A edição autorizada

USANDO MS-DOS® QBasic™

Michael Halvorson
David Rygmyr

- **Comece a usar o novo ambiente de programação da versão 5 do MS-DOS**
- **Aprenda técnicas de programação fundamentais e avançadas**
- **Escreva programas modernos e estruturados em Basic**

Microsoft PRESS®

Editora Campus

USANDO MS-DOS® QBasic™

Sérgio Coelho Cardoso

A todos os nossos amigos que
trabalham pesado na Microsoft Press.

Sérgio Coelho Cardoso

USANDO MS-DOS® QBasic™

Michael Halvorson
David Rygmyr

- **Comece a usar o novo ambiente de programação da versão 5 do MS-DOS**
- **Aprenda técnicas de programação fundamentais e avançadas**
- **Escreva programas modernos e estruturados em Basic**

Tradução
EDIDATA
Publicações de Informática Ltda.

Consultor Editorial
Fernando B. Ximenes
Flow Informática
Uma empresa associada à KPMG

Editora Campus

Do original:
Running MS-DOS QBasic
Tradução autorizada do idioma inglês da edição publicada por Microsoft Press

Capa:
Otavio Studart

Copy-desk:
Kátia Regina Alves Rosa da Silva

Composição:
Edidata

Revisão:
Nair Dametto
Luiz Antonio Oliveira da Silva

Índice Remissivo:
Edidata

Projeto Gráfico:
Editora Campus Ltda.
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Sete de Setembro, 111 – 16º andar
Telefone: (021)221-5340 FAX (021)252-2904
20169-900 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO

ISBN 85-7001-699-9
(Edição original: ISBN 1-55615-340-6, Microsoft Press, a division of Microsoft Corporation, Redmond, Washington, USA.)

Ficha Catalográfica
CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ

H184u

Halvorson, Michael
Usando MS-DOS QBASIC / Michael Halvorson, David Rygmyr; tradução de Edidata Publicações de Informática. — Rio de Janeiro: Campus, 1992.

Tradução de: Running MS-DOS QBasic
ISBN 85-7001-699-9

1. BASIC (Linguagem de programação de computador). 2. MS-DOS (Sistema operacional). 3. Programação (Computadores). I. Rygmyr, David. II. Título.

CDD — 001.6424
001.6425
91-0571 CDU — 800-92BASIC
681.32

94 95 96 97 98 5 4 3 2 1

Agradecimentos

Vários profissionais habilitados na Microsoft Press trabalharam intensamente na edição americana. Somos gratos a Darcie Furlan pelo seu eficiente projeto; Becky Geisler-Johnson e Roger Collier pelas incríveis ilustrações; Debbie Kem e Judith Bloch pelo rápido processamento de texto e codificação; Peggy Herman pela habilidosa coordenação do layout e arte-final; Shawn Peck e Deborah Long pela cuidadosa revisão das provas; e Carolyn Magruder pela ótima composição. Somos especialmente gratos ao editor de projeto Erin O'Connor, por suas brilhantes aptidões editoriais e gerenciais, e ao editor técnico Dail Magee, Jr., por criar as imagens de tela e perseguir incansavelmente a perfeição.

Michael Halvorson
David Rygmyr
Novembro, 1990

Introdução

Há dez anos, em agosto de 1981, a IBM lançou o Computador Pessoal IBM original. O modelo PC básico, vendido por US$ 1.565, possuía um microprocessador 8088, 16 KB de memória principal, e exigia que você fornecesse um gravador comum para armazenamento de dados e uma televisão para ver pedidos e resultados. Esse sistema inicial parece ridículo para os padrões atuais, mas uma tradição que surgiu com o primeiro IBM PC permanece intacta: os IBM PCs e seus compatíveis são sempre vendidos com uma versão do MS-DOS e uma versão da linguagem de programação Basic.

Este livro o ajudará a fazer parte da estória bem-sucedida do MS-DOS e do Basic, e a juntar-se aos milhões de usuários de computador que usam o Basic para tornar seus computadores úteis e aumentar sua produtividade diária. Alguns usam o Basic para criar aplicações do MS-DOS visando lucros — há uma grande demanda de profissionais habilitados em Basic —, enquanto outros escrevem programas em Basic apenas por diversão.

A versão do Basic que lhe ensinaremos a usar é o MS-DOS QBasic, o novo e poderoso Basic que é entregue grátis com a versão 5 do MS-DOS e com o MS-DOS 5 Retail Upgrade Package. Se você já possui o MS-DOS 5 instalado no seu computador, terá tudo o que precisa para iniciar — poderá escrever seu primeiro programa em QBasic ainda hoje.

Este Livro É para Você?

Este livro foi escrito na forma de uma introdução à moderna programação QBasic para o iniciante, aquele que não possui qualquer experiência anterior em programação. Se já ficou entusiasmado com a programação, mas ainda não sabe como iniciar, este livro é para você. Apresentamos cada conceito importante em programação por sua vez — começando do *início* — e reforçamos as lições no decorrer do livro com sessões práticas, perguntas e exercícios de programação. Ao final do livro, mostramos as respostas às perguntas e exercícios.

A programação pode ser um trabalho duro, mas os seus esforços serão recompensados. Além de aprender sobre as bases do QBasic, você criará aplicações úteis e interessantes para o MS-DOS — programas que poderão ser colocados em funcionamento a cada dia. Aqui estão algumas aplicações que você construirá neste livro:

- Um programa que calcula o preço de um item, incluindo o imposto.
- Um programa de venda de bagatelas por televisão.
- Um programa simples para orçamento mensal.
- Um programa que rola dados eletrônicos.
- Um programa que produz efeitos sonoros interessantes.
- Um programa que mostra as estrofes e o coro da canção tradicional americana "Clementine".
- Um programa que acompanha a informação mensal de vendas para um distribuidor de bebidas.
- Um programa de banco de dados de música que você pode modificar para utilizar qualquer tipo de informação de banco de dados.
- Um programa que simula a entrada no espaço.

No todo, este livro contém mais de 130 programas completos que você poderá digitar e rodar — muita escolha para praticar.

Você Já Usou o Basic Antes?

Se você já usou alguma outra versão do Basic, como o GW-BASIC ou o BASICA, este livro salientará os benefícios da linguagem QBasic, moderna e melhorada, e lhe mostrará como você pode converter seus programas Basic existentes para que rodem sob o QBasic. O poderoso ambiente de edição do QBasic, as facilidades de ajuda em linha e o aperfeiçoamento da linguagem melhorarão a qualidade dos seus programas existentes e lhe ajudarão a construir novos programas ainda mais rápido do que antes. Se você já conhece todos os fundamentos da linguagem Basic, recomendamos que salte os Capítulos de 1 a 6 e comece convictamente no Capítulo 7. Também recomendamos que leia o Apêndice D, que resume as diferenças entre o QBasic e o GW-BASIC, oferecendo dicas para conversão dos seus programas do GW-BASIC para o QBasic.

Como Este Livro É Organizado

Se você está começando agora, recomendamos que leia os tópicos de programação na seqüência em que o livro os apresenta. A capacidade do Basic em executar algumas tarefas impressionantes baseia-se em fundamentos simples, de modo que nenhum detalhe inicial deve ser deixado de lado. Os Capítulos de 1 a 6 apresentam os primeiros conceitos importantes da programação Basic — projeto de programas,

uso do ambiente de programação do QBasic, entrada de instruções, trabalho com variáveis, preparação de estruturas de decisão e uso de laços.

Os Capítulos de 7 a 12 baseiam-se nesses fundamentos à medida que apresentam práticas mais sofisticadas — uso de subprogramas e funções para organizar seus programas em módulos, vetores para administrar dados, cadeias para processar texto e arquivos para armazenar informações. Você aprenderá a criar listagens em seus programas, usar cores e gráficos para tornar as coisas mais atraentes e usar o som para impressionar seus amigos. Finalmente, você aprenderá práticas de depuração que facilitarão a dolorosa tarefa de consertar erros ocasionais.

Usando MS-DOS QBasic termina com cinco apêndices. O Apêndice A resume os comandos de mouse e teclado no ambiente de programação do QBasic e analisa as opções de adaptação e partida disponíveis. Recomendamos que consulte o Apêndice A se ainda não se acostumou com uma aplicação controlada por menus e precisa rever operações gerais, como selecionar texto e usar um quadro de diálogo. O Apêndice B lista os conjuntos de caracteres ASCII e estendido da IBM que, juntos, documentam cada caracter que você pode mostrar usando o QBasic. O Apêndice C lista o conjunto completo de comandos e funções do QBasic. O Apêndice D descreve a conversão de programas do GW-BASIC para que rodem no QBasic — leitura recomendada se você já possui experiência anterior com o Basic. O Apêndice E mostra respostas completas para todas as perguntas e exercícios de programação deste livro.

O Futuro do Basic

Aprender a programar em QBasic exigirá alguma paciência, mas tenha certeza de que o seu investimento será recompensado por muitos anos. Além do QBasic, a Microsoft Corporation vende e dá suporte a três produtos Basic poderosos para programadores profissionais e eventuais: Microsoft QuickBasic para o Apple Macintosh, Microsoft QuickBasic para o IBM PC e Microsoft Basic Professional Development System. Se você precisar de algo além do QBasic, haverá versões avançadas que poderão ser utilizadas.

A Microsoft inclui uma linguagem de macro tipo Basic chamada *WordBasic* nas suas aplicações para processamento de texto Microsoft Word for Windows e Microsoft Word for OS/2. Se você já possui um desses produtos, poderá colocar suas novas habilidades no Basic em prática imediatamente.

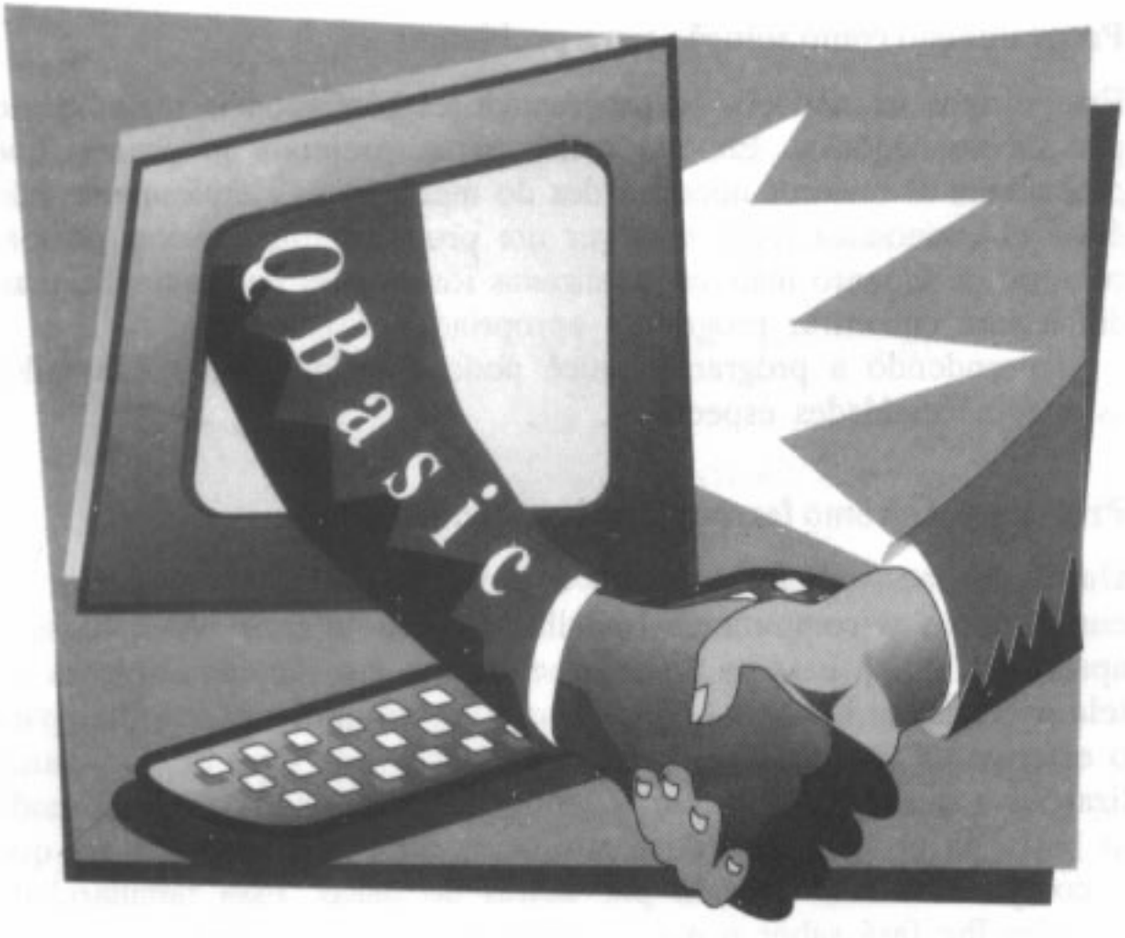
A Microsoft planeja usar um subconjunto do Basic como linguagem unificadora dentro de um ou mais dos seus ambientes operacionais.

Isso significa que você será capaz de controlar, com o Basic, a operação geral do seu computador, fazendo com que as aplicações trabalhem em conjunto, como uma equipe de jogadores, em vez de lutarem entre si como oponentes. O QBasic, agora, permitirá que você escreva programas úteis hoje mesmo, preparando-se para o futuro da computação.

Sumário

CAPÍTULO 1

Introdução à Programação e ao QBasic



POR QUE AS PESSOAS USAM COMPUTADORES?

Um conceito é simplesmente uma ferramenta. Sua finalidade é servi-lo — ajudá-lo a realizar uma tarefa rápida e eficientemente. Você não precisa estar convencido disso. O próprio fato de ter comprado este livro significa que reconhece que um computador pode ser-lhe útil. Mas você poderia se perguntar por que desejaria escrever programas de computador por conta própria.

Por Que *Eu* Devo Aprender a Programar?

Afinal, hoje você pode escolher entre milhares de programas já prontos — programas que o guiam por um labirinto de preparação de devolução de imposto tão facilmente quanto o desafiam por um labirinto de um jogo de *aventura*. Frente a uma variedade tão grande de software comercial, você pode acreditar que o mercado está recheado — que existe um programa adequado para cada necessidade.

Mas isso não é verdade.

Programação como solução para problemas

Estereótipos irresistíveis, os programadores profissionais também são pessoas de negócios. Em sua maior parte, projetam programas que preenchem as maiores necessidades do mercado — simplesmente não é viável economicamente escrever um programa que poucas pessoas comprarão. Quanto mais especializadas forem suas necessidades, mais difícil será encontrar programas apropriados já prontos.

Aprendendo a programar, você pode criar programas adaptados às suas necessidades específicas.

Programação como ferramenta de aprendizado

Um outro motivo para aprender a programar é que o processo lhe ensina como o computador trabalha em seu interior: você passa a apreciar a lógica passo a passo, que resulta em rápidas exibições na tela e elegantes listagens. Assim como um arquiteto pode olhar para o exterior de um prédio e ver a organização das vigas de aço, canalizações e estruturas de suporte dentro dele, uma pessoa que aprende as bases da programação do computador pode ter uma idéia do que o computador está fazendo por detrás do palco. Essa familiaridade também lhe fará saber o que o computador pode e não pode fazer. Por este motivo, um curso introdutório sobre programação de computador é normalmente a primeira aula exigida no estudo de computadores no segundo grau ou na faculdade. Além disso, aprendendo sobre as bases, você estará aprendendo a falar a linguagem dos

profissionais. A familiaridade com termos e conceito de programação o ajudará a expressar-se com mais clareza.

Programação como recreação

O último motivo para escrever seus próprios programas é que isso pode ser positivamente *divertido*. Para programadores iniciantes e experientes, a programação do computador pode ser agradável e recompensadora.

Como vimos, existem vários motivos para você aprender a programar. Agora vamos falar um pouco sobre terminologia. Afinal, o que *é* um programa de computador?

Programação Personalizada

O que acontece quando as suas necessidades são muito específicas? Talvez você precise apenas de alguns progrramas organizacionais para simplificar sua vida. O QBasic lhe permite criar programas simples ou muito detalhados, conforme a necessidade. Aqui estão algumas pessoas que poderiam se beneficiar escrevendo seus próprios programas em QBasic:

- Um entusiasta por baseball que deseja registrar estatísticas de time e jogador e obter instantaneamente itens de interesse especial
- Um usuário dedicado a video-games que deseja construir um jogo de *aventura* melhor
- Um gerente ou proprietário de um pequeno comércio que deseja registrar compromissos e datas importantes e ser notificado automaticamente sobre os eventos pendentes
- Um estudante ou cientista que precisa realizar cálculos matemáticos especializados
- Um chefe de família que deseja calcular as finanças domésticas
- Um médico que deseja acompanhar os registros de pacientes, receitas e contas, mas que deseja ir além das limitações das fichas tradicionais

O QUE É UM PROGRAMA DE COMPUTADOR?

Para realizar um trabalho útil, um computador deve ter duas coisas: *hardware* e *software*.

- Hardware são as partes físicas do computador, que você pode ver e tocar, como o teclado e a tela.
- Software é um pouco mais abstrato: é a inteligência que controla o hardware e permite que você realmente use o computador.

Você não pode usar um sem o outro: hardware sem software é como um toca-discos sem discos.

Um *programa de computador* é software. É simplesmente um conjunto de instruções que faz com que o computador realize alguma tarefa útil, como um cálculo ou um processamento de texto. Um programa pode ter uma única instrução ou até centenas ou milhares de instruções individuais.

Computadores: Heróis ou Vilões?

O cinema e a televisão fizeram um bom trabalho mostrando-nos como os computadores podem assumir cargos, tornar a vida mais difícil e controlar o mundo. E esse não é exatamente o tipo de ferramenta que você deseja levar para a sua casa ou para o trabalho, muito menos usar ou programar!

Apesar desta má impressão, os computadores pessoais entraram em firmas e lares de milhões de pessoas, permitindo a realização de um trabalho muito útil, que facilita suas vidas, tornando-as mais produtivas. As pessoas estão começando a mudar de idéia a respeito desses estranhos dispositivos eletrônicos. O que antes engolia e cuspia milhares de cartões perfurados em código e ocupava a maior parte de uma sala da sua firma agora é colocado sobre uma mesa e apresenta imagens úteis e significativas. Os computadores se tornaram parte integral do comércio, comunicações, entretenimento e pesquisa científica. Fáceis de serem usados e cada vez mais poderosos, os computadores pessoais de hoje são as ferramentas úteis que uma indústria recente um dia sonhou que seriam.

COMO POSSO ESCREVER UM PROGRAMA DE COMPUTADOR?

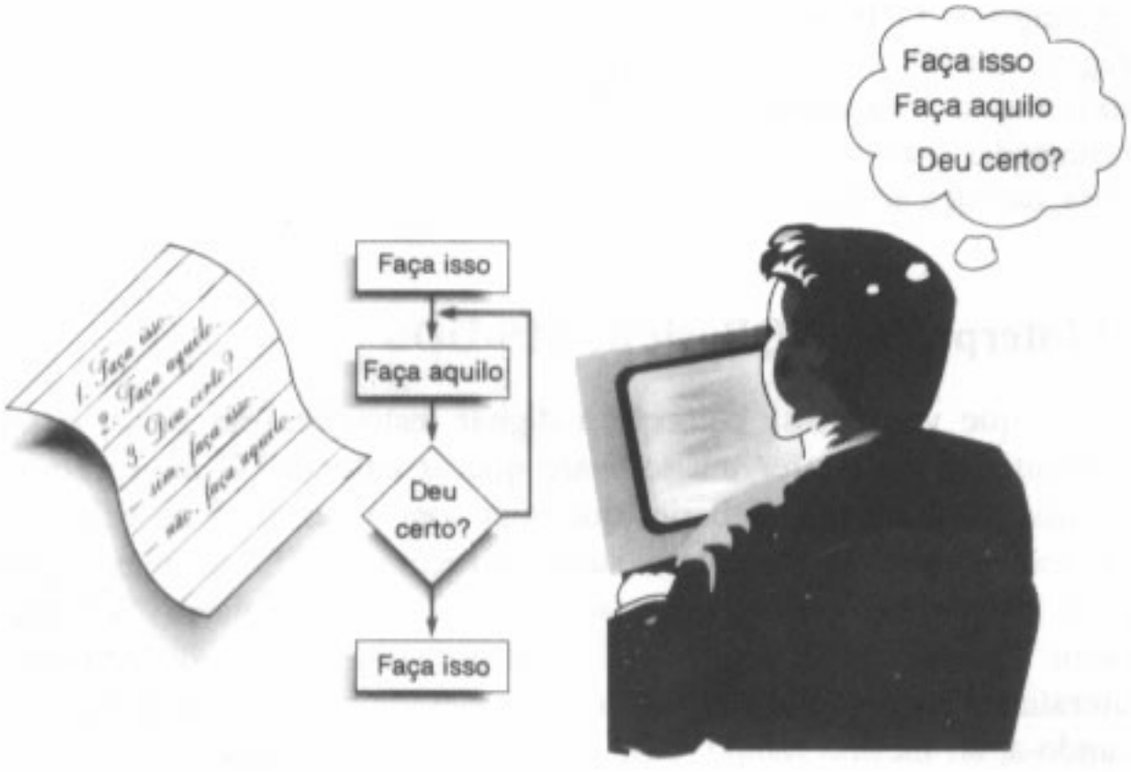
Preparação e organização são os primeiros passos importantes no processo de programação. Duas perguntas o ajudarão a organizar-se melhor:

1. O que desejo que este programa faça?

2. Que passos o programa deve tomar para isso?

O conjunto de passos gerais que você imagina é chamado *algoritmo*. Um algoritmo não é um programa de computador por si só; é apenas uma coleção de notas ordenadas descrevendo o que o seu programa fará a cada passo do caminho. Na verdade, se você já seguiu uma receita, já terá usado um algoritmo. Fazer um bolo, por exemplo, significa seguir um processo passo a passo que produz um resultado útil (e gostoso!).

Como um algoritmo é uma ferramenta para solução de problemas, projetado para ajudá-lo a obter o programa escrito, não existe um único meio correto de prepará-lo. Alguns programadores escrevem um algoritmo como uma lista de passos ordenados em uma folha de papel, outros mostram os passos graficamente em um diagrama, chamado fluxograma, e ainda outros (que vêm trabalhando com isso por algum tempo) simplesmente organizam os passos de programação em suas cabeças.



Uma lista | Um fluxograma | Um programador experiente

Qualquer que seja o método escolhido, comece a pensar no seu programa em termos gerais bem antes de sentar-se em frente ao computador.

A Linguagem de Programação Basic

Parte do motivo para a ênfase no projeto inicial para solução de problemas é que o computador não conhece muitas palavras: você deve expressar-se em uma linguagem que possa ser entendida.

O Basic é uma das muitas linguagens de programação disponíveis para computadores pessoais. Assim como as linguagens faladas, cada linguagem de computador tem o seu próprio "vocabulário" e o seu próprio conjunto de regras. Além do mais, cada linguagem possui pontos fortes e fracos. Algumas linguagens são mais eficientes para fazer certas tarefas do que outras; algumas são mais fáceis de trabalhar do que outras. Considerando todos os fatores, o Basic é uma linguagem excelente para a programação do computador, pois é poderosa, flexível e — acima de tudo — segue uma sintaxe parecida com a da linguagem falada e é fácil de usar.

Aprender a "falar" em Basic, ou em qualquer outra linguagem de programação para computador, é como aprender a falar uma linguagem estrangeira. O Basic tem seu próprio vocabulário e um conjunto de regras que devem ser seguidas quando usamos esse vocabulário. Felizmente, o vocabulário inteiro do Basic possui menos de 200 palavras (chamadas *palavras-chave*) e as regras (chamadas regras de *sintaxe*) para uso dessas palavras-chave são relativamente simples.

O Interpretador QBasic do MS-DOS

Antes que você possa começar a digitar instruções em Basic, o seu computador deverá ter um software que as entenda. O interpretador QBasic do MS-DOS (QBasic) que vem com a versão 5 do MS-DOS faz exatamente isso — lê as suas instruções em Basic e as traduz para uma forma que o computador possa usar. A palavra "interpretador" é bastante apropriada: O interpretador QBasic do MS-DOS literalmente interpreta cada sentença à medida que você a digita, checando-a, ao mesmo tempo, para ter certeza de que você não cometeu algum erro de digitação, usou uma palavra-chave de forma errada ou quebrou alguma regra de sintaxe.

O QBasic também oferece um ambiente de programação completo para o seu trabalho. O QBasic é um único pacote que lhe permite escrever e rodar programas imediatamente. Usando-o, você poderá salvar seus programas no disco, imprimi-los na impressora e obter ajuda quando precisar dela. O ambiente QBasic contém ainda muitas características tipo processador de texto, que facilitam a escrita e a alteração dos seus programas.

As Origens do QBasic

QBasic é um superconjunto moderno da linguagem BASIC original, criada por John G. Kemeny e Thomas E. Kurtz em 1963 e 1964 no Dartmouth College, e das outras versões do BASIC para o Computador Pessoal IBM e compatíveis (como BASICA e GW-BASIC). QBasic é não apenas um superconjunto dessas versões mais antigas do BASIC — ou seja, ele entende todas as palavras-chave e a sintaxe das versões mais antigas —, mas também oferece palavras-chave e características de outras linguagens de computador populares, tornando-o um sistema completo para a escrita dos seus próprios programas.

Uma versão mais avançada do Basic, o Microsoft QuickBasic Compiler, também é comercializada pela Microsoft. Falaremos um pouco sobre este produto de linguagem no Capítulo 13.

COMO ESTE LIVRO ME AJUDARÁ A APRENDER SOBRE O QBASIC?

Além de apresentá-lo à linguagem de programação de computador QBasic, *Usando MS-DOS QBasic* lhe permite colocar em prática suas novas habilidades. Ao aprender, você escreverá programas que mostram caracteres na tela, lêem caracteres digitados pelo teclado, trabalham com palavras e números, e armazenam informações em um disco, para que novamente sejam recuperadas. Você criará até mesmo programas que usam gráficos e som. Após ter estudado neste livro, você poderá criar programas úteis em QBasic, fazendo com que o seu computador — sua ferramenta — trabalhe por você.

NOTA: A partir do Capítulo 2, cada capítulo deste livro termina com perguntas de revisão e exercícios. Gaste algum tempo preparando-os — eles são uma boa oportunidade para revisar o que acabou de aprender e ter certeza de que entendeu tudo, antes de ir para o capítulo seguinte. Você verá as respostas às perguntas e exercícios no Apêndice E.

QBASIC FAZ PARTE DO MS-DOS 5

O QBasic vem grátis com o MS-DOS versão 5 e Retail Upgrade Package da versão 5. Se você possui um desses produtos (ou um produto compatível do fabricante do seu computador), também possui o QBasic. Antes de começar, esteja certo de que o MS-DOS versão 5

está instalado no seu computador (se tiver um disco rígido) ou de que você possui os discos de trabalho da versão 5 (se tiver um sistema com dois disquetes e nenhum disco rígido). O próximo capítulo lhe dará instruções passo a passo para iniciar o QBasic e escrever seu primeiro programa!

NOTA: Presume-se que você possui algum conhecimento elementar sobre o sistema operacional MS-DOS (formatação de discos, mudança de diretórios, definição de caminhos etc.). Se precisar dar uma olhada nos passos básicos, veja a documentação do DOS que veio com o seu computador ou consulte Usando MS-DOS, *de Van Wolverton (Editora Campus, 1991).*

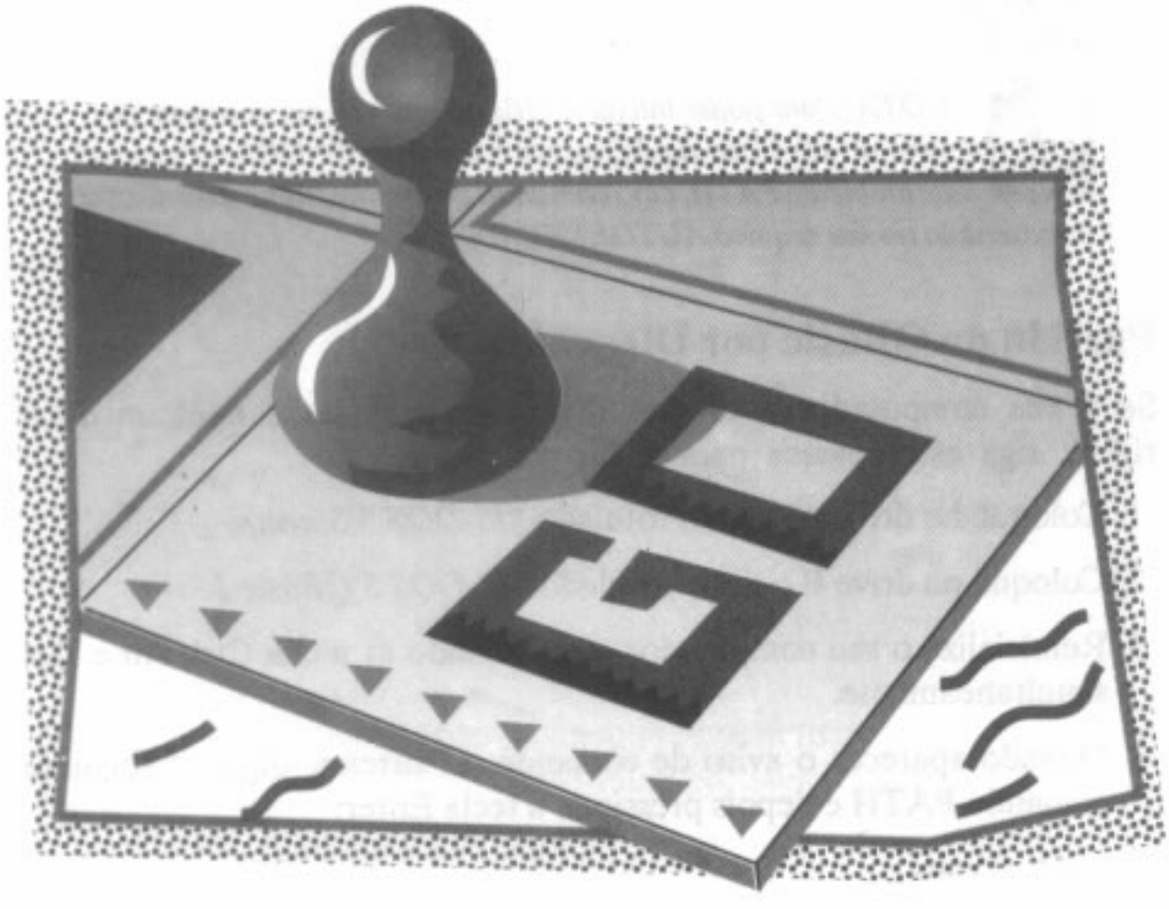
Apêndice A: Sua Referência para o QBasic

Nos capítulos seguintes, pressupomos que você esteja acostumado com os ambientes baseados em menu, como o MS-DOS EDIT (o editor de texto da versão 5 do MS-DOS) ou com o Microsoft Works (uma aplicação comercial de múltipla finalidade). Se você precisar rever o uso do mouse ou do teclado para *navegar* em um ambiente controlado por menu, ou se precisar de informações específicas sobre a adaptação do ambiente QBasic, veja o Apêndice A, "Uso de Menus e Opções do QBasic". O Apêndice A contém informações sobre estes tópicos:

- Seleção de menus e comandos
- Uso de quadros de diálogo
- Uso dos comandos de teclado do QBasic
- Seleção de texto
- Obtenção de ajuda em linha
- Impressão
- Mudança de cores na tela
- Uso de opções de partida

CAPÍTULO 2

Primeiros Passos com o QBasic



Agora que você está preparado para o básico, vamos direto ao assunto e começar aprendendo a programar.

