTRO"
3680 PRINT "9. SAIR E GRAVAR"
3690 PRINT
3700 PRINT
3710 REM *ATRESC*
3720 REM
3730 LET L=0
3740 LET I=0
3750 FOR L=1 TO 1
3760 PRINT "ESCOLHER DE 1 A 9"
3770 FOR I=1 TO 1
3780 LET A$=INKEY$
3790 IF A$=" " THEN I=0
3800 NEXT I
3810 LET ESCL=VAL(A$)
3820 IF ESCL <1 THEN L=0
3830 IF ESCL >9 THEN L=0
3840 NEXT L
3850 RETURN
3860 REM SUB-ROTINA *PRIMEI* (DA MENSAGEM)
3870 LET ESCL=6
3880 PRINT CHR$(12): REM LIMPA TELA
3890 PRINT
3900 PRINT TAB(8);"NAO EXISTEM REGISTROS NO"
3910 PRINT TAB(8);"ARQUIVO. VOCE DEVERA COMECAR"
3920 PRINT TAB(8);"ACRESCENTANDO UM REGISTRO"
3930 PRINT
3940 PRINT "(PRESSIONAR A BARRA DE ESPACO PARA CONTINUAR)"
3950 FOR B=1 TO 1
3960 IF INKEY$ <> " " THEN B=0
3970 NEXT B
3980 PRINT CHR$(12): REM LIMPA TELA
3990 RETURN
4000 REM SUB-ROTINA *EXECUT*
4010 REM
4020 IF ESCL=6 THEN GOSUB 10000
4030 REM
4040 REM 1 = *ENCREG*
4050 REM 2 = *ENCNOM*
4060 REM 3 = *ENCCID*
4070 REM 4 = *ENCINI*
4080 REM 5 = *LISREG*
4090 IF ESCL=6 THEN GOSUB 10000
4100 REM 7 = *MODREG*
4110 REM 8 =6 *ELMREG*
4120 IF ESCL=9 THEN GOSUB 11000
4130 REM
4140 RETURN
10000 REM SUB-ROTINA *ACRREG*
10010 PRINT CHR$(12): REM LIMPA TELA
10020 INPUT "FORNECER O NOME";CAMPNOM$(TAMA)
10030 INPUT "FORNECER A RUA";CAMPRUA$(TAMA)
10040 INPUT "FORNECER A CIDADE";CAMPCID$(TAMA)
10050 INPUT "FORNECER O ESTADO";CAMPEST$(TAMA)
10060 INPUT "FORNECER O NUMERO TELEFONICO";
      CAMPTEL$(TAMA)
10070 LET RMOD=1: REM FLAG DE REGISTRO MODIFICADO
10080 LET CAMPIND$(TAMA)=STR$(TAMA)
10090 LET TEST$=" "
10100 GOSUB 10200: REM *MODNOM*
10110 LET ESCL=0
10120 LET TAMA=TAMA+1
10130 REM
10140 REM
10150 RETURN
10200 REM ROTINA *MODNOM*
10210 REM CONVERTE O CONTEUDO DE CAMPNOM$ EM MAIUSCULAS,
10220 REM REMOVE SINAIS E ARQUIVA NA ORDEM:
10230 REM SOBRENOME+ESPACO+PRIMEIRONOME EM CAMPMOD$
10240 REM
10250 LET N$=CAMPNOM$(TAMA)
10260 FOR L=1 TO LEN(N$)
10270 LET TEMP$=MID$(N$,L,1)
10280 LET T=ASC(TEMP$)
10290 IF T >|= 97 THEN T=T-32
10300 LET TEMP$=CHR$(T)
10310 LET P$=P$+TEMP$
10320 NEXT L
10330 LET N$=P$
10340 REM ENCONTRAR ULTIMO ESPACO
10350 FOR L=1 TO LEN(N$)
10360 IF MID$(N$,L,1)=" " THEN S=L
10370 NEXT L
10380 REM ELIMINAR SINAIS E ARQUIVAR PRIMEIRONOME
10390 REM EM CNOM$
10400 FOR L=1 TO S-1
10410 IF ASC(MID$(N$,L,1)) > 64 THEN
      CNOM$=CNOM$+MID$(N$,L,1)
10420 NEXT L
10430 REM ELIMINA SINAIS E ARQUIVA SOBRENOME
10440 REM EM SNOM$
10450 FOR L=S+1 TO LEN(N$)
10460 IF ASC(MID$(N$,L,1)) > 64 THEN
      SNOM$=SNOM$+MID$(N$,L,1)
10470 NEXT L
10480 LET CAMPMOD$(TAMA)=SNOM$+" "+CNOM$
10490 LET P$=" "; LET N$=" "; LET SNOM$=" "; LET CNOM$=" ";
      LET S=0
10500 RETURN
11000 REM SUB-ROTINA *SAIPRG*
11010 REM CLASSIFICA E GRAVA O ARQUIVO
11020 REM SE ALGUM REGISTRO FOI
11030 REM MODIFICADO(RMOD=1)
11040 REM
11050 IF RMOD=0 THEN RETURN
11060 REM
11070 GOSUB 11200: REM *CLAREG*
11080 REM
11090 GOSUB 12000: REM *GRAREG*
11100 RETURN
11200 REM SUB-ROTINA *CLAREG*
11210 REM CLASSIFICA TODOS OS REGISTROS POR ORDEM
11220 REM ALFABETICA SEGUNDO CAMPMOD$ E ATUALIZA
11230 REM CAMPIND$
11240 REM
11250 LET S=0
11260 FOR L=1 TO TAMA-2
11270 IF CAMPMOD$(L) > CAMPMOD$(L+1) THEN GOSUB 11350
11280 NEXT L
11290 IF S=1 THEN 11250
11300 REM
11310 REM
11320 LET RMOD=0: REM ZERA A FLAG DE REGISTRO MODIFICADO
11330 REM
11340 RETURN
11350 REM SUB-ROTINA *INVREG*
11360 LET TCAMNOM$=CAMPNOM$(L)
11370 LET TCAMMOD$=CAMPMOD$(L)
11380 LET TCAMRUA$=CAMPRUA$(L)
11390 LET TCAMCID$=CAMPCID$(L)
11400 LET TCAMEST$=CAMPEST$(L)
11410 LET TCAMTEL$=CAMPTEL$(L)
11420 REM
11430 LET CAMPNOM$(L)=CAMPNOM$(L+1)
11440 LET CAMPMOD$(L)=CAMPMOD$(L+1)
11450 LET CAMPRUA$(L)=CAMPRUA$(L+1)
11460 LET CAMPCID$(L)=CAMPCID$(L+1)
11470 LET CAMPEST$(L)=CAMPEST$(L+1)
11480 LET CAMPTEL$(L)=CAMPTEL$(L+1)
11490 LET CAMPIND$(L)=STR$(L)
11500 REM
11510 LET CAMPNOM$(L+1)=TCAMNOM$
11520 LET CAMPMOD$(L+1)=TCAMMOD$
11530 LET CAMPRUA$(L+1)=TCAMRUA$
11540 LET CAMPCID$(L+1)=TCAMCID$
11550 LET CAMPEST$(L+1)=TCAMEST$
11560 LET CAMPTEL$(L+1)=TCAMTEL$
11570 LET CAMPIND$(L+1)=STR$(L+1)
11580 LET S=1
11590 REM
11600 RETURN
12000 REM SUB-ROTINA *GRAREG*
12010 REM
12020 REM
12030 OPEN "O", #1, "AGEN.DAD"
12040 REM
12050 FOR L=1 TO TAMA-1
12060 PRINT #1,CAMPNOM$(L);",";
      CAMPMOD$(L);",";CAMPRUA$(L);","CAMPCID$(L)
12070 PRINT #1,CAMPEST$(L);",";CAMPTEL$(L);","CAMPIND$(L)
12080 NEXT L
12090 REM
12100 REM
12110 REM
12120 REM
12130 CLOSE #1
12140 REM
12150 RETURN
12500 REM SUB-ROTINA *TAMARQ*
12510 IF CAMPNOM$(L)="  " THEN LET L=50
12520 IF CAMPNOM$(L)="  " THEN RETURN
12530 LET TAMA=TAMA+1
12540 REM
12550 REM
12560 RETURN
```

Mandado de busca

O tempo necessário para se localizar um registro pode ser muito reduzido usando-se "busca binária", contanto que o arquivo já tenha sido adequadamente ordenado.

As três atividades mais importantes do programa da agenda de endereços — acréscimo de novos registros, gravações do arquivo em fita ou em disco e leitura do arquivo a partir do armazenamento em massa, ao se processar o programa pela primeira vez — já foram desenvolvidas. Mas a agenda de endereços será inútil se você só puder acrescentar (e não extrair) dados. Uma rotina que encontre os registros faz-se necessária.

Localizar o registro completo de um nome será com certeza a atividade mais freqüente; portanto, a primeira opção do menu de escolha (*ESCLHA*) é ENCONTRAR UM REGISTRO (PELO NOME). A busca constitui atividade importantíssima em muitos programas de computador, sobretudo nos de bancos de dados, em que com freqüência se necessita da recuperação de itens específicos de um arquivo. De modo geral, há dois métodos de busca: o linear e o binário. A busca linear examina os elementos numa matriz, desde seu início, até a localização do item desejado. Se os itens de dados na matriz não estiverem ordenados, a busca linear será o único método eficaz. O tempo para localização do item, usando-se a busca linear numa matriz de N itens, tem valor médio proporcional de N/2. Se houver poucos itens a pesquisar, considera-se N/2 adequado, mas, à medida que o número de itens aumentar, o tempo consumido pela execução da busca pode tornar-se excessivo.

Contudo, se soubermos que os dados no arquivo estão em determinada ordem, há um método de busca muito mais eficiente, conhecido como "busca binária".

Suponha que alguém queira encontrar no dicionário a definição da palavra "levulose". Não sabendo que o dicionário está em ordem alfabética, começará pela primeira página, para ver se a palavra se encontra ali. Não a encontrando, passa à segunda, e assim por todo o dicionário até localizá-la! No entanto, em vez disso, você (que não ignora que os termos se apresentam ordenadamente) coloca o polegar numa página próxima ao meio do volume, abre-o e verifica as palavras que ali se encontram. Se a página aberta começar pela palavra "metatarso", você constata que avançou muito; assim, a segunda metade do volume torna-se irrelevante e a palavra que se busca estará na primeira metade do livro. Então você repete o procedimento, considerando a página que começa com "metatarso" como se fosse o final do dicionário. A seguir, abre a primeira metade do dicionário na página em que está, digamos, "dodecaedro". Desta vez, você percebe que a página aberta está muito "aquém" do "l" que procura, pois essa letra se situa "além" de "d". As páginas que começam por "dodecaedro" e "metatarso" são agora consideradas a "primeira" e a "última" do volume. De novo você coloca o polegar na parte relevante e abre, por exemplo, na palavra "jasmim". Está ainda "aquém" e, desse modo, a palavra que você busca deve estar entre esta página e a de "metatarso". A repetição desse procedimento é eficaz na localização da palavra procurada — desde que ela esteja registrada no dicionário.

Nesse exemplo, a palavra "levulose" é a "chave de busca", o item que estamos tentando encontrar. Toda vez que examinamos um registro, comparamos a chave de busca com a "chave de registro" para localizar o "alvo" ou "objetivo". Esperamos encontrar, junto com a chave de registro, o que é chamado de "informação adicional". A informação adicional da chave de registro "levulose" é a definição no dicionário — neste caso, "açúcar levogiro encontrado no mel e em alguns frutos".

A analogia da busca num arquivo de banco de dados com um arquivo alvo é apropriada, desde que os registros estejam previamente ordenados como os verbetes de um dicionário. Imagine como seria difícil consultar o dicionário se os verbetes se apresentassem na ordem em que o autor os fosse coligindo ou segundo a importância que ele lhes atribuísse.

A rotina de busca necessária para nossa agenda de endereços precisa ser mais complexa do que se pensa, por razões que ficarão evidentes. A primeira operação da rotina de busca — vamos chamá-la *BUSREG*, por exemplo — é solicitar o nome a ser encontrado. Esse procedimento denomina-se "chave de busca". Suponha que haja, em algum ponto do arquivo, o registro de alguém chamado Pedro Jonas. O registro dessa pessoa tem um campo (com o nome da forma padronizada) que contém JONAS PEDRO.