PARTIDA NO QBASIC

Para começar a programar, você deve iniciar o QBasic. Isso pode ser feito de duas formas, dependendo de você estar rodando o QBasic por um disco rígido ou por disquete.

NOTA: Em todo este livro, o texto que você (ou os seus usuários) deverá entrar pelo teclado aparece em `tipo diferente.`

Partida do QBasic por um Disco Rígido

Se você tiver a versão 5 do MS-DOS instalada em um disco rígido, o programa QBasic e arquivos de suporte deverão estar localizados no diretório do DOS no drive C. Para iniciar o QBasic, digite os seguintes comandos, pressionando a tecla Enter após cada um:

```
c:
cd \dos
qbasic
```

NOTA: Para poder iniciar o QBasic estando em qualquer diretório do seu disco rígido, inclua o diretório C:\DOS na variável de ambiente PATH, cujo comando do mesmo nome deverá estar armazenado no seu arquivo AUTOEXEC.BAT.

Partida do QBasic por Disquetes

Se o seu computador tiver dois drives de disquete e nenhum disco rígido, siga estes passos para rodar o QBasic:

1. Coloque no drive A o disco rotulado *MS-DOS 5 Startup.*
2. Coloque no drive B o disco rotulado *MS-DOS 5 QBasic 1.*
3. Reinicialize o seu computador pressionando as teclas Ctrl, Alt e Del simultaneamente.
4. Quando aparecer o aviso de comando do sistema, digite o seguinte comando PATH e depois pressione a tecla Enter:

   ```
   path a:;b:
   ```

5. Inicie o QBasic digitando o seguinte comando e depois pressionando a tecla Enter:

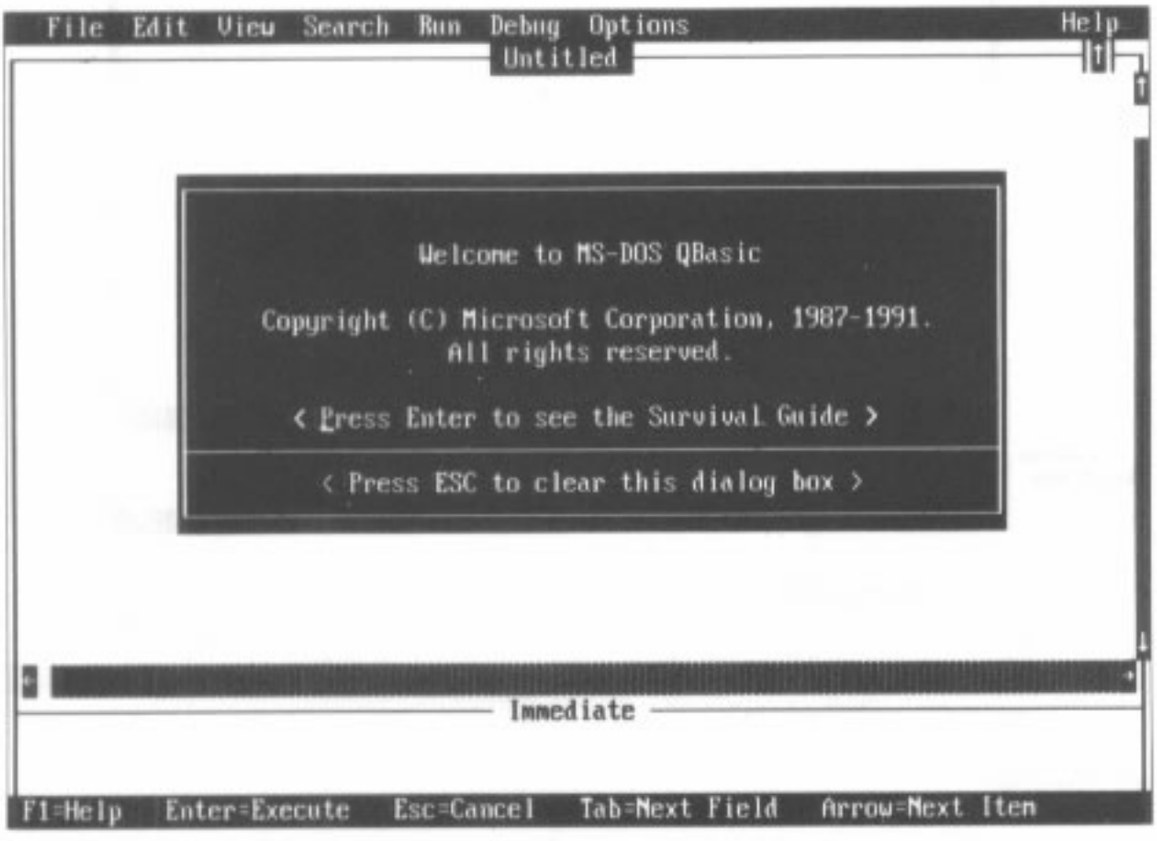
   ```
   qbasic
   ```

Você verá a luz do drive B se acender e o QBasic aparecer na tela. Quando a luz do drive se apagar, remova o disco *MS-DOS 5 Startup* do drive A e insira o disco *MS-DOS 5 QBasic 2* no mesmo drive. Rode o QBasic a partir dos dois discos até que tenha saído do programa. Depois remova o disco *MS-DOS 5 QBasic 2* do drive A e insira o disco *MS-DOS 5 Startup* no drive A.

NOTA: À medida que os seus programas se tornam maiores, você achará cada vez mais difícil rodar programas a partir de um sistema sem disco rígido. Seu maior problema ao rodar o QBasic será esgotar o espaço no disco e ter que trocar discos com muita freqüência. Se você gasta muito tempo trabalhando com o seu computador, pense na compra de um disco rígido ou na troca do seu computador por um que o tenha.

A TELA DO QBASIC

Após o QBasic ter sido carregado, sua tela deverá ficar assim:



Esta mensagem de "boas-vindas" aparece toda vez que você inicia o QBasic. A tela lhe oferece duas opções:

- Você pode pressionar Enter para ver o sistema de ajuda embutido do QBasic.

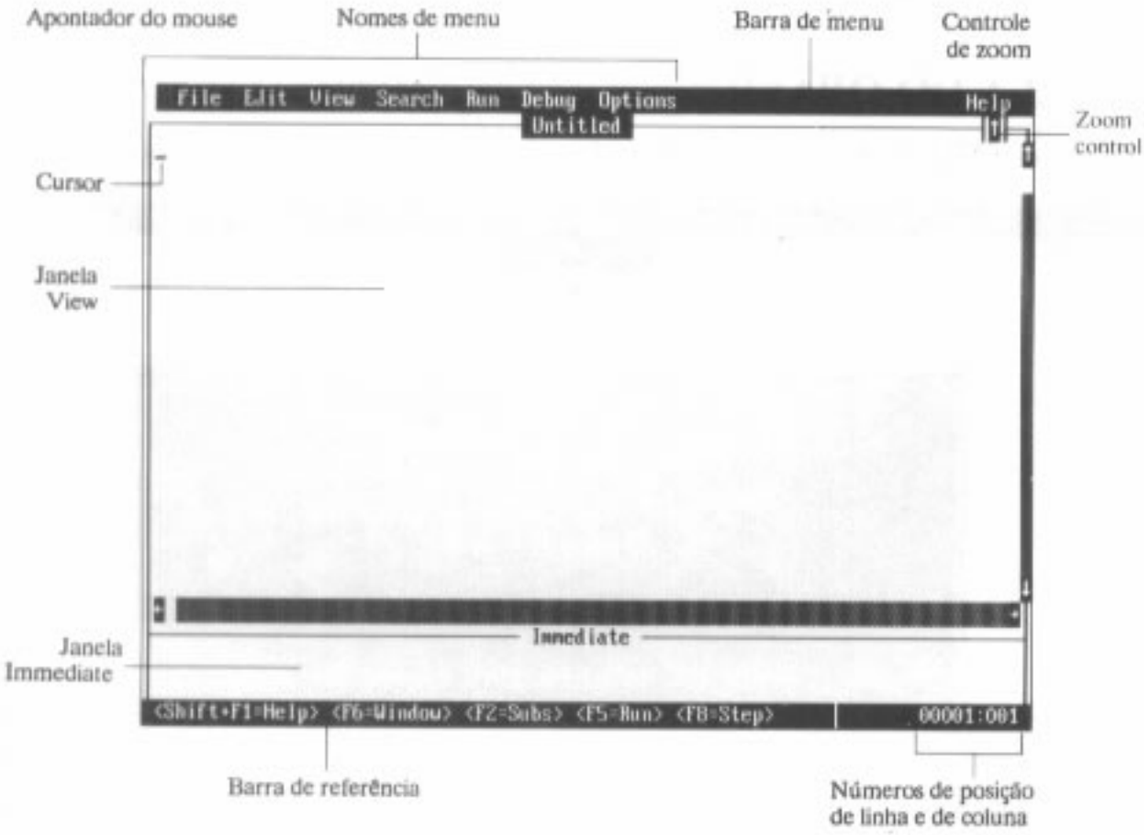
- Você pode pressionar a tecla Esc para apagar a mensagem da tela e começar a programar.

Prática: Apagando a tela para começar a programar

Como você estará programando neste capítulo, pressione Esc para limpar a mensagem da tela.

O Ambiente do QBasic

Vamos começar vendo os vários elementos da tela do QBasic, que o ajudarão a criar programas e processar texto.



O cursor

O sublinhado piscante no canto superior esquerdo da tela provavelmente será o primeiro elemento a atrair sua atenção. Ele é chamado *cursor*. O cursor indica onde o próximo caracter digitado aparecerá na tela.

O apontador do mouse

Se você tiver um mouse instalado no seu computador, um *apontador de mouse* retangular aparecerá no canto superior esquerdo da tela. O apontador do mouse lhe permite movimentar-se pelo programa rapidamente, escolhendo itens de menu com mais facilidade.

Os números de posição de linha e de coluna

No canto inferior direito da tela existem dois números separados por um sinal de dois pontos. Estes indicam o local atual do cursor. O primeiro número lhe dirá em que linha o cursor se encontra; o segundo número (após os dois pontos) lhe dirá em que coluna o cursor se encontra. Como vemos na Figura 2-1, os números indicam que o cursor está na linha 1, coluna 1.

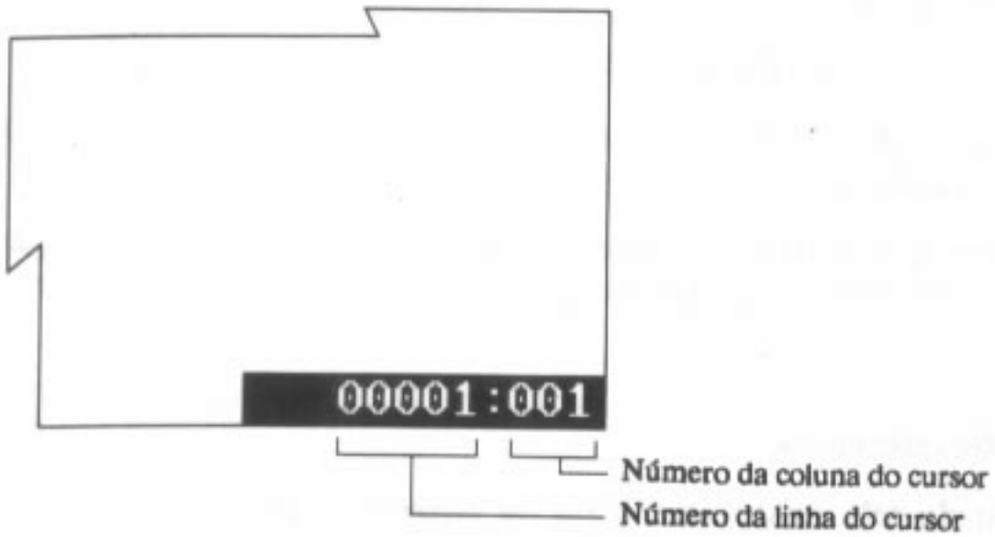


FIGURA 2-1.
Números de posição de linha e de coluna.

Prática: Alterando os números de coluna

1. Pressione a tecla de seta à direita no teclado numérico algumas vezes e veja o que acontece com o número da coluna.
2. Agora pressione a tecla de seta à esquerda até que o cursor retorne à coluna 1. (Mais tarde, quando você começar a digitar programas, poderá praticar a mudança de números de linha com as teclas de seta para cima e para baixo.)

A barra de nenu e os nomes de menu

Horizontalmente, no topo da tela, existe uma *barra de menu*. A barra de menu contém os *nomes de menu* para todos os menus drop-down do QBasic.

A janela View e a janela Immediate

O QBasic inicialmente lhe dará duas janelas para trabalhar:

- A janela superior — onde você digitará e trabalhará com seus próprios programas — é chamada *janela View*.
- A janela inferior — onde você poderá testar uma instrução de programação antes de realmente digitá-la no seu programa — é chamada *janela Immediate*.

Você pode trabalhar em apenas uma janela de cada vez, embora seja fácil trocar de janela. A janela onde você trabalha atualmente é chamada *janela ativa*. É muito fácil identificar a janela ativa:

- Procure o cursor piscante; ele sempre estará na janela ativa.
- Veja o título da janela no alto; se o título estiver destacado, a janela estará ativa.

Observe que o título da janela View permanece *Untitled* até que você dê um nome ao programa que está sendo escrito e o salve-o no disco.

A barra de referência

No fundo da tela existe uma *barra de referência*. Essa barra contém uma lista de *teclas de controle* que você pode usar na janela ativa. Essa lista de teclas de controle muda sempre que você seleciona uma nova janela ativa.

Prática: Trocando de janela com um comando da barra de referência

Observe que a tecla de função F6 é a tecla Window. Você usa F6 para trocar de janela. (Para trocar de janela com o mouse, basta mover o apontador para qualquer lugar dentro da nova janela a ser ativada e clicar o botão esquerdo.) Pressione F6 agora e veja o que acontece.

Como podemos ver na Figura 2-2, o cursor piscante agora encontra-se na janela Immediate. As barras de deslocamento horizontal e vertical desapareceram do menu View, e o destaque que estava na palavra *Untitled* na janela View agora passou para a palavra *Immediate* no topo da janela Immediate. Essas mudanças lhe dirão que a janela ativa é a janela Immediate. Se você digitasse caracteres pelo teclado agora, estes apareceriam na janela Immediate, e não na janela View. Observe que, diferente do título da janela View, o título da janela Immediate nunca muda.

Observe, também, que um conjunto diferente de teclas de controle aparece agora na barra de referência no fundo da tela. Estas são as teclas de controle da janela Immediate. Por enquanto você

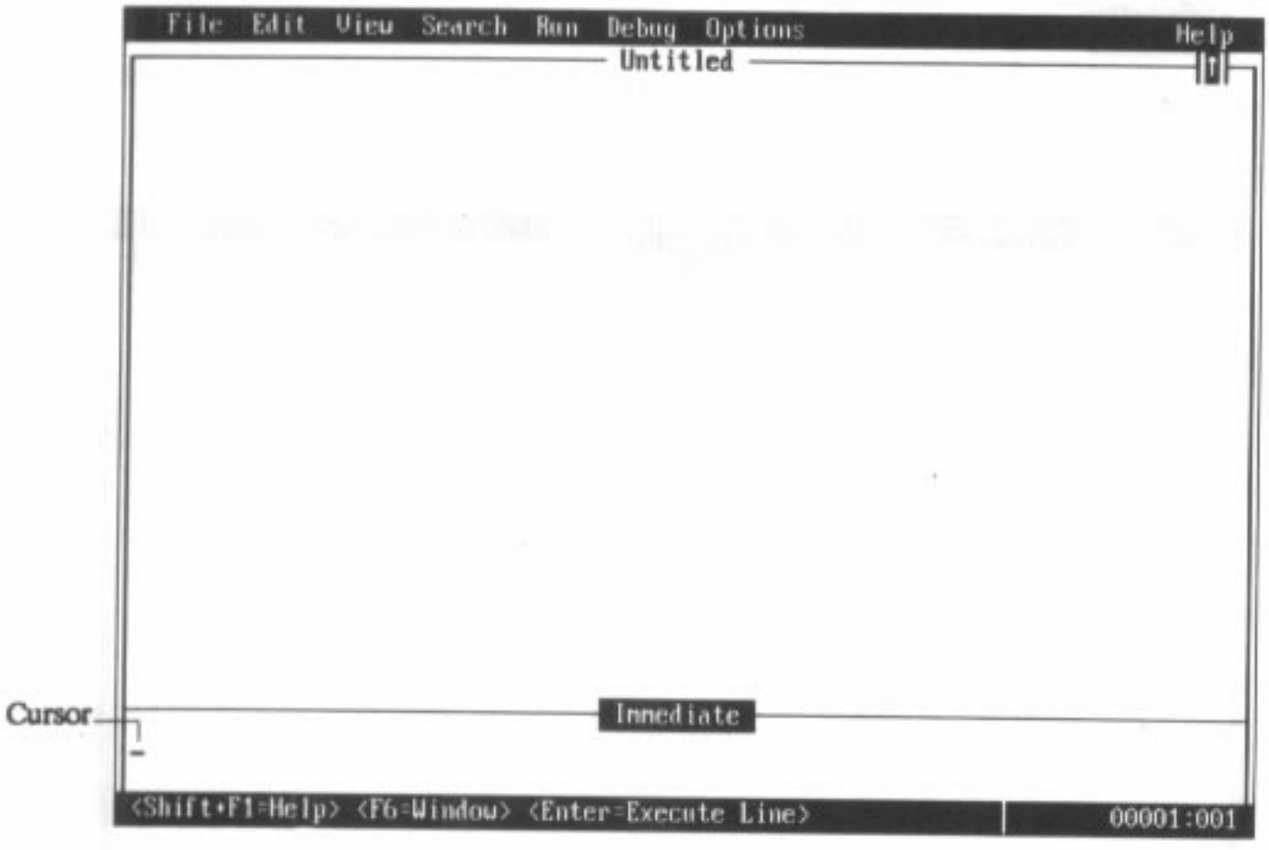


FIGURA 2-2.
Tornando a janela Immediate a janela ativa.

não usará a janela Immediate; logo, pressione F6 para ativar a janela View novamente.

O controle de zoom

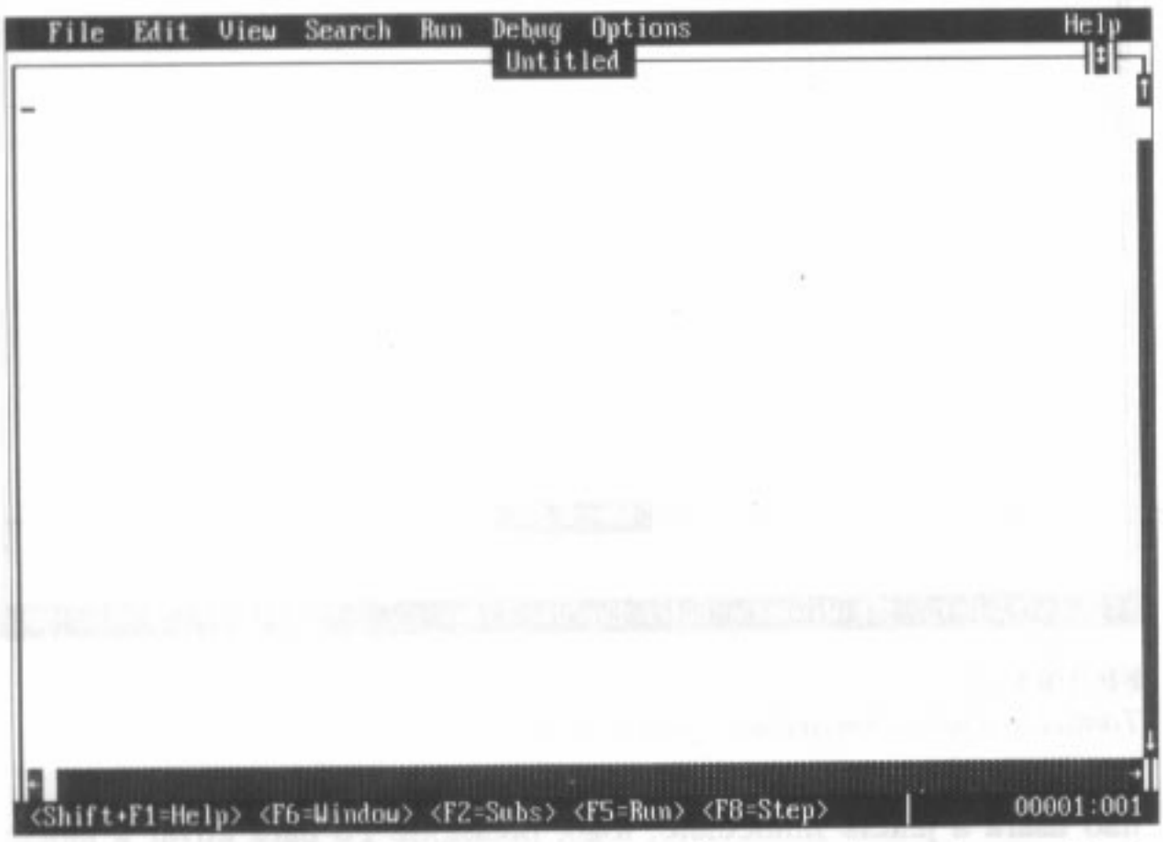
Você pode fazer com que a janela Immediate desapareça da tela. No canto superior da tela, logo abaixo do nome de menu Help, existe um retângulo vertical com uma seta apontando para cima. Este é o *controle de zoom*, que faz com que a janela View se expanda e cubra a janela Immediate.

Após usar o controle de zoom, o caracter de seta para cima muda para uma seta de duas pontas, significando que a janela View está "em zoom" ao máximo possível. Você pode usar o controle de zoom do QBasic com o mouse ou com o teclado:

- Para usar o controle de zoom com o mouse, clique no controle de zoom. Para restaurar a janela View ao seu tamanho original, clique novamente no controle de zoom.
- Para usar o controle de zoom com o teclado, mantenha pressionada a tecla Ctrl e pressione F10. Para restaurar a janela View ao seu tamanho original, pressione novamente Ctrl-F10.

Prática: Expandindo a janela View com o controle de zoom

1. Use o mouse ou o teclado para expandir a janela View com o controle de zoom. Após fazer isso, sua tela ficará assim:



2. Agora restaure a janela View ao seu tamanho normal para tornar visível a janela Immediate.

SEU PRIMEIRO PROGRAMA

Agora que você já teve alguma experiência prática com o QBasic, está pronto para digitar seu primeiro programa. Digitar um programa é muito simples. Tudo que é preciso fazer é digitar as linhas exatamente como as mostramos neste livro e pressionar Enter ao final de cada uma.

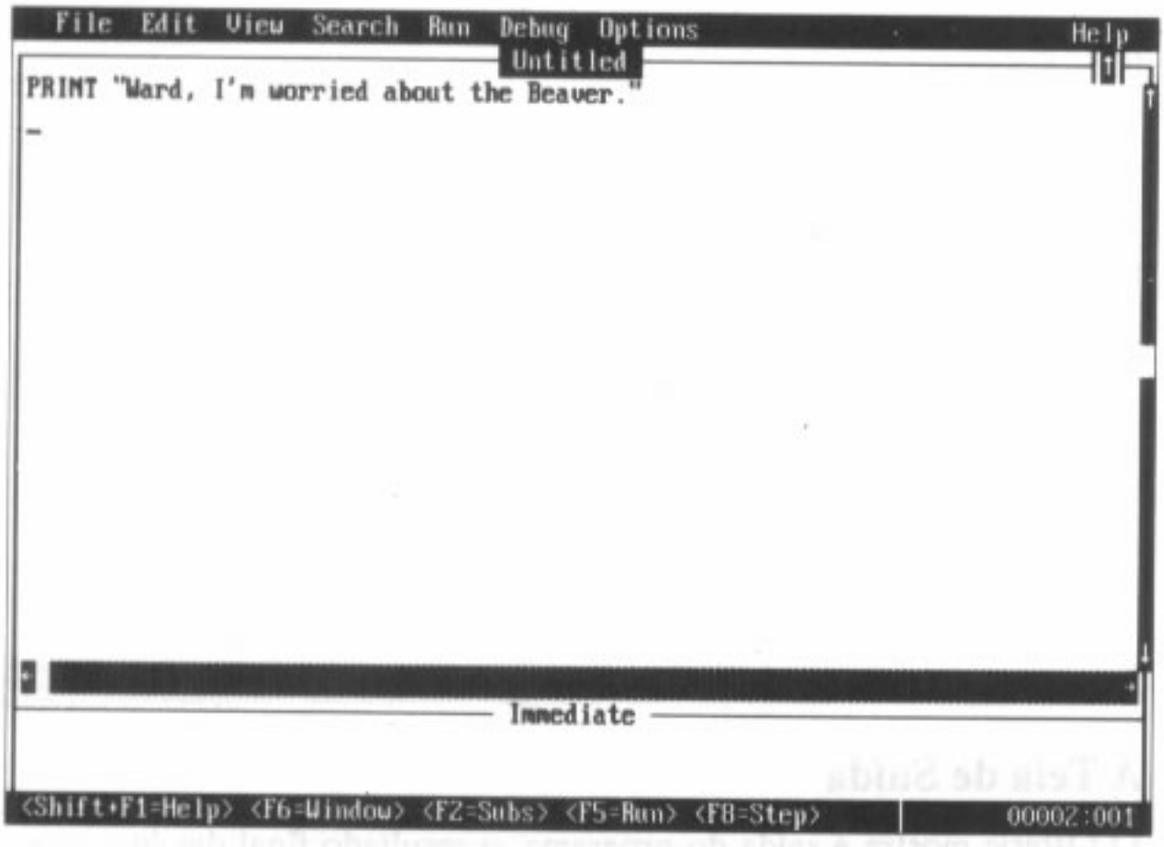
Prática: Digitando seu primeiro programa

1. Retorne o cursor à linha 1, coluna 1, e digite a seguinte linha exatamente como a mostramos — espaços, aspas duplas etc.:

```
print "Ward, I'm worried about the Beaver."
```

2. Verifique a linha cuidadosamente para ter certeza de que não digitou algo errado. Se o fez, use a tecla Backspace (retrocesso) para retornar ao erro. Conserte-o e digite o restante da linha.

3. Quando estiver certo de que não há erros, pressione Enter. Após pressionar Enter, sua tela ficará assim:



Observe que a palavra *print* encontra-se agora em letras maiúsculas, pois a palavra *print* é uma *palavra-chave* do QBasic — ou seja, ela faz parte do "vocabulário" do QBasic. O QBasic sempre converte palavras-chave para letras maiúsculas, para que se possa distingui-las com facilidade de todo o restante de uma linha.

O que você acabou de digitar, embora tendo apenas uma linha, é um programa QBasic completo.

Rodando um Programa

Para rodar um programa, escolha o comando Start pelo menu Run. Fazendo isso, você estará dizendo ao QBasic para executar as instruções que se encontram atualmente na janela View.

Prática: Rodando seu primeiro programa

Rode agora o seu programa: pressione Alt para ativar a barra de menu, escolha Run e depois, Start. (Para relembrar o uso do teclado ou do mouse para selecionar itens do menu, veja o Apêndice A, "Uso de Menus e Opções do QBasic".) Sua tela deverá ser algo assim:

```
C:\DOS>qbasic
Ward, I'm worried about the Beaver.


Press any key to continue
```

A Tela de Saída

O QBasic mostra a saída do programa, o resultado final das instruções no seu programa, na *tela de saída*. Quando você roda o programa, o QBasic passa imediatamente para a tela de saída, evitando que a saída do seu programa seja escrita sobre as instruções do programa (ou *código*) na janela View.

A tela de saída mostra a saída de qualquer programa que você tenha rodado desde a última vez em que apagou a tela. É por isso que você vê a linha de comando do DOS que usou para iniciar o QBasic. A mensagem no fundo da tela de saída, *Press any key to continue* (pressione uma tecla qualquer para continuar), aparece toda vez que um programa termina sua execução.

Prática: Retornando à janela View

Pressione qualquer tecla agora. O QBasic o retornará à janela View. Parabéns! Você escreveu e executou o seu primeiro programa em BASIC!

Alteração do Programa

Você aprendeu sobre uma palavra-chave do QBasic, PRINT, que permite que o seu programa coloque informações na tela. Agora você aprenderá outra palavra-chave do QBasic, CLS (de CLear Screen),

que permite que seu programa apague completamente a tela de saída. Você incluirá CLS ao programa que já escreveu, e com isso aprenderá a modificar um programa.

Ordenação correta das instruções do QBasic

Antes de incluir CLS ao seu programa, pense na ordem em que você deseja que o programa execute suas instruções. Em quase todos os programas que serão criados neste livro, cada instrução encontrar-se-á em uma linha separada. Ao rodar o seu programa, o QBasic começará no topo do programa e executará suas instruções uma a uma na ordem em que você as listou, até que chegue ao fim da lista.

Seu primeiro passo, portanto, é decidir onde colocar a instrução CLS no seu programa.

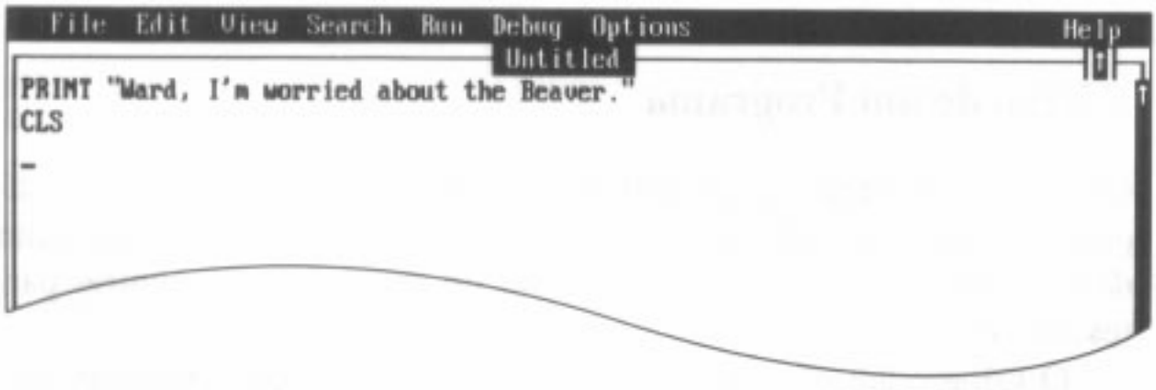
Prática: Inclusão de outra instrução ao seu programa

Neste momento, o cursor estará uma linha abaixo da letra P da sua instrução PRINT.

1. Digite o seguinte e pressione Enter:

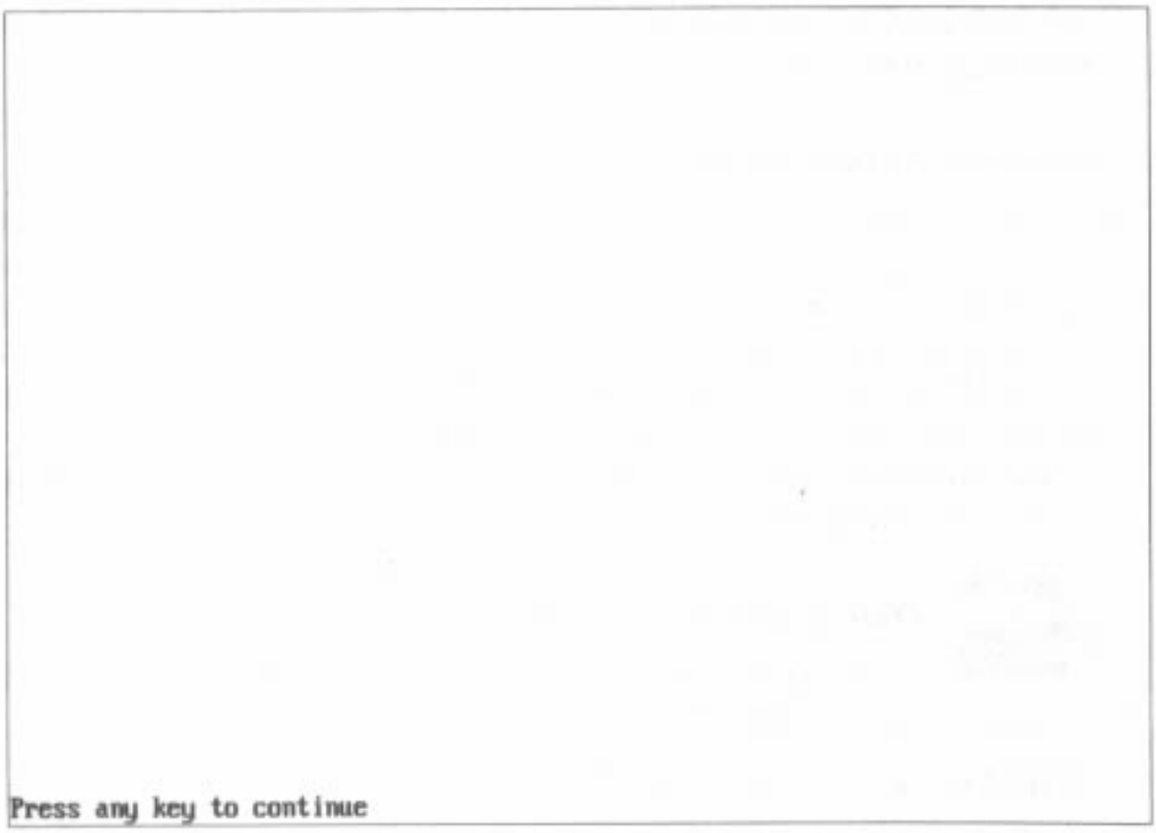
   ```
   CLS
   ```

   Sua tela deverá estar como esta:



   (Como antes, se você digitou a instrução CLS com letras minúsculas, o QBasic as converterá para maiúsculas após pressionar Enter, pois CLS é uma palavra-chave do QBasic.)

2. Rode o programa novamente: escolha Start pelo menu Run. Se você olhar atentamente, poderá ver a mensagem *Ward, I'm worried about the Beaver* piscando rapidamente na tela, mas logo depois a tela deverá ficar como a tela seguinte. Não existe muito sentido em usar uma instrução PRINT se você irá segui-la imediatamente por uma instrução CLS. Agora você pode ver por que a ordem das instruções é tão importante. Você precisa colocar a instrução CLS *antes* da instrução PRINT para que o programa se comporte como você espera.



3. Pressione qualquer tecla para retornar à janela View.

Edição de um Programa

Quando você altera um programa, realiza um processo chamado *edição*. A edição compreende a inclusão de caracteres, o apagamento de caracteres e o movimento de linhas de instruções para outras partes do seu programa.

O QBasic lhe oferece várias ferramentas de edição. Algumas dessas ferramentas estão no menu adequadamente chamado Edit; outras são teclas do seu próprio teclado. Use as práticas a seguir como visão geral. Se você achar que precisa de mais informações, veja o Apêndice A, "Uso de Menus e Opções do QBasic", que contém instruções detalhadas sobre como usar essas ferramentas.

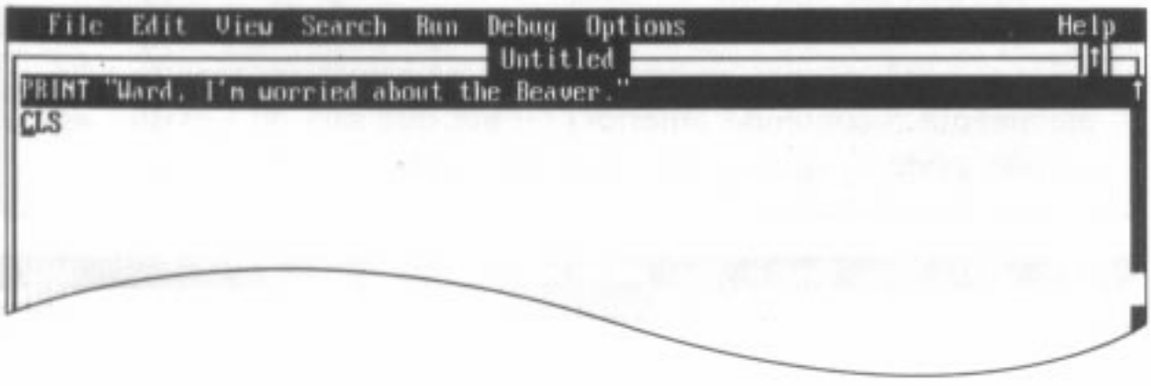
Os comandos Cut, Copy e Paste

Os comandos Cut (apagar), Copy (copiar) e Paste (inserir) lhe permitem editar seus programas de forma eficaz. Eles não apenas economizam tempo e muita digitação, mas também reduzem as chances de erros de digitação repetida.

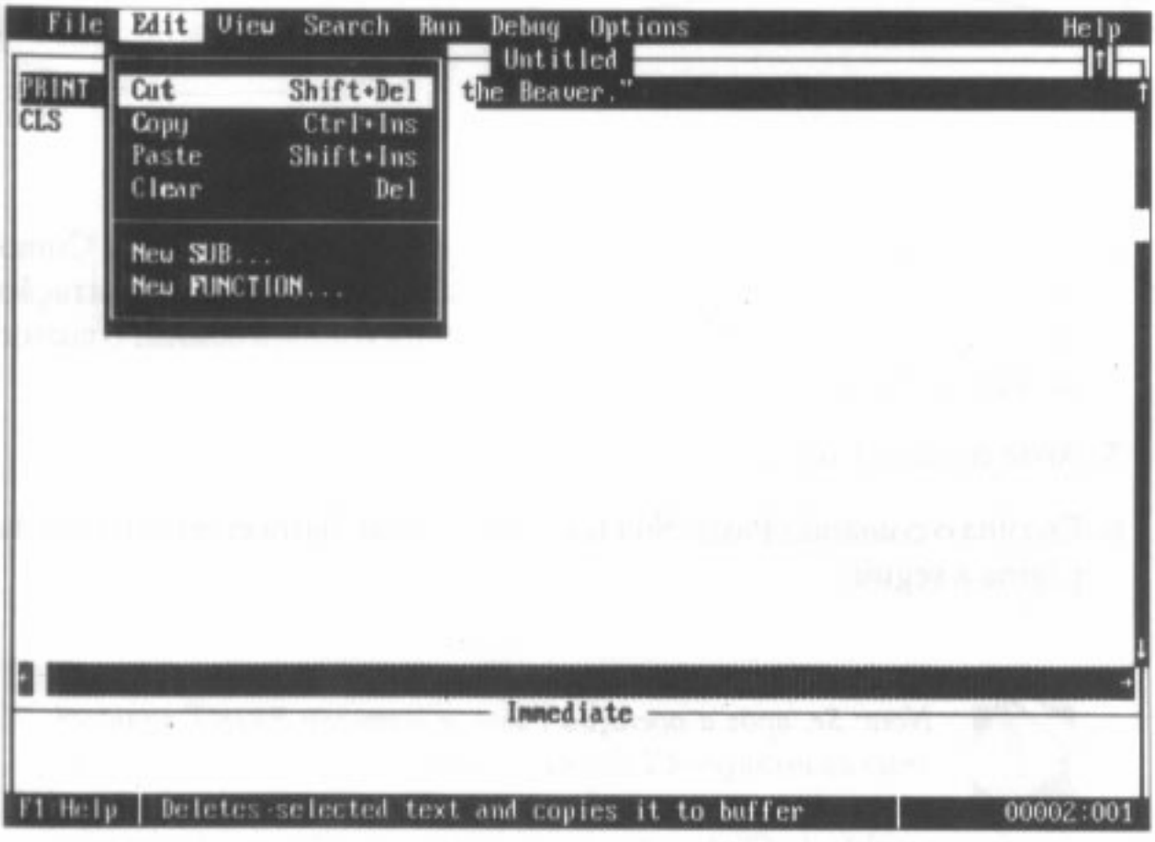
Prática: Editando um programa com os comandos Cut e Paste

Como a instrução PRINT precisa ir após a instrução CLS, você precisa mover a instrução PRINT inteira. Faça isso usando os comandos Cut e Paste no menu Edit.

1. Mova o cursor sob o P de PRINT. Em seguida, mantenha pressionada a tecla Shift e pressione a tecla de seta para baixo para selecionar a linha da instrução PRINT. Sua tela deverá se parecer com a figura seguinte.



2. Arrie o menu Edit:

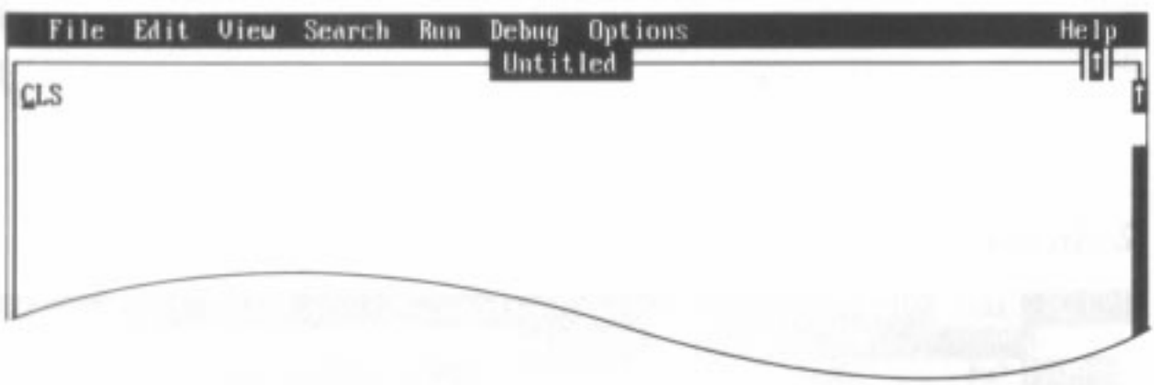


3. Escolha o comando Cut pelo menu Edit. A instrução PRINT deverá desaparecer da janela View, deixando-o apenas com a instrução CLS,

como mostramos na figura seguinte. Embora não estando mais na tela, a instrução PRINT não está perdida. Ela se encontra agora no *Clipboard* (área de transferência) do QBasic. Seu próximo passo será inserir o conteúdo do Clipboard de volta ao programa.

O Clipboard

O Clipboard do QBasic armazena o texto do programa apagado ou copiado mais recentemente. O texto permanece no Clipboard até que você apague ou copie algo novamente (apagando completamente o conteúdo anterior) ou até que saia do QBasic. Você não pode apagar o conteúdo do Clipboard do QBasic.



4. Mova o cursor até o local em que você deseja inserir o texto. Como você quer colocar a linha da instrução PRINT após a linha da instrução CLS, pressione a tecla de seta para baixo uma vez para colocar o cursor uma linha abaixo do C da instrução CLS.

5. Arrie o menu Edit novamente.

6. Escolha o comando Paste. Sua tela deverá estar agora como a figura da página a seguir.

Nota: Se, após a operação Paste, a instrução PRINT aparecer antes da instrução CLS, você provavelmente errou no posicionamento do cursor antes de escolher Paste. Lembre-se: se quiser inserir o conteúdo do Clipboard após uma instrução qualquer, deverá ter o cuidado de posicionar o cursor uma linha abaixo dessa instrução antes de escolher Paste. Repita as operações Cut e Paste, se for necessário, de modo que a instrução PRINT fique abaixo da instrução CLS.

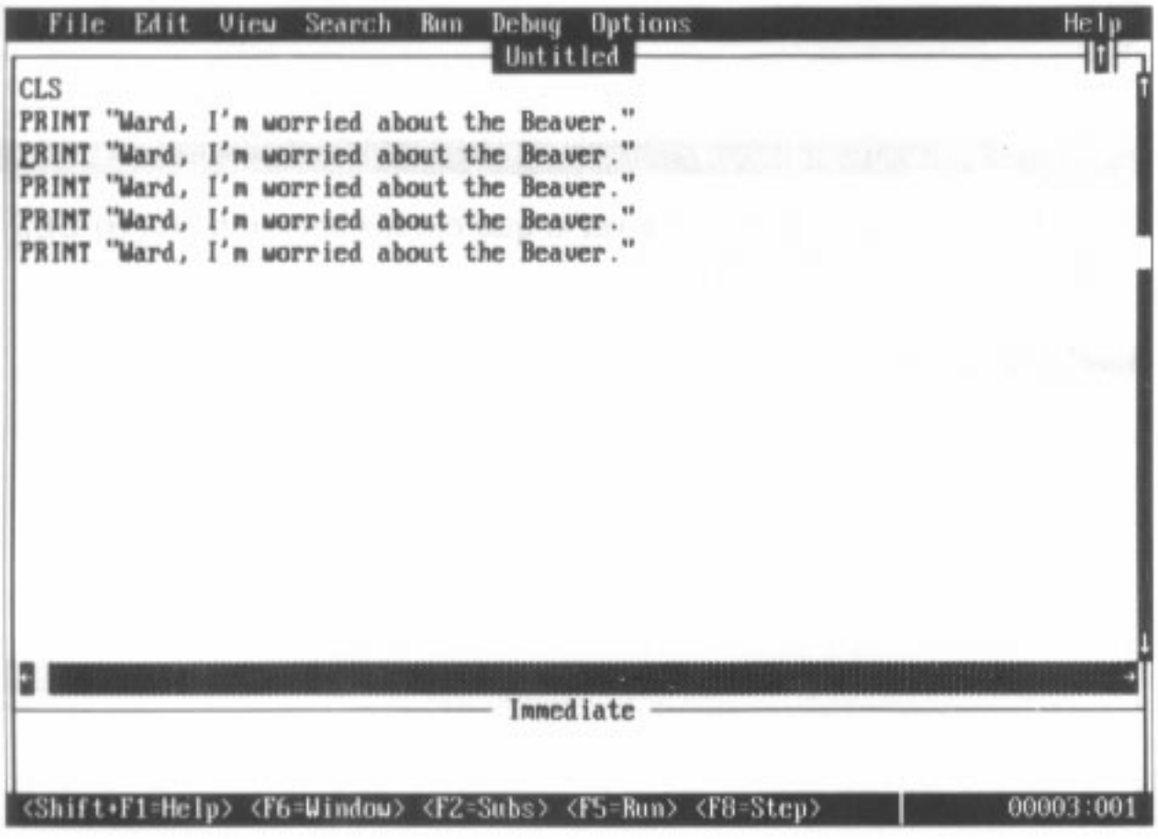
7. Rode o programa agora para ver o que ele faz com a instrução CLS antes da instrução PRINT:

8. Pressione qualquer tecla para retornar à janela View.

Prática: Editando um programa com os comandos Copy e Paste

Agora que você posicionou corretamente a instrução CLS, é hora de esticar seu programa usando os comandos Copy e Paste.

1. Selecione a instrução PRINT inteira mais uma vez.
2. Escolha o comando Copy pelo menu Edit. Observe que a linha selecionada permanece no seu programa.
3. Mova o cursor de modo que esteja uma linha abaixo da instrução PRINT. Observe que o destaque desaparece.
4. Escolha Paste pelo menu Edit.
5. Escolha Paste novamente — até que tenha cinco instruções Print na tela. Seu programa deverá agora se parecer com este:



Rode o programa novamente, se quiser, mas provavelmente você ficará cansado de ver a mesma mensagem repetidamente. Como prática, esta seria uma boa ocasião para apagar algumas dessas instruções.

Prática: Apagando texto com a tecla Del

1. Mova o cursor até o P na última instrução PRINT e pressione a tecla Del várias vezes. Observe como o Del apaga o caracter onde o cursor está localizado.
2. Continue pressionando Del até que a linha inteira seja apagada.

3. Pressione Del mais algumas vezes; eventualmente, você ouvirá um bipe toda vez que pressionar Del. Este bipe é o modo como o QBasic informa que não existem mais caracteres a apagar nessa linha. (Sempre que você ouvir isso, provavelmente estará fazendo algo que o QBasic não permite.)

Prática: Apagando texto com a tecla Backspace

1. Mova o cursor uma linha para cima.
2. Pressione a tecla End para mover o cursor até o final da linha.
3. Pressione Backspace repetidamente, até que a instrução inteira seja apagada. Observe que, se continuar pressionando Backspace após a linha ser apagada, o cursor passará para o final da linha anterior e começará a apagar caracteres dessa linha.

Prática: Apagando texto selecionado com a tecla Del

1. Selecione parte ou toda a última instrução PRINT.
2. Pressione Del e o texto selecionado será apagado. (Se você selecionou apenas parte da linha, volte e destaque o restante para apagá-lo também.)

Pressionando Del em vez de escolher Cut, você diz ao QBasic que *não* deseja que o texto selecionado seja armazenado no Clipboard — você deseja apagá-lo de vez.

Prática: Apagando uma linha inteira com Ctrl-Y

1. Mova o cursor para cima até algum lugar dentro da próxima instrução PRINT.
2. Segure a tecla Ctrl e pressione Y. Isso possui o mesmo efeito de destacar a linha inteira e escolher Cut pelo menu Edit.

Já chega de mexer nas instruções PRINT! Se você seguiu as instruções exatamente, estará agora com a instrução CLS e uma instrução PRINT. Se tiver mais instruções do que isso, apague-as agora.

Alteração do Texto

Às vezes você pode querer mudar parte de uma instrução. Você poderia simplesmente apagar a instrução inteira e depois redigitá-la, mas verá que alterar apenas parte de uma instrução fará com que economize tempo e reduza a chance de cometer erros de digitação.

Analisemos a instrução PRINT. Uma instrução PRINT imprime qualquer coisa entre os dois sinais de aspas. Se você usou uma instrução PRINT em seu programa mas decidiu mudar a mensagem, tudo que você tem a fazer é mudar a mensagem entre as aspas. Você pode fazer isso de várias maneiras.

Você poderia apagar o texto existente (usando um dos métodos de apagamento que acabou de praticar) e digitar o novo texto em seu lugar. Ou então poderia simplesmente digitar um novo texto usando um dos dois cursores: o *cursor de inserção* ou o *cursor de sobreposição.*

O cursor de inserção

O cursor com que esteve trabalhando, o sublinhado piscante, é chamado *cursor de inserção*; isto porque você pode usá-lo para inserir texto no meio de uma linha sem destruir qualquer texto existente. Para inserir um texto com esse cursor, siga estes passos:

1. Mova o cursor até o local em que deseja inserir o texto.
2. Digite o texto que deverá ser inserido.

Prática: Inserindo texto com o cursor de inserção

1. Mova o cursor até a primeira letra após as aspas iniciais da mensagem de PRINT.
2. Digite algumas letras e veja o que acontece. As letras na mensagem, incluindo a letra onde o cursor se encontra, serão movidas para a direita à medida que você digita, dando espaço para as novas letras digitadas.
3. Quando terminar, use Backspace para apagar os novos caracteres que acabou de digitar, restaurando a mensagem original.

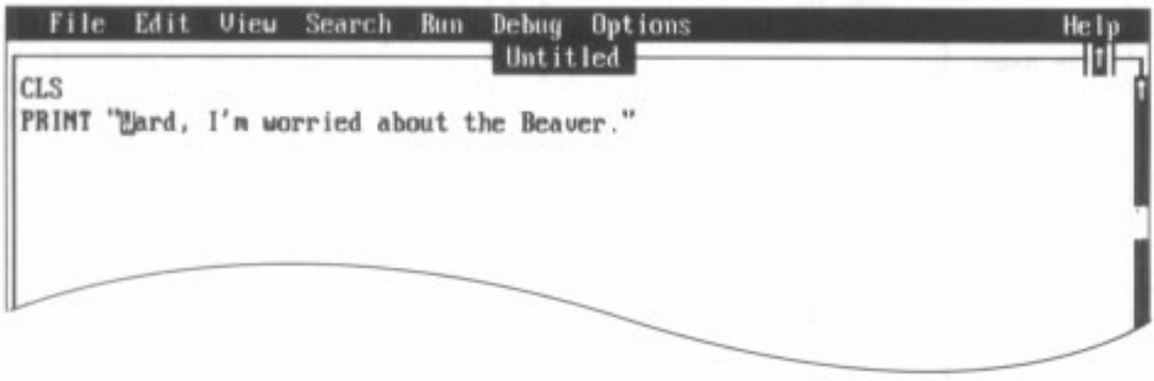
O cursor de sobreposição

O outro cursor é chamado *cursor de sobreposição.* Ele é chamado assim porque substitui os caracteres existentes, sobrepondo-os por novos caracteres. O uso desse tipo de cursor poderá ser muito útil, pois você não terá que apagar os caracteres, como seria preciso se tivesse apenas o cursor de inserção. Para usar o cursor de sobreposição, siga estes passos:

1. Mova o cursor até o início dos caracteres que deseja sobrepor.
2. Pressione a tecla Ins. O cursor passará para um retângulo piscante.
3. Digite os caracteres que desejar.

Prática: Usando o cursor de sobreposição

1. Mova o cursor até a primeira letra após as aspas iniciais da mensagem de PRINT.
2. Pressione a tecla Ins. O cursor passará de um sublinhado piscante para um retângulo vertical piscante:



3. Comece a digitar uma nova mensagem e veja o que acontece. Ao digitar, os novos caracteres substituirão os antigos.
4. Se a sua nova mensagem for mais curta do que a antiga, use Del para apagar os caracteres restantes (exceto pelas aspas finais). Se a sua mensagem for maior do que a antiga, você terá escrito sobre as aspas finais, devendo digitá-las novamente.
5. Rode o programa novamente, para ver sua nova mensagem, e depois pressione qualquer tecla para retornar à janela View.
6. Pressione Ins para mudar o cursor de sobreposição e retorná-lo ao modo de inserção. Você provavelmente desejará deixar o cursor no modo de inserção enquanto utiliza este livro.

Como Salvar o Seu Programa em Disco

Após digitar um programa, sobretudo um que deseje manter, você deverá salvá-lo no disco. Se não o fizer e desligar o computador, o programa ficará perdido, e você terá que digitá-lo novamente desde o início.

O programa que você acabou de digitar não é grande, e não seria preciso muito esforço para redigitá-lo. Mas quando os seus programas ficarem maiores, você definitivamente desejará salvá-los no disco, pois a redigitação seria um trabalho desnecessário, extra.

Prática: Salvando um programa em disco

1. Escolha o comando Save As (salvar como) pelo menu File. O comando Save As mostra o quadro de diálogo da página seguinte.
2. Digite o seguinte no quadro de texto File Name (nome do arquivo):

```
my-first.bas
```

NOTA: Como o QBasic pressupõe que o arquivo que está sendo salvo é um programa que garante uma extensão BAS, ele colocará esta extensão no nome do arquivo se você não o fizer.

3. Pressione Enter para executar o comando Save As. O QBasic salvará o seu programa no disco sob o nome de arquivo MY-FIRST.BAS.

Após ter salvo o programa no disco, o QBasic o levará à janela View, que agora mostra o nome do arquivo no topo da janela. Isso lhe permite saber a qualquer momento qual o programa apresentado na janela. Se o texto for *Untitled*, você saberá que ainda não salvou o programa no disco.

Um programa não precisa estar completo para que seja salvo. Assim, como você saberá quando salvá-lo pela primeira vez e com que freqüência depois disso terá que salvá-lo novamente? Para responder a esta pergunta, basta descobrir quanto trabalho perderia e como ficaria chateado se faltasse energia *neste momento*.

Nomeação de Arquivos

As seguintes dicas o ajudarão a nomear seus arquivos de forma eficaz:

- Um nome de arquivo não pode ter mais do que oito caracteres.
- Quanto mais descritivo o nome, melhor.
- Um nome de arquivo pode ser qualquer combinação de letras, números e alguns símbolos especiais; você não pode usar espaços e nem os seguintes caracteres:

  * = + [ ] ; : " ? < > / \ |

- A extensão do nome de arquivo do seu programa sempre deverá ser BAS (separada do nome por um ponto). Isso dirá ao QBasic que este é um arquivo de programa QBasic legítimo.

Alteração de um Arquivo Salvo

Se, após usar Save As para salvar um programa no disco, você fizer mudanças no programa, o QBasic tratará o programa na janela View como um programa diferente daquele que foi armazenado no disco. O QBasic não lhe permitirá trabalhar em outro programa até que saiba o que deseja fazer com a versão modificada do programa atual: Deseja modificar a versão *salva* e deixar em seu lugar a versão *alterada* do arquivo? Ou deseja abandonar as mudanças completamente?

Prática: Alterando um arquivo salvo

1. Faça algumas mudanças no programa existente na janela View. As mudanças ficam por sua conta — talvez possa passar a mensagem de PRINT para o português ou incluir uma segunda instrução PRINT.
2. Após fazer as mudanças, escolha Save As pelo menu File.

Observe que o quadro de texto File Name já possui o nome que você deu ao programa. Você possui várias opções neste ponto:

- Escolha OK para salvar a versão alterada do programa no disco, substituindo a versão original.
- Digite um novo nome para o programa alterado e escolha OK. Isso salvará a versão alterada do programa sob um nome diferente, deixando intacta a versão original.
- Escolha Cancel para cancelar a operação Save As e retornar à janela View.
- Escolha Help para ver um quadro de diálogo explicando a respeito do quadro Save As.

3. Pressione Enter para escolher OK e salve o programa alterado sob o mesmo nome no disco.

Salve Freqüentemente

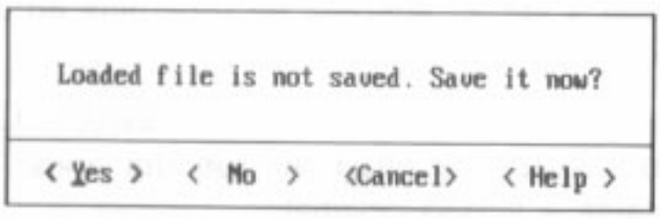
Até que você tenha salvo o programa no disco, tudo que é digitado na janela View é mantido na memória temporária do computador. Salve os seus programas regularmente no disco, evitando perdas no trabalho. Se o seu computador ficasse momentaneamente sem energia (talvez devido a um problema na transmissão ou um cabo que se soltou da tomada), tudo o que existe na memória — incluindo o seu programá — seria perdido para sempre. Se você tivesse armazenado esse programa no disco, no entanto, poderia simplesmente carregá-lo novamente.

INÍCIO DE UM NOVO PROGRAMA

Você só pode ter um programa de cada vez na janela View, seja um programa recém-digitado ou um programa carregado do disco. Para apagar o programa corrente da janela View e iniciar a escrita de um novo programa em QBasic, escolha o comando New (novo) pelo menu File. Se o programa na janela View tiver mudanças não salvas, o QBasic não permitirá que inicie um novo programa até que descubra o que você deseja fazer com o programa corrente alterado.

Prática: Iniciando um novo programa

1. Faça outra mudança no programa da janela View, talvez alterando a mensagem que PRINT apresenta.
2. Após fazer a mudança, escolha New pelo menu File. O quadro de diálogo a seguir aparecerá na tela.



Se você pressionar Enter para escolher a resposta default *Yes*, o QBasic salvará no disco a versão alterada do programa, usando o nome atualmente mostrado no topo da janela View. Isso modifica a versão mais antiga já existente no disco. Você não possui a opção de digitar outro nome de arquivo — para isso, deverá primeiro pressionar Esc para sair

deste quadro de diálogo e depois escolher Save As pelo menu File para dar um outro nome ao programa.