A rotina de busca pode advertir-nos com uma mensagem como QUE NOME VOCÊ ESTÁ PROCURANDO?, e nós respondemos PEDRO JONAS, ou talvez P. Jonas, ou ainda Pedrinho Jonas. Antes de examinar formas mais complexas, vamos admitir que respondemos com o nome completo Pedro Jonas. A primeira coisa a ser feita pela rotina de busca será a conversão da resposta na forma padronizada: JONAS, PEDRO. A seguir, ela compara o item que fornecemos (a chave de busca) com os vários conteúdos dos campos CAMPMOD$. Se o programa utiliza uma busca linear, ele compara a chave de busca ao campo CAMPMOD$ em seqüência, até que encontre uma correspondência ou constate não existir equivalência exata.

Porém, como já observamos, no caso de os dados estarem ordenados, a busca linear não se mostra eficiente, em comparação com a binária. A rotina de busca pode garantir a ordenação dos registros a partir de uma instrução IF RMOD = 1 THEN GOSUB *CLAREG*. O programa tem a informação de que o menor

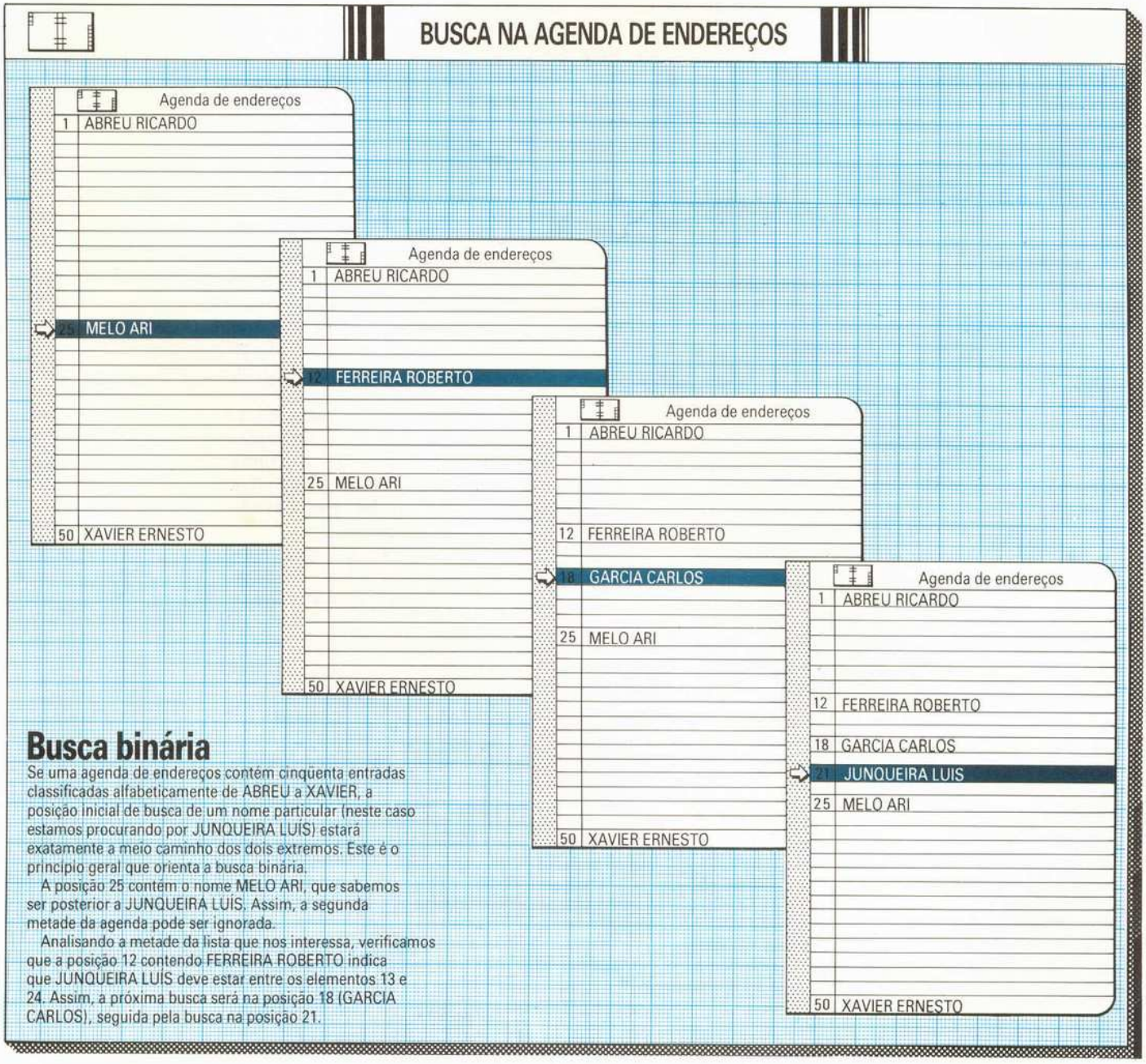
elemento na tabela em que será realizada a busca é CAMPMOD$(1) e que o maior corresponde a CAMPMOD$(TAMA − 1). Para organizar a busca, necessitamos de três variáveis: FIM, para o final da matriz (CAMPMOD$(1), no início); INI, para o início da matriz (CAMPMOD$(TAMA − 1), no início); e MEI, para o valor correspondente ao elemento do meio.

Usando a analogia do dicionário, podemos supor que FIM = MATRIZ(1) e INI = MATRIZ(TAMA − 1). Ou seja, a matriz que temos de levar em conta com relação à busca se inicia com o "menor" elemento e se encerra com o "maior". Podemos então executar os procedimentos LET FIM = 1 e LET INI = TAMA − 1. Lembre-se de que TAMA é sempre uma unidade maior que o total de registros presentes na agenda.

Caso haja 21 entradas válidas na agenda de endereços, os valores serão 22 (para TAMA), 1 (para FIM) e 21 (para INI). O valor de MEI — ou seja, a posição do elemento do meio — pode derivar em BASIC de INT ((FIM + INI)/2). Se o valor de FIM é 1, e o valor de INI, 21, o valor de MEI corresponde a 11.

Para organizar uma busca binária, precisamos de início admitir a validade de todo o arquivo e encontrar o ponto médio INT((FIM + INI)/2) no interior de um loop que se encerra — quer o objeto seja encontrado quer não haja correspondência possível. A seguir, verificamos se a chave de busca (CHAVE$) equivale ao valor MEI da matriz. Caso esse valor seja muito pequeno, inferimos que MATRIZ(MEI) corresponde à parte mais baixa da matriz a ser considerada; desse modo, podemos atribuir MEI a FIM. No entanto, revela-se mais eficiente atribuir MEI + 1 a FIM, uma vez que MATRIZ(MEI) não corresponde à chave de busca. De modo semelhante, se tivermos o procedimento IF MATRIZ(MEI) > CHAVE$, pode-se atribuir MEI − 1 a INI.

Como estágio intermediário ao desenvolvimento de rotinas completamente eficientes, o programa apresentado pode receber uma entrada fictícia (que deve estar no mesmo formato que os campos



Busca binária

Se uma agenda de endereços contém cinquenta entradas classificadas alfabeticamente de ABREU a XAVIER, a posição inicial de busca de um nome particular (neste caso estamos procurando por JUNQUEIRA LUIS) estará exatamente a meio caminho dos dois extremos. Este é o princípio geral que orienta a busca binária.

A posição 25 contém o nome MELO ARI, que sabemos ser posterior a JUNQUEIRA LUIS. Assim, a segunda metade da agenda pode ser ignorada.

Analisando a metade da lista que nos interessa, verificamos que a posição 12 contendo FERREIRA ROBERTO indica que JUNQUEIRA LUIS deve estar entre os elementos 13 e 24. Assim, a próxima busca será na posição 18 (GARCIA CARLOS), seguida pela busca na posição 21.

CAMPMOD$). Ele vai imprimir REGISTRO NÃO ENCONTRADO, se não houver correspondência, ou O REGISTRO É NO (MEI), se encontrar alguma correspondência. Como se inicia com o número de linha 13000, a rotina pode ser incluída no final do programa apresentado na página 399, e funcionará se a linha 4040 for alterada para IF ESCL = 1 THEN GOTO 13000.

A linha 13240 apresenta a instrução STOP, que encerra o programa por algum tempo, tão logo seja apresentada a mensagem REGISTRO NÃO ENCONTRADO ou O REGISTRO É NO (MEI). O programa se reinicia no mesmo número de linha, sem perda de dados, pela digitação de CONT. Sem a instrução STOP, o programa segue de imediato para a instrução RETURN na linha 13250, e a mensagem apresentada desaparece muito depressa, o que impossibilita sua leitura.

Consideremos em pormenores esse fragmento de programa. Na linha 13100, FIM recebe o valor 1, que corresponde à posição do menor elemento na matriz CAMPMOD$. INI recebe o valor TAMA − 1, na linha 13110. Essa é a posição na matriz CAMPMOD$ em que se localiza o elemento mais elevado. A linha 13120 inicializa um loop que se encerra quando é encontrada uma correspondência ou se verifica que não existe correspondência possível.

A linha 13130 determina o ponto médio, dividindo a soma dos índices superior e inferior da matriz (INT é utilizada para arredondar a divisão, de modo que MEI jamais possa assumir valores como 1,5). Existe a possibilidade de os conteúdos de CAMPMOD$(MEI) serem os mesmos que a chave de busca (CHAVE$). Mas se não forem, como é provável, L receberá o valor 0, garantindo que o loop seja repetido. Se o teste na linha 13140 falhar, CAMPMOD$(MEI) terá valor maior ou menor que CHAVE$. O valor de FIM será então uma unidade acima do valor anterior de MEI (linha 13150), ou o valor de INI corresponderá a uma unidade a menos que o valor anterior de MEI. O valor de MEI sozinho não é usado porque a resposta negativa do teste na linha 13140 já demonstrou que CAMPMOD$(MEI) não é o elemento que estamos procurando. Logo, não teria sentido examinar esse ponto da matriz na próxima passagem pelo loop.

Se não foi encontrada correspondência, o valor de FIM terminará por superar o valor de INI. O loop pode ser encerrado (linha 13170) e a mensagem REGISTRO NÃO ENCONTRADO será impressa (linha 13200).

Apresentamos esse fragmento de programa para demonstração e para possibilitar o teste da rotina de busca. Do modo como está, seu emprego se mostra bem limitado. Sem a instrução STOP, na linha 13240, não haveria tempo sequer para a leitura da mensagem na tela. Convém apresentar o registro completo, como foi digitado da primeira vez. Tão logo se conheça o número de registro, um procedimento simples recupera quaisquer dados adicionais necessários — CAMPMOD$, CAMPRUA$ etc. Abaixo da apresentação do registro convém que constem uma mensagem como PRESSIONAR A BARRA DE ESPAÇO PARA CONTINUAR (retornando ao menu principal) e opções complementares como PRESSIONAR "P" PARA IMPRIMIR.

Não é tão fácil decidir como lidar com a entrada de *ENCREG*. No fragmento do programa, a entrada esperada (na linha 13020) deve estar em forma padronizada — JONAS PEDRO, por exemplo. É claro que isso não é o mais adequado. Não costumamos pensar primeiro no sobrenome e depois no prenome da pessoa, e não constitui exigência razoável o fornecimento dos nomes em letras maiúsculas. Além do mais, a menor diferença em relação ao nome inicialmente fornecido resultará em mensagem de REGISTRO NÃO ENCONTRADO. *MODNOM* não basta para resolver os dois primeiros problemas. O terceiro — como lidar com uma correspondência aproximada — mostra-se bem mais interessante, mas de solução difícil.

Antes de considerar este problema, vejamos por que motivo *MODNOM* não pode resolver os dois primeiros. Se você voltar e examinar *MODNOM*, que começa na linha 10200, encontrará um bom exemplo de uma das armadilhas mais comuns em que os programadores caem — falta de generalidade. Essa sub-rotina deveria lidar com conversões de nomes "normais" em formas padronizadas, sempre que essa operação se fizesse necessária. Embora desenvolvida como sub-rotina isolada, ela foi escrita tendo em vista *ACRREG*. Isso faz supor que o nome a ser convertido estará sempre em CAMPNOM$(TAMA), e que, após a conversão, o campo modificado será armazenado em CAMPMOD$(TAMA). Diante desse fato, o programador tem três escolhas: refaz *MODNOM* para torná-la mais abrangente, o que por sua vez exige alterações adicionais em outras partes do programa; desenvolve uma rotina quase idêntica só para lidar com a entrada para *ENCREG* (o que significa esforço desperdiçado e exige mais espaço na memória); ou recorre a alguma técnica de programação que permita o uso da rotina *MODNOM* não modificada. Esta última alternativa é, sob certos aspectos, a menos atraente. Resolve o problema, mas a própria parte operante do programa que foi modificado fica obscura, mesmo para o programador, e se converte num pesadelo para quem tentar usá-lo.

Disso tudo se conclui que o ideal é fazer as sub-rotinas tão abrangentes quanto possível, de modo que possam ser chamadas por qualquer parte do programa.

Para exemplificar a má técnica de programação (ou programação "suja") e mostrar como ela pode tornar o programa obscuro, observe a linha 13020 do fragmento de programa INPUT "FORNECER A CHAVE";CHAVE$, e a seguir examine a modificação ou "dificuldade" que permitiria o emprego de *MODNOM*:

```
13020 INPUT "FORNECER A CHAVE";
      CAMPNOM$(TAMA)
13030 GOSUB 10200: REM SUB-ROTINA
      *MODNOM*
13040 LET CHAVE$=CAMPMOD$(TAMA)
13050...
```

O valor de TAMA tem sempre uma unidade a mais que o maior registro válido. Ou seja, não há registro na posição TAMA nas matrizes e assim esse problema não altera qualquer registro existente. Mas, sem algumas instruções REM abrangentes que expliquem

os procedimentos que estão se realizando, imagine a confusão que essas três linhas criariam para quem não estivesse vinculado ao desenvolvimento do programa.

Voltemos ao problema mais interessante de lidar com "quase erros". Suponha que demos entrada ao nome de alguém como Pedrinho Jonas na operação *ACRREG*, mas como Pedro Jonas em *ENCREG*. Ambas as formas seriam convertidas às formas padronizadas JONAS PEDRINHO e JONAS PEDRO. Portanto, não seria encontrada correspondência durante a busca. Não tentaremos resolver por completo esse problema porque a solução adequada constitui uma tarefa maior de programação. No entanto, para o leitor interessado em experimentar, eis algumas indicações:

```
INICIAR {busca de matriz para correspondência
exata}
  IF correspondência exata for encontrada
    THEN PRINT registro completo
    ELSE buscar matriz para correspondência
    aproximada
      IF correspondência aproximada for encontrada
        THEN PRINT registro da correspondência
        aproximada
        ELSE PRINT "NENHUM REGISTRO
        ENCONTRADO"
      ENDIF
  ENDIF
END
```

O procedimento para correspondência aproximada poderia ser semelhante a:

```
INICIAR {correspondência aproximada}
Buscar matriz para correspondência exata do
sobrenome
  IF correspondência exata do sobrenome
    THEN buscar prenomes para correspondência
    máxima
    PRINT registro de correspondência máxima
    ELSE buscar sobrenomes para correspondência
    máxima
      IF correspondência máxima de sobrenome for
      encontrada
        THEN PRINT registro de correspondência
        máxima
      ENDIF
  ENDIF
END
```

O procedimento de correspondência máxima pode ser definido de modo aproximativo como o encontro da variável alfanumérica objeto com o número máximo de caracteres em comum com a variável alfanumérica chave. Admita uma situação na qual a "variável chave" está inteiramente contida pela "variável objeto", ou vice-versa. Não há solução simples, apenas campo suficiente para programação criativa.

Existe um efeito paralelo no fragmento de programa apresentado. Admitamos que a seguinte seqüência de eventos venha a ocorrer. Há dez registros no arquivo de dados. Você processa o programa e a seguir utiliza *ACRREG* para incluir novo registro, seguido por *ENCREG*, para localização de registros. Quando *SAIPRG* é processado, para gravar o arquivo e encerrar o programa, o registro que você acrescentou não será gravado (embora todos os outros registros sejam). Essa é uma conseqüência direta de algo que aconteceu na execução de *ENCREG*. Você percebe por que o registro incluído não será gravado?

O próximo fascículo deste curso explica como evitar essa perda de dados; mostra com que finalidade se emprega a variável CORR e descreve como eliminar ou modificar um registro. Outras opções no menu principal (*ENCCID* etc.) são muito semelhantes às rotinas já examinadas.

Por fim, observe o que aconteceria se houvesse exatamente cinqüenta registros no arquivo de dados e fosse modificada a rotina *ENCREG*, que chama *MODNOM*. (Uma dica: TAMA valerá 51.)

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
OUT porta, valor	Endereça um valor à porta especificada			+	
PAUSE n	Pára a execução e apresenta imagem de n quadros				+
PEEK (n)	Fornece o byte correspondente à posição n da memória	+	+	+	+
PI	Fornece o valor 3,14159265...				+
PLOT x,y	Mostra o ponto (x,y) em gráfico de baixa resolução		+		+

```
13000 REM VERSAO DE *ENCREG* PARA TESTE
13010 IF RMOD=1 THEN GOSUB 11200
13020 INPUT "ENTRE COM A CHAVE";CHAVE$
13030 REM
13040 REM
13050 REM
13060 REM
13070 REM
13080 REM
13090 REM
13100 LET FIM=1
13110 LET INI=TAMA-1
13120 FOR L=1 TO 1
13130 LET MEI=INT((FIM+INI)/2)
13140 IF CAMPMOD$(MEI) <> CHAVE$ THEN L=0
13150 IF CAMPMOD$(MEI) < CHAVE$ THEN FIM=MEI+1
13160 IF CAMPMOD$(MEI) > CHAVE$ THEN INI=MEI-1
13170 IF FIM > INI THEN L=1
13180 NEXT L
13190 REM
13200 IF FIM > INI THEN PRINT "REGISTRO NAO ENCONTRADO"
13210 IF FIN <= INI THEN PRINT "O REGISTRO E O NO. ";MEI
13220 REM
13230 REM
13240 STOP
13250 RETURN
```

Recursos extras

A eliminação das anomalias ocasionadas pela unificação de módulos e o acréscimo de alguns recursos continuam nossa agenda de endereços.

No fascículo anterior deste curso, os leitores ficaram com a tarefa de imaginar o motivo pelo qual o processamento de um programa de agenda de endereços, com o acréscimo de um registro (empregando *ACRREG*), a localização de outro (*ENCREG*) e ainda o desvio do programa (*SAIPRG*), resultaria na não gravação do registro incluído. O problema surgiu a partir do uso da variável RMOD como flag indicativa da modificação de um registro (significando que o arquivo poderia estar fora de ordem). A sub-rotina *CLAREG* colocaria o arquivo em ordem alfabética e a seguir atribuiria o valor 0 a RMOD, supondo-se o arquivo ordenado. A execução de *SAIPRG* verificava se o arquivo estava em ordem (RMOD=0), mas não previa sua gravação, em caso positivo.

A inclusão de novo registro (empregando *ACRREG*) atribuiria o valor 1 a RMOD (se o arquivo tivesse sido modificado, isto é, um novo registro tivesse sido acrescentado). Porém, *CLAREG* atribuiria o valor 0 a RMOD, indicando que o arquivo estava ordenado. No entanto, tornam-se necessárias — não importando se o arquivo tenha sido ordenado ou não — uma flag que sinalize quando um registro for modificado e uma flag individual para mostrar se o arquivo está ordenado ou não. Assim, sub-rotinas que requeiram informe sobre a classificação do arquivo podem consegui-lo por meio da flag "classificada", ao passo que as sub-rotinas que necessitam de informações sobre modificação de registros vão obtê-las por meio da flag "modificada".

RMOD, para indicar se um registro foi modificado, e CLAS, para indicar se o arquivo foi classificado, são nomes apropriados para as duas flags.

Quando apresentamos o programa, na página 399, a linha 1230 continha a instrução LET GRAVO=0. A variável GRAVO não fora utilizada, mas, com a inclusão da linha, percebemos que RMOD apenas não seria suficiente.

Um nome mais adequado para essa flag seria CLAS (para indicar que o arquivo está classificado). A linha 1230 original foi alterada para:

```
1230 LET CLAS = 1
```

Há agora quatro estados possíveis com relação às condições do arquivo de dados, que são:

RMOD	CLAS	
0	0	Não modificado, não classificado (inconsistente)
1	0	Modificado, não classificado
0	1	Não modificado, classificado
1	1	Modificado, classificado

RMOD=0 e CLAS=0 são procedimentos inconsistentes porque o programa garante que o arquivo de dados sempre esteja classificado, antes da gravação. Quando se processa o programa, RMOD recebe o valor 0 (linha 1220), indicando que não ocorreram modificações, e CLAS recebe o valor 1 (linha 1230), significando que o arquivo foi classificado.

Alguma operação que altere um registro (como *ACRREG*, *ELMREG* ou *MODREG*) atribui a RMOD o valor 1 e essa flag não será recolocada em 0 por alguma operação posterior. CLAS, que recebe de início o valor 1, passa de novo a 0 por meio de um procedimento que pode significar que os dados saíram da ordem (como em *MODREG*, se o campo do nome for alterado). Qualquer rotina que parte da suposição de que os dados já estão classificados (como *ENCREG*) sempre verifica CLAS e chama a rotina de classificação, se CLAS=0. Por meio do emprego dessas duas flags, em vez de apenas RMOD, o programa pode se encerrar sem gravar o arquivo de dados, caso não tenham ocorrido alterações de ordem no processamento atual. Nada o induz a isso, se ocorrer classificação após a alteração de um registro.

Outra variável até agora não utilizada é CORR. Ela serve para reter a posição atual do registro "corretamente em uso" na matriz, após a rotina de busca ter localizado essa posição. Depois de receber a atribuição de um valor, CORR transporta dados relativos ao registro objeto para outras rotinas do programa. O final da rotina *ENCREG* (busca) sofreu modificação nas linhas 13320 e 13330 para passar o valor de CORR a 0, se a busca não encontrar o registro objeto; e a MEI, se a busca tiver bom resultado.

A linha 13340 desvia para a sub-rotina *NAOREG* se CORR corresponder a 0. Esse procedimento exibe mensagem indicando que o registro não foi encontrado e apresenta a chave de busca, CAMPNOM$(TAMA). *NAOREG* retorna ao menu principal após o pressionamento da barra de espaço. Pode ser modificada com facilidade, dando ao usuário as duas oportunidades seguintes:

```
PRESSIONE RETURN PARA NOVA TENTATIVA OU A
BARRA DE ESPAÇO PARA CONTINUAR
```

O modo mais simples para conseguir isso parece ser uma nova chamada de *ENCREG*, se a tecla RETURN for pressionada. No entanto, a chamada de uma rotina de dentro de si mesma, embora válida em BASIC, causará "confusão" no endereço de retorno e fará com que a sub-rotina se repita, mesmo quando desnecessária. Há maneiras de contornar o problema, mas elas complicam a programação.

O método mais fácil seria usar uma flag (como NREG, para não registro) e atribuir-lhe o valor 0 em *NAOREG*, possibilitando que a sub-rotina retornasse pelo procedimento normal e forçasse uma volta para *EXECUT* no programa principal. Por exemplo: 95 IF NREG = 0 THEN 80. Com essa aborda-

gem, porém, a codificação começa a ficar desordenada. De acordo com nossa regra de evitar GOTOs, decidimos simplificar e apenas retornar ao menu principal se *ENCREG* não encontrar registro.

Houve um pequeno acréscimo em *MODNOM*, na linha 10490. A variável numérica S também deve receber valor 0 (LET S=0). A não execução desse procedimento, em algumas circunstâncias inabituais, ocasiona o funcionamento ineficaz de *MODNOM*.

MODREG corresponde a outra rotina implementada nesta versão final do programa. Ela primeiro localiza o registro a ser modificado, chamando *ENCREG* (linha 14120). Essa linha, por sua vez, chama a 13030, não a 13000, para suprimir a instrução de limpeza da tela de *ENCREG*. Caso não se localize o registro, o programa retornará ao menu principal pelo procedimento habitual (linha 14130). Se localizado, o registro aparece na tela e o usuário recebe a seguinte solicitação:

QUER MODIFICAR O NOME?
TECLE RETURN PARA ENTRAR COM O NOVO NOME
OU A BARRA DE ESPAÇO PARA O PRÓXIMO CAMPO

A rotina que determina a opção necessária pode ser encontrada nas linhas de 14190 a 14280.

As linhas de 14190 a 14220 constituem um loop simples que se encerra apenas se a barra de espaço ou RETURN for pressionada. Se A$ não for CHR$(13) — o valor ASCII para RETURN —, nem um espaço (você pode também usar CHR$(32) em vez de " "), I recebe o valor 0 e o loop se repete. Caso a tecla pressionada seja RETURN (e o campo do nome deva ser alterado), as poucas linhas seguintes vão preencher CAMPNOM$(CORR) com o novo nome; determinar RMOD; atribuir o valor 0 a CLAS; e chamar *MODNOM*, além de preencher CAMPMOD$(CORR) com o nome padronizado criado por *MODNOM* e localizado em CAMPMOD$(TAMA).

O restante de *MODNOM* funciona do mesmo modo. Observe, porém, que a modificação de outros campos determina RMOD sem atribuir o valor 0 a CLAS (ver a linha 14490, por exemplo). Isso porque a alteração apenas do campo do nome significa que o arquivo de dados pode estar fora de ordem, pois é classificado pelo nome. A alteração de qualquer outro campo indica que um registro foi modificado (RMOD=1) e que o arquivo deve ser gravado ao se encerrar o programa.

Outra rotina implementada é *ELMREG*, para eliminar um registro. Trata-se de um procedimento bastante direto. *ELMREG* limpa a tela (linha 15020) e apresenta mensagem informando sobre o que acontece no programa. A seguir, chama *ENCREG* para localizar o registro a ser eliminado. Uma escolha é então oferecida: tecle RETURN para eliminar ou a BARRA DE ESPAÇO para continuar. Também se apresenta uma mensagem de advertência (linha 15160). Abordagem ainda melhor pode ser a mensagem VOCÊ TEM CERTEZA?, se RETURN for pressionada e, em seguida, a eliminação do registro, se a tecla S for pressionada (isto é, IF INKEY$ = "S" THEN...).

ELMREG não atribui o valor 0 à flag CLAS. Uma vez que o arquivo já está em ordem alfabética, a eliminação de todo um registro não altera essa ordem. Isso significa que o arquivo foi modificado e, desse modo, RMOD recebe o valor 0 na linha 15340 e TAMA reduz-se em uma unidade na linha 15350 (pois o arquivo agora tem um registro válido a menos). Todos os registros deslocaram-se para "uma unidade abaixo" nas linhas de 15260 a 15320.

Você deve ter notado que *ENCREG* inclui a chamada condicional de uma sub-rotina denominada *IMPCOR* para impressão do registro atual localizado por *ENCREG*. Se você não tem impressora, apenas substitua a linha 13540 por uma instrução REM (para ampliação futura) e omita as linhas de 13600 a 13690.

Desenvolvemos, na agenda de endereços, todas as opções fundamentais apresentadas no menu principal: o encontro, a inclusão, a alteração e a eliminação de registros, além do desvio do programa. O objetivo dessa agenda de endereços computadorizada é exemplificar o modo como se deve especificar, elaborar e desenvolver programas. Uma modificação fundamental para quem deseja um software de utilização mais flexível consiste em resolver o problema que surgirá se TAMA chegar a ser 51. Isso acontecerá tão logo haja cinqüenta registros no arquivo.

No próximo fascículo, apresentaremos um exemplo de programa das demais funções de nossa agenda, ou seja, encontrar nomes, registros de uma cidade, registros de uma inicial e listar todos os registros.

Programa para agenda de endereços