Se escolher *No*, o QBasic não salvará as mudanças feitas desde a última vez em que o arquivo foi salvo. Não tenha pressa para escolher *No* — esteja certo de que não estará perdendo algo que você queria salvar!

3. Se quiser salvar seu programa alterado e gravar sobre a versão anterior, pressione Enter. Caso contrário, pressione Tab uma vez para destacar o botão de comando No e depois pressione Enter.

   O QBasic apagará a janela View e você estará pronto para digitar um programa totalmente novo. Tente entrar agora com alguns comandos PRINT para praticar o que aprendeu. Quando tiver terminado, salve o programa no disco sob um nome apropriado.

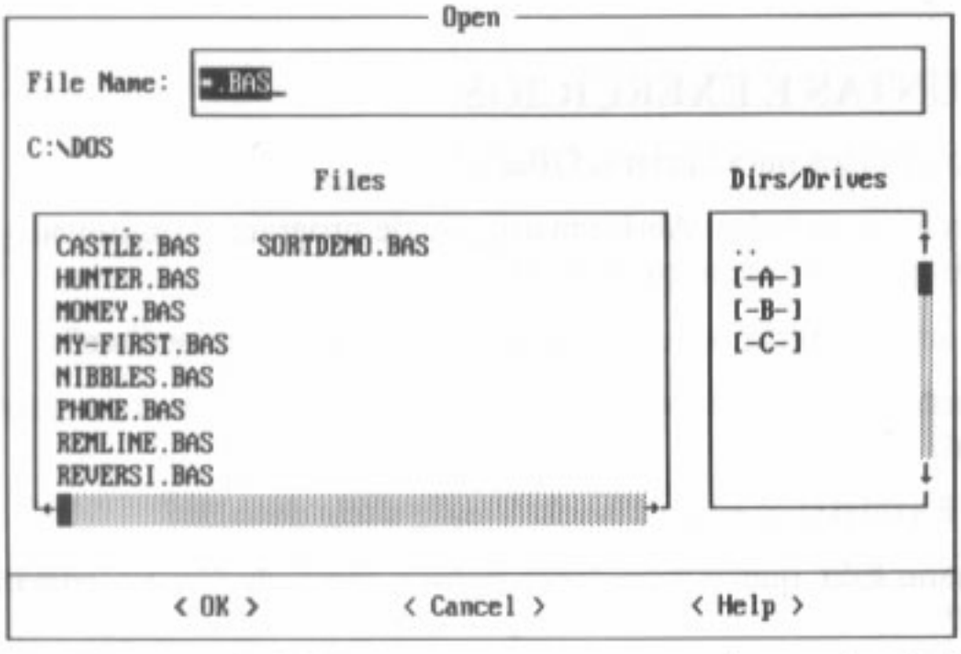
ABRINDO UM PROGRAMA EXISTENTE

Para trabalhar com um programa armazenado no disco, você deverá primeiro abri-lo, assim como você deve abrir um livro antes que possa lê-lo. Abrir um programa significa carregá-lo na janela View. Para abrir um programa, selecione o comando Open pelo menu File.

Prática: Abrindo um programa existente

1. Arrie o menu File. Observe que o comando Open é seguido por reticências (...).

2. Escolha o comando Open e verá o quadro de diálogo Open.



3. Pressione a tecla Tab para mover o cursor até o quadro de lista Files (arquivos).

4. Destaque REVERSI.BAS.
5. Escolha OK. Isso fará com que o QBasic abra o programa e o carregue na janela View.

Agora você estará numa posição em que pode editar ou rodar o programa de jogo REVERSI.BAS do QBasic. Use este método para carregar qualquer arquivo existente no disco. Use a lista Dirs/Drives (diretórios/unidades de disco) para encontrar programas em qualquer outro local no sistema.

SAÍDA DO QBASIC

Quando tiver terminado a programação, você sempre deverá sair do QBasic antes de desligar o computador. Se não o fizer, poderá lembrar tarde demais que se esqueceu de salvar um programa.

Para sair do QBasic, escolha Exit (sair) pelo menu File. Lembre-se de que, se tiver um programa na janela View e não o tiver salvo, ou então se tiver alterado um programa que já estava salvo, o QBasic perguntará se você deseja salvá-lo.

RESUMO

Você já está bem adiantado no aprendizado da programação em QBasic. Você já sabe como iniciar, usar e sair do QBasic, além de escrever, rodar, editar e salvar os seus próprios programas. No próximo capítulo, você aprenderá mais sobre instruções QBasic, vendo-as mais detalhadamente.

PERGUNTAS E EXERCÍCIOS

1. Como fazemos para iniciar o QBasic?
2. Verdadeiro ou Falso: Abrir um arquivo de programa armazenado em um disco apaga esse arquivo do disco.
3. Diga uma diferença entre a janela View e a janela Immediate?
4. No menu File, qual é a diferença entre o comando New e o comando Open?
5. Qual é a diferença entre a janela View e a tela de saída?
6. No menu Edit, qual é a diferença entre o comando Cut e o comando Copy?
7. Quais são os quatro meios de apagar um texto de um programa, e como eles diferem?

8. Qual é a diferença entre o cursor de inserção e o cursor de sobreposição? Como você pode diferenciá-los? Como você pode passar de um para outro?

9. Sem incluir a extensão BAS, qual é o número máximo de caracteres que você pode usar para nomear o seu arquivo?

10. Quais dos nomes a seguir você *não* pode usar para nomear seus programas? Por que não?

IMPRINOME.BAS	HOM-E-MUL.BAS	50%OK.BAS
OLA!.BAS	NEGOCIOS.BAS	PROG*1.BAS
BOM+MAU.BAS	BALOES.BAS	MEU_MODO.BAS
BOM MAU.BAS	PROG[1].BAS	BOB'S-OK.BAS

11. Como fazemos para sair do QBasic? O que acontece se você tentar sair antes de salvar o seu trabalho e quais são as suas opções?

CAPÍTULO 3

Introdução à Linguagem QBasic

No capítulo anterior você se familiarizou com o QBasic e usou algumas instruções do QBasic para escrever um pequeno programa. Neste, você verá a linguagem mais de perto e aumentará o conhecimento já adquirido.

ANATOMIA DE UMA INSTRUÇÃO QBASIC

Até aqui, chamamos as linhas do QBasic que você digitou como *instruções*. Como vimos no Capítulo 1, cada linha em um programa do QBasic é simplesmente uma instrução que o computador executa quando você roda o programa. O QBasic reconhece dois tipos de instruções: *comandos* e *funções*.

Comandos e Funções

Comandos e funções são parecidos e muito fáceis de usar. A diferença principal está na finalidade:

- Um comando é geralmente um verbo. Ele simplesmente faz o que você diz para fazer quando o programa é executado. Quando o seu programa executa um comando, o resultado é geralmente aparente. O comando PRINT, por exemplo, coloca caracteres na tela de saída, e o comando CLS apaga a tela de saída.

- Uma função retorna um valor que o seu programa pode usar; geralmente trabalha de forma menos óbvia do que um comando. Ela aparece dentro de um comando e realiza seu trabalho quando o comando é executado.

Cada comando e função possui uma *sintaxe*, que indica como o comando ou função deve ser usado no seu programa.

Sintaxe do QBasic

A sintaxe de uma instrução em QBasic, seja comando ou função, é simplesmente uma palavra-chave seguida pela informação que a instrução precisa para realizar seu trabalho.

A sintaxe de algumas instruções do QBasic, como BEEP, consiste apenas no nome do comando ou função. Inicie o QBasic (se ainda não o fez) e digite o seguinte programa de única linha:

```
BEEP
```

Rode o programa e veja o que acontece. O comando BEEP faz com que o alto-falante dentro do seu computador emita um som rápido.

Agora vejamos algumas instruções com maiores detalhes. No restante deste capítulo usaremos o comando PRINT como nosso exemplo de comando.

Sintaxe do comando PRINT

O comando PRINT tem a seguinte sintaxe:

PRINT [*listaexpressão*][,|;]

Parece um pouco diferente do comando PRINT do Capítulo 2, não é? O comando PRINT é muito versátil; no capítulo anterior, você usou apenas uma das muitas características do comando PRINT.

Vamos por enquanto aprender mais sobre a estrutura de uma linha de sintaxe (Figura 3-1):

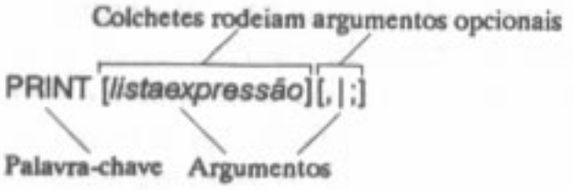


FIGURA 3-1.
Estrutura de uma linha de sintaxe.

Cada item em uma linha de sintaxe, exceto pela palavra-chave, é chamado *argumento*. Os argumentos dão à instrução informações adicionais que ela precisa para realizar seu trabalho. Alguns deles são opcionais; eles aparecem dentro de colchetes tanto neste livro quanto no sistema de ajuda embutido no QBasic. Se você decidir usar um desses argumentos opcionais na sua instrução, não digite os colchetes. (Seguindo o mesmo princípio, não digite o símbolo | — ele simplesmente indica que você deve escolher um dos argumentos ao redor de |. No caso da Figura 3-1, você escolheria a vírgula ou o ponto-e-vírgula.)

Em um comando PRINT, os dois argumentos são opcionais: você poderá usar a palavra-chave PRINT sem qualquer argumento.

Prática: Usando um comando PRINT sem argumentos

1. Selecione o comando New pelo menu File.

2. Digite o seguinte programa e execute-o:

```
CLS
PRINT "Este comando usa um argumento."
PRINT
PRINT "Este também usa."
```

Sua tela se parecerá com esta:

```
Este comando usa um argumento.

Este também usa.
```

Como você pode ver, um comando PRINT sem qualquer argumento imprime uma linha em branco. (Simplesmente pressionar Enter para criar linhas em branco não funcionará.) Vejamos três tipos de argumentos que você pode usar com um comando PRINT: texto, expressões numéricas e funções.

Uso de texto como argumento

No capítulo anterior, usamos um texto cercado por aspas ("Ward, I'm worried about the Beaver.") como um argumento *listaexpressão* para o comando PRINT. Esse texto é chamado *cadeia*. Você pode usar quaisquer números, letras, espaços e sinais de pontuação — exceto as aspas duplas — em uma cadeia. (O QBasic reconhece um sinal de aspas duplas dentro de uma cadeia como o final da cadeia.)

Uso de expressões numéricas como argumentos

Um outro argumento *listaexpressão* válido é uma *expressão numérica*. Uma expressão numérica é um número, uma equação matemática ou, como veremos no próximo capítulo, um tipo especial de palavra chamado *variável* numérica. Expressões numéricas não podem ser cercadas por aspas.

Prática: Usando expressões numéricas como argumentos

1. Modifique o seu programa para que contenha apenas o seguinte comando PRINT:

```
CLS
PRINT 42
```

2. Agora rode o programa. Sua tela de saída se parecerá com isto:

```
42
```

Uso de funções como argumentos

Um terceiro argumento *listaexpressão* válido é uma *função* do QBasic. Uma função geralmente realiza uma tarefa em segundo plano e retorna a informação que você pode usar no seu programa. Assim como as expressões numéricas, uma função não pode ser cercada por aspas.

Prática: Usando funções como argumentos

O QBasic possui duas funções, chamadas DATE$ e TIME$, que obtêm a data e a hora correntes do relógio do sistema embutido no seu computador. Você não pode usar essas funções isoladamente. (Em outras palavras, você não pode digitar simplesmente DATE$ ou TIME$ em uma linha e esperar que algo aconteça.) Entretanto, você pode usá-las como argumentos para um comando PRINT.

1. Selecione o comando New pelo menu File.
2. Digite o seguinte programa e execute-o:

```
CLS
PRINT DATE$
PRINT TIME$
```

Sua tela de saída ficará semelhante a esta:

```
03-25-1991
21:13:09
```

Embora a data e a hora mostradas aqui não combinem com o que você vê na sua tela de saída, o princípio fundamental ainda permanece: DATE$ e TIME$ realizaram uma tarefa e retornaram um valor sem muito trabalho. Isso é o que existe de melhor a respeito das funções do QBasic: Você simplesmente as utiliza, e elas realizam um trabalho útil sem que você tenha que se preocupar com os detalhes.

Impressão de Mais de Um Item com PRINT

Até aqui, usamos um único argumento, como uma única cadeia ou uma única função, para cada um dos comandos PRINT. Você pode usar múltiplos argumentos em um único comando PRINT, mas somente se usar um caracter especial chamado *separador* entre eles. O comando PRINT reconhece dois caracteres separadores: uma vírgula e um ponto-e-vírgula.

O separador vírgula

Você pode pensar no separador vírgula como o equivalente em programação da tecla Tab. Quando PRINT encontra uma vírgula, ele imprime o valor do próximo argumento no início da próxima zona de impressão. (As zonas de impressão são os equivalentes em programação das paradas de tabulação; elas possuem 14 caracteres de extensão.) Esta capacidade de imprimir valores em locais específicos torna os separadores vírgula uma escolha natural quando você deseja imprimir informações em colunas.

Prática: Usando separadores vírgula

1. Selecione o comando New pelo menu File.
2. Digite e execute o seguinte programa para ver como os separadores vírgula funcionam:

```
CLS
PRINT "Separador", "vírgula", "separa", "argumentos"
PRINT "Ele", "também", "alinha", "argumentos"
```

Sua tela de saída se parecerá com esta:

```
Separador     vírgula       separa        argumentos
Ele           também        alinha        argumentos
```

NOTA: Se você não colocou um espaço após cada vírgula, o QBasic os colocou por você quando a tecla Enter foi pressionada ao final de cada linha. O QBasic sempre inclui um espaço após um separador se você se esquecer, dando aos seus programas uma aparência melhor e mais coerente.

Você pode até mesmo usar vírgulas isoladas sem ter um argumento entre elas, como mostra a sessão de prática a seguir.

Prática: Posicionando argumentos com separadores vírgula

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que usa vírgulas para posicionar valores próximos ao centro da tela:

```
CLS
PRINT , , "Use separadores vírgula"
PRINT , , "para posicionar argumentos"
```

Sua saída deverá se parecer com esta:

```
                Use separadores vírgula
                para posicionar argumentos
```

O separador ponto-e-vírgula

Quando PRINT encontra um separador ponto-e-vírgula, ele imprime o próximo argumento imediatamente após o argumento que acabou de imprimir. PRINT não insere espaços entre esses argumentos: Você deverá fornecê-los se assim desejar.

Prática: Usando separadores ponto-e-vírgula

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que usa separadores ponto-e-vírgula com e sem espaços incluídos:

```
CLS
PRINT "Isso"; "é"; "o"; "que"; "ponto-e-vírgulas";
"fazem"
PRINT "Se "; "quiser "; "espaços, "; "inclua-os"
```

Sua tela de saída deverá ser parecida com esta:

```
Issoéoqueponto-e-vírgulasfazem
Se quiser espaços, inclua-os
```

Uso de separadores vírgula e ponto-e-vírgula

Assim como você pode usar mais de um tipo de argumento em cada linha, pode usar também mais de um tipo de separador em uma linha, misturando-os como desejar; eles sempre funcionarão como dissemos.

Prática: Usando separadores vírgula e ponto-e-vírgula

1. Selecione o comando New pelo menu File.
2. Digite e execute o seguinte programa, que usa vírgulas e ponto-e-vírgulas na mesma linha:

```
CLS
PRINT "Estas"; "palavras"; "ficam"; "juntas",
"Estas", ficam", "separadas"
```

Sua tela de saída deverá ficar assim:

```
Estaspalavrasficamjuntas      Estas      ficam      separadas
```

Uso de um separador ao final de um comando PRINT

Quando você colocar uma vírgula ou um ponto-e-vírgula ao final de um comando PRINT, estará indicando onde o resultado do comando PRINT *subseqüente* aparecerá na tela de saída:

- Uma vírgula imprime a saída do próximo comando PRINT no início da próxima zona de impressão.
- Um ponto-e-vírgula imprime a saída do próximo comando PRINT imediatamente após a saída do comando PRINT corrente.

Prática: Usando separadores ao final de um comando PRINT

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que usa um ponto-e-vírgula ao final de um comando PRINT:

```
CLS
PRINT "Esta é a primeira linha ";
PRINT "e esta é a segunda"
```

Sua tela de saída ficará assim:

```
Esta é a primeira linha e esta é a segunda
```

3. Agora mude o ponto-e-vírgula para uma vírgula e execute o programa novamente.

NOTA: Se você usar o ponto-e-vírgula ao final de um comando PRINT e não houver mais comandos PRINT em seu programa, ele não terá qualquer efeito.

RESUMO

Neste capítulo você aprendeu mais sobre as instruções de um programa em QBasic e como as várias opções de sintaxe tornam um comando mais versátil. No capítulo seguinte, você conhecerá dois elementos importantes do QBasic, que darão ainda mais flexibilidade aos programas: variáveis e operadores.

PERGUNTAS E EXERCÍCIOS

1. Descreva resumidamente a diferença entre um comando e uma função.
2. Quais das seguintes palavras-chave do QBasic são comandos e quais são funções?

BEEP	DATE$	TIME$
CLS	PRINT	

3. Em uma linha de sintaxe do QBasic, qual é o significado dos colchetes ao redor de um item? O que significa o caracter |?
4. O que é um argumento?
5. Qual a diferença entre uma cadeia e uma expressão numérica?
6. Qual será a diferença entre as saídas dos seguintes comandos PRINT?

```
PRINT "Olá", ; "você"
PRINT "Olá"; , "você"
```

7. O que acontece quando você coloca um ponto-e-vírgula ou uma vírgula ao final de um comando PRINT?

CAPÍTULO 4

Variáveis e Operadores do QBasic

Ao programar, você pode ter que imprimir certos números ou cadeias de caracteres mais de uma vez. O QBasic oferece um método fácil para fazer exatamente isso — um método que não exige muita redigitação da sua parte. No QBasic, você pode armazenar dados e usá-los quando desejar e com qualquer freqüência, bastando usar o nome do local de armazenagem no seu programa. Você nomeia o local de armazenagem, que é conhecido como *variável*, de acordo com o tipo e tamanho dos dados que ele contém.

O QUE VOCÊ DESEJA ARMAZENAR?

As variáveis possuem dois tipos básicos: *de cadeia* e *numérica*. Uma variável de cadeia é um nome representando um local de armazenagem para uma cadeia. Uma variável numérica é um nome representando um local de armazenagem para um número. Além do mais, as variáveis numéricas possuem vários subtipos. A Figura 4-1 mostra os tipos válidos no QBasic.

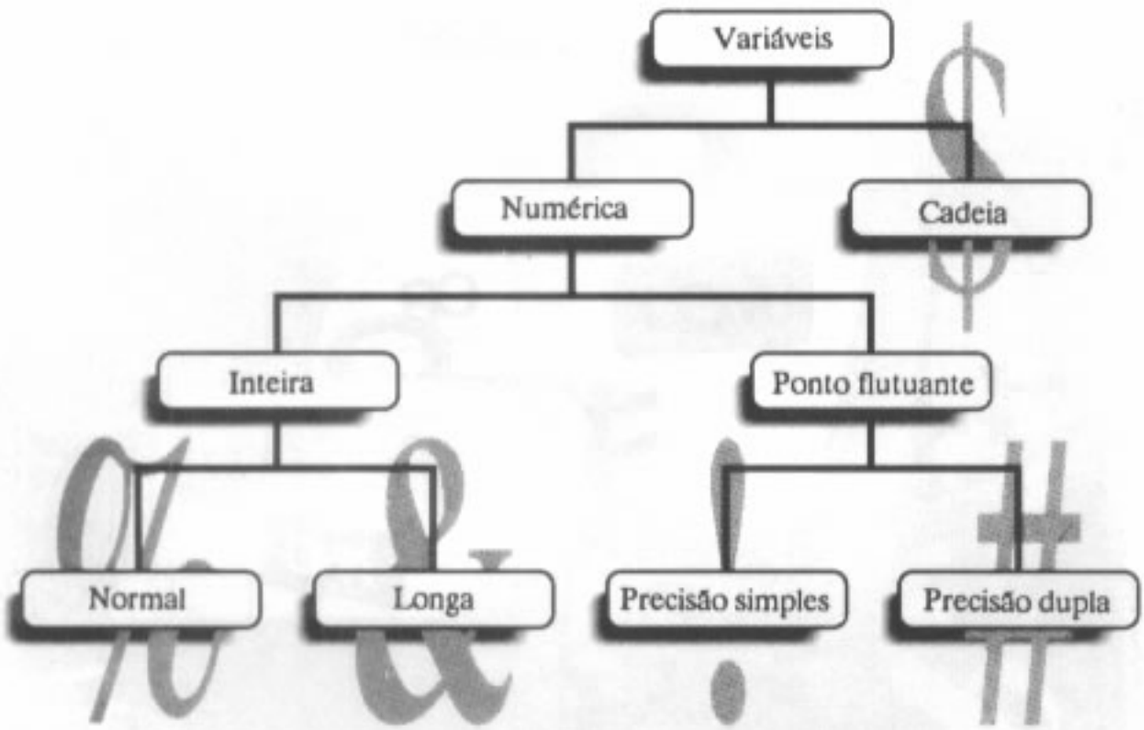


FIGURA 4-1.
Uma visão geral dos tipos de variáveis usados no QBasic.

Ao ler este capítulo, tenha em mente as perguntas a seguir. Como veremos, as respostas a essas perguntas o ajudarão a determinar qual o tipo de variável mais adequado para as suas finalidades.

- **Seus dados são texto ou números?** Texto é armazenado em uma variável de cadeia; um número é armazenado em uma variável numérica.

- **Se estiver armazenando um número, ele possui ponto decimal?** Um inteiro — ou seja, um número sem um ponto decimal — é armazenado em uma *variável inteira*. Um número com um ponto decimal é armazenado em uma *variável de ponto flutuante*.
- **Qual o tamanho do número que você quer armazenar?** O tamanho do número desempenha um papel importante quando você detetemina qual a variável a usar.

USO DE VARIÁVEIS – UMA VISÃO GERAL

Quando você decidir usar uma variável, deverá primeiro *declará-la* no seu programa; ou seja, deverá informar ao QBasic três coisas:

- O nome da variável
- O tipo da variável
- O valor da variável

Você poderá ver os elementos de uma declaração na Figura 4-2, que mostra uma declaração de variável de cadeia como exemplo.

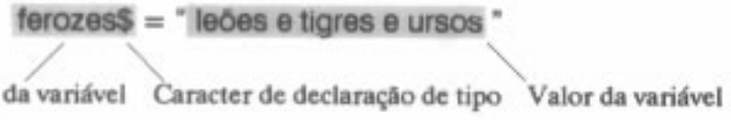


FIGURA 4-2.
Os elementos de uma declaração de variável.

As seções a seguir descrevem como usar cada um desses elementos.

Nomeação de uma Variável

Essas regras e sugestões o ajudarão a escolher nomes apropriados para as suas variáveis do QBasic:

- Um nome de variável pode ter até 40 caracteres de extensão.
- Um nome pode ter qualquer combinação de letras maiúsculas e minúsculas. (Entretanto, saiba que os nomes em maiúsculas podem apresentar dificuldade na distinção entre as palavras-chave do QBasic.)
- Para os nomes das variáveis, você não pode usar as palavras-chave do QBasic, como PRINT ou BEEP.
- O caracter de declaração de tipo apropriado deve ser o último caracter do nome da variável. (Veja "Declaração do Tipo de Variável".)

- Em geral, nomes de variável mais descritivos são mais eficientes. Por exemplo, os nomes de variável *primeiroNome$* e *sobrenome$* não deixam dúvida em sua mente quanto ao que elas representam.

NOTA: O QBasic diferencia o uso de letras maiúsculas e minúsculas para variáveis. Tanto que, na verdade, se você digitar um nome de variável que já foi usado com um tipo de letra diferente, o QBasic ajusta os nomes digitados anteriormente para que reflitam a nova nomenclatura.

Declaração do Tipo de Variável

Se você olhar novamente a Figura 4-2, notará que o caracter final do nome da variável é o *caracter de declaração de tipo*. O caracter de declaração de tipo diz ao QBasic o tipo de variável que está sendo declarada. Cada tipo possui seu próprio símbolo:

Se a variável for	*Use isto como seu caracter de declaração de tipo*
Cadeia	$
Inteira	
Normal	%
Longa	&
Ponto flutuante	
Precisão simples	!
Precisão dupla	#

Quando o QBasic encontrar o caracter de declaração de tipo no nome da variável, ele saberá qual o tipo de dado armazenado nessa variável.

Declaração do Valor de uma Variável

Quando você declarar o valor de uma variável, assegure-se de que o seu tipo combina com o caracter de declaração de tipo. Por exemplo, uma variável inteira não poderá conter um valor de cadeia.

E Depois?

Agora que você já conhece a estrutura geral de uma declaração de variável, poderá começar a aprender sobre os diferentes tipos de variáveis. Primeiro você aprenderá sobre variáveis de cadeia e variáveis numéricas. Depois aprenderá sobre a obtenção de informações pelo teclado e seu armazenamento em variáveis. A parte final deste capítulo descreve como o QBasic trabalha com a matemática em geral.

VARIÁVEIS DE CADEIA NO QBASIC

Um nome de variável de cadeia representa um local de armzenagem para uma cadeia de texto que você deseja usar num programa inteiro.

Para declarar uma variável de cadeia, use o caracter de declaração de tipo cifrão ($) ao final do nome da variável. Delimite a cadeia com aspas, exatamente como faria em um comando PRINT. Por exemplo:

```
filmesBons$ = "Rambo, Aliens, ET"
```

O tamanho e conteúdo das variáveis de cadeia podem ser alterados dentro de um programa, como veremos no Capítulo 9.

Prática: Declarando e usando uma variável de cadeia

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que atribui um valor a uma variável de cadeia e depois imprime a cadeia:

```
CLS
tele$ = "Educativa, Globo, Manchete, TVS"
PRINT tele$
```

Sua tela de saída deverá ficar assim:

```
Educativa, Globo, Manchete, TVS
```

VARIÁVEIS NUMÉRICAS EM QBASIC

O QBasic usa dois tipos de variáveis numéricas: *variáveis numéricas inteiras* e *variáveis numéricas de ponto flutuante*. Cada tipo possui sub-tipos, concebidos para números de diferentes tamanhos. (Lembre-se do que já dissemos: o tamanho do número ajuda a determinar qual o tipo de variável a ser usado.) A Figura 4-3 mostra as relações entre os tipos de variável numérica em QBasic.

Primeiro você aprenderá sobre os dois tipos de variáveis inteiras e quando deverá usá-las; depois, aprenderá sobre os dois tipos de variáveis de ponto flutuante.

Variáveis Inteiras

O QBasic usa dois tipos de variáveis inteiras: *variáveis inteiras normais* e *variáveis inteiras longas*. A diferença entre as duas encontra-se no tamanho do número inteiro que cada uma representa.

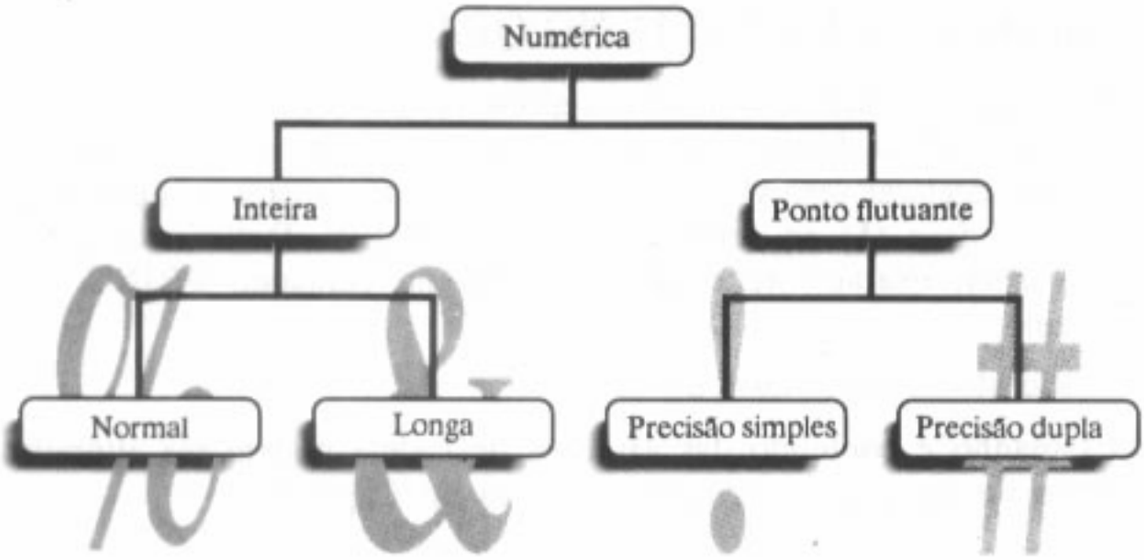


FIGURA 4-3.
Variáveis numéricas no QBasic.

Como o QBasic Vê os Números

Esteja você calculando o saldo da sua conta ou preparando relatórios e propostas, provavelmente gastará algum tempo do seu dia trabalhando com números. O que você provavelmente não observa é que está na realidade trabalhando com *tipos* diferentes de números. Para muitas pessoas, os nomes "apropriados" para esses números — números naturais, inteiros etc. — são apenas uma vaga memória das aulas de matemática do primeiro grau. Porém, para o QBasic, as diferenças entre dois tipos particulares de números são cruciais:

- Um inteiro é um número sem ponto decimal.
- Um número de ponto flutuante é um número com um ponto decimal.

Variáveis inteiras normais

Uma variável inteira normal pode armazenar qualquer número de −32.768 até 32.767.

Para declarar uma variável inteira normal, use o sinal de porcentagem (%) como caracter de declaração de tipo ao final do nome da variável. Por exemplo:

```
esposas% = 6
```

Prática: Declarando e usando uma variável inteira normal

1. Selecione mais uma vez o comando New pelo menu File.
2. Digite e execute o seguinte programa, que declara uma variável inteira normal e imprime o seu valor:

```
CLS
diasparaNatal% = 12
PRINT diasparaNatal%
```

Sua tela de saída deverá mostrar:

```
12
```

Um Espaço Extra para Números

Quando você usar o comando PRINT no seu programa em QBasic para mostrar números, poderá notar que todos os números são precedidos por um intrigante espaço adicional. Se um número for negativo, o QBasic usará este espaço para mostrar um sinal de menos. Se for positivo, ele mostrará um espaço no lugar do sinal de mais.

Variáveis inteiras longas

Para trabalhar com um inteiro fora da faixa de uma variável inteira normal, use uma variável inteira longa. Uma variável inteira longa pode representar um número inteiro de –2.147.483.648 até 2.147.483.647.

Para declarar uma variável inteira longa, use o símbolo & como caracter de declaração de tipo ao final do nome da variável. Por exemplo:

```
populCidade& = 175000
```

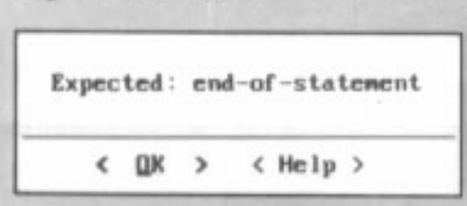
Prática: Declarando e usando uma variável inteira longa

1. Selecione o comando New pelo menu File.

Você Não Pode Usar Vírgulas em Números

Em nosso uso diário, usamos vírgulas para separar as partes inteira e decimal dos números. (Por exemplo, no sistema americano, o número 1,653,892,000 é certamente mais fácil de ser compreendido do que 1653892000.) Mas como as vírgulas possuem um uso especial como separadores de caracteres no QBasic, você deverá aprender a conviver sem elas quando entrar com números.

Se você se esquecer e usar uma vírgula como parte de um número dentro do seu programa, o QBasic mostrará uma mensagem de erro:



(Ou seja: *Fim de comando esperado*) Igualmente, se alguém rodando o seu programa digitar um número com vírgulas, o QBasic pedirá que o usuário digite o número novamente.

2. Digite e execute o programa a seguir, que declara uma variável inteira longa e imprime o seu valor:

```
CLS
distância& = 84700
PRINT "A maior distância, em metros, foi"; distancia&
```

Sua tela de saída deverá ficar assim:

```
A maior distância, em metros, foi 84700
```

Números de Ponto Flutuante

Até aqui você trabalhou apenas com inteiros. No entanto, na vida real você normalmente encontrará números com uma parte fracionária — ou seja, um número seguido por um ponto (ou vírgula) decimal e a expressão decimal de uma fração.

O QBasic oferece dois tipos de variáveis para representar números de ponto flutuante: *variáveis de ponto flutuante com precisão simples* e *variáveis de ponto flutuante com precisão dupla*. A principal diferença entre os dois tipos de variáveis encontra-se na precisão com que podem representar um determinado número fracionário.

Uso de uma Variável com o Tamanho Errado

Como vimos, o trabalho com números em QBasic exige muita atenção no tamanho de um número. Se você acidentalmente atribuir um número fora de faixa a uma variável, o QBasic mostrará um quadro de diálogo que indica que o número é inválido para esse tipo de variável:

(Ou seja: *Estouro*) Você pode ajudar a evitar esses enganos escolhendo variáveis do tamanho correto e mostrando a faixa de valores válidos ao usuário.

- Uma variável de ponto flutuante com precisão simples pode representar um número com até 7 dígitos significativos. O ponto decimal pode aparecer em qualquer lugar dentro desses dígitos.
- Uma variável de ponto flutuante com precisão dupla pode representar um número com até 15 dígitos de extensão. O ponto decimal pode aparecer em qualquer lugar dentro desses dígitos.

A tabela a seguir mostra alguns valores de exemplo com ponto flutuante:

Precisão simples	*Precisão dupla*
412.002	1.000032478
19246.34	4280000.0055
.00025	7160.0000000005
657926.3	.10000000001

Variáveis de ponto flutuante com precisão simples

Para declarar uma variável de ponto flutuante com precisão simples, use o ponto de exclamação (!) como caracter de declaração de tipo ao final do nome da variável. Por exemplo:

```
VelocLp! = 33.3333
```

Prática: Declarando e usando uma variável de ponto flutuante com precisão simples

1. Selecione o comando New pelo menu File.

2. Digite e execute o programa a seguir, que declara uma variável de ponto flutuante com precisão simples e mostra seu valor.

```
CLS
precoCarro! = 12999.99
PRINT "Este carro esportivo custa US$";precoCarro!
```

Sua tela de saída deverá ficar como esta:

```
Este carro esportivo custa US$ 12999.99
```

Variáveis de ponto flutuante com precisão dupla

As variáveis de ponto flutuante com precisão dupla, que podem conter até 15 dígitos de extensão, são úteis para trabalhos científicos que exijam números exatos.

Para declarar uma variável de ponto flutuante com precisão dupla, use o sinal de libra (#) como caracter de declaração de tipo ao final do nome da variável. Por exemplo:

```
grandeFlut# = 5.0000000127
```

Prática: Declarando e usando uma variável de ponto flutuante com precisão dupla

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que declara uma variável de ponto flutuante com precisão dupla e mostra seu valor.

```
CLS
pi# = 3.141592653589793
PRINT "O valor de pi é"; pi#
```

Sua tela de saída deverá ficar como esta:

```
O valor de pi é 3.141592653589793
```

Que Tipo de Variável Numérica Deve Ser Usado?

Você já aprendeu que o QBasic oferece quatro tipos de variáveis numéricas para você usar nos seus programas: inteira normal, inteira longa, ponto flutuante com precisão simples e ponto flutuante com precisão dupla. Quando você quiser atribuir um valor numérico a um nome de variável no seu programa, terá que escolher o tipo apropriado de variável. Faça as seguintes perguntas a si mesmo:

- O número é um inteiro? Ele permanecerá inteiro durante todo o programa? Caso afirmativo, use uma variável inteira.

- O número possui um ponto decimal? Ou, mais importante, *poderá* esse número obter uma parte fracionária mais adiante no programa? Caso afirmativo, use uma variável de ponto flutuante.

Depois de determinar o tipo da variável — inteira ou ponto flutuante —, você terá que determinar o tamanho que a variável usará.

Escolha do tamanho de variável apropriado

A princípio pode parecer que, por conterem números maiores, as variáveis inteiras longas e as variáveis de ponto flutuante com precisão dupla seriam as escolhas óbvias para usar o tempo inteiro. Afinal, o uso dessas variáveis maiores certamente reduziria suas chances de ver mensagens de erro *Overflow* (estouro).

Entretanto (você já sabia que sempre existe um inconveniente, não é?), essas variáveis maiores possuem uma desvantagem da qual você deverá estar ciente. Seu tamanho oferece um maior espaço para armazenamento dos seus valores, mas também ocupam muita memória preciosa no seu computador. Quando você usa variáveis grandes desnecessariamente, é como usar um silo de grãos para armazenar um pacote de cinco quilos de farinha. A memória do computador, assim como uma extensão de terra definida, é limitada. Se você construir um silo toda vez que comprar um saco de farinha, eventualmente usará toda a sua terra para isso. Gaste algum tempo fazendo um uso sensato dos seus recursos; determine o menor tipo de variável possível que você pode usar — depois use-a.

Não use uma variável maior do que você precisa.

Alteração do conteúdo de uma variável

A palavra "variável" por si só é uma dica para a utilidade das variáveis: seu conteúdo pode variar dependendo das necessidades do programa ou do usuário. Você pode mudar o conteúdo de uma variável simplesmente atribuindo um novo valor à mesma. Lembre-se, porém, que o *valor corrente* de uma variável é o valor que foi atribuído por último.

Prática: Alterando o valor de uma variável

1. Selecione o comando New pelo menu File.
2. Digite e execute o programa a seguir, que declara e depois muda o valor de uma variável:

```
CLS
fruta$="uva"
PRINT fruta$
fruta$ = "laranja"
PRINT fruta$
fruta$ = "pêra"
PRINT fruta$
```

Sua tela de saída deverá mostrar o seguinte resultado:

```
uva
laranja
pêra
```

USO DA INFORMAÇÃO FORNECIDA PELO USUÁRIO EM UMA VARIÁVEL

Agora que você já se acostumou com variáveis, vejamos uma área da programação em QBasic que se baseia bastante no uso de variáveis, a saber, a obtenção de caracteres da pessoa que usa o programa. Este processo também é chamado "leitura de caracteres pelo teclado".

Leitura de Caracteres com o Comando INPUT

O comando QBasic mais usado para a leitura de caracteres do teclado é o comando INPUT. No entanto, você não pode usar um comando INPUT por si só. Ele deve ser seguido pelo nome da variável (incluindo o seu tipo) que vooê deseja atribuir aos caracteres a serem digitados pelo usuário.

Prática: Usando o comando INPUT

1. Selecione o comando New pelo menu File.

2. Digite e execute o seguinte programa:

```
CLS
PRINT "Digite uma palavra e pressione Enter"
INPUT palavra$
PRINT "A palavra digitada foi "; palavra$
```

Sua tela de saída mostrará o resultado da operação:

```
Digite uma palavra e pressione Enter
? Bucareste
A palavra digitada foi Bucareste
```

Examine o que aconteceu:

- Primeiro, o seu programa apagou a tela e depois usou um comando PRINT para dizer ao usuário o que fazer.
- Após as instruções do comando PRINT, você digitou uma palavra e pressionou Enter. A palavra digitada foi atribuída pelo comando INPUT à variável *palavra$*.
- O comando PRINT final usou uma cadeia e a variável *palavra$* para mostrar o que você digitou.

Solicitação de entrada do usuário

O programa anterior usou um comando PRINT para pedir ao usuário para fornecer informações. Entretanto, você pode fazer com que a mensagem pedindo entrada seja parte do seu comando INPUT. Isso força o comando INPUT a fazer um duplo serviço: ele tanto pede a entrada ao usuário *quanto* lê o que o usuário digita.

Prática: Inserindo uma mensagem em INPUT

Mude o programa que você acabou de digitar para isto:

```
CLS
INPUT "Digite uma palavra e pressione Enter"; palavra$
PRINT "A palavra digitada foi "; palavra$
```

Após executar o programa, a sua tela de saída deverá ser semelhante a esta:

```
Digite uma palavra e pressione Enter? Bucareste
A palavra digitada foi Bucareste
```

Observe as diferenças entre este programa e aquele entrado anteriormente:

- Como a solicitação feita ao usuário faz parte do comando INPUT, o cursor espera pela entrada na mesma linha do pedido, em vez de numa linha separada.
- O comando INPUT mostra um ponto de interrogação ao final da mensagem. (Para eliminá-lo, basta trocar o ponto-e-vírgula ao final da mensagem de INPUT por uma vírgula e reexecutar o programa.)

Prática: Criatividade com INPUT

Agora que você já sabe um pouco sobre variáveis e como obter informações pelo teclado usando INPUT, poderá colocar um pouco de tempero nos seus programas.

1. Selecione o comando New pelo menu File.
2. Digite e execute o seguinte programa:

```
CLS
INPUT "Digite seu primeiro nome e pressione Enter:",
nome$
PRINT
INPUT "Qual a sua idade (prometo não espalhar)? ",
idade%
PRINT
INPUT "Quanto você tem de trocado no bolso? CR$",
dinheiro!
PRINT
PRINT "Obrigado, "; nome$; ". Você disse que tem";
idade%
PRINT "anos e possui CR$"; dinheiro!; "de trocado."
```

Depois de rodar o programa, a sua tela de saída ficará semelhante a esta:

```
Digite seu primeiro nome e pressione Enter: Paulo

Qual a sua idade (prometo não espalhar)? 45

Quanto você tem de trocado no bolso? CR$116.50

Obrigado, Paulo. Você disse que tem 45
anos e possui CR$ 116.50 de trocado.
```

OPERADORES MATEMÁTICOS DO QBASIC

O QBasic oferece vários símbolos que você pode usar para realizar cálculos matemáticos. Estes símbolos, chamados *operadores*, lhe permitem realizar tarefas como adição, subtração, multiplicação e divisão.

Vários operadores do QBasic são iguais àqueles que você usa normalmente. Por exmplo, no QBasic você usa + para adição e – para subtração. Outros operadores do QBasic são representados por símbolos especiais. A tabela a seguir mostra os operadores que você pode usar no QBasic:

Operador	*Operação matemática*
+	Adição
–	Subtração
*	Multiplicação
/	Divisão
\	Divisão inteira
MOD	Divisão com resto
^	Exponenciação (elevando a uma potência)

Os três últimos operadores (\, MOD e ^) são operadores de finalidade especial que descreveremos em seguida.

Comentando Seus Programas

Para ajudá-lo a entender o que o programa faz, o QBasic lhe permite colocar comentários dentro dos programas. Comentários são precedidos pelo comando REM ou pelo símbolo ', servindo apenas para o uso do programador. Comentários não aparecem na saída produzida pelo programa. Digite e execute o programa a seguir para ver como o comando REM funciona:

```
REM EXEMPLO.BAS
REM Programa exemplo mostrando o uso de comentários.
REM
REM Programadores: Marcos e Davi
REM Data: 1 de novembro de 1991

CLS

PRINT "Este é um programa com comentários!"
```

O programa produz a seguinte saída:

```
Este é um programa com comentários!
```

O símbolo ' também pode ser usado como equivalente mais discreto do comando REM, como vemos no programa a seguir:

```
' EXEMPLO.BAS
' Programa exemplo mostrando o uso de comentários.
'
' Programadores: Marcos e Davi
' Data: 1 de novembro de 1991

CLS

PRINT "Este é um programa com comentários!"
```

Quando você rodar um destes programas, ele produzirá o mesmo resultado:

```
Este é um programa com comentários!
```

Usaremos o segundo estilo de comentário nos programas deste livro. Ao escrever seus próprios programas, não se esqueça de incluir comentários não apenas para si mesmo, mas para outros que possam ter que estudar o seu programa algum dia.

Uso de Operadores do QBasic

Para realizar cálculos em QBasic, basta usar os operadores como faria na "vida real". Por exemplo, para adicionar os números 12 e 16, você digitaria

```
12 + 16
```

Entretanto, você não pode apenas colocar uma operação matemática isolada como esta em uma linha. Você pode fazer duas coisas:

- Atribuir o resultado da operação matemática a uma variável numérica:

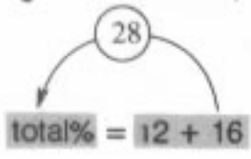
  ```
  total% = 12 + 16        ' Atribui o resultado a uma variável
  ```

- Usar o resultado da operação matemática como argumento para um comando do QBasic:

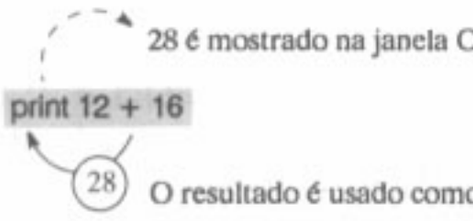
  ```
  PRINT 12 + 16           ' Imprime o resultado diretamente
  ```

Quando você atribuir uma operação matemática a uma variável, estará atribuindo o *resultado* da operação à variável, como mostramos a seguir:

O resultado é atribuído à variável.



Quando você usar uma operação matemática como argumento do comando, estará usando o *resultado* da operação como argumento do comando, como mostramos a seguir:



Prática: Trabalhando com operadores do QBasic

No Capítulo 2, você aprendeu que a janela Immediate é um local conveniente para testar uma linha de programa antes de colocá-la no seu programa. Usaremos a janela Immediate para experimentar os operadores do QBasic.

Uso da Janela Immediate

A janela Immediate lhe permite testar linhas de programa individuais antes que você as digite na janela View. As seguintes dicas o ajudarão a usar a janela View de forma mais eficaz:

- Você só pode executar uma linha de cada vez na janela Immediate.
- Para executar uma linha na janela Immediate, basta pressionar Enter. Você não precisa escolher Run.
- Você não pode salvar qualquer linha entrada na janela Immediate, a não ser que use os comandos de edição para mover as linhas para a janela View.

1. Pressione F6 para tornar a janela Immediate ativa e entre com a seguinte linha:

   ```
   CLS
   ```

2. Pressione qualquer tecla para retornar à janela Immediate, e depois entre com a seguinte linha:

   ```
   PRINT 7 * 7
   ```

   Sua tela de saída mostrará o resultado do comando PRINT:

   ```
   49
   ```

3. Pressione qualquer tecla para retornar à janela Immediate e depois entre com a seguinte linha:

   ```
   PRINT 75 / 6
   ```

A tela de saída mostrará o resultado do comando PRINT:

```
12.5
```

Os Operadores \, MOD e ^

O QBasic possui três operadores para uso especial: \, MOD e ^. Esses operadores oferecem complementos úteis para os operadores mais comuns de adição, subtração, multiplicação e divisão.

O operador \ (divisão inteira)

O operador de divisão inteira (\) trabalha exatamente como o operador de divisão padrão (/), mas quando divide um número por outro ele abandona a parte fracionária do resultado e retorna apenas a parte inteira.

***Prática: Usando o operador ***

Entre com a linha a seguir na janela Immediate:

```
PRINT 5 \ 2
```

Sua tela de saída mostrará o resultado da operação:

```
2
```

Enquanto a divisão padrão gera um resultado de 2.5 ou 2 com um resto de 1, a divisão inteira abandona o resto e gera o resultado 2.

O operador MOD

Embora MOD não se pareça com um operador típico, representado por um único símbolo, ele é um verdadeiro operador do QBasic.

O operador MOD também é chamado operador de *resto*. Seu resultado é oposto ao do operador de divisão inteira (\): Quando você divide um número por outro com MOD, o QBasic abandona a parte inteira do resultado e retorna apenas o resto.

Prática: Usando o operador MOD

Entre com a linha a seguir na janela Immediate:

```
PRINT 5 MOD 2
```

Sua tela de saída mostrará o resultado da operação:

```
1
```

Novamente, com a divisão padrão o resultado desta operação seria 2 com um resto de 1. Como você usou o operador MOD, o QBasic abandonou a parte inteira do resultado e retornou apenas o resto.

Regras Diferentes para a Divisão

O operador de divisão padrão (/) divide totalmente um número por outro. Por exemplo, o comando a seguir mostra um resultado de 2.5 — o mesmo resultado fornecido por uma calculadora:

```
PRINT 5 / 2
```

Os operadores \ e MOD, no entanto, usam o método mais tradicional, ou mais "longo". Ou seja, os resultados dados por esses dois operadores levam em consideração uma parte dividida exatamente e uma parte com o resto.

O operador ^ (exponenciação)

O operador de exponenciação (^) lhe permite elevar um número a uma potência de si mesmo. Por exemplo, o equivalente do QBasic de 10^3 (dez à terceira potência) é representado assim:

```
10 ^ 3
```

Prática: Usando o operador ^

Entre com a linha a seguir na janela Immediate:

```
PRINT 10 ^ 3
```

A sua tela de saída mostrará o resultado da operação:

```
1000
```

Expressões Numéricas

Uma *expressão numérica*, comumente chamada fórmula, é simplesmente uma combinação de números, variáveis numéricas, operadores numéricos e funções numéricas que, coletivamente, geram um único resultado. Os exercícios de multiplicação, divisão e exponenciação que você fez foram exemplos simples de expressões numéricas.

O QBasic lhe permite criar uma grande faixa de expressões numéricas, desde as mais simples, que você já praticou, até as mais complexas. Melhor que isso, a criação de expressões numéricas não é complicada. Tudo o que você precisa fazer é aprender algumas regras básicas, e poderá calcular quase tudo.

Uso de mais de um operador

Todos os exemplos com que você trabalhou usavam um único operador. O QBasic lhe permite usar mais de um operador na mesma

expressão numérica, permitindo que vários cálculos sejam efetuados ao mesmo tempo.

Prática: Uso de mais de um operador

Entre com o comando a seguir na janela Immediate:

```
PRINT 14 + 26 + 15.75 - 33.2
```

A tela de saída deverá mostrar o resultado da operação:

22.55

Como você pode ver, a janela Immediate pode ser uma calculadora muito prática!

Ordem de cálculo

O QBasic segue um conjunto estrito de normas quando calcula uma expressão numérica contendo mais de um operador. Considere a seguinte expressão numérica:

*3 + 4 * 5*

Qual é o resultado desta expressão? Na realidade, existem dois resultados diferentes, dependendo de como você a calcula. Se você realizar primeiro a adição, a resposta será 35. Se você realizar primeiro a multiplicação, a resposta será 23.

Para evitar confusão, o QBasic calcula as operações na seguinte ordem:

- A exponenciação (^) é realizada em primeiro lugar.
- A multiplicação e a divisão (*, /, \ e MOD) são realizadas em seguida.
- A adição e a subtração (+ e –) são realizadas por último.

Essas regras, no entanto, não cobrem todas as circunstâncias. E se as expressões tiverem operadores do mesmo tipo ou "nível de importância"? Por exemplo:

*3 * 5 MOD 2*

ou

*100 / 4 * 3*

Quando o QBasic encontra essas expressões, ele as calcula da esquerda para a direita.

Prática: Trabalhando com a precedência dos operadores

Entre com o seguinte comando na janela Immediate:

```
PRINT 3 + 4 * 5
```

Sua tela de saída deverá mostrar o resultado da operação:

```
23
```

Como o QBasic realiza a multiplicação antes da adição, o resultado será 23. Pressione qualquer tecla para retornar à janela Immediate e depois entre com o seguinte comando:

```
PRINT 100 / 4 * 3
```

Sua tela de saída mostrará o resultado da operação:

```
75
```

Como os operadores de divisão (/) e multiplicação (*) possuem o mesmo peso — ou seja, a mesma prioridade na lista de precedência dos operadores — o QBasic calcula a expressão da esquerda para a direita.

Uso de parênteses para controlar a ordem do cálculo

Considere a expressão numérica vista anteriormente:

*3 + 4 * 5*

Como já vimos, o QBasic realiza a multiplicação antes da adição, resultando no valor 23. E se você *quisesse* que a adição fosse calculada antes da multiplicação?

O QBasic lhe permite controlar como uma expressão numérica é calculada. Se você colocar parte de uma expressão numérica entre *parênteses*, o QBasic realizará essa parte do cálculo antes de qualquer outra. Por exemplo, a expressão numérica seguinte produz um resultado de 35, pois o QBasic calcula o conteúdo entre parênteses antes de calcular o restante da expressão:

*(3 + 4) * 5*

Se você usar dois ou mais operadores dentro do mesmo conjunto de parênteses, o QBasic calculará o conteúdo dos parênteses usando as normas comuns. Por exemplo, a expressão numérica a seguir produz um resultado de 23:

*(3 + 4 * 5)*

Prática: Usando parênteses para controlar a ordem de cálculo de uma expressão numérica

1. Entre com o seguinte comando na janela Immediate:

```
PRINT 3 + 4 * 5
```

Sua tela de saída mostrará o resultado da operação:

```
23
```

2. Pressione qualquer tecla para retornar à janela Immediate, depois entre com o seguinte comando:

```
PRINT (3 + 4) * 5
```

Sua tela de saída mostrará o resultado da operação:

```
35
```

Uso de parênteses dentro de parênteses

Se você precisar usar uma expressão numérica que exija mais controle do que pode ser oferecido pelos conjuntos de parênteses separados, você poderá usar *parênteses embutidos* — ou seja, um conjunto de parênteses dentro de outro.

Considere a seguinte expressão numérica:

*2 + 3 * 4 ^ 2*

O QBasic calcularia primeiro a exponenciação, depois a multiplicação e, finalmente, a adição. Mas e se você quisesse executar primeiro a adição, depois a multiplicação, e depois a exponenciação? Se você usasse

*(2 + 3 * 4) ^ 2*

o QBasic calcularia primeiro o conteúdo entre parênteses e faria a exponenciação por último, mas ainda faria a multiplicação antes da adição. A resposta seria usar um conjunto de parênteses embutido, como este:

*((2 + 3) * 4) ^ 2*

Quando o QBasic encontra parênteses embutidos, ele calcula o conteúdo entre os parênteses mais internos antes de calcular o conteúdo dos parênteses externos. No exemplo mostrado, o QBasic realizaria primeiro a adição, pois está dentro dos parênteses mais internos, depois a multiplicação e, por último, a exponenciação.

Funções Matemáticas do QBasic

Como vimos no Capítulo 3, uma função retorna um valor a um programa. O QBasic oferece várias funções matemáticas. (Como qualquer outra função do QBasic, você deverá usar essas funções dentro de um comando do QBasic.)

O símbolo *n* na tabela seguinte representa o número, variável numérica ou expressão sobre os quais você deseja que a função opere. Observe que *n* está cercado por parênteses. Você deverá usar esses parênteses sempre que oferecer um valor como argumento para uma função.

Função	*Finalidade*
ABS(*n*)	Retorna o valor absoluto de *n*
ATN(*n*)	Retorna o arco-tangente, em radianos, de *n*
COS(*n*)	Retorna o cosseno do ângulo *n* expresso em radianos
EXP(*n*)	Retorna o logaritmo de *n*
SGN(*n*)	Retorna –1 se *n* for menor que zero, retorna 0 se *n* for zero e retorna +1 se *n* for maior que zero
SIN(*n*)	Retorna o seno do ângulo *n* expresso em radianos
SQR(*n*)	Retorna a raiz quadrada de *n*
TAN(*n*)	Retorna a tangente do ângulo *n* expresso em radianos

Prática: Trabalhando com funções numéricas do QBasic

Entre com o seguinte comando na janela Immediate:

```
PRINT SQR(36)
```

A sua tela de saída deverá mostrar o resultado da operação, neste caso a raiz quadrada de 36:

```
6
```

Uso de funções matemáticas em expressões numéricas

Você pode usar funções matemáticas em uma expressão numérica junto com outros operadores do QBasic. No exemplo a seguir, a função SQR soluciona um problema envolvendo um triângulo retângulo.

Prática: Usando a função SQR

Suponha que você tenha um poste de 4 metros de altura e uma estaca de metal presa ao chão a 4,6 metros da base do poste (Figura 4-4). Você pretente prender um cabo do topo do poste até a estaca, mas não tem certeza do tamanho que deverá comprar.

Você pode escrever um programa em QBasic usando a função SQR para calcular o tamanho do cabo. A fórmula para o cálculo é

$$\sqrt{altura^2 + distância^2}$$

1. Pressione a tecla de função F6 para ativar a janela View.
2. Selecione o comando New pelo menu File e depois digite o programa a seguir.

```
CLS

PRINT "Este programa calcula quanto cabo é preciso
para"
PRINT "prender o topo de um poste a uma estaca no
chão."
PRINT
INPUT "Entre com a altura, em metros, do poste: ",
altura!
PRINT
PRINT "Entre com a distância, em metros, da base do"
INPUT "poste à estaca no chão: ", distancia!

cabo! = SQR((altura! ^ 2) + (distancia! ^ 2))

PRINT
PRINT "Você deverá comprar"; cabo!; "metros de cabo."
```

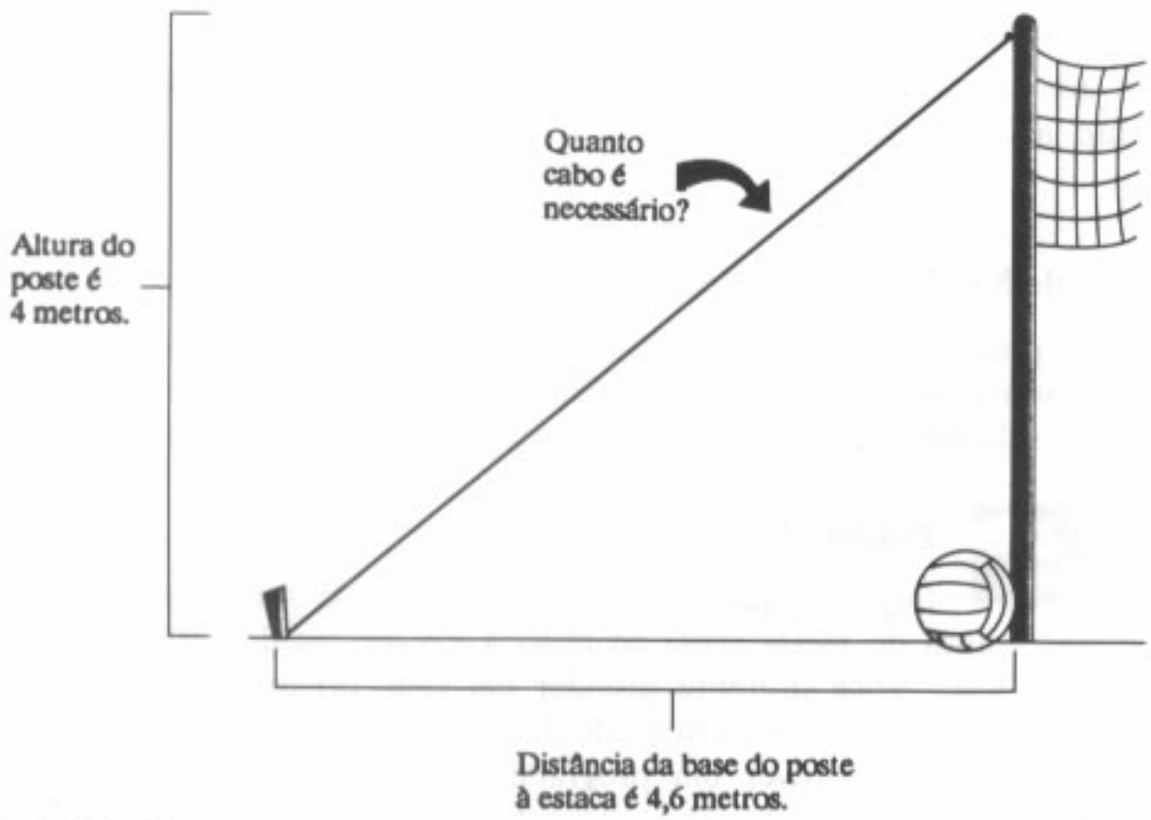


FIGURA 4-4.
Descobrindo o tamanho do cabo.

3. Execute o programa e entre com as medidas da altura e da distância, dadas na Figura 4-4 — 4 e 4.6, respectivamente. A sua saída ficará assim:

```
Este programa calcula quanto cabo é preciso para
prender o topo de um poste a uma estaca no chão.
```

```
Entre com a altura, em metros, do poste: 4

Entre com a distância, em metros, da base do
poste à estaca no chão: 4 . 6

Você precisa comprar 19.52562 metros de cabo.
```

Observe que o programa usa variáveis de ponto flutuante com precisão simples o tempo inteiro. Usando este tipo de variável, você não estará limitado ao uso de números inteiros por parte do usuário.