```
10 REM "PRGPRN"
20 REM *INICIL*
30 GOSUB 1000
40 REM *SAUDAR*
50 GOSUB 3000
60 REM *ESCLHA*
70 GOSUB 3500
80 REM *EXECUT*
90 GOSUB 4000
100 IF ESCL <> 9 THEN 60
110 END
1000 REM SUB-ROTINA *INICIL*
1010 GOSUB 1100: REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1020 GOSUB 1400: REM SUB-ROTINA *LERARQ* (LE ARQUIVOS)
1030 GOSUB 1600: REM SUB-ROTINA *ESTFLG* (ESTABELECE FLAGS)
```

```
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN
1100 REM SUB-ROTINA *CRIMAT* (CRIA MATRIZES)
1110 DIM CAMPNOM$(50)
1120 DIM CAMPMOD$(50)
1130 DIM CAMPRUA$(50)
1140 DIM CAMPCID$(50)
1150 DIM CAMPEST$(50)
1160 DIM CAMPTEL$(50)
1170 DIM CAMPIND$(50)
1180 REM
1190 REM
1200 REM
1210 LET TAMA=0
1220 LET RMOD=0
1230 LET CLAS=1
```

```
1240 LET CORR=0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN
1400 REM SUB-ROTINA *LERARQ*
1410 OPEN "I", #1, "AGEN.DAD"
1420 INPUT #1, TEST$
1430 IF TEST$="@PRIMEIRO" THEN GOTO 1540: REM CLOSE E RETURN
1440 LET CAMPNOM$(1)=TEST$
1450 INPUT #1,CAMPMOD$(1),CAMPRUA$(1),CAMPCID$(1),
     CAMPEST$(1),CAMPTEL$(1)
1460 INPUT #1,CAMPIND$(1)
1470 LET TAMA=2
1480 FOR L=2 TO 50
1490 INPUT #1,CAMPNOM$(L),CAMPRUA$(L),CAMPCID$(L),
     CAMPEST$(L)
1500 INPUT #1,CAMPTEL$(L),CAMPIND$(L)
1510 LET TAMA=TAMA+1
1520 IF EOF(1)=-1 THEN LET L=50
1530 NEXT L
1540 CLOSE #1
1550 RETURN
1600 REM SUB-ROTINA *ESTFLG*
1610 REM ESTABELECE FLAGS APOS *LERARQ*
1620 REM
1630 REM
1640 IF TEST$="@PRIMEIRO" THEN LET TAMA=1
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
3000 REM SUB-ROTINA *SAUDAR*
3010 PRINT CHR$(12): REM LIMPA TELA
3020 PRINT
3030 PRINT
3040 PRINT
3050 PRINT
3060 PRINT TAB(14);"*BENVINDO AO*"
3070 PRINT TAB(4);"*MICROCOMPUTADOR — CURSO BASICO*"
3080 PRINT TAB(1);"*AGENDA DE ENDERECOS
     COMPUTADORIZADA*"
3090 PRINT
3100 PRINT " (PRESSIONAR A BARRA DE ESPACO PARA CONTINUAR) "
3110 FOR L=1 TO 1
3120 IF INKEY$ < > " " THEN L=0
3130 NEXT L
3140 PRINT CHR$(12)
3150 RETURN
3500 REM SUB-ROTINA *ESCLHA*
3510 REM
3520 IF TEST$="@PRIMEIRO" THEN GOSUB 3660: REM SUB-ROTINA
     *PRIMEI*
3530 IF TEST$="@PRIMEIRO" THEN RETURN
3540 REM "MENESC"
3550 PRINT CHR$(12)
3560 PRINT "ESCOLHER UM DOS SEGUINTES:"
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. ENCONTRAR UM REGISTRO (PELO NOME) "
3610 PRINT "2. ENCONTRAR NOMES (POR TRECHO DE UM NOME) "
3620 PRINT "3. ENCONTRAR REGISTROS (DE UMA CIDADE) "
3630 PRINT "4. ENCONTRAR REGISTROS (DE UMA INICIAL) "
3640 PRINT "5. LISTAR TODOS OS REGISTROS"
3650 PRINT "6. ACRESCENTAR UM REGISTRO"
3660 PRINT "7. MODIFICAR UM REGISTRO"
3670 PRINT "8. ELIMINAR UM REGISTRO"
3680 PRINT "9. SAIR E GRAVAR"
3690 PRINT
3700 PRINT
3710 REM "ATRESC"
3720 REM
3730 LET L=0
3740 LET I=0
3750 FOR L=1 TO 1
3760 PRINT "ESCOLHER DE 1 A 9"
3770 FOR I=1 TO 1
3780 LET A$=INKEY$
3790 IF A$=" " THEN I=0
3800 NEXT I
3810 LET ESCL=VAL(A$)
3820 IF ESCL<1 THEN L=0
3830 IF ESCL>9 THEN L=0
3840 NEXT L
3850 RETURN
3860 REM SUB-ROTINA *PRIMEI* (DA MENSAGEM)
3870 LET ESCL=6
3880 PRINT CHR$(12): REM LIMPA TELA
3890 PRINT
3900 PRINT TAB(8);"NAO EXISTEM REGISTROS NO"
3910 PRINT TAB(8);"ARQUIVO. VOCE DEVERA COMECAR"
3920 PRINT TAB(8);"ACRESCENTANDO UM REGISTRO"
3930 PRINT
3940 PRINT " (PRESSIONAR A BARRA DE ESPACO PARA
     CONTINUAR) "
3950 FOR B=1 TO 1
3960 IF INKEY$ <> " " THEN B=0
3970 NEXT B
3980 PRINT CHR$(12): REM LIMPA TELA
3990 RETURN
4000 REM SUB-ROTINA *EXECUT*
4010 REM
4020 REM
4030 REM
4040 IF ESCL=1 THEN GOSUB 13000: REM *ENCREG*
4050 REM 2 = *ENCNOM*
4060 REM 3 = *ENCCID*
4070 REM 4 = *ENCINI*
4080 REM 5 = *LISREG*
4090 IF ESCL=6 THEN GOSUB 10000: REM *ACRREG*
4100 IF ESCL=7 THEN GOSUB 14000: REM *MODREG*
4110 IF ESCL=8 THEN GOSUB 15000: REM *ELMREG*
4120 IF ESCL=9 THEN GOSUB 11000: REM *SAIPRG*
4130 REM
4140 RETURN
10000 REM SUB-ROTINA *ACRREG*
10010 PRINT CHR$(12): REM LIMPA TELA
10020 INPUT "FORNECER O NOME";CAMPNOM$(TAMA)
10030 INPUT "FORNECER A RUA";CAMPRUA$(TAMA)
10040 INPUT "FORNECER A CIDADE";CAMPCID$(TAMA)
10050 INPUT "FORNECER O ESTADO";CAMPEST$(TAMA)
10060 INPUT "FORNECER O NUMERO TELEFONICO";CAMPTEL$(TAMA)
10070 LET RMOD=1: LET CLAS=0: REM MODIFICADO E NAO
      CLASSIFICADO
10080 LET CAMPIND$(TAMA)=STR$(TAMA)
10090 LET TEST$=" "
10100 GOSUB 10200: REM *MODNOM*
10110 LET ESCL=0
10120 LET TAMA=TAMA+1
10130 REM
10140 REM
10150 RETURN
10200 REM ROTINA *MODNOM*
10210 REM CONVERTE O CONTEUDO DE CAMPNOMS EM MAIUSCULAS,
10220 REM REMOVE SINAIS E ARQUIVA NA ORDEM:
10230 REM SOBRENOME + ESPACO + PRIMEIRONOME EM CAMPMOD$
10240 REM
10250 LET NS=CAMPNOM$(TAMA)
10260 FOR L=1 TO LEN(NS)
10270 LET TEMP$=MID$(N$,L,1)
10280 LET T=ASC(TEMP$)
10290 IF T>=97 THEN T=T-32
10300 LET TEMP$=CHR$(T)
10310 LET PS=PS+TEMP$
10320 NEXT L
10330 LET NS=PS
10340 REM ENCONTRAR ULTIMO ESPACO
10350 FOR L=1 TO LEN(NS)
10360 IF MID$(NS,L,1)=" " THEN S=L
10370 NEXT L
10380 REM ELIMINAR SINAIS E ARQUIVAR PRIMEIRONOME
10390 REM EM CNOMS
10400 FOR L=1 TO S-1
10410 IF ASC(MID$(NS,L,1)) > 64 THEN
      CNOMS=CNOMS+MID$(NS,L,1)
10420 NEXT L
10430 REM ELIMINA SINAIS E ARQUIVA SOBRENOME
10440 REM EM SNOMS
10450 FOR L=S+1 TO LEN(NS)
10460 IF ASC(MID$(NS,L,1)) > 64 THEN SNOMS=SNOMS+MID$(NS,L,1)
10470 NEXT L
10480 LET CAMPMOD$(TAMA)=SNOMS+" "+CNOMS
10490 LET PS=" ": LET NS=" ": LET SNOMS=" ": LET CNOMS=" ": LET S=0
10500 RETURN
11000 REM SUB-ROTINA *SAIPRG*
11010 REM CLASSIFICA E GRAVA O ARQUIVO
11020 REM SE ALGUM REGISTRO FOI
11030 REM MODIFICADO (RMOD=1)
11040 REM OU NAO CLASSIFICADO (CLAS=0)
11050 REM RMOD=0 E CLAS=0 NAO E LEGAL
11060 REM
11070 IF RMOD=0 AND CLAS=1 THEN RETURN
11080 IF RMOD=1 AND CLAS=0 THEN GOSUB 11200: REM
      *CLAREG*
11090 GOSUB 12000: REM *GRAREG*
11100 RETURN
11200 REM SUB-ROTINA *CLAREG*
11210 REM CLASSIFICA TODOS OS REGISTROS POR ORDEM
11220 REM ALFABETICA SEGUNDO CAMPMOD$ E ATUALIZA
11230 REM CAMPIND$
11240 REM
11250 LET S=0
11260 FOR L=1 TO TAMA-2
11270 IF CAMPMOD$(L) > CAMPMOD$(L+1) THEN GOSUB 11350
11280 NEXT L
11290 IF S=1 THEN 11250
11300 REM
11310 REM
11320 LET CLAS=1: REM FLAG DE ARQUIVO CLASSIFICADO
11330 REM
11340 RETURN
11350 REM SUB-ROTINA *INVREG*
11360 LET TCAMNOM$=CAMPNOM$(L)
11370 LET TCAMMOD$=CAMPMOD$(L)
11380 LET TCAMRUA$=CAMPRUA$(L)
11390 LET TCAMCID$=CAMPCID$(L)
11400 LET TCAMEST$=CAMPEST$(L)
11410 LET TCAMTEL$=CAMPTEL$(L)
11420 REM
11430 LET CAMPNOM$(L)=CAMPNOM$(L+1)
11440 LET CAMPMOD$(L)=CAMPMOD$(L+1)
11450 LET CAMPRUA$(L)=CAMPRUA$(L+1)
11460 LET CAMPCID$(L)=CAMPCID$(L+1)
11470 LET CAMPEST$(L)=CAMPEST$(L+1)
11480 LET CAMPTEL$(L)=CAMPTEL$(L+1)
11490 LET CAMPIND$(L)=STR$(L)
11500 REM
11510 LET CAMPNOM$(L+1)=TCAMNOM$
11520 LET CAMPMOD$(L+1)=TCAMMOD$
11530 LET CAMPRUA$(L+1)=TCAMRUA$
11540 LET CAMPCID$(L+1)=TCAMCID$
11550 LET CAMPEST$(L+1)=TCAMEST$
11560 LET CAMPTEL$(L+1)=TCAMTEL$
11570 LET CAMPIND$(L+1)=STR$(L+1)
11580 LET S=1
11590 REM
11600 RETURN
12000 REM SUB-ROTINA *GRAREG*
12010 REM
12020 REM
```