Além disso, observe que a linha

```
cabo! = SQR((altura! ^ 2) + (distancia! ^ 2))
```

usa parênteses embutidos. Neste caso, devido às regras de precedência do QBasic, os parênteses embutidos não são realmente necessários. Você obteria o mesmo resultado se não os tivesse usado. Entretanto, geralmente você poderá descobrir que suas fórmulas ficam mais fáceis de se ler quando parênteses extras são usados dessa forma.

4. Execute o programa novamente, dessa vez usando medidas de altura e distância escolhidas por você mesmo. (Se você realmente fosse usar um programa como este, precisaria calcular alguma folga na linha, bem como uma distância adicional para amarrar o cabo.)

NOTA: Se você tivesse incluído as medidas como parte do seu programa, como em

```
altura! = 4
distancia! = 4.6
```

teria que mudar o próprio programa se quisesse usar um outro conjunto de medidas. Permitindo que as medidas sejam entradas pelo usuário, no entanto, você pode rodar o programa várias vezes sem ter que mudar coisa alguma.

O COMANDO CONST

Quando você tiver um valor que nunca mudará em todo o programa, como uma taxa de vendas local ou estadual, ou uma taxa de desconto para compra à vista, declare-o como uma constante com o comando CONST.

Para usar o comando CONST, simplesmente coloque a palavra-chave antes da declaração da constante. Por exemplo, o comando a seguir diz ao QBasic que você quer que a constante de ponto flutuante com precisão simples chamada *TAXA!* sempre represente o valor de 0.081:

```
CONST PTAXA! = .081
```

Se você tentar atribuir outro valor a *TAXA!* mais adiante no programa, o QBasic mostrará o seguinte quadro de diálogo ao tentar rodar o programa:

```
Duplicate definition
< OK >   < Help >
```

Prática: Declarando constantes com o comando CONST

Suponha que você queira calcular o preço *real* de um ou mais itens — ou seja, o preço do item mais a taxa pela venda. Você poderá escrever um programa em QBasic para ajudá-lo a fazer isso de forma rápida e eficaz.

1. Selecione o comando New pelo menu File.
2. Digite e execute o seguinte programa:

```
CLS

CONST PTAXA! = .081        ' Taxa de vendas em Seattle

PRINT "Entre com o preço do item. Favor não usar"
INPUT "o sinal de cifrão ou a vírgula: CR$", valor!

taxa! = valor! * PTAXA!    ' Calcula o valor da taxa
total! = valor! + taxa!    ' Calcula o custo total
PRINT
PRINT "O custo do item é CR$"; valor!
PRINT "A taxa de vendas seria CR$"; taxa!
PRINT "- - - - - - - - - - - - - - - - - - - - - - -"
PRINT "O custo total seria CR$"; total!
```

A sua tela de saída ficaria como esta:

```
Entre com o preço do item. Favor não usar
o sinal de cifrão ou a vírgula: CR$2500

O custo do item é CR$ 2500
A taxa de vendas seria CR$ 202.5
- - - - - - - - - - - - - - - - - - - - -
O custo total seria CR$ 2702.5
```

Você pode rodar este programa quantas vezes desejar. Se tiver muitos itens para calcular, verá que o programa é muito mais rápido e preciso do que uma calculadora.

Este programa também demonstra que você pode usar variáveis e constantes para criar novas variáveis. Observe as linhas

```
taxa! = valor! * PTAXA!     ' Calcula o valor da taxa
total! = valor! + taxa!     ' Calcula o custo total
```

Não existem números envolvidos aqui, somente variáveis e constantes numéricas. Como o QBasic trata as variáveis e constantes numéricas como trataria os números, você poderá usá-los em qualquer lugar onde normalmente usaria um número.

RESUMO

Parabéns! Você já deu os principais passos na sua jornada para se tornar um programador em QBasic. As variáveis (tanto de cadeia quanto numéricas) e os operadores são tópicos importantes — tópicos que aparecem em quase todos os programas em QBasic. No capítulo seguinte, você dará alguma inteligência aos seus programas, permitindo que tomem decisões por conta própria, com base em conjuntos de regras indicadas por você.

PERGUNTAS E EXERCÍCIOS

1. Quais são os quatro tipos de variáveis numéricas do QBasic e como elas diferem entre si?
2. Por que o QBasic coloca um espaço na frente de um número positivo?
3. O que significa quando o QBasic mostra um quadro de diálogo *Expected: end-of-statement*?
4. O que significa quando o QBasic mostra um quadro de diálogo *Overflow* ou uma mensagem de erro?
5. Que tipo de variável você usaria para cada um dos seguintes números?

a. -32679	d. 14.000001	g. 268110
b. 12.3774	e. -1286.0	h. –10.222222
c. 142286.9	f. -268.0005	i. –.0000001

6. Quais são as diferenças entre a divisão normal com o operador /, a divisão inteira com o operador \ e a divisão com resto, que usa o operador MOD?
7. Da maior à menor prioridade, qual é a ordem de precedência que o QBasic aplica aos operadores matemáticos?
8. Qual é o resultado deste cálculo?

   (((5 + 8) – (1 + 3) / 4) * ((7 – 2) ^ 2))
9. Escreva um programa (completo, com comentários) que calcule e imprima os seguintes valores:

   □ ABS(–10)+5

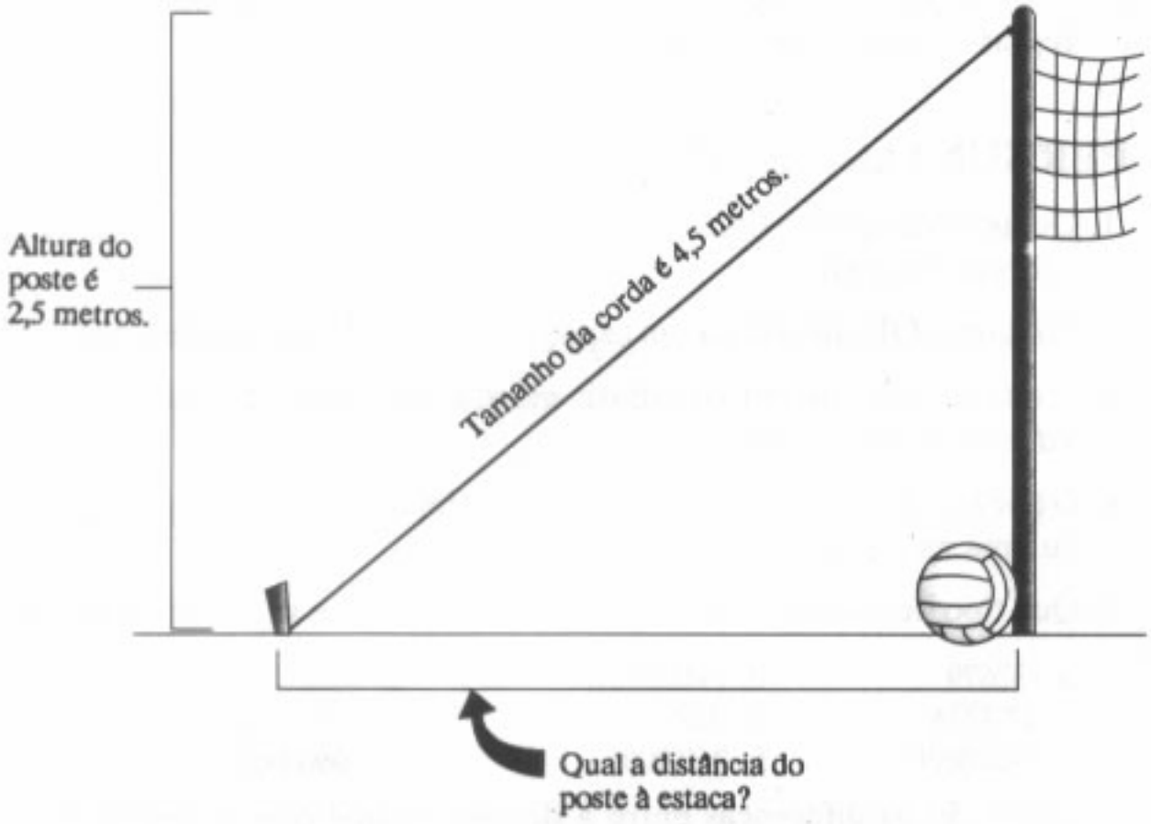
- SQR(36)
- SQR(4)^2
- COS(3.141592654)

10. O valor da constante matemática π pode ser aproximado para 3.141592654. A fórmula para a circunferência de um círculo é

    $2 \times \pi \times raio$

    (*raio* é a distância do centro do círculo até a sua borda.) Escreva um programa que peça o raio de um círculo ao usuário e depois mostre uma mensagem informando a circunferência do mesmo.

11. (BRINDE) Você está armando uma rede de vôlei. O topo do poste está a 2,5 metros do solo e a corda ligada ao topo do poste tem 4,5 metros de comprimento:



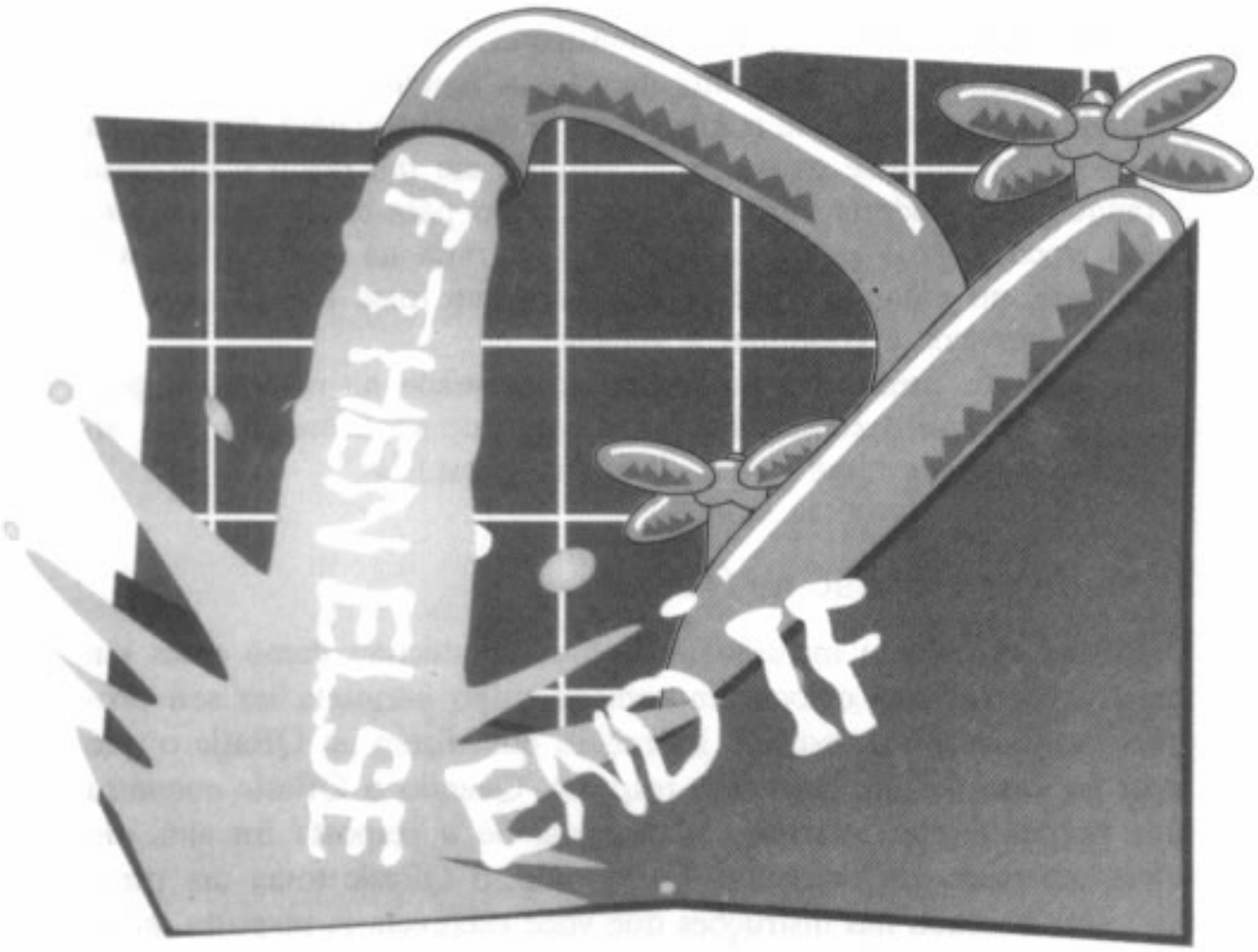
A fórmula para calcular a distância da base do poste até onde a estaca se encontra é

$$\sqrt{tamanho\_cadeia^2 - altur^2}$$

Escreva um programa em QBasic que peça ao usuário a altura do poste e o tamanho da corda e depois imprima a quantos metros de distância da base do poste deverá ficar a estaca.

CAPÍTULO 5

Controle de Fluxo do Programa



Os programas que você escreveu até aqui eram executados de forma bastante direta: o QBasic executa a primeira instrução e depois continua progressivamente até a última instrução. Às vezes, no entanto, você pode desejar que o QBasic execute certas partes do seu programa somente em algumas circunstâncias. Este capítulo lhe ensina como programar desta forma.

INTRODUÇÃO À TOMADA DE DECISÃO

Você toma milhares de decisões todo dia. Algumas delas exigem que você considere duas opções; outras exigem pouca ou nenhuma consideração. A decisão de tomar o café da manhã com frutas ou com gelatina poderia levar algum tempo, mas a decisão de levantar a mão e coçar a barba provavelmente será espontânea.

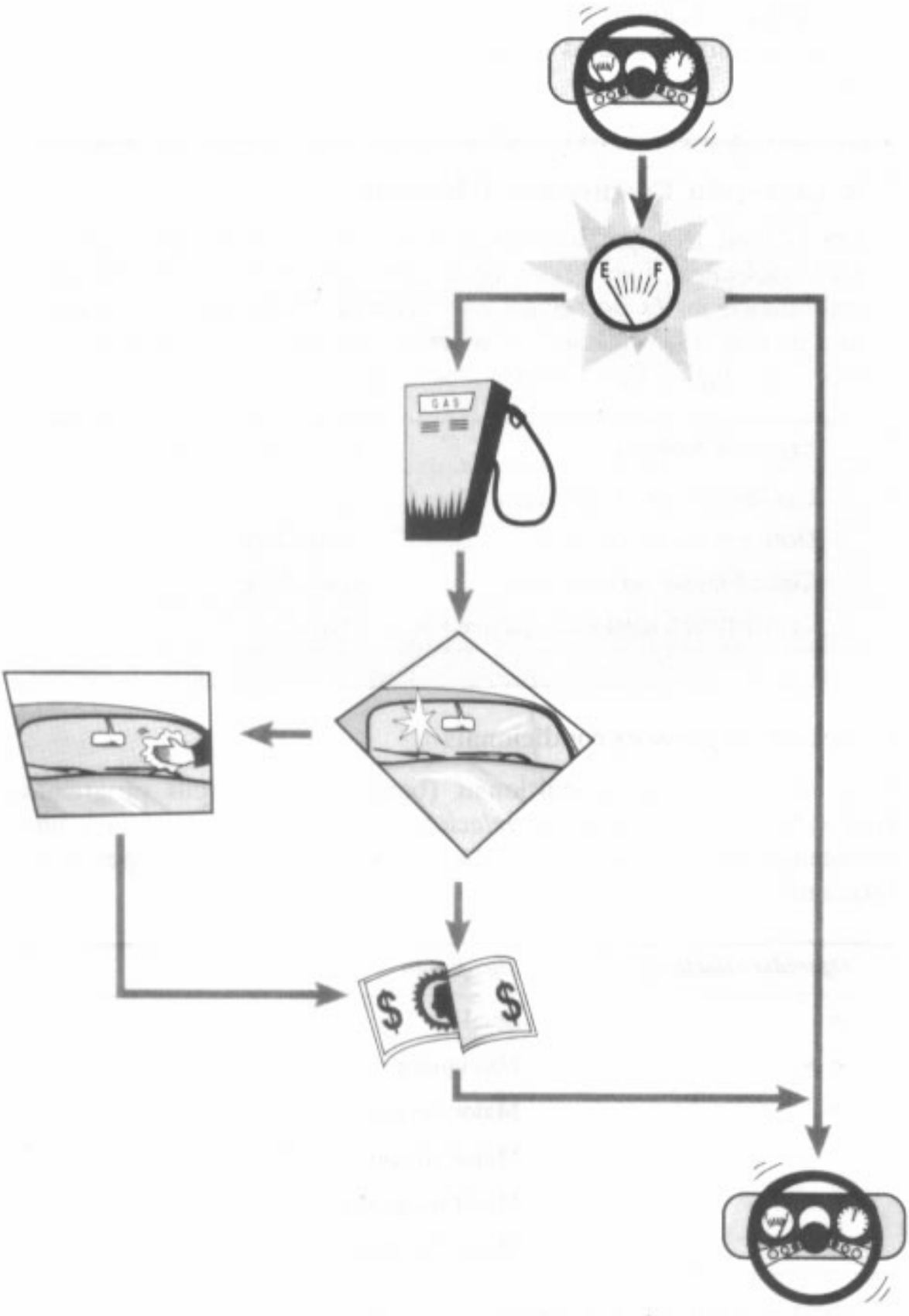
Outras decisões podem levar a grupos de ação separados. Por exemplo, se você estivesse em uma longa viagem de carro, precisaria ter certeza de que não faltará combustível. Vamos examinar um processo típico de tomada de decisão, que seria executado toda vez que você verificasse o medidor de combustível.

Como vemos no fluxograma da página seguinte, você verificaria o estado do seu tanque de combustível. Se estivesse cheio, continuaria dirigindo. Se estivesse quase vazio, entraria em um posto e encheria o tanque. Lá, poderia verificar o estado do pára-brisa. Se estivesse limpo, você pagaria pelo combustível e entraria na estrada. Se estivesse sujo, no entando, você pediria para que fosse lavado antes de pagar os serviços e continuar a viagem.

É fácil ver como uma única decisão, baseada na resposta à pergunta *Tenho pouco combustível?*, faz com que você escolha uma alternativa e depois tome outras decisões necessárias.

Tomada de Decisão no QBasic

O QBasic permite a inclusão de pontos de decisão como estes em um programa. Você coloca um comando tipo pergunta no seu programa e, junto a ele, escreve instruções que dizem ao QBasic o que fazer no caso de uma ou outra resposta. Quando o QBasic encontra uma pergunta, ele determina a resposta. Se a resposta for sim, ele segue um rumo em particular. Se for não, o QBasic toma um rumo diferente, baseado nas instruções que você escreveu. A resposta *talvez* nunca ocorre no QBasic ou em qualquer outra linguagem de programação. A resposta a uma pergunta é sempre *sim* ou *não*.



Tomada de decisão em ação: verificando o medidor de combustível.

Condições Verdadeiras e Falsas

Na verdade, você não coloca perguntas do tipo sim ou não em um programa QBasic. Em vez disso, você estabelece "perguntas" incluindo *expressões condicionais* nos seus programas. O QBasic avalia essas expressões condicionais — não para determinar respostas sim ou não, mas para determinar se as expressões condicionais são *verdadeiras* ou *falsas*.

A Expressão Condicional é Booleana

Em QBasic, as expressões condicionais são, na realidade, *expressões booleanas*. Expressões booleanas, cujo nome deriva-se do matemático inglês do século XIX George Boole, são expressões que podem ser avaliadas como *verdadeiras* ou *falsas*. Aqui estão alguns exemplos de expressões boolcanas:

Expressão booleana	*Avaliação*
Um litro é maior do que um galão.	Falso
Doze é maior do que dez.	Verdadeiro
Cinco é menor ou igual a seis.	Verdadeiro
Cem metros é igual a um quilômetro.	Falso

Criação de expressões condicionais

Para criar expressões condicionais (booleanas) nos seus programas, você deve usar um *operador relacional*, um *operador lógico* ou uma combinação dos dois tipos. O QBasic oferece os seguintes operadores relacionais:

Operador relacional	*Significado*
=	Igual a
< >	Não igual a
>	Maior do que
<	Menor do que
> =	Maior ou igual a
< =	Menor ou igual a

A tabela a seguir mostra alguns exemplos de expressões condicionais, que usam operadores relacionais, e os seus resultados:

Condição	*Resultado*
3 < 7	Verdadeiro (3 é menor do que 7)
14 > = 22	Falso (14 não é maior nem igual a 22)
11 < > 16	Verdadeiro (11 não é igual a 16)
5 = -5	Falso (5 positivo não é igual a 5 negativo)
12 > 14	Falso (12 não é maior do que 14)
11 > = 11	Verdadeiro (11 é maior ou igual a 11)
total% < 5	Verdadeiro se o valor de *total%* for menor do que 5; caso contrário, falso
num1% = num2%	Verdadeiro se o valor de *num1%* e *num2%* forem iguais; caso contrário, falso

Logo você terá uma chance de trabalhar com operadores relacionais e expressões condicionais. Por enquanto, vejamos alguns dos comandos do QBasic que lhe permitem usar essas expressões condicionais nos seus programas.

Expressões Numéricas e Condicionais

No capítulo anterior, você aprendeu sobre expressões *numéricas*. As expressões numéricas e as expressões condicionais são semelhantes: ambas consistem em um operador e dados. A diferença entre uma expressão numérica e uma expressão condicional é que uma expressão numérica usa um operador numérico (como +, -, / ou MOD) e uma expressão condicional usa um operador condicional (como >=, <, <> ou <=). Uma expressão numérica gera um resultado numérico, enquanto uma expressão condicional gera um resultado lógico (falso ou verdadeiro).

O COMANDO IF

O comando IF lhe permite avaliar uma condição, trabalhando em conjunto com a cláusula THEN para agir com base na avaliação.

Em sua forma mais simples, IF e THEN formam um único comando. A sintaxe de um comando condicional de única linha usando IF e THEN é a seguinte:

IF *condição* THEN *comando*

A parte *condição* do comando é uma das expressões condicionais que você acabou de ver. A parte *comando* é um outro comando do QBasic que é executado apenas se a *condição* for verdadeira. Se a *condição* for falsa, o QBasic ignorará o *comando* e seguirá para a próxima linha no seu programa. Eis um exemplo:

Se esta condição for verdadeira, o QBasic executa este comando.

```
IF numUsu% > 1000 THEN PRINT "O número é muito grande!"
```

Se esta condição for falsa, o QBasic ignora o resto da linha e executa o próximo comando no programa.

O comando após THEN pode ser qualquer comando do QBasic válido. Mas lembre-se de que o QBasic executa o comando após o THEN *apenas* se a condição após o IF for verdadeira.

Salvando Seus Programas de Prática

Deste ponto em diante, os programas deste livro aparecem em fundo reticulado e possuem nomes. Recomendamos que você digite esses programas enquanto prossegue, a fim de praticar os conceitos de programação do QBasic à medida que forem sendo discutidos. Quando tiver terminado, salve os programas em disco para futura referência. A construção de um grupo de programas próprios e a revisão de suas rotinas fará com que ganhe muito tempo ao realizar projetos de programação maiores.

Prática: Usando o comando IF

1. Digite o programa IF-1.BAS (Figura 5-1) e execute-o.

```
' IF-1.BAS
' Este programa demonstra o comando IF.

CLS

INPUT "Quantas vezes o Jornal Nacional vai ao ar
por semana? ", resp%
IF resp% = 6 THEN PRINT "Correto!!"
IF resp% <> 6 THEN PRINT "Sinto, mas a resposta é
6!"
PRINT "No domingo as notícias são no Fantástico."
```

FIGURA 5-1.
IF-1.BAS: Um programa demonstrando o comando IF.

2. Entre com o número *6*. A sua tela de saída deverá ficar como esta:

```
Quantas vezes o Jornal Nacional vai ao ar por semana? 6
Correto!!
No domingo as notícias são no Fantástico.
```

Como você entrou com o valor *6*, a condição no primeiro comando IF do programa foi verdadeira, de modo que o QBasic executou o comando PRINT ao final da linha. Além disso, como a condição no segundo comando IF *não* foi verdadeira, o QBasic não executou o comando PRINT ao final do segundo comando IF.

3. Execute o programa novamente, entrando com um valor diferente de 6. A sua tela de saída deverá mostrar algo assim:

```
Quantas vezes o Jornal Nacional vai ao ar por semana? 4
Sinto, mas a resposta é 6!
No domingo as notícias são no Fantástico.
```

Desta vez, o valor que você entrou fez com que ocorresse um conjunto diferente de ações. Como o valor *4* não é igual ao valor *6*, a condição do primeiro comando IF foi falsa, de modo que o QBasic ignorou o resto do comando. Entretanto, desta vez a condição no segundo comando IF foi verdadeira, de modo que o QBasic executou o restante desse comando.

Observe que nos dois casos o comando PRINT final do programa imprimiu a mensagem *No domingo as notícias são apresentadas pelo Fantástico*. Isso demonstra que, mesmo que a condição em um comando IF seja falsa, o QBasic executa o resto do programa como faria normalmente. Somente o comando após a parte THEN de um comando IF é ignorado se a condição não for verdadeira.

Uso de Mais de uma Condição com IF

No programa anterior, o QBasic avaliou uma única condição em cada comando IF. Você pode especificar múltiplas condições em um comando IF usando os *operadores lógicos* AND e OR. Você usa os operadores lógicos em expressões condicionais assim como usaria operadores matemáticos em expressões numéricas.

O operador lógico AND

O operador AND (e) lhe permite especificar múltiplas condições que devem ser verdadeiras antes que uma ação possa acontecer. Veja a linha de sintaxe para um comando IF que usa o operador AND:

IF *condição1* AND *condição2* THEN *comando*

Tanto *condição1* quanto *condição2* devem ser verdadeiros antes que o QBasic possa executar o *comando*. Veja um exemplo:

Se esta condição for verdadeira *e*
esta condição for verdadeira, o QBasic
executa este comando.

```
IF num1% > 10 AND num2% < 20 THEN PRINT "Correto!"
```

Se esta condição for falsa *ou* esta condição for falsa, ou se *ambas* forem falsas,
o QBasic ignora o resto da linha e executa o próximo comando no programa.

Observe que o comando IF tem duas condições e que, por estarem conectadas pelo operador lógico AND, tanto *num1% > 10* quanto *num2% < 20* devem ser verdadeiras para que o QBasic execute o comando PRINT após o THEN.

Prática: Trabalhando com o operador AND

1. Digite o programa IF-2.BAS (Figura 5-2) e execute-o.

```
' IF-2.BAS
' Este programa demonstra o operador lógico AND.

CLS

INPUT "Quantos adolescentes cabem dentro de uma
cabine de telefone? ", resp%
PRINT
IF resp% > 9 AND resp% < 13 THEN PRINT "Correto!!"
PRINT "Dependendo do tamanho, aproximadamente de 10
a 12"
PRINT "adolescentes cabem dentro de uma cabine de
telefone."
```

FIGURA 5-2.
IF-2.BAS: Um programa demonstrando o operador lógico AND.

2. Entre com o número *10*. Sua tela de saída deverá ficar como esta:

```
Quantos adolescentes cabem dentro de uma cabine de telefone? 10

Correto!!
Dependendo do tamanho, aproximadamente de 10 a 12
adolescentes cabem dentro de uma cabine de telefone.
```

Como você entrou com o valor *10*, as duas condições no comando IF foram verdadeiras. Se você rodar o programa novamente e entrar com um inteiro menor ou que 10 ou maior que 12, uma das condições será falsa, e o QBasic não executará o comando PRINT ao final do comando IF.

O operador lógico OR

O operador OR (ou) lhe permite criar um conjunto mais flexível de condições que devem ser atendidas antes que uma ação possa ocorrer. A linha de sintaxe para um comando IF usando o operador OR é a seguinte:

IF *condição1* OR *condição2* THEN *comando*

Observe que o comando IF contém duas condições. Basta *uma* dessas condições ser verdadeira para que o QBasic execute o comando THEN seguinte. O QBasic também executa o comando THEN seguinte se as duas condições forem verdadeiras. Veja um exemplo:

Se esta condição for verdadeira *ou*
esta condição for verdadeira, o QBasic
executa este comando.

```
IF quota% > 10 OR vendas% > 1000 THEN PRINT "Bom trabalho!"
```

Se as duas condições forem falsas, o QBasic ignora o resto da linha
e executa o próximo comando no programa.

Prática: Trabalhando com o operador OR

1. Digite o programa IF-3.BAS (Figura 5-3).

```
' IF-3.BAS
' Este programa demonstra o operador lógico OR.

CLS

PRINT "Estou pensando em um número de 1 a 100."
PRINT "Você consegue adivinhar qual é?"
INPUT "Entre com um número de 1 a 100: ", chute%
PRINT
IF chute% = 63 PRINT "Correto!!"
IF chute% < 53 OR chute% > 73 THEN PRINT "Nem
chegou perto!"
PRINT "Obrigado por jogar!"
```

FIGURA 5-3.
IF-3.BAS: Um programa demonstrando o operador lógico OR.

2. Execute o programa. Entre com o valor *63*, o número que o programa está "pensando". Sua tela de saída deverá mostrar o seguinte:

```
Estou pensando em um número de 1 a 100.
Você consegue adivinhar qual é?
Entre com um número de 1 a 100: 63
```

```
Correto!!
Obrigado por jogar!
```

3. Execute o programa novamente, mas use um número menor do que 53 ou maior do que 73. Sua tela de saída ficará assim:

```
Estou pensando em um número de 1 a 100.
Você consegue adivinhar qual é?
Entre com um número de 1 a 100: 76

Nem chegou perto!
Obrigado por jogar!
```

Veja o segundo comando IF. Entrando com um valor pelo menos 11 a mais ou a menos do que o número que o programa estava "pensando", você fez com que o QBasic imprimisse a mensagem *Nem chegou perto!*

O operador lógico NOT

O operador NOT lhe permite *negar* uma condição. Se uma condição for falsa, o operador NOT tornará a condição verdadeira; se a condição for verdadeira, NOT a tornará falsa. Aqui está a sintaxe para um comando IF usando um operador NOT:

IF NOT *condição* THEN *comando*

NOT é útil quando você deseja executar um comando quando uma condição *não* é verdadeira. Veja um exemplo:

```
Se esta condição for falsa, o QBasic executará este comando.

IF NOT idade% >= 16 THEN PRINT "Você não pode votar."

Se esta condição for verdadeira, o QBasic ignorará o resto da
linha e executará o próximo comando no programa.
```

Uso de ELSE com IF e THEN

Agora você já sabe como fazer com que o QBasic avalie uma condição e execute uma ação se a condição for verdadeira ou falsa. Porém, e se você quiser que o QBasic escolha entre *duas* ações com base na condição?

Quando usado em conjunto com IF e THEN, ELSE lhe permite especificar duas ações separadas para o QBasic: uma ação (após o THEN) se a condição for verdadeira e outra ação (após o ELSE) se a condição for falsa.

Eis a sintaxe para um comando IF usando THEN e ELSE:

IF *condição* THEN *comando1* ELSE *comando2*

onde *condição* é a condição lógica que você quer que o QBasic avalie, *comando1* é o comando que o QBasic executará se a *condição* for verdadeira, e *comando2* é o comando que o QBasic executará se a *condição* for falsa. IF, THEN e ELSE devem aparecer na mesma linha de comando. Veja um exemplo:

Se esta condição for verdadeira, o QBasic executará este comando.

```
IF numUsu% > 1000 THEN PRINT "Verdadeiro" ELSE PRINT "Falso"
```

Se esta condição for falsa, o QBasic executará este comando.

Prática: Trabalhando com IF, THEN e ELSE

1. Digite o programa IF-4.BAS (Figura 5-4).

```
' IF-4.BAS
' Este programa demonstra o comando ELSE.

CLS

PRINT "Estou pensando em um número entre 1 e 5."
PRINT "Você consegue adivinhar qual é?"
INPUT "Entre com um número entre 1 e 5: ", chute%
PRINT
IF chute% = 3 THEN PRINT "Correto!" ELSE PRINT
"Lamento!"
PRINT "Obrigado por jogar!"
```

FIGURA 5-4.
IF-4.BAS: Um programa demonstrando a clausula ELSE.

2. Execute o programa e entre com o número *3*. Sua tela de saída deverá ficar assim:

```
Estou pensando em um número entre 1 e 5.
Você consegue adivinhar qual é?
Entre com um número entre 1 e 5: 3

Correto!
Obrigado por jogar!
```

   Como você entrou com o valor *3*, a condição foi verdadeira, de modo que o QBasic executou o comando PRINT após o THEN.

3. Rode o programa novamente, entrando com um valor diferente de 3. Sua tela de saída se parecerá com esta:

```
Estou pensando em um número entre 1 e 5.
Você consegue adivinhar qual é?
Entre com um número entre 1 e 5: 4
```

```
Lamento!
Obrigado por jogar!
```

Desta vez a condição não foi verdadeira, de modo que o QBasic executou o comando PRINT após o ELSE.

Criando Comandos Condicionais Maiores com END IF

Usar ELSE com IF e THEN é um meio prático de dar ao QBasic duas opções a seguir, dependendo da avaliação de uma condição em particular. Entretanto, como acabamos de ver, colocar toda essa informação em uma única linha não permite muito espaço para criatividade.

É aí que entra o END IF — você pode criar programas maiores colocando cada ação possível em uma linha separada. Veja a sintaxe para um comando IF que usa THEN, ELSE e END IF:

```
IF condição THEN
    comandos executados se a condição for verdadeira
ELSE
    comandos executados se a condição for falsa
END IF
```

Este tipo de comando IF consiste em várias linhas individuais, juntamente conhecidas como *bloco*. A *condição* do comando é a expressão condicional que você deseja que o QBasic avalie. A palavra-chave THEN termina a linha. THEN deve sempre aparecer na mesma linha de IF.

Se a *condição* for verdadeira, o QBasic

1. Executará os comandos entre THEN e ELSE
2. Saltará os comandos entre ELSE e END IF
3. Continuará a execução do programa

Você pode incluir qualquer quantidade de comandos entre THEN e ELSE.

Se a *condição* for falsa, o QBasic

1. Saltará os comandos entre THEN e ELSE
2. Executará os comandos entre ELSE e END IF
3. Continuará a execução do programa

Você pode incluir qualquer quantidade de comandos entre ELSE e END IF.

Veja um exemplo:

Se esta condição for verdadeira, o QBasic executará estes comandos.

```
IF escolha$ = "SIM" THEN
  PRINT "Sim senhor! O Ford 1958 é o seu carro!"
  PRINT "Potência aqui e ali, e pura diversão!"
ELSE
  PRINT "Talvez você devesse ver outro modelo"
  PRINT "de carro esportivo do ano de 1958."
END IF
```

Se esta condição for falsa, o QBasic executará estes comandos.

Observe como a endentação dos comandos relacionados facilita a leitura do programa. Você deverá adquirir o hábito de usar a endentação nos seus próprios programas.

Prática: Trabalhando com IF, THEN, ELSE e END IF

1. Digite o programa IF-5.BAS (Figura 5-5).

```
' IF-5.BAS
' Este programa demonstra o comando IF em bloco.

CLS

PRINT "Bem-vindo ao teste sobre automóveis!"
PRINT
PRINT "Em que ano Karl-Friedrich Benz testou o
primeiro"
INPUT "automóvel a gasolina bem-sucedido? ", ano%
PRINT
IF ano% = 1885 THEN
  PRINT "Correto! Você entende muito de carros!"
ELSE
  PRINT "Não, ele o dirigiu em Mannheim, Alemanha,"
  PRINT "em 1885. (Foi patenteado em 29/01/1886.)"
END IF
PRINT "Obrigado por usar o programa!"
```

FIGURA 5-5.
IF-5.BAS: Um programa demonstrando o comando IF em bloco.

2. Execute o programa. Entre com o valor *1885*, que é o valor que torna verdadeira a condição após o IF. Sua tela de saída deverá se parecer com esta:

```
Bem-vindo ao teste sobre automóveis!

Em que ano Karl-Friedrich Benz testou o primeiro
automóvel a gasolina bem-sucedido? 1885
```

```
Correto! Você entende muito de carros!
Obrigado por usar o programa!
```

3. Execute o programa novamente, mas desta vez entre com um valor diferente de 1885. Sua tela de saída se parecerá com esta:

```
Bem-vindo ao teste sobre automóveis!

Em que ano Karl-Friedrich Benz testou o primeiro
automóvel a gasolina bem-sucedido? 1776

Não, ele o dirigiu em Mannheim, Alemanha,
em 1885. (Foi patenteado em 29/01/1886.)
Obrigado por usar o programa!
```

Como você pode ver, o uso de END IF com IF, THEN e ELSE permite que você use blocos inteiros de comandos.

A Palavra-Chave ELSEIF

ELSEIF é semelhante a ELSE por fornecer uma ação alternativa se a *condição* for falsa. Com ELSEIF, no entanto, você fornece outra condição para o QBasic avaliar. Veja a sintaxe para um comando IF que usa ELSEIF:

```
IF condição1 THEN
    comandos executados se condição1 for verdadeira
ELSEIF condição2 THEN
    comandos executados se condição2 for verdadeira
ELSEIF condição3 THEN
    comandos executados se condição3 for verdadeira
...
ELSE
    comandos executados se todas as condições forem falsas
END IF
```

A linha com reticências entre ELSEIF e ELSE na sintaxe anterior indica que você pode ter mais comandos ELSEIF seguidos por outras condições, se desejar. Não existe limite ao número de comandos ELSEIF e condições associadas.

NOTA: Você deverá usar THEN ao final de um comando ELSEIF. O uso de ELSE é opcional. Nós o incluímos na sintaxe anterior para mostrar onde ele apareceria se você decidisse incluí-lo.

Se *condição1* for verdadeira, o QBasic

1. Executará os comandos nas linhas seguintes até encontrar o primeiro ELSEIF
2. Saltará até o END IF
3. Continuará a execução do programa

Se *condição1* for falsa, o QBasic saltará para o primeiro ELSEIF e avaliará *condição2*. Se *condição2* for verdadeira, o QBasic

1. Executará os comandos nas linhas seguintes até encontrar o próximo ELSEIF ou, se você a tiver incluído, a cláusula ELSE

2. Saltará para o END IF e continuará a execução do programa

Se a condição associada com um comando ELSEIF for falsa, o QBasic saltará para o próximo comando ELSEIF e avaliará sua condição. O QBasic continuará neste processo até que um comando ELSEIF for avaliado como verdadeiro, ou até que encontre um comando ELSE.

Veja um exemplo:

Se esta condição for verdadeira, o QBasic executará este comando.

```
IF escolha% = 1 THEN
   PRINT "Você escolheu a opção 1!"
ELSEIF escolha% = 2 THEN
   PRINT "Você escolheu a opção 2!"
ELSEIF escolha% = 3 THEN
   PRINT "Você escolheu a opção 3!"
ELSE
   PRINT "Você não entrou com 1, 2 ou 3."
END IF
```

Se a primeira condição for falsa, e esta for verdadeira, o QBasic executa este comando.

Se a primeira e segunda condições forem falsas e esta condição for verdadeira, o QBasic executará este comando.

Se nenhuma dessas condições forem verdadeiras, o QBasic executará este comando.

Observe como o uso da palavra-chave ELSEIF lhe permite especificar diferentes condições e ações para o QBasic tomar.

Prática: Trabalhando com ELSEIF

1. Digite o programa ELSEIF.BAS (Figura 5-6).

```
' ELSEIF.BAS
' Este programa demonstra o uso de ELSEIF.

CLS

PRINT "Bem-vindo ao teste sobre cinema!"
PRINT "Em que ano o filme Ben Hur ganhou"
PRINT "o prêmio da academia pelo melhor filme?"
PRINT
INPUT "Favor entrar com um ano entre 1950 e 1959:
", ano%
```

FIGURA 5-6. (*continua*)
ELSEIF.BAS: Um jogo de pergunta sobre cinema usando ELSEIF.

FIGURA 5-6. *continuação*

```
PRINT
IF ano% = 1950 THEN
  PRINT "Errado. Em 1950, o prêmio da academia pelo"
  PRINT "melhor filme foi para All About Eve."
ELSEIF ano% = 1951 THEN
  PRINT "Errado. Em 1951, o prêmio da academia pelo"
  PRINT "melhor filme foi para An American in Paris."
ELSEIF ano% = 1952 THEN
  PRINT "Errado. Em 1952, o prêmio da academia pelo"
  PRINT "melhor filme foi para The Greatest Show on
Earth."
ELSEIF ano% = 1953 THEN
  PRINT "Errado. Em 1953, o prêmio da academia pelo"
  PRINT "melhor filme foi para From Here to Eternity."
ELSEIF ano% = 1954 THEN
  PRINT "Errado. Em 1954, o prêmio da academia pelo"
  PRINT "melhor filme foi para On the Waterfront."
ELSEIF ano% = 1955 THEN
  PRINT "Errado. Em 1955, o prêmio da academia pelo"
  PRINT "melhor filme foi para Marty."
ELSEIF ano% = 1956 THEN
  PRINT "Errado. Em 1956, o prêmio da academia pelo"
  PRINT "melhor filme foi para Around the World in 80
Days."
ELSEIF ano% = 1957 THEN
  PRINT "Errado. Em 1957, o prêmio da academia pelo"
  PRINT "melhor filme foi para The Bridge on the
River Kwai."
ELSEIF ano% = 1958 THEN
  PRINT "Errado. Em 1958, o prêmio da academia pelo"
  PRINT "melhor filme foi para Gigi."
ELSEIF ano% = 1959 THEN
  PRINT "Correto! Ben Hur, dirigido por William
Wyler,"
  PRINT "ganhou 11 prêmios da academia em 1959,
incluindo"
  PRINT "melhor filme."
ELSE
  PRINT "Você não entrou com um número de 1950 a
1959."
  PRINT "Favor rodar o programa novamente e entrar"
  PRINT "com um ano apropriado."
END IF
```

2. Execute o programa e entre com um valor de *1959*, que é a resposta correta à pergunta. Sua tela de saída se parecerá com esta:

```
Bem-vindo ao teste sobre cinema!
Em que ano o filme Ben Hur ganhou
o prêmio da academia pelo melhor filme?

Favor entrar com um ano entre 1950 e 1959: 1959

Correto! Ben Hur, dirigido por William Wyler,
ganhou 11 prêmios da academia em 1959, incluindo
melhor filme.
```

3. Execute o programa novamente, desta vez entrando com um valor ainda na faixa, mas diferente de 1959. Sua tela de saída mostrará algo assim:

```
Bem-vindo ao teste sobre cinema!
Em que ano o filme Ben Hur ganhou
o prêmio da academia pelo melhor filme?

Favor entrar com um ano entre 1950 e 1959: 1950

Errado. Em 1950, o prêmio da academia pelo
melhor filme foi para All About Eve.
```

4. Execute o programa mais uma vez, desta vez entrando com um valor fora da faixa de 1950 a 1959. Sua tela de saída será parecida com esta:

```
Bem-vindo ao teste sobre cinema!
Em que ano o filme Ben Hur ganhou
o prêmio da academia pelo melhor filme?

Favor entrar com um ano entre 1950 e 1959: 1596

Você não entrou com um número de 1950 a 1959.
Favor rodar o programa novamente e entrar
com um ano apropriado.
```

O COMANDO SELECT CASE

Uma outra ferramenta que lhe permite trabalhar com comandos condicionais é o comando SELECT CASE. O comando SELECT CASE é semelhante em função ao comando IF. Na verdade, em muitos casos você poderá usar tanto o comando IF quanto um comando SELECT CASE para realizar a mesma tarefa. Entretanto, as diferenças entre os dois tornam-nos mais adequados para certas circunstâncias em particular.

As Palavras-Chave CASE e END SELECT

Assim como o comando IF precisa de outras palavras-chave para realizar o seu trabalho corretamente, o comando SELECT CASE precisa

usar outras palavras-chave do QBasic — a saber, CASE e END SELECT.

Aqui está a sintaxe para um comando SELECT CASE (comandos relacionados foram endentados para dar mais clareza):

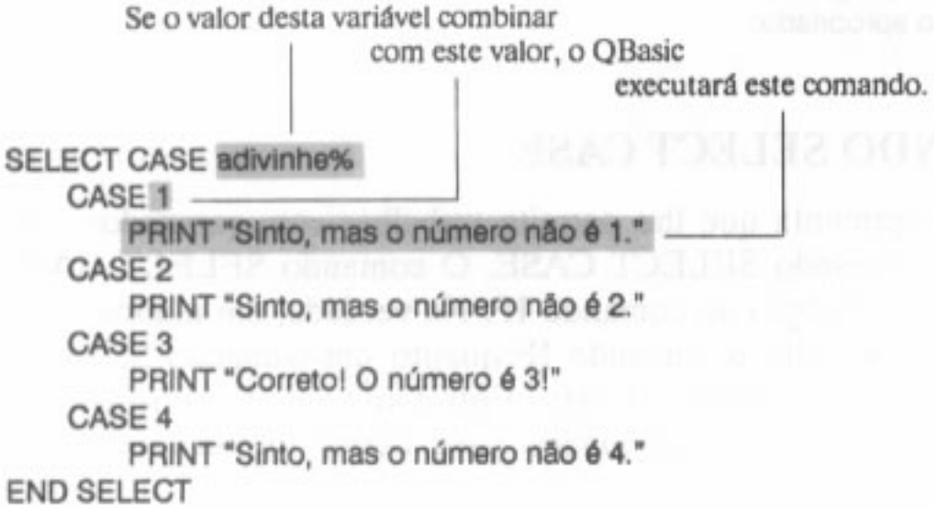
```
SELECT CASE variável
    CASE valor1
        comandos a executar se valor1 combinar com variável
    CASE valor2
        comandos a executar se valor2 combinar com variável
    CASE valor3
        comandos a executar se valor3 combinar com variável
    ...
END SELECT
```

Diferente de um comando IF, um comando SELECT CASE completo não pode ser especificado em uma única linha. Você deverá usar a sintaxe de bloco mostrada.

Variável pode ser uma variável numérica ou de cadeia. Você pode pensar nessa variável após o SELECT CASE como a "chave" para as cláusulas CASE individuais abaixo dela — o QBasic usa o valor da *variável* para determinar qual a cláusula CASE a usar.

As cláusulas CASE são comandos condicionais separados. O *valor* de cada uma está relacionado à *variável*. Não existe limite para o número de cláusulas CASE individuais que você pode colocar entre SELECT CASE e END SELECT.

Cada cláusula CASE é seguida por um ou mais comandos QBasic que o QBasic executará se o valor na cláusula CASE que os precede combinar com o valor da variável após o SELECT CASE. Veja um exemplo:



Quando o QBasic encontra um comando SELECT CASE, ele anota o valor da variável (neste exemplo, o valor de *adivinhe%*) e depois examina o valor especificado na primeira cláusula CASE. Se esse valor combinar com o valor da variável, o QBasic

1. Executará os comandos após esse CASE até que encontre o próximo CASE ou o comando END SELECT

2. Saltará para o comando após END SELECT e continuará executando o programa

Se o valor na primeira cláusula CASE não combinar com o valor da variável, o QBasic verificará o valor de cada cláusula CASE até que encontre um valor que combine.

Se nenhuma cláusula CASE tiver um valor igual ao valor da variável após SELECT CASE, o QBasic saltará para o comando após END SELECT e continuará executando o programa.

Prática: Trabalhando com o comando SELECT CASE

1. Digite o programa SELECT-1.BAS (Figura 5-7).

```
' SELECT-1.BAS
' Este programa pede ao usuário seu nome, imprime
' um menu, e pede que escolha um item do menu.
' Depois o programa mostra uma mensagem com base no
' item de menu escolhido.

CLS

INPUT "Favor entrar com seu nome: ", nomeUsu$
PRINT
PRINT "Parabéns, "; nomeUsu$; "! Você ganhou o"
PRINT "jogo de Controle Remoto!! Favor escolher"
PRINT "o prêmio que deseja:"
PRINT
PRINT , "1. Porta 1"
PRINT , "2. O que está atrás da cortina"
PRINT , "3. A Grande Caixa Rosada"
PRINT
INPUT "Favor entrar com 1, 2 ou 3: ", numMenu%
PRINT

SELECT CASE numMenu%
  CASE 1
    PRINT "** É um carro novo!!! **"
    PRINT "Sim, é um Ford do ano,"
    PRINT "com som e ar-condicionado!"
  CASE 2
```

FIGURA 5-7. *(continua)*
SELECT-1.BAS: Um programa demonstrando o comando SELECT CASE.

FIGURA 5-7. *continuação*

```
    PRINT "** 500 bolas de gude!!! **"
    PRINT "Sim, você e sua família inteira"
    PRINT "poderão se divertir a valer!"
  CASE 3
    PRINT "** Um estoque de picles em conserva!!! **"
    PRINT "Sim, doze mil jarras de picles em"
    PRINT "conserva entregues na porta de casa!"
END SELECT

PRINT
PRINT "Obrigado por disputar!"
```

2. Execute o programa para ver uma saída como esta:

```
Favor entrar com seu nome: Evaldo

Parabéns, Evaldo! Você ganhou o
jogo de Controle Remoto!! Favor escolher
o prêmio que deseja:

        1. Porta 1
        2. O que está atrás da cortina
        3. A Grande Caixa Rosada

Favor entrar com 1, 2 ou 3: 2

** 500 bolas de gude!!! **
Sim, você e sua família inteira
poderão se divertir a valer!

Obrigado por disputar!
```

Após ter entrado com um valor para *numMenu%* , o QBasic o usou para verificar o valor em cada cláusula CASE. Como o usuário entrou com o valor 2 neste caso, o QBasic encontrou uma combinação na segunda cláusula CASE. Depois, por ter encontrado uma combinação, o QBasic saltou sobre a cláusula CASE restante e executou o restante do programa.

3. Execute o programa novamente, mas desta vez use um valor diferente de 1, 2 ou 3. Sua tela de saída deverá se parecer com esta:

```
Favor entrar com seu nome: Deise

Parabéns, Deise! Você ganhou o
jogo de Controle Remoto!! Favor escolher
o prêmio que deseja:
```

```
1. Porta 1
2. O que está atrás da cortina
3. A Grande Caixa Rosada

Favor entrar com 1, 2 ou 3: 5

Obrigado por disputar!
```

Desta vez o QBasic não mostrou a mensagem do prêmio ganho, pois não conseguiu achar, nas cláusulas CASE, um valor que combinasse com o valor entrado.

Uso de CASE ELSE

Lembre-se de que, quando usada em um comando IF, a cláusula ELSE lhe permite especificar uma ou mais ações que serão executadas quando as expressões condicionais forem avaliadas como falsas. Você usa CASE ELSE para fazer a mesma coisa com um comando SELECT CASE.

A sintaxe de um comando SELECT CASE usando CASE ELSE é

```
SELECT CASE variável
    CASE valor
        comandos que serão executados se valor = variável
    ...
    CASE ELSE
        comandos que serão executados se valor <> variável
END SELECT
```

Prática: Usando um comando CASE ELSE

1. Digite o programa SELECT-2.BAS (Figura 5-8).

```
' SELECT-2.BAS
' Este programa demonstra o uso de CASE ELSE.

CLS

PRINT "Bem-vindos ao jogo de perguntas de TV"!
PRINT "Em que ano o famoso show I Love Lucy"
PRINT "apareceu inicialmente na televisão?"
PRINT
INPUT "Favor entrar com um ano de 1950 a 1959: ",
ano%
PRINT
SELECT CASE ano%
  CASE 1950
     PRINT "Errado. Em 1950, o programa popular"
     PRINT "Your Show of Shows estreou na TV."
```

FIGURA 5-8. (*continua*)
SELECT-2.BAS: Um jogo de perguntas de TV usando CASE ELSE.

FIGURA 5-8. *continuação*

```
  CASE 1951
    PRINT "Certo! I Love Lucy apareceu inicialmente"
    PRINT "em outubro de 1951. De 1952 a 1954 foi o
show"
    PRINT "de TV mais popular nos Estados Unidos."
  CASE 1952
    PRINT "Errado. Em 1952, o programa Dragnet"
    PRINT "fez sua estréia na televisão."
  CASE 1953
    PRINT "Errado. Em 1953, o programa Name That Tune"
    PRINT "fez sua estréia na televisão."
  CASE 1954
    PRINT "Errado. Em 1954, o programa Lassie"
    PRINT "fez sua estréia na televisão."
  CASE 1955
    PRINT "Errado. Em 1955, o programa Gunsmoke"
    PRINT "fez sua estréia na televisão."
  CASE 1956
    PRINT "Errado. Em 1956, o programa Playhouse 90"
    PRINT "fez sua estréia na televisão."
  CASE 1957
    PRINT "Errado. Em 1957, o programa Leave It to"
    PRINT "Beaver fez sua estréia na televisão."
  CASE 1958
    PRINT "Errado. Em 1958, o programa Peter Gunn"
    PRINT "fez sua estréia na televisão."
  CASE 1959
    PRINT "Errado. Em 1959, o programa The Twilight"
    PRINT "Zone fez sua estréia na televisão."
  CASE ELSE
    PRINT "Você não entrou com um número de 1950 a
1959."
    PRINT "Favor rodar o programa novamente e entrar"
    PRINT "com um valor apropriado."
END SELECT
```

2. Rode o programa e entre com o valor *1951* — a resposta correta à pergunta. Sua tela de saída deverá mostrar o seguinte:

```
Bem-vindo ao jogo de perguntas de TV!
Em que ano o famoso show I Love Lucy
apareceu inicialmente na televisão?

Favor entrar com um ano de 1950 a 1959: 1951

Certo! I Love Lucy apareceu inicialmente
em outubro de 1951. De 1952 a 1954 foi o show
de TV mais popular nos Estados Unidos.
```

3. Agora rode o programa novamente, mas desta vez entre com um valor fora da faixa correta. Sua tela de saída deverá mostrar algo assim:

```
Bem-vindo ao jogo de perguntas de TV!
Em que ano o famoso show I Love Lucy
apareceu inicialmente na televisão?

Favor entrar com um ano de 1950 a 1959: 1598

Você não entrou com um número de 1950 a 1959.
Favor rodar o programa novamente e entrar
com um valor apropriado.
```

Como o QBasic não encontrou uma cláusula CASE com o valor entrado, ele executou os comandos após CASE ELSE. O uso de CASE ELSE é um meio conveniente de enfrentar dificuldades não previstas nos seus comandos de SELECT CASE.

Uso de IS com CASE

A palavra-chave IS lhe permite usar uma *expressão* condicional (em vez de apenas um *valor* numérico ou de cadeia) em qualquer cláusula CASE. O QBasic só executa o comando após esse CASE se a condição for verdadeira. Eis a sintaxe para uma cláusula CASE usando IS:

```
SELECT CASE variável
    CASE IS condição
        comandos a serem executados se condição for verdadeira
    ...
END SELECT
```

Esta sintaxe é idêntica ao comando SELECT CASE padrão, exceto pelo acréscimo de "IS *condição*" na cláusula CASE. Quando estiver avaliando a condição, o QBasic usará o valor da variável de SELECT CASE como base da comparação.

Veja um exemplo:

Se o valor de *numUsu%* for menor ou igual a 4, o QBasic executará este comando.

Se o valor de *numUsu%* for 5, o QBasic executará este comando.

```
SELECT CASE numUsu%
    CASE IS <= 4
        PRINT "O número que você entrou foi 4 ou menor."
    CASE 5
        PRINT "O número que você entrou foi 5."
    CASE IS >= 6
        PRINT "O número que você entrou foi 6 ou maior."
END SELECT
```

Se o valor de *numUsu%* for maior ou igual a 6, o QBasic executará este comando.

O QBasic compara o valor entrado pelo usuário com as partes condicionais das cláusulas CASE. Neste exemplo, se o usuário entrasse com um valor 4 ou menor, a condição na primeira cláusula CASE seria verdadeira e, portanto, o QBasic executaria o comando PRINT associado. (A segunda cláusula CASE simplesmente usa um valor. Você pode misturar valores e expressões condicionais IS.)

Prática: Trabalhando com SELECT CASE e IS

1. Digite o programa SELECT-3.BAS (Figura 5-9).

```
' SELECT-3.BAS
' Este programa analisa o consumo diário de café.

CLS

INPUT "Quantos copos de café você beberá hoje? ",
coposCafe%
PRINT

SELECT CASE coposCafe%
  CASE 0
    PRINT "Não gosta do meu café??"
  CASE IS <= 3          ' 1 a 3 copos por dia
    PRINT "Um nível adequado."
  CASE IS <= 7          ' 4 a 7 copos por dia
    PRINT "Melhor reduzir a quantidade..."
  CASE IS >= 8          ' 8 ou mais copos por dia
    PRINT "Sobrecarga de cafeína!"
END SELECT
```

FIGURA 5-9.
SELECT-3.BAS: Um programa de "análise" de consumo de café usando IS.

2. Execute o programa e entre com um valor. A sua tela de saída ficará como esta:

```
Quantos copos de café você beberá hoje? 3

Um nível adequado.
```

Pressione uma tecla para retornar à janela View. No programa, observe como as condições são estabelecidas. Se você entrou com o valor *3* quando rodou o programa, como fizemos, terá visto que ele fez com que a segunda cláusula CASE fosse verdadeira. No entanto, observe também que tecnicamente o valor 3 faria com que a terceira cláusula CASE também fosse verdadeira, e você poderá questionar por que o QBasic não executou o comando PRINT associado. Lembre-se: Assim

que o QBasic encontra uma cláusula CASE com uma condição verdadeira, ele salta sobre as cláusulas CASE restantes sem ao menos considerá-las, e continua executando o seu programa.

Uso de TO com SELECT CASE

Para especificar uma faixa inclusiva de valores em uma cláusula CASE, use a palavra-chave TO. A sintaxe para usar TO em uma cláusula CASE é a seguinte:

```
SELECT CASE variável
    CASE valor1 TO valor2
        comandos a serem executados se CASE for verdadeiro
    ...
END SELECT
```

O dois valores na cláusula CASE podem ser dois valores numéricos ou duas cadeias de texto, sendo que estas deverão estar entre aspas.

Uso de valores numéricos com a palavra-chave TO

Aqui está um exemplo do uso de dois valores numéricos com a palavra-chave TO em uma cláusula CASE:

```
Se o valor de numUsu% for de 1 a 5,
inclusive, o QBasic executará este comando.


SELECT CASE numUsu%
    CASE 1 TO 5
        PRINT "O número entrado foi de 1 a 5."
    CASE 6 TO 10
        PRINT "O número entrado foi de 6 a 10."
END SELECT


Se o valor de numUsu% for de 6 a 10,
inclusive, o QBasic executará este comando.
```

Observe que o menor dos dois valores aparece à esquerda da palavra-chave TO. Você deverá listar os valores nesta ordem; se não o fizer, o QBasic não entenderá a expressão, retornando sempre um valor falso.

Isso também vale para números negativos. Eis um exemplo usando um número negativo na posição correta:

```
CASE -12 TO 3
```

Como 12 negativo é menor do que 3 positivo, você deverá colocar o 12 negativo à esquerda da palavra-chave TO.

Uso de cadeias de texto com a palavra-chave TO

Você também pode usar uma faixa de cadeias de texto em uma cláusula CASE se usar a palavra-chave TO. Eis um exemplo:

```
SELECT CASE palavra$
    CASE "a" TO "m"
        PRINT "A palavra entrada está na faixa de a a m."
    CASE "m" TO "z"
        PRINT "A palavra entrada está na faixa de m a z."
END SELECT
```

A princípio, a faixa de cadeias de texto pode não parecer tão óbvia quanto uma faixa de números, mas ambos funcionam de modo semelhante.

Lembre-se de que o QBasic trata cada caracter separadamente. Por exemplo, no que se refere ao QBasic, as letras *a* e *A* são diferentes.

O motivo para isto está no modo como o computador trabalha com as informações. Embora você digite letras maiúsculas e minúsculas no seu teclado, e embora o seu computador possa mostrar letras maiúsculas e minúsculas na tela, o computador não pode trabalhar diretamente com letras. Os microchips dentro do seu computador só trabalham com números. (E você pensava que os computadores fossem inteligentes....) Quando você pressiona uma tecla de caracter pelo teclado, seu computador traduz temporariamente esse caracter em um número com que possa trabalhar. Depois, quando precisar fazer algo com esse número na forma de caracter, como apresentá-lo na sua tela, o QBasic voltará com o número à forma de caracter inserida originalmente.