```
12030 OPEN "O",#1, "AGEN.DAD"
12040 REM
12050 FOR L=1 TO TAMA-1
12060 PRINT #1,CAMPNOMS(L);",";CAMPMODS(L);",";
      CAMPRUAS(L);","CAMPCIDS(L)
12070 PRINT #1,CAMPESTS(L);",";CAMPTELS(L);","CAMPINDS(L)
12080 NEXT L
12090 REM
12100 REM
12110 REM
12120 REM
12130 CLOSE #1
12140 REM
12150 RETURN
13000 REM SUB-ROTINA *ENCREG* (ENCONTRAR UM REGISTRO)
13010 PRINT CHR$(12): REM LIMPA TELA
13020 REM
13030 IF CLAS=0 THEN GOSUB 11200: REM *CLAREG*
13040 PRINT
13050 PRINT
13060 PRINT TAB(3);"*PROCURANDO UM REGISTRO PELO NOME*"
13070 PRINT
13080 PRINT
13090 PRINT TAB(9);"DIGITE O NOME COMPLETO"
13100 PRINT TAB(7);"NA ORDEM DE NOME SOBRENOME"
13110 PRINT
13120 PRINT
13130 REM
13140 INPUT "O NOME E ";CAMPNOMS(TAMA)
13150 GOSUB 10200: REM SUB-ROTINA *MODNOM*
13160 LET CHAVES=CAMPMODS(TAMA)
13170 REM
13180 REM
13190 REM
13200 REM
13210 REM
13220 LET FIM=1
13230 LET INI=TAMA-1
13240 FOR L=1 TO 1
13250 LET MEI=INT((FIM+INI)/2)
13260 IF CAMPMODS(MEI) < > CHAVES THEN L=0
13270 IF CAMPMODS(MEI) < CHAVES THEN FIM=MEI+1
13280 IF CAMPMODS(MEI) > CHAVES THEN INI=MEI-1
13290 IF FIM > INI THEN L=1
13300 NEXT L
13310 REM
13320 IF FIM > INI THEN LET CORR=0
13330 IF FIM <= INI THEN LET CORR=MEI
13340 IF CORR=0 THEN GOSUB 13700: REM *NAOREG*
13350 IF CORR=0 THEN RETURN
13360 REM
13370 REM
13380 PRINT CHR$(12)
13390 PRINT
13400 PRINT TAB(15);"*REGISTRO ENCONTRADO*"
13410 PRINT
13420 PRINT "NOME: ";CAMPNOMS(CORR)
13430 PRINT "RUA: ";CAMPRUAS(CORR)
13440 PRINT "CIDADE: ";CAMPCIDS(CORR)
13450 PRINT "ESTADO: ";CAMPESTS(CORR)
13460 PRINT "TELEFONE: ";CAMPTELS(CORR)
13470 PRINT
13480 PRINT TAB(7);"PRESSIONE QUALQUER LETRA PARA IMPRIMIR"
13490 PRINT TAB(7);"OU A BARRA DE ESPACO PARA CONTINUAR"
13500 FOR I=1 TO 1
13510 LET A$=INKEY$
13520 IF A$="" THEN I=0
13530 NEXT I
13540 IF A$ < > " " THEN GOSUB 13600: REM *IMPCOR*
13550 RETURN
13600 REM SUB-ROTINA *IMPCOR* (IMPRIME O REGISTRO CORRENTE)
13610 LPRINT
13620 LPRINT "NOME: ";CAMPNOMS(CORR)
13630 LPRINT "RUA: ";CAMPRUAS(CORR)
13640 LPRINT "CIDADE: ";CAMPCIDS(CORR)
13650 LPRINT "ESTADO: ";CAMPESTS(CORR)
13660 LPRINT "TELEFONE: ";CAMPTELS(CORR)
13670 LPRINT
13680 LPRINT
13690 RETURN
13700 REM SUB-ROTINA *NAOREG* (REGISTRO NAO ENCONTRADO)
13710 PRINT CHR$(12): REM LIMPA TELA
13720 PRINT TAB(11);"REGISTRO NAO ENCONTRADO"
13730 PRINT TAB(4);"NO FORMATO: ";CAMPNOMS(TAMA)
13740 PRINT
13750 PRINT " (PRESSIONAR A BARRA DE ESPACO PARA CONTINUAR) "
13760 FOR I=1 TO 1
13770 IF INKEY$ < > "" THEN I=0
13780 NEXT I
13790 RETURN
14000 REM SUB-ROTINA *MODREG* (MODIFICAR UM REGISTRO)
14010 REM
14020 PRINT CHR$(12): REM LIMPA TELA
14030 PRINT
14040 PRINT
14050 PRINT
14060 PRINT
14070 PRINT TAB(10);"*PARA MODIFICAR UM REGISTRO*"
14080 PRINT TAB(5);"*PRIMEIRO ENCONTRE O REGISTRO DESEJADO*"
14090 REM
14100 REM
14110 REM
14120 GOSUB 13030: REM SUB-ROTINA *ENCREG* SEM LIMPAR A TELA
14130 IF CORR=0 THEN RETURN: REM REGISTRO NAO ENCONTRADO
14140 PRINT
14150 PRINT TAB(8);"QUER MODIFICAR O NOME?"
14160 PRINT
14170 PRINT TAB(2);"TECLE RETURN PARA ENTRAR COM O NOVO
      NOME"
14180 PRINT TAB(2);"OU A BARRA DE ESPACO PARA O PROXIMO
      CAMPO"
14190 FOR I=1 TO 1
14200 LET A$=INKEY$
14210 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
14220 NEXT I
14230 IF A$=CHR$(13) THEN INPUT "NOVO NOME: ";
      CAMPNOMS(CORR)
14240 IF A$=CHR$(13) THEN RMOD=1
14250 IF A$=CHR$(13) THEN CLAS=0
14260 IF A$=CHR$(13) THEN CAMPNOMS(TAMA)=
      CAMPNOMS(CORR)
14270 IF A$=CHR$(13) THEN GOSUB 10200: REM SUB-ROTINA
      *MODNOM*
14280 IF A$=CHR$(13) THEN LET CAMPMODS(CORR)=
      CAMPMODS(TAMA)
14290 PRINT
14300 PRINT TAB(8);"QUER MODIFICAR A RUA?"
14310 PRINT
14320 PRINT TAB(2);"TECLE RETURN PARA ENTRAR COM A
      NOVA RUA"
14330 PRINT TAB(2);"OU A BARRA DE ESPACO PARA O PROXIMO
      CAMPO"
14340 FOR I=1 TO 1
14350 LET A$=INKEY$
14360 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
14370 NEXT I
14380 IF A$=CHR$(13) THEN RMOD=1
14390 IF A$=CHR$(13) THEN INPUT "NOVA RUA: ";
      CAMPRUAS(CORR)
14400 PRINT
14410 PRINT TAB(8);"QUER MODIFICAR A CIDADE?"
14420 PRINT
14430 PRINT TAB(2);"TECLE RETURN PARA ENTRAR COM A NOVA
      CIDADE"
14440 PRINT TAB(2);"OU A BARRA DE ESPACO PARA O PROXIMO
      CAMPO"
14450 FOR I= 1 TO 1
14460 LET A$=INKEY$
14470 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
14480 NEXT I
14490 IF A$=CHR$(13) THEN RMOD=1
14500 IF A$=CHR$(13) THEN INPUT "NOVA CIDADE: ";
      CAMPCIDS(CORR)
14510 PRINT
14520 PRINT TAB(8);"QUER MODIFICAR O ESTADO?"
14530 PRINT
14540 PRINT TAB(2);"TECLE RETURN PARA ENTRAR COM O NOVO
      ESTADO"
14550 PRINT TAB(2);"OU A BARRA DE ESPACO PARA O PROXIMO
      CAMPO"
14560 FOR I=1 TO 1
14570 LET A$=INKEY$
14580 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
14590 NEXT I
14600 IF A$=CHR$(13) THEN RMOD=1
14610 IF A$=CHR$(13) THEN INPUT "NOVO ESTADO: ";
      CAMPESTS(CORR)
14620 PRINT
14630 PRINT TAB(8);"QUER MODIFICAR O TELEFONE?"
14640 PRINT
14650 PRINT TAB(2);"TECLE RETURN PARA ENTRAR COM O NOVO
      TELEFONE"
14660 PRINT TAB(2);"OU A BARRA DE ESPACO PARA CONTINUAR"
14670 FOR I=1 TO 1
14680 LET A$=INKEY$
14690 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
14700 NEXT I
14710 IF A$=CHR$(13) THEN RMOD=1
14720 IF A$=CHR$(13) THEN INPUT "NOVO TELEFONE: ";
      CAMPTELS(CORR)
14730 REM
14740 REM
14750 RETURN
15000 REM SUB-ROTINA *ELMREG* (ELIMINA UM REGISTRO)
15010 REM
15020 PRINT CHR$(12): REM LIMPA TELA
15030 PRINT
15040 PRINT
15050 PRINT
15060 PRINT
15070 PRINT TAB(10);"*PARA ELIMINAR UM REGISTRO*"
15080 PRINT TAB(5);"*LOCALIZE PRIMEIRO O REGISTRO DESEJADO*"
15090 REM
15100 REM
15110 REM
15120 GOSUB 13030: REM SUB-ROTINA *ENCREG* SEM LIMPAR
      A TELA
15130 IF CORR=0 THEN RETURN: REM REGISTRO NAO ENCONTRADO
15140 PRINT
15150 PRINT TAB(3);"VOCE DESEJA ELIMINAR ESTE REGISTRO?"
15160 PRINT TAB(3);"*ATENCAO* - NAO HAVERA OUTRA CHANCE"
15170 PRINT
15180 PRINT TAB(7);"TECLE RETURN PARA ELIMINAR"
15190 PRINT TAB(3);"OU A BARRA DE ESPACO PARA CONTINUAR"
15200 FOR I=1 TO 1
15210 LET A$=INKEY$
15220 IF A$ < > CHR$(13) AND A$ < > " " THEN I=0
15230 NEXT I
15240 IF A$=" " THEN RETURN
15250 FOR L=CORR TO TAMA-2
15260 LET CAMPNOMS(L)=CAMPNOMS(L+1)
15270 LET CAMPMODS(L)=CAMPMODS(L+1)
15280 LET CAMPRUAS(L)=CAMPRUAS(L+1)
15290 LET CAMPCIDS(L)=CAMPCIDS(L+1)
15300 LET CAMPESTS(L)=CAMPESTS(L+1)
15310 LET CAMPTELS(L)=CAMPTELS(L+1)
15320 LET CAMPINDS(L)=STR$(L)
15330 NEXT L
15340 LET RMOD=1
15350 LET TAMA=TAMA-1
15360 REM
15370 REM
15380 REM
15390 RETURN
```

Questão de estilo

Conhecidas as regras fundamentais da linguagem BASIC, podemos nos deter agora no estilo de programação e em novos comandos que aperfeiçoarão nossa técnica.

O programa da agenda de endereços computadorizada, desenvolvido nos fascículos anteriores, emprega muitos recursos importantes da linguagem BASIC, mas não todos. Agora, na conclusão deste curso, examinaremos até onde o BASIC pode levá-lo, se você quer se tornar um programador de nível adiantado.

Linguagem de máquina

A maioria das versões do BASIC permite incluir como parte do programa rotinas desenvolvidas em código de máquina. De maneira geral, há dois métodos de fazer isso. O mais simples consiste em utilizar os comandos PEEK e POKE. PEEK é uma instrução empregada para examinar endereços específicos de memória. Por exemplo, LET X = PEEK(1000) faz com que o valor armazenado na posição de endereço 1000 seja atribuído à variável X. A execução de PRINT X imprime, então, o valor que estava (e ainda deve estar) na posição 1000. O pequeno programa que apresentamos a seguir examina os conteúdos de dezesseis posições de memória, mediante o comando PEEK, e as imprime na tela:

```
10 INPUT "FORNECER ENDERECO DE INICIO PARA
   'PEEK'"; S
20 PRINT
30 FOR L = 1 TO 16
40 LET A = PEEK(S)
50 PRINT "A POSICAO"; S; "CONTEM: ";A
60 LET S = S + 1
70  NEXT L
80 PRINT "PRESSIONAR A BARRA DE ESPACO
   PARA EXAMINAR"
90 PRINT "AS PROXIMAS 16 POSICOES OU
   RETURN PARA TERMINAR"
100 FOR I = 1 TO 1
110 LET C$ = INKEY$
120 IF C$ < > CHR$(13) AND C$ < > " " THEN I = 0
130 NEXT I
140 IF C$ = CHR$(13) THEN GOTO 160
150 GOTO 30
160 END
```

O loop nas linhas de 100 a 130 verifica a entrada a partir do teclado e em seguida vai para o fim do programa, se o caractere digitado foi RETURN (13, em ASCII), ou volta ao início, saltando a instrução INPUT.

O caractere ASCII da posição de memória também pode ser impresso usando PRINT CHR$(A). Cuidado, porém, pois os valores ASCII menores que o decimal 32 (ASCII para o caractere de "espaço") não estão definidos de modo uniforme. Todos os valores ASCII de 0 a 31 representam caracteres não imprimíveis ou funções especiais, como controles de cursor. Um dos raros padrões comuns dos fabricantes de computadores consiste em ASCII 13 corresponder ao retorno do carro (RETURN) e em ASCII 7 produzir som ou um "bip" no alto-falante interno.

O comando POKE é o inverso de PEEK. Permite que você registre qualquer valor, de 0 até 255, em alguma posição da RAM. No entanto, deve-se empregar esse recurso com cautela, pois o registro numa parte da memória já utilizada pelo programa causa às vezes resultados catastróficos. As rotinas escritas em código de máquina são armazenadas por meio do comando POKE no endereço adequado e recuperadas, no processamento, pela instrução CALL.

O método para desenvolver programas em código de máquina vai além dos objetivos deste curso de BASIC. Basta lembrar que o código de máquina opera de maneira muito mais rápida do que as mais eficientes versões do BASIC. Quando a velocidade de execução ou uma grande precisão forem exigidas, o código de máquina revela-se a melhor opção.

Movendo o cursor

Muitos dos microcomputadores permitem o endereçamento das posições na própria tela. Mesmo que seu equipamento não se enquadre nesse caso, é possível mover o cursor na tela (para a esquerda, para a direita, para cima e para baixo) com relativa facilidade. Em primeiro lugar, você precisa saber quais os códigos ASCII usados para representar as teclas de controle do cursor. Este pequeno programa solicita a digitação de uma tecla e indica o valor ASCII correspondente:

```
1 REM DESCOBRINDO OS CODIGOS ASCII PARA
  AS TECLAS DO CURSOR
10 PRINT "PRESSIONAR UMA TECLA";
20 FOR I = 1 TO 1
30 LET K$ = INKEY$
40 IF K$ = " " THEN I = 0
50 NEXT I
60 PRINT ASC(K$)
70 GOTO 10
80 END
```

Essa rotina também permite encontrar o código da tecla RETURN (em geral, 13), ESCape (quase sempre, 27) e a de espaço (geralmente, 32), além dos códigos para as teclas de controle do cursor. Existem micros que utilizam estes valores: 8, para deslocar o cursor para a esquerda; 28, para a direita; 29, para cima e 30, para baixo. Seu computador, com certeza, tem valores diferentes. Para substituí-los no programa citado, experimente o seguinte:

```
10 PRINT CHR$(12): REM UTILIZAR O CODIGO CLS
   OU O CODIGO CORRESPONDENTE
20 FOR L = 1 TO 39
30 PRINT "*";
40 NEXT L
50 FOR L = 1 TO 22
60 PRINT CHR$(8); :REM UTILIZAR O CODIGO
   "CURSOR ESQUERDO"
70 NEXT L
80 FOR L = 1 TO 4
90 PRINT "@";
100 NEXT L
110 END
```

Esse programa imprimirá na tela uma linha parecida com a que se segue:

**********@@@@**********

As linhas de 20 a 40 teriam imprimido apenas uma linha com 39 asteriscos. Contudo, as linhas de 50 a 70 "imprimiram" o "caractere" cursor para a esquerda 22 vezes; desse modo, o cursor se deslocou 22 posições para trás. As linhas de 80 a 100 imprimiram @ quatro vezes, e o programa se encerrou. Técnicas como essa permitem deslocar o cursor pela tela para impressão de caracteres em novas posições que podem não ser conhecidas até que seus valores sejam calculados pelo programa. Essa técnica tem a vantagem de possibilitar o emprego de caracteres comuns de tela para traçar gráficos simples, sem usar os recursos especiais do computador (se houver).

Para verificar de que modo esse tipo de controle de cursor pode ser feito na produção de gráficos como saída de seus programas, experimente a listagem abaixo:

```
10 PRINT "ESTE PROGRAMA IMPRIME UM GRAFICO
   DE BARRAS DE TRES VARIAVEIS"
20 INPUT "FORNECER OS TRES VALORES ";X,Y,Z
30 PRINT
40 FOR L = 1 TO 2
50 FOR A = 1 TO X
60 PRINT "*";
70 NEXT A
80 PRINT CHR$(13)
90 NEXT L
100 FOR L = 1 TO 2
110 FOR A = 1 TO Y
120 PRINT "+";
130 NEXT A
140 PRINT CHR$(13)
150 NEXT L
160 FOR L = 1 TO 2
170 FOR A = 1 TO Z
180 PRINT "#";
190 NEXT A
200 PRINT CHR$(13)
210 NEXT L
220 PRINT
230 END
```

Esse programa imprime um gráfico de barras das três variáveis. As barras são impressas em linhas horizontais, a partir da esquerda, e seguem o movimento "natural" do cursor. Observe que é necessária uma instrução PRINT CHR$(13) nas linhas 80, 140 e 200. Isso porque os sinais de ponto e vírgula no final das instruções PRINT suprimem os RETURN (13 é o código ASCII para <CR>).

Mais sobre variáveis

Até aqui, tratamos das variáveis como se fossem apenas de dois tipos (numéricas e alfanuméricas). Na verdade, há muitos tipos de variáveis numéricas que o BASIC pode reconhecer. O bom programador sempre especifica o tipo correto, para economizar capacidade de memória e garantir a precisão.

Quando se declara uma variável em linguagem de programação, certa quantidade de espaço na memória é reservada para seu armazenamento. Se o programa souber que a variável sempre será inteira (por exemplo, LET CONTAGEM = TOTAL + BONIFICAÇÃO – MULTA), reservará menor capacidade de memória. Se tivermos uma variável que pode assumir infinitos valores reais (por exemplo, LET ÁREA = PI *RAIO* RAIO), o espaço de memória requerido será maior.

No desenvolvimento de nossa agenda de endereços computadorizada, adotamos a convenção de especificar variáveis alfanuméricas usando o sinal $ após o nome da variável (por exemplo, LET CHAVE$ = CAMPMOD$(TAMA)). As que aparecem sem o sinal de cifrão foram consideradas numéricas comuns. Porém, convenções semelhantes podem ser usadas depois de nomes de variáveis para especificar o tipo de variável numérica. Quando não há especificador, consideramos variáveis numéricas reais de precisão simples. Outros sinais admitidos pela maioria das linguagens BASIC são: % para especificação de variáveis inteiras; ! para as de precisão simples; e # para as de precisão dupla (isto é, as variáveis que podem armazenar o dobro de dígitos significativos). Eis um fragmento de programa hipotético que emprega esses três sinais:

```
70 LET JOGADOR = "JOAO": REM UMA
   VARIAVEL ALFANUMERICA
80 LET PLACAR% = 0: REM UMA
   VARIAVEL INTEIRA
90 LET PI! = 3.1416: REM UMA VARIAVEL
   DE PRECISAO SIMPLES
```

```
100 LET AREA# = PI*R*R: REM VARIAVEL
    DE PRECISAO DUPLA
110 LET JOGADAS = 6: REM ASSUME
    COMO SENDO REAL DE PRECISAO
    SIMPLES
```

Nem todas as versões da linguagem BASIC possuem todos esses tipos de variáveis. O Spectrum da Sinclair, por exemplo, não dispõe de variáveis inteiras. Os números inteiros são armazenados como reais de precisão simples. Também não possui números de dupla precisão. Contudo, no BASIC do Spectrum, números de precisão simples são calculados com até nove algarismos significativos, enquanto no BASIC da Microsoft a precisão resume-se a sete algarismos significativos. Vários micros admitem variáveis do tipo inteiro e números reais de precisão simples calculados até nove algarismos significativos. O BASIC da Microsoft admite variáveis de precisão dupla para dezesseis posições significativas.

Os computadores que efetivamente trabalham com variáveis inteiras utilizam quase sempre 2 bytes de memória para armazenar o número, que pode estar entre −32.768 e 32.767. Essa amplitude, em geral, mostra-se bem adequada para variáveis como resultados, quantidade de funcionários, contadores de loops FOR-NEXT e outros números que devem ter valores inteiros. Uma vez que se empregam apenas 2 bytes para armazenar cada número, o uso de variáveis inteiras, se houver, economiza tempo, embora para muitas versões de BASIC isso aconteça apenas com matrizes inteiras e não com variáveis individuais.

Exemplificamos abaixo as quatro funções restantes de nossa agenda computadorizada:

Opção 2 - Encontrar nomes (por trecho de um nome)
Opção 3 - Encontrar registros (de uma cidade)
Opção 4 - Encontrar registros (de uma inicial)
Opção 5 - Listar todos os registros

Para acrescentar o programa listado a seguir em nosso programa principal, devemos substituir as linhas de 4050 a 4080 do programa original para:

```
4050 IF ESCL=2 THEN GOSUB 16000: REM *ENCNOM*
4060 IF ESCL=3 THEN GOSUB 17000: REM *ENCCID*
4070 IF ESCL=4 THEN GOSUB 18000: REM *ENCINI*
4080 IF ESCL=5 THEN GOSUB 19000: REM *LISREG*
```

Por outro lado, todo o programa da agenda computadorizada foi desenvolvido tomando-se por base a versão BASIC da Microsoft. Caso seu equipamento tenha outra versão do BASIC, o programa pode ser convertido se você adaptá-lo a partir das indicações de nossos quadros "A propósito..." e também estudando o manual BASIC do fabricante de seu equipamento.