Relaxe — você não precisa aprender como funciona esta fase de numeração, mas precisa saber que ela acontece se pretende entender como o QBasic trabalha com caracteres individuais ou cadeias de caracteres. Mesmo que você manipule diretamente caracteres ou cadeias de caracteres, o seu computador e o QBasic simplesmente consideram-nos números.

No exemplo anterior, observe a superposição entre as duas cláusulas CASE. Na primeira cláusula CASE a faixa vai de *a* a *m*, e na segunda cláusula CASE ela vai de *m* a *z*. Por que a superposição? A sessão de prática a seguir demonstrará o motivo.

Prática: Trabalhando com uma faixa de cadeias de texto em uma cláusula CASE

1. Digite o programa TO-1.BAS (Figura 5-10).

```
' TO-1.BAS
' Este programa demonstra a palavra-chave TO
' em uma cláusula CASE.

CLS

INPUT "Favor entrar com um nome: ", nome$
PRINT

SELECT CASE nome$
  CASE "a" TO "m"
    PRINT "O nome está na faixa de a a m."
  CASE "m" TO "z"
    PRINT "O nome está na faixa de m a z."
  CASE "S" TO "M"
    PRINT "O nome está na faixa de A a M."
  CASE "M" TO "Z"
    PRINT "O nome está na faixa de M a Z."
END SELECT
```

FIGURA 5-10.
TO-1.BAS: Um programa de comparação de texto usando TO.

2. Execute o programa e entre com o nome *beto*. A sua tela de saída ficará assim:

   Favor entrar com um nome: `beto`

   O nome está na faixa de a a m.

   Como a primeira letra de *beto* é a letra *b* minúscula, isso fez com que o primeiro comando CASE fosse verdadeiro, pois *b* está dentro da faixa de *a* a *m*.

3. Execute o programa novamente, mas entre com a letra *m*. A sua tela de saída ficará assim:

   Favor entrar com um nome: `m`

   O nome está na faixa de a a m.

   Mais uma vez, como a letra *m* minúscula retornou um valor verdadeiro no primeiro CASE, o QBasic executou o comando após o primeiro CASE.

4. Rode o programa novamente, e entre com o nome *marcos*. A sua tela de saída ficará assim:

   Favor entrar com um nome: `marcos`

   O nome está na faixa de m a z.

Observe que a entrada da palavra *marcos* fez com que o segundo CASE fosse verdadeiro, mas no exemplo anterior a entrada somente da letra *m* fez com que o primeiro CASE fosse verdadeiro. Por que isso aconteceu?

Lembre-se de que, dentro de um computador, todas as letras são tratadas como números. De certa forma, então, você pode pensar em uma cadeia de caracteres como uma série de números. No exemplo anterior, a palavra *marcos* teve um "valor" maior do que a letra *m*, assim como o número 15 tem um valor maior do que o número 1. Embora os dois números — 1 e 15 — comecem com um 1, as regras da numeração e da matemática dizem que 15 tem um valor maior do que 1.

Esta regra também se aplica às cadeias de caracteres, mas existe uma diferença. No exemplo anterior, o QBasic não considerou a palavra inteira *marcos* quando examinou as cláusulas CASE. Ele precisava simplesmente examinar as duas primeiras letras — *ma*. Como a combinação de letras *ma* tem um "valor" maior do que a simples letra *m*, e como a letra *m* isolada está no topo da faixa de *a* a *m*, o QBasic não precisou verificar mais — ela já sabia que a primeira condição não era verdadeira.

Entretanto, o QBasic achou uma combinação na segunda condição. Embora a cadeia *marcos* seja maior do que a combinação em única letra que o QBasic estava procurando, a primeira letra de *marcos* cai na faixa de *m* a *z*, e, portanto, o QBasic determinou que a segunda condição era verdadeira.

Prática: Mais trabalho com uma faixa de cadeias

1. Modifique o programa TO-1.BAS de modo que se iguale ao programa TO-2.BAS (Figura 5-11).

```
' TO-2.BAS
' Este programa demonstra a palavra-chave TO
' em uma cláusula CASE.

CLS

INPUT "Favor entrar com um nome: ", nome$
PRINT

SELECT CASE nome$
  CASE "a" TO "mz"
    PRINT "O nome está na faixa de a a m."
```

FIGURA 5-11. (*continua*)
TO-2.BAS: Uma versão modificada de TO-1.BAS.

FIGURA 5-11. *continuação*

```
    CASE "n" TO "z"
      PRINT "O nome está na faixa de n a z."
    CASE "A" TO "Mz"
      PRINT "O nome está na faixa de A a M."
    CASE "N" TO "Z"
      PRINT "O nome está na faixa de N a Z."
END SELECT
```

TO-2.BAS é idêntico a TO-1.BAS, exceto por algumas mudanças nas condições — observe sobretudo a primeira e terceria cláusulas CASE.

2. Execute o programa e entre com a letra *M* maiúscula. (Para variar, usaremos letras maiúsculas desta vez.) A sua tela de saída deverá ficar assim:

```
Favor entrar com um nome: M

O nome está na faixa de A a M.
```

3. Agora execute o programa novamente, entrando com o nome *Marcos*. A sua tela de saída deverá ser a seguinte:

```
Favor entrar com um nome: Marcos

O nome está na faixa de A a M.
```

Desta vez, a condição na terceira cláusula CASE é verdadeira, pois *Mz* tem um "valor" maior do que as duas primeiras letras de *Marcos*, o nome que você inseriu.

Uso de Múltiplas Condições com CASE

Até aqui, a condição que você usou em cada cláusula CASE foi um único valor, condição ou faixa de valores. Para especificar múltiplas condições dentro de uma única cláusula CASE, basta separar as condições com vírgulas. Você verá um exemplo em seguida.

Se um desses valores combinar com o valor de *numero%*, o QBasic executará este comando.

```
SELECT CASE num%
        CASE 1, 3, 5, 7, 9
          PRINT "O número entrado foi ímpar."
        CASE 2, 4, 6, 8, 10
          PRINT "O número entrado foi par."
END SELECT
```

Se um desses valores combinar com o valor de num%, o QBasic executará este comando.

O uso de uma vírgula em uma cláusula CASE é semelhante ao uso do operador OR em um comando IF: Se um dos itens fizer com que o CASE seja verdadeiro, o QBasic executará os comandos associados.

Os itens não precisam ser do mesmo tipo: Você pode misturar valores, condições e faixas de valores. Por exemplo, a cláusula CASE a seguir é válida:

```
CASE 5, IS <> 6, 20 TO 30
```

Prática: Trabalhando com vírgulas e CASE

Digite o programa TO-3.BAS (Figura 5-12) e execute-o. Como você pode ver, o uso de vírgulas para especificar múltiplos itens permite maior flexibilidade nos seus comandos SELECT CASE.

```
' TO-3.BAS
' Este programa demonstra o uso de vírgulas com
' CASE.

CLS

PRINT "Entre com um mês, e lhe direi quantos"
PRINT "feriados nos E.U.A. existem nesse mês."
PRINT "Favor entrar com um número de 1 a 12: ", mes%
PRINT

SELECT CASE mes%
  CASE 8
    PRINT "Não existem feriados nos E.U.A. nesse mês."
  CASE 3, 4, 7
    PRINT "Existe 1 feriado nos E.U.A nesse mês."
  CASE 6
    PRINT "Existem 2 feriados nos E.U.A nesse mês."
  CASE 1, 2, 9 TO 11
    PRINT "Existem 3 feriados nos E.U.A nesse mês."
  CASE 12
    PRINT "Existem 4 feriados nos E.U.A nesse mês."
  CASE 5
    PRINT "Existem 5 feriados nos E.U.A nesse mês."
END SELECT
```

FIGURA 5-12.
TO-3.BAS: Um programa demonstrando o uso de múltiplas condições dentro de cláusulas CASE.

QUE TIPO DE COMANDO CONDICIONAL VOCÊ DEVE USAR?

Tanto o comando IF quanto SELECT possuem muitas opções e configurações diferentes. Quando você deseja incluir um ou mais comandos condicionais no seu programa, as respostas às seguintes perguntas ajudarão a escolher o comando mais apropriado às suas necessidades.

- **Que comando você está mais acostumado a usar?** Através da ajuda de palavras-chave associadas, como ELSE e ELSEIF com um comando IF, e CASE e CASE ELSE com um comando SELECT CASE, você poderá obter o mesmo resultado usando qualquer comando. Pense no uso do comando com que esteja mais acostumado.

- **Quantas condições existem no seu comando?** SELECT CASE exige um mínimo de três instruções separadas para estabelecer até mesmo uma única condição! Assim, se tiver apenas uma condição, um simples comando IF é a sua melhor escolha. Se o seu comando tiver duas ou mais condições, IF e SELECT CASE são ambos fáceis de se usar.

  Se tiver muitas condições, SELECT CASE é uma boa escolha. Ele é visualmente mais arejado e, portanto, mais fácil de ler e compreender do que um comando IF. Um comando SELECT CASE também é um bom candidato a situações em que você esteja procurando uma série de valores específicos que não se encontram em uma faixa bem definida.

RESUMO

Os comandos condicionais permitem que seus programas sejam muito mais flexíveis e "inteligentes". Usando comandos condicionais, você pode controlar como os seus programas são executados, com base na informação digitada pelo usuário ou na informação alterada pelo programa à medida que é executado. Você acabou de dar mais um passo em direção ao estado de programador em QBasic — usará comandos condicionais como parte primordial de quase todos os programas criados.

PERGUNTAS E EXERCÍCIOS

1. Quais as diferenças entre uma expressão numérica e uma expressão condicional?

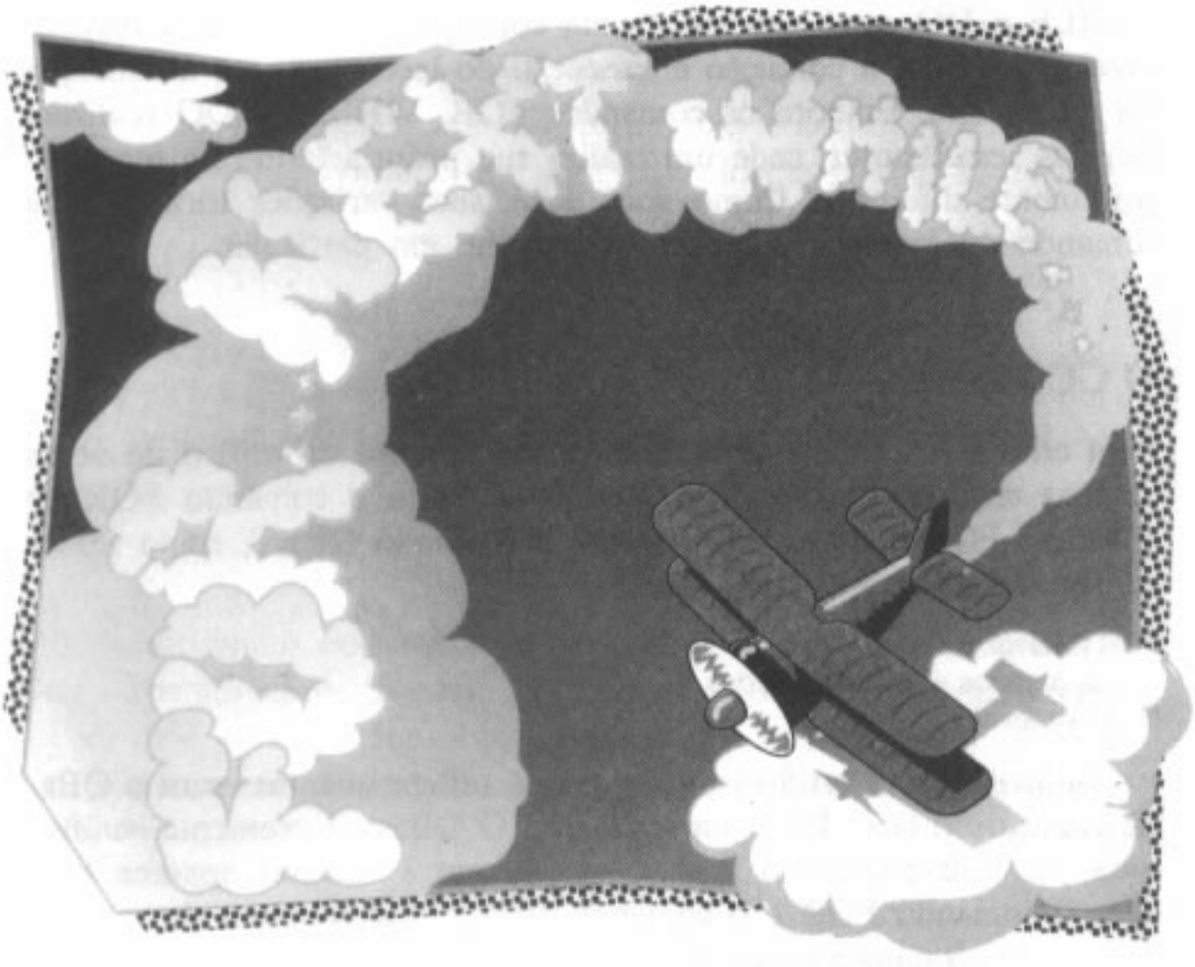
2. Quais dos seguintes *não* são operadores condicionais?

a. <=	d. <	g. <>	j. /
b. ><	e. >=	h. >	k. <<
c. ==	f. ^	i. =<	l. =

3. Verdadeiro ou Falso: Você sempre deverá usar a palavra-chave THEN na mesma linha da palavra-chave IF.
4. Qual a diferença entre o operador AND e o operador OR?
5. O que a palavra-chave ELSE lhe permite fazer? Como ela difere da palavra-chave THEN?
6. Como funciona a palavra-chave ELSEIF? Que outra palavra-chave do QBasic deve ser usada com ELSEIF?
7. Escreva um programa que faça ao usuário uma pergunta do tipo sim-ou-não. Instrua o usuário a entrar com um *S* de sim ou *N* de não. Inclua um conjunto de comandos que serão executados se o usuário entrar com *S* e um conjunto de comandos separado que será executado se o usuário entrar com *N*. Inclua um terceiro conjunto de comandos que será executado se o usuário não entrar com uma resposta adequada. (Dica: Lembre-se de verificar letras maiúsculas e minúsculas.)
8. Como funciona o comando SELECT CASE?
9. O que faz a cláusula CASE ELSE?
10. Em um comando CASE, o que fazem as palavras-chave IS e TO?
11. Escreva um programa que ofereça a descrição de três itens ao usuário. Faça com que o programa mostre uma lista numerada de itens e depois peça ao usuário para entrar com um número correspondendo ao item a ser descrito. Use um comando SELECT CASE para mostrar a informação apropriada, e não se esqueça de incluir comandos que serão executados se o usuário não entrar com uma resposta apropriada.

CAPÍTULO 6

Uso de Laços do QBasic



Você já aprendeu a maior parte dos fundamentos da linguagem QBasic. Neste capítulo, você aprenderá a usar uma das ferramentas mais poderosas do QBasic: o *laço*. Usando laços no QBasic, você poderá fazer com que o computador realize certas tarefas repetitivas em uma fração do tempo que você levaria para fazê-las pessoalmente.

INTRODUÇÃO AOS LAÇOS DO QBASIC

Um *laço* é simplesmente um ou mais comandos do QBasic que você direciona o QBasic a repetir. Os laços podem ser repetidos de duas formas:

- Um número específico de vezes
- Até que uma certa condição seja atendida

Em QBasic, três comandos — FOR, WHILE e DO — permitem que você inclua laços ao seu programa. No comando FOR, você especifica o número de vezes que deseja executar um laço. No comando WHILE e DO, você especifica uma condição, e o QBasic é responsável por avaliar a condição e execução do laço enquanto a condição for verdadeira. Embora os comandos FOR, WHILE e DO realizem funções semelhantes, cada um realiza sua própria tarefa de forma ligeiramente diferente. Como você verá, suas variações tornam cada comando adequado a um tipo de trabalho em particular.

O COMANDO FOR

Para criar um laço que é executado um número específico de vezes (o tipo mais comum de laço no QBasic), use o comando FOR. O comando FOR sempre termina com o comando NEXT, como vemos na sintaxe a seguir:

```
FOR variável = início TO fim
    comandos a serem repetidos
NEXT variável
```

variável é uma variável numérica que reflete quantas vezes o QBasic executou o laço. Ele é um contador. O QBasic incrementa *variável* toda vez que executa o laço. Observe que a *variável* aparece logo após o comando FOR e o comando NEXT. Esses nomes de variável devem ser exatamente iguais.

início é o valor numérico, seja um número ou uma expressão, em que você deseja que o QBasic comece a contar.

fim é um valor numérico, seja um número ou uma expressão numérica, que diz ao QBasic até onde ele deverá contar, ou seja,

quantas vezes ele deve repetir o laço. Quando atribuir valores a *início* e *fim*, tenha em mente as seguintes dicas:

- Você poderá usar valores positivos e negativos para *início* e *fim*.
- Você poderá usar valores inteiros e de ponto flutuante para *início* e *fim*.
- O valor de *início* deve ser menor ou igual ao valor de *fim*.
- O valor de *início* não precisa ser 1.

Os comandos entre o comando FOR e o comando NEXT são os comandos que o QBasic executará por um número especificado de vezes. Não existe limite para o número de comandos que você pode usar, e os comandos não precisam ser do mesmo tipo.

O QBasic usa o comando NEXT para contar quantas vezes ele executou os comandos entre o comando FOR e o comando NEXT. Eis um exemplo de um comando FOR que imprime uma mensagem cinco vezes, incrementando a variável *i%* toda vez que faz isso:

```
FOR i% = 1 TO 5
    PRINT "Estou em um laço."
NEXT i%
```

Toda vez que completa o laço, o QBasic volta ao comando FOR e compara o valor de *i%* com o valor de *fim* (que é 5). Assim que *i%* ultrapassar *fim*, o QBasic saltará para o comando após NEXT e continuará executando o programa.

Prática: Trabalhando com um laço FOR

1. Digite o programa FOR-1.BAS (Figura 6-1).

```
' FOR-1.BAS
' Este programa demonstra o laço FOR.

CLS

INPUT "Favor entrar com um número de 2 a 5: ",
vezes%
PRINT

FOR i% = 1 TO vezes%
  PRINT "Essas linhas aparecerão"; vezes%; "vezes."
  PRINT "Esta é a vez"; i%
  PRINT
NEXT i%
```

FIGURA 6-1.
FOR-1.BAS: Um laço FOR simples.

2. Execute o programa. A sua tela de saída deverá ficar assim:

```
Favor entrar com um número de 2 a 5: 4

Essas linhas aparecerão 4 vezes.
Esta é a vez 1.

Essas linhas aparecerão 4 vezes.
Esta é a vez 2.

Essas linhas aparecerão 4 vezes.
Esta é a vez 3.

Essas linhas aparecerão 4 vezes.
Esta é a vez 4.
```

***Prática: Quando o* início *é maior do que o* fim**

1. Digite o programa FOR-2.BAS (Figura 6-2). Observe que *início* é maior do que *fim*.

```
' FOR-2.BAS
' Este programa demonstra um laço FOR
'  nunca executado.

CLS

FOR i% = 6 TO 5
  PRINT "O valor corrente de i% é"; i%
NEXT i%
```

FIGURA 6-2.
FOR-2.BAS: Um laço FOR nunca executado.

2. Execute o programa. Exceto pela mensagem *Press any key to continue*, sua tela de saída ficará em branco. O QBasic abriu o contador em *6*, um valor maior do que *fim*, e, portanto, considerou a tarefa terminada.

***Prática: Usando valores iguais para* início *e* fim**

1. Mude o comando FOR no programa FOR-2.BAS de modo que *início* e *fim* tenham o mesmo valor:

```
FOR i% = 5 TO 5
```

2. Execute o programa. A sua tela de saída ficará como esta:

```
O valor corrente de i% é 5
```

Dessa vez, o QBasic executou o corpo do comando FOR uma vez. Como os valores de *início* e *fim* são os mesmos, o QBasic supôs que esta era a última "volta" e executou o comando PRINT uma vez antes de seguir adiante.

Por Que Usar *i%*?

Na maior parte dos laços FOR deste livro, você notará que *i%* é a variável do laço. Por que isto?

Antes que o Basic fosse inventado, muitos programadores usavam uma linguagem chamada FORTRAN. Em FORTRAN, a primeira letra de um nome de variável especificava seu tipo. Por exemplo, qualquer nome de varnavel começado com uma letra de *I* a *N* indicava uma variável inteira, assim como um sinal de porcentagem (%) anexado a uma variável do QBasic indica uma variável inteira.

Quase todos os contadores de laço FOR são inteiros. Assim, para ganhar tempo na digitação, um programador FORTRAN normalmente usava a variável *i* como contador do laço. Os programadores que precisassem endentar dois ou mais laços (veja "Endentação de Laços FOR" mais adiante neste capítulo) usariam *j*, depois *k* e assim por diante. Esta tradição foi seguida por muitos programadores em Basic.

Você precisa usar *i%*, *j%* e *k%*? Não. Você pode usar qualquer nome de variável válido em QBasic — por exemplo, *conta%*, *numLinhas%* ou *diasSemana%*.

Laços e o Comando COLOR

Você aprenderá mais sobre o comando COLOR no Capítulo 11, mas por enquanto basta sentar-se e ver esta demonstração, que usa tanto o comando COLOR quanto um laço FOR dentro de um programa.

Prática: Mudando cores de primeiro plano com o laço FOR

1. Digite o programa FOR-3.BAS (Figura 6-3).

```
' FOR-3.BAS
' Este programa usa um laço FOR e o comando COLOR
'   para demonstrar várias cores de segundo plano.

CLS

FOR i% = 1 TO 15     ' usa cores de 1 a 15
  COLOR i%           ' usa valor de i% para a cor
  PRINT "Esta linha usa a cor"; i%
NEXT i%
```

FIGURA 6-3.
FOR-3.BAS: Usando o comando COLOR em um laço FOR.

NOTA: Você pode rodar este programa mesmo que não tenha um monitor colorido conectado ao seu computador. Sua experiência não será tão colorida, mas você ficará esclarecido da mesma forma.

O Comando COLOR

O comando COLOR lhe permite especificar as cores de primeiro plano (texto) e segundo plano para a sua tela de saída. Aqui está a sintaxe para o comando COLOR:

COLOR [*primeiro plano*][, *segundo plano*]

Tanto *primeiro plano* quanto *segundo plano* são números que representam cores:

Valor	*Cor resultante*
0	Preto (segundo plano default)
1	Azul
2	Verde
3	Ciano
4	Vermelho
5	Magenta
6	Marrom
7	Branco (primeiro plano default)
8	Cinza
9	Azul claro
10	Verde claro
11	Ciano claro
12	Vermelho claro
13	Magenta claro
14	Amarelo
15	Branco intenso

Por exemplo, o comando

```
COLOR 14, 1
```

faz com que o QBasic mostre texto amarelo sobre um fundo azul.

2. Execute o programa. Sua tela de saída deverá se parecer com esta, exceto por cada linha ser mostrada em uma cor diferente:

```
Esta linha usa a cor 1
Esta linha usa a cor 2
Esta linha usa a cor 3
Esta linha usa a cor 4
Esta linha usa a cor 5
Esta linha usa a cor 6
Esta linha usa a cor 7
Esta linha usa a cor 8
Esta linha usa a cor 9
Esta linha usa a cor 10
Esta linha usa a cor 11
Esta linha usa a cor 12
Esta linha usa a cor 13
Esta linha usa a cor 14
Esta linha usa a cor 15
```

Prática: Alteração das cores de primeiro e segundo planos com um laço FOR

1. Digite o programa FOR-4.BAS (Figura 6-4).

```
' FOR-4.BAS
' Este programa usa um laço FOR e o comando COLOR
' para alterar cores de primeiro e segundo planos.

CLS

FOR i% = 1 TO 15        ' usa cores de 1 a 15
  COLOR i%, i% - 1
  PRINT "Cor de primeiro plano é"; i%;
  PRINT "; cor de segundo plano é"; i% - 1
NEXT i%
```

FIGURA 6-4.
FOR-4.BAS: Alteração das cores de primeiro e segundo planos com um laço FOR.

2. Execute o programa. Sua tela de saída deverá ser como esta, mas a cores:

```
Cor de primeiro plano é 1; cor de segundo plano é 0
Cor de primeiro plano é 2; cor de segundo plano é 1
Cor de primeiro plano é 3; cor de segundo plano é 2
Cor de primeiro plano é 4; cor de segundo plano é 3
Cor de primeiro plano é 5; cor de segundo plano é 4
Cor de primeiro plano é 6; cor de segundo plano é 5
Cor de primeiro plano é 7; cor de segundo plano é 6
Cor de primeiro plano é 8; cor de segundo plano é 7
Cor de primeiro plano é 9; cor de segundo plano é 8
Cor de primeiro plano é 10; cor de segundo plano é 9
```

```
Cor de primeiro plano é 11; cor de segundo plano é 10
Cor de primeiro plano é 12; cor de segundo plano é 11
Cor de primeiro plano é 13; cor de segundo plano é 12
Cor de primeiro plano é 14; cor de segundo plano é 13
Cor de primeiro plano é 15; cor de segundo plano é 14
```

3. Pressione uma tecla para retornar à janela View, e depois veja o comando COLOR no laço FOR. Observe os valores que definem as cores de primeiro e segundo planos. A cor de primeiro plano é definida com o valor corrente de *i%*, e a cor de segundo plano é definida com o valor de *i%* menos 1. O uso de técnicas como esta evita que a mesma cor seja usada para primeiro e segundo planos — se os valores de primeiro e segundo planos forem iguais, você não conseguirá ler o texto.

Laços e o Comando SOUND

Um outro comando divertido do QBasic é SOUND. (Mais uma vez, você verá mais detalhes sobre o comando SOUND no Capítulo 11.) Na prática a seguir, o comando FOR combinado com o comando SOUND fornece uma prova audível do poder dos laços.

O Comando SOUND

O comando SOUND faz exatamente o que o significado do nome sugere — ele emite um som pelo computador. Veja a sintaxe do comando SOUND:

SOUND *freqüência, duração*

O argumento *freqüência* é um inteiro de 37 até 32767, indicando a freqüência do som em ciclos por segundo, ou hertz. O argumento *duração* é um número inteiro ou de ponto flutuante de 0 até 65535, indicando o tempo em que o som permanecerá. (O valor 18.2 é igual a um segundo.) Por exemplo, o comando a seguir faz com que o QBasic emita um som de 500 hertz por meio segundo:

```
SOUND 500, 9.1
```

Prática: Usando os comandos FOR e SOUND

1. Digite o programa FOR-5.BAS (Figura 6-5).

```
' FOR-5.BAS
' Este programa usa o comando SOUND dentro
'   de um laço FOR para gerar efeitos sonoros.

CLS

PRINT ,"GERADOR DE EFEITOS SONOROS"
PRINT ,"Favor escolher o som que deseja"
PRINT ,"ouvir pelo seguinte menu:"
PRINT
PRINT ,"1.  Arma de raios"
PRINT ,"2.  Sirene européia"
PRINT ,"3.  Decolagem!"
PRINT

INPUT "        Favor entrar com 1, 2 ou 3: ", esc%
PRINT
SELECT CASE esc%
  CASE 1          ' arma de raios
    FOR i% = 1 TO 50
      SOUND 850, .3
      SOUND 800, .3
      SOUND 825, .3
    NEXT i%
  CASE 2          ' sirene européia
    FOR i% = 1 TO 5
      SOUND 500, 10
      SOUND 450, 10
    NEXT i%
  CASE 3          ' decolagem!
    FOR i% = 500 TO 2500
      SOUND i%, .03  ' usa valor de i% para o som
    NEXT i%
END SELECT
```

FIGURA 6-5.
FOR-5.BAS: Um gerador de efeitos sonoros.

2. Execute o programa. A sua tela de saída deverá se parecer com esta:

```
GERADOR DE EFEITOS SONOROS
Favor escolher o som que deseja
ouvir pelo seguinte menu:

1. Arma de raios
2. Sirene européia
3. Decolagem!

Favor entrar com 1, 2 ou 3:
```

3. Entre com o valor apropriado para ouvir o som que desejar.

Execute o programa várias vezes para testar cada um dos sons. Em seguida, experimente este programa alternando alguns dos valores nos vários comandos SOUND e rodando o programa novamente. Observe que a maior parte dos valores de duração são pequenos. Valores de duração pequenos funcionam bem para efeitos sonoros, mas você precisará de valores maiores para criar uma melodia.

Controle do Contador com STEP

Como vimos, quando o QBasic executa um laço FOR ele começa atribuindo um valor de *início* à variável seguinte ao FOR e depois executa o bloco de comandos até que o valor da variável contadora seja maior do que o valor de *fim*. Toda vez que encontra o comando NEXT, o QBasic incrementa o valor do contador em 1.

Você pode dizer ao QBasic para incrementar o contador por um valor diferente de 1 se usar a cláusula STEP. Veja um exemplo:

```
FOR i% = 5 TO 25 STEP 5
    PRINT "Esta mensagem aparecerá 5 vezes."
NEXT i%
```

Quando o QBasic encontrar o comando NEXT, ele incrementará o contador pelo número que você especificou em STEP.

O número na cláusula STEP não precisa ser positivo. Se colocar um número negativo após o STEP, o QBasic *decrementará* o valor do contador toda vez que encontrar o comando NEXT.

O valor de STEP também afeta os valores atribuídos a *início* e *fim*:

- Se o valor de STEP for positivo, o valor de *início* deverá ser menor ou igual ao valor de *fim*.
- Se o valor de STEP for negativo, o valor de *início* deverá ser maior ou igual ao valor de *fim*.

Pense nisso: se você usar um valor de STEP negativo e *fim* for maior do que *início*, o QBasic achará que o trabalho terminou, saltará sobre o bloco de comandos sem executá-lo e continuará executando o restante do programa.

Prática: Usando STEP

1. Digite o programa FOR-6.BAS (Figura 6-6).

```
' FOR-6.BAS
' Este programa demonstra o uso de STEP.

CLS

FOR i% = 15 TO 1 STEP -1
  COLOR i%
  PRINT "Caindo!"
  SOUND (50 * i%), 1
NEXT i%
```

FIGURA 6-6.
FOR-6.BAS