```
16000 REM SUB-ROTINA *ENCNOM*
16010 REM ESTA SUB-ROTINA ENCONTRA
16020 REM UM REGISTRO A PARTIR DE UM
16030 REM TRECHO DE PRIMEIRO NOME OU
16040 REM DE UM TRECHO DE SOBRENOME
16050 PRINT CHR$(12): REM LIMPA TELA
16060 PRINT
16070 PRINT
16080 PRINT
16090 PRINT "FORNECER UM TRECHO DE UM PRIMEIRO NOME"
16100 PRINT "OU TRECHO DE UM SOBRENOME A SER
      ENCONTRADO"
16110 TRECHO$=" "
16120 INPUT TRECHO$
16130 LET W=LEN(TRECHO$)
16140 REM CONVERTER TRECHO$
      EM MAIUSCULAS
16150 LET N$=TRECHO$
16160 GOSUB 20000: REM SUB-ROTINA
      *MAIUSCULAS*
16170 REM
16180 REM
16190 REM
16200 REM
16210 REM
16220 REM
16230 LET TRECHO$=P$
16240 LET P$=" ": LET N$=" "
16250 REM FAZER COMPARACAO DE TRECHO COM
16260 REM TODOS OS REGISTROS DA AGENDA
16270 REM ATE ENCONTRAR O TRECHO
16280 FOR L=1 TO TAMA-1
16290 FOR K=1 TO LEN(CAMPMOD$(L))-W
16300 TEMP$=MID$(CAMPMOD$(L),K,W)
16310 IF TRECHO$=TEMP$ THEN GOTO 16500
16320 NEXT K
16330 NEXT L
16340 REM REGISTRO NAO ENCONTRADO
16350 PRINT CHR$(12)
16360 PRINT
16370 PRINT
16380 PRINT "INFELIZMENTE NAO FOI ENCONTRADO"
16390 PRINT "UM REGISTRO CONTENDO O
      TRECHO: ";TRECHO$
16400 PRINT
16410 PRINT"(PRESSIONAR A BARRA DE ESPACO PARA
      CONTINUAR)"
16420 FOR I=1 TO 1
16430 IF INKEY$ < >" " THEN I=0
16440 NEXT I
16450 RETURN
16500 LET CORR=L
16510 GOSUB 13380: REM REGISTRO ENCONTRADO
16520 RETURN
17000 REM SUB-ROTINA *ENCCID*
17010 REM ESTA SUB-ROTINA ENCONTRA TODOS OS
17020 REM REGISTROS DE UMA MESMA CIDADE
17030 REM
17040 PRINT CHR$(12): REM LIMPA TELA
17050 PRINT
17060 PRINT
17070 PRINT
17080 PRINT "FORNECER O NOME COMPLETO DA CIDADE"
17090 PRINT "PARA ENCONTRAR OS REGISTROS"
17100 LET CIDADES=" "
17110 INPUT CIDADES
17120 REM CONVERTER CIDADES EM MAIUSCULAS
17130 LET N$=CIDADES
17140 GOSUB 20000: REM SUB-ROTINA *MAIUSCULAS*
17150 LET CIDADES=P$
17160 LET P$=" ": LET N$=" "
17170 REM
```

```
17180 REM
17190 REM
17200 FOR L=1 TO TAMA-1
17210 REM CONVERTER CAMPCID$ EM MAIUSCULAS
17220 LET CID$=" ": LET CID$=CAMPCID$(L)
17230 LET NS=CID$
17240 GOSUB 20000: REM SUB-ROTINA "MAIUSCULAS"
17250 LET CID$=PS
17260 LET PS=" ": LET NS=" "
17270 IF CIDADES < > CID$ THEN GOTO 17300
17280 CORR=L: GOSUB 13400: REM REGISTRO ENCONTRADO
17290 REM
17300 NEXT L
17310 PRINT
17320 PRINT "ISSO E TUDO QUE FOI ENCONTRADO"
17330 PRINT "PARTIR DA CIDADE: ";CIDADES
17340 PRINT
17350 PRINT " (PRESSIONAR A BARRA DE ESPACO PARA
      CONTINUAR) "
17360 FOR I=1 TO 1
17370 IF INKEYS < > " " THEN I=0
17380 NEXT I
17390 RETURN
18000 REM SUB-ROTINA "ENCINI"
18010 REM ESTA SUB-ROTINA ENCONTRA REGISTROS
18020 REM A PARTIR DAS INICIAIS DO NOME
18030 REM NO FORMATO INICIAL DO PRIMEIRO NOME
18040 REM E INICIAL DO SOBRENOME
18050 PRINT CHR$(12): REM LIMPA TELA
18060 PRINT
18070 PRINT
18080 PRINT
18090 PRINT "FORNECER AS INICIAIS DO REGISTRO"
18100 PRINT "A SER ENCONTRADO NO FORMATO DE"
18110 PRINT "INICIAL DO PRIMEIRO NOME E INICIAL"
18120 PRINT "DO SOBRENOME."
18130 PRINT "EXEMPLO: RA PARA RAUL AGUIAR"
18140 LET INICIALS=" "
18150 INPUT INICIALS
18160 REM CONVERTER INICIALS EM MAIUSCULAS
18170 LET NS=INICIALS
18180 GOSUB 20000: REM SUB-ROTINA "MAIUSCULAS"
18190 LET INICIALS=PS
18200 LET PS=" ": LET NS=" "
18210 REM
18220 FOR L=1 TO TAMA-1
18230 REM ENCONTRAR INICIAIS DE CAMPMOD$
18240 FOR K=1 TO LEN(CAMPMOD$(L))
18250 IF MID$(CAMPMOD$(L),K,1)=" " THEN GOTO 18270
18260 NEXT K
18270 LET INIC$=MID$(CAMPMOD$(L),
      K+1,1)+MID$(CAMPMOD$(L),1,1)
18280 IF INICIALS < > INIC$ THEN GOTO 18310
18290 LET CORR=L: GOSUB 13400
18300 REM
18310 NEXT L
18320 PRINT "ISSO E TUDO QUE FOI ENCONTRADO"
18330 PRINT "A PARTIR DAS INICIAIS: ";INICIALS
18340 PRINT
18350 PRINT " (PRESSIONAR A BARRA DE ESPACO
      PARA CONTINUAR) "
18360 FOR I=1 TO 1
18370 IF INKEY$ < > " " THEN I=0
18380 NEXT I
18390 RETURN
19000 REM SUB-ROTINA "LISREG"
19010 REM ESTA SUB-ROTINA LISTA TODOS OS
19020 REM REGISTROS DA AGENDA COMPUTADORIZADA
19030 REM
19040 PRINT CHR$(12): REM LIMPA TELA
19050 PRINT
19060 PRINT
19070 PRINT
19080 PRINT "PRESSIONE: "
19090 PRINT "T PARA LISTAR TODOS OS REGISTROS NA TELA"
19100 PRINT "I PARA LISTAR TODOS OS REGISTROS NA
      IMPRESSORA"
19110 PRINT "A BARRA DE ESPACO PARA VOLTAR AO MENU
      PRINCIPAL"
19120 FOR I=1 TO 1
19130 LET A$=INKEY$
19140 IF A$=" " THEN I=0
19150 NEXT I
19160 IF A$="T" THEN GOTO 19300
19170 IF A$="I" THEN GOTO 19600
19180 IF A$=" " THEN RETURN
19190 GOTO 19120
19300 REM IMPRIMIR TODOS OS REGISTROS NA TELA
19310 PRINT CHR$(12): REM LIMPA TELA
19320 REM
19330 LET T=0
19340 FOR L=1 TO TAMA-1
19350 PRINT: LET T=T+1
19360 PRINT "REGISTRO NO. ";CAMPIND$(L)
19370 PRINT "NOME: ";CAMPNOM$(L)
19380 PRINT "RUA: ";CAMPRUA$(L)
19390 PRINT "CIDADE: ";CAMPCID$(L)
19400 PRINT "ESTADO: ";CAMPEST$(L)
19410 PRINT "TELEFONE: ";CAMPTEL$(L)
19420 IF T=3 THEN GOSUB 19500
19430 NEXT L
19440 PRINT
19450 PRINT "***FIM DO ARQUIVO***"
19460 PRINT
19470 GOTO 19080
19500 PRINT
19510 REM
19520 PRINT "PRESSIONE QUALQUER TECLA PARA CONTINUAR"
19530 FOR I=1 TO 1
19540 IF INKEY$=" " THEN I=0
19550 NEXT I
19560 LET T=0
19570 RETURN
19600 REM IMPRIMIR TODOS OS REGISTROS NA IMPRESSORA
19610 PRINT CHR$(12): REM LIMPA TELA
19620 PRINT
19630 PRINT
19640 PRINT "VERIFIQUE A IMPRESSORA E O PAPEL"
19650 PRINT
19660 PRINT "PRESSIONE QUALQUER TECLA PARA INICIAR A
      IMPRESSAO"
19670 FOR I=1 TO 1
19680 IF INKEY$=" " THEN I=0
19690 NEXT I
19700 FOR L=1 TO TAMA-1
19710 LET CORR=L
19720 GOSUB 13620: REM SUB-ROTINA "IMPCOR"
19730 NEXT L
19740 GOTO 19040
20000 REM SUB-ROTINA "MAIUSCULAS"
20010 REM ESTA SUB-ROTINA CONVERTE UMA VARIAVEL
20020 REM ALFANUMERICA QUALQUER EM MAIUSCULAS
20030 FOR M=1 TO LEN(NS)
20040 LET TEMP$=MID$(NS,M,1)
20050 LET T=ASC(TEMP$)
20060 IF T >= 97 THEN T=T-32
20070 LET TEMP$=CHR$(T)
20080 LET PS=PS+TEMP$
20090 NEXT M
20100 RETURN
```

Linguagem alternativa

Encerrando nosso curso, faremos a avaliação crítica da linguagem BASIC e de algumas alternativas com relação a ela.

Como conclusão de nosso curso de programação em BASIC, examinaremos rapidamente os pontos fortes e os fracos dessa linguagem, em comparação com outras de programação.

O BASIC é uma ramificação da linguagem em FORTRAN, uma das primeiras de programação. Ao contrário da maioria delas, o BASIC é interpretado. Isso significa que, quando um programa em BASIC é executado, um outro, especial, em algum ponto da memória do computador, interpreta o código linha por linha e converte as instruções do BASIC em código de máquina. Eis o que acontece em um pequeno programa desse tipo:

```
10 CLS
20 PRINT "DIGITE UM NUMERO"
30 INPUT X
40 PRINT "DIGITE UM SEGUNDO NUMERO"
50 INPUT Y
60 "PRINT "O PRODUTO DOS DOIS NUMEROS
   E: ";
70 PRINT X*Y
80 PRINT
90 PRINT "VOCE DESEJA OUTRO LANCE?"
100 PRINT "PRESSIONAR "S" PARA NOVA
    TENTATIVA"
110 PRINT "OU "N" PARA ENCERRAR"
120 FOR X = 1 TO 1
130 LET A$ = INKEY$
140 IF A$ <> "N" AND A$ <> "S" THEN X = 0
150 NEXT X
160 IF A$ = "S" THEN GOTO 10
170 END
```

Ao encontrar a linha 10, o interpretador BASIC opera o código de máquina necessário para limpar a tela. Na linha 20, o interpretador dá as instruções necessárias para enviar à tela a mensagem DIGITE UM NÚMERO. Na linha 30, determina o espaço de memória para armazenar um número real, aguarda o fornecimento de um item pelo teclado e a seguir converte o número digitado em código binário, armazenando-o no espaço reservado para a variável X. Todo esse procedimento se repete nas linhas de 40 a 60. Se o usuário quer repetir o programa por meio da digitação de S, o interpretador desvia de novo para a linha 10 e faz todos os cálculos e as computações novamente.

A maioria das outras linguagens é "compilada". Ou seja, após ter sido desenvolvido, o programa processa-se por um "compilador" antes de ser rodado. O compilador, um programa separado, passa pelo "código fonte" (o programa original) e produz uma segunda versão em código de máquina. Ao se processar, o programa compilado opera, possivelmente, de modo muito mais rápido que o programa interpretado, pois todas as traduções para código de máquina que exigem tempo já se terão realizado.

Uma vez que os programas compilados funcionam muito mais depressa que os interpretados, talvez você pergunte por que nem todas as linguagens empregam compiladores. Há várias vantagens no emprego de programas interpretados, como no BASIC, sobressaindo o aspecto interativo da linguagem. Isso significa que ela pode ser testada e seus erros eliminados "a partir do teclado", à medida que o programa se desenvolve. O BASIC, por exemplo, permite a inserção do comando STOP em qualquer ponto do programa. Ao encontrar essa instrução, o interpretador interrompe seu trabalho e passa a admitir a emissão de "comandos" a partir do teclado.

Comandos são instruções executáveis diretamente pelo interpretador, quando o programa não está sendo processado. O BASIC possui grande número desses comandos, que podem ser imprescindíveis na eliminação de erros. Após a execução de um programa em linguagem BASIC, isto é, quando o interpretador encontra a instrução END ou a instrução STOP, pode-se imprimir (PRINT) os valores de todas as variáveis. Experimente processar o programa da agenda de endereços, por exemplo, e digite o número 9 para desvio do programa. Se for processado por inteiro sem mensagens de erro, deverá encerrar-se com uma indicação em BASIC que, em geral, é OK, > ou *. Você então digita PRINT RMOD. O interpretador imprime o número 0 na tela (desde que você não tenha acrescentado mais algum registro!). A seguir, digite PRINT TAMA. O interpretador imprime na tela um número, uma unidade maior que o número de registros que você tiver no arquivo de dados.

Vantagens do BASIC

Considera-se a linguagem BASIC ideal para o programador inexperiente, porque admite a eliminação dos erros por meio do teclado. Outra grande vantagem reside na facilidade de seu aprendizado. Note, por exemplo, que em nosso curso de programação em BASIC examinamos todos os pontos fundamentais da linguagem e muitos de seus aspectos mais avançados em apenas 86 páginas. Erros de sintaxe, como 40 PRINT A(12), quase sempre resultarão em mensagens de erro fáceis de ser entendidas, na execução do programa, como SYNTAX ERROR IN 40. Uma olhada na instrução referente ao número de linha mencionado é suficiente para mostrar onde se en-

contra o erro; e a retificação, comumente, consiste em digitar EDIT 40 (seguido por alguns comandos de correção) ou refazer a linha, digitando-a de modo correto. Ao encontrar um erro lógico ou de sintaxe, o interpretador BASIC interrompe a execução do programa e indica o erro. A correção de erros consiste em experimentar uma nova linha em substituição à errada e digitar RUN de novo.

Desvantagens do BASIC

A programação em BASIC apresenta uma série de inconvenientes, alguns sutis, outros mais evidentes. Por se tratar de uma linguagem interpretada (embora existam algumas versões compiladas do BASIC), seu processamento é de lenta execução. Se rapidez não for essencial (como num programa para cálculo de conta bancária, por exemplo), a lentidão do BASIC interpretado não traz inconvenientes. Mas se a rapidez for importante (como num programa para animação de tela que emprega gráficos, ou num "relógio" para cronometrar as reações em experiência de laboratório), é muito provável que o BASIC interpretado se mostre excessivamente lento.

Se você precisar acelerar seus programas, há duas alternativas: a programação em código de máquina ou em linguagem Assembly (ver p. 448), processo difícil, que consome tempo, ou programação em linguagem compilada, como PASCAL ou FORTH. As linguagens compiladas não se mostram difíceis de aprender, mas é muito provável que o código fonte (o programa original) contenha erros, detectáveis ao se compilar o programa. As retificações não são fáceis, em comparação com as do BASIC. Depois de feitas as correções do código fonte, deve-se compilar o programa outra vez por inteiro. A maior parte dos compiladores executa duas ou três passagens pelo código fonte e cada uma delas na certa resulta em mensagens de erro, que devem ser corrigidas antes da recompilação do programa.

A produção de um programa compilado de maneira correta pode requerer muito mais tempo que a de um em BASIC interpretado. Todavia, o BASIC tem maior possibilidade de afastar o programador principiante da objetividade e da concisão, estimulando técnicas de programação viciosas, que linguagens bem estruturadas como PASCAL não admitiriam. O BASIC permite ao usuário escrever programas com pouco cuidado, cheios de GOTOs, por exemplo — são maus hábitos, que dificultam a transição para linguagens mais avançadas.

E o que mais, após o BASIC?

A linguagem BASIC mostra-se flexível e fácil. Tem excelentes recursos de manipulação de séries, mas é lenta e não tira proveito máximo da capacidade do microcomputador. Linguagens mais modernas, como PASCAL e FORTH, apresentam bons recursos de programação, difíceis ou impossíveis no BASIC.

A linguagem PASCAL também foi criada para aprendizado e projetada especificamente para incentivar o desenvolvimento de programas "estruturados" e bem construídos. Constitui linguagem compilada, ou seja, o usuário encontrará numerosos erros detectados pelo compilador (após o código fonte ter se desenvolvido e antes que o "código objeto" compilado se processe), e isso pode ser muito frustrante. Os programadores principiantes em PASCAL consideram as restrições da linguagem — a necessidade de declarar todas as variáveis no início do programa e indicar a que tipo pertencem: reais, inteiras etc. — como uma limitação à programação livre e flexível.

Em PASCAL, o programador tem de refletir cuidadosamente sobre a estrutura lógica do programa antes de desenvolvê-lo. Nos programas em PASCAL, há grande possibilidade de numerosos erros de sintaxe no código fonte. Mas esses programas são mais bem estruturados e têm menor probabilidade de conter erros básicos de lógica.

A linguagem FORTH vem se tornando uma alternativa muito popular ao BASIC, para microcomputadores. Embora seu aprendizado não se mostre difícil como Assembly ou a linguagem de código de máquina, devemos salientar que é muito menos "intuitiva" que o BASIC ou PASCAL. Mesmo assim, a linguagem FORTH possui muitas qualidades exclusivas que a tornam séria candidata a segunda linguagem do programador.

Embora de alto nível, a linguagem FORTH processa com rapidez muito próxima à do código de máquina, em decorrência do modo excepcional com que opera. Enquanto linguagens como o BASIC têm número fixo de instruções e comandos, os usuários da FORTH podem definir seu próprio vocabulário.

A palavra-chave PRINT, em BASIC, significa que os caracteres que a seguem, colocados entre aspas, serão apresentados na tela. Na linguagem FORTH, PRINT pode ser definido para apresentar na tela, digamos, uma relação dos equivalentes hexadecimais dos códigos ASCII relativos aos caracteres de determinada variável, impressos numa coluna vertical.

A linguagem FORTH permite que o programador dê a qualquer palavra a significação que queira e apresente os resultados desejados sempre que utilizada daí por diante. FORTH não é apenas muito flexível nesse sentido; também apresenta programas que podem ser compilados para código objeto (ver p. 184), os quais são quase tão compactos e rápidos no processamento quanto os de linguagem de máquina.

Embora haja muitas linguagens de programação, a maioria dos amadores que deixam o BASIC prefere escolher entre Assembly, PASCAL e FORTH. Estas são, em resumo, as vantagens e desvantagens de cada uma delas:

BASIC
Aprendizado fácil
Memorização fácil
Correção de erros fácil
Execução lenta
Grande utilização de espaço na memória
Desincentivo à programação estruturada

Linguagem Assembly
Aprendizado não muito fácil
Memorização não muito fácil
Correção difícil
Execução muito rápida
Permissão de controle total do microprocessador

PASCAL
Aprendizado relativamente fácil

Memorização moderadamente fácil
Eliminação de erros mais difícil que em BASIC
Incentivo a melhores técnicas de programação
Execução mais rápida que o BASIC; mais lenta que o Assembly
Necessidade de ser compilada, o que toma tempo; uma vez compilada, processa quase tão rapidamente quanto Assembly
Possibilidade de controle relativo sobre o microprocessador, porém menos que Assembly; a manipulação de variáveis alfanuméricas não é tão fácil como em BASIC

FORTH
Aprendizado não muito fácil; mais fácil para os principiantes, não tão fácil para os programadores em BASIC
Memorização relativamente fácil
Correção de erros no modo interpretado muito fácil
Compilação possível; executa quase tão rapidamente quanto a linguagem Assembly
Permissão de controle total do microprocessador
Consumo de pouca capacidade de memória
Aprendizado mais fácil que a linguagem Assembly, embora menos "intuitiva" que o BASIC

A propósito...

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
POINT (x,y)	Fornece a cor do ponto especificado	+		+	
POKE n,m	Aloca m na posição de memória n	+	+	+	+
POP	Esquecer a posição de retorno do último GOSUB executado		+		
POS (n)	Fornece a coluna da posição atual do cursor	+	+	+	
PR # slot	Seleciona o número do slot para a saída		+		
PRINT @ n;expressão	Inicia a impressão na tela a partir da posição n			+	
PRINT TAB(n);expressão	Move o cursor para a posição n da tela e imprime			+	+
PRINT USING x$;expressão	Especifica o formato de saída da impressão			+	
PRINT lista de expressões	Imprime na tela uma lista de variáveis ou expressões	+	+	+	+
PRINT #-1,lista de variáveis	Grava dados em fita cassete			+	
PRINT"arquivo",lista de expr	Grava dados em um arquivo seqüencial	+			
PUT"arquivo",número	Grava um registro de um buffer para um arquivo	+			
RAND n	Atribui o valor n à variável do sistema para gerar RND				+
RANDOM	Aciona o gerador de números aleatórios			+	
RANDOMIZE n	Aloca o valor n ao sistema para gerar um n.º aleatório	+			
READ "nomearquivo"	Dá o nome do arquivo para pegar dados com GET e INPUT		+		
READ lista de variáveis	Lê os valores da instrução DATA e os aloca às variáveis	+	+	+	
RECALL variável	Traz os valores da variável de uma fita cassete		+		
REM comentário	Insere comentários em um programa	+	+	+	+
RENUM novo, velho, incremento	Renumera as linhas do programa	+			
RESET	Reinicia informações do disquete	+			
RESET (x,y)	Desativa a posição (x,y) da tela	+		+	
RESTORE	Endereça o pointer do DATA no início da lista		+	+	
RESTORE linha	Permite a execução do DATA a partir da linha indicada	+			
RESUME	Faz voltar a execução onde ocorreu um erro		+		
RESUME linha	Continua a execução do programa na linha após erro	+		+	
RETURN	Retorna a execução do programa para o último GOSUB	+	+	+	+
RETURN linha	Retorna a execução do programa para a linha desejada	+			
RIGHT$(x$,n)	Fornece o enésimo caractere mais à direita de x$	+	+	+	
RND	Fornece um número aleatório entre 0 e 1				+
RND (0)	Fornece um número aleatório entre 0 e 1	+		+	
RND (x)	Fornece um número aleatório entre 0 e 1	+	+	+	
ROT=x	Muda a orientação do desenho em alta resolução		+		
RUN "arquivo"	Executa um programa	+	+		

Comando/Instrução/Função	Ação/Resultado	MS BASIC	Applesoft	TRS-80	Sinclair
RUN linha	Executa o programa em memória a partir da linha indicada	+	+	+	+
SAVE "arquivo"	Grava o programa em memória com o nome definido	+	+		+
SCALE=x	Fornece a escala para gráficos em alta resolução		+		
SCREEN(linha,coluna,z)	Dá o código ASCII do caractere na posição da tela	+			
SCRN(x,y)	Dá a cor do ponto (x,y) em gráfico de baixa resolução		+		
SCROLL	Empurra a imagem da tela uma linha para cima				+
SET (x,y)	Ativa a posição (x,y) da tela	+		+	
SGN (x)	Fornece o sinal de x(1,0 ou -1)	+	+	+	+
SHLOAD	Carrega um desenho em alta resolução da fita cassete		+		
SIN (x)	Fornece o cálculo do seno de x, x em radianos	+	+	+	+
SLOW	Apresenta o display de forma contínua				+
SOUND freq,duração	Gera um som com freqüência e duração indicadas	+			
SPACE$(n)	Fornece uma variável alfanumérica com n espaços	+			
SPC(n)	Pula n espaços em uma instrução PRINT	+	+		
SPEED=x	Altera a freqüência de saída dos caracteres		+		
SQR(x)	Fornece o valor da raiz quadrada de x	+	+	+	+
STOP	Termina a execução de um programa	+	+	+	+
STORE variável	Grava uma matriz em fita cassete		+		
STR$(x)	Fornece uma representação alfanumérica do valor de x	+	+	+	+
STRING$(n,m)	Fornece uma var alfanum de n caract de código ASCII m	+		+	
STRING$(n,x$)	Idem anterior apenas com n caracteres do 1.º de x$	+			
SWAP variável1,variável2	Troca os valores de duas variáveis entre si	+			
SYSTEM	Retorna ao sistema operacional	+		+	
TAB(n)	Posiciona na tela para a posição n	+	+		
TAN(x)	Fornece o valor da tangente de x, x em radianos	+	+	+	+
TEXT	Retorna a tela para modo texto depois de gráfico		+		
TIME$	Fornece data e hora	+		+	
TRACE	Mostra o número da linha que está sendo executada		+		
TROFF	Desliga a marcação das linhas executadas	+		+	
TRON	Liga a marcação das linhas executadas	+		+	
UNLOCK "nomearquivo"	Desbloqueia um arquivo em disco		+		
UNPLOT x,y	Limpa o ponto da tela x,y mostrado em PLOTx,y				+
USR x	Vai para sub-rotina em linguagem de máquina		+	+	+
USRn (x)	Chama a sub-rotina indicada em linguagem de máquina	+			
VAL (x$)	Dá o valor numérico da variável x$	+	+	+	+
VARPTR(variável)	Dá o endereço onde estão nome, valor e ind de uma var			+	
VLIN x1,x2,AT y	Desenha linha vertical na tela em baixa resolução		+		
VTAB x	Posiciona o cursor na linha x da coluna atual		+		
WAIT port,n,m	Suspende a execução do programa enquanto monitora a porta	+	+		
WEND	Encerra um WHILE expressão	+			
WHILE expressão	Executa instruções de um loop enquanto expr for verdadeira	+			
WIDTH tamanho	Define o tamanho de saída da linha em n.º de caracteres	+			
WRITE "nomearquivo"	Dá o nome do arquivo em disco para as próximas saídas		+		
WRITE lista de expressões	Dá saída de dados na tela	+			
WRITE #arquivo,lista de expr	Grava dados em um arquivo seqüencial	+			
XDRAW expressão AT x,y	Desenha uma forma gráfica em alta resolução		+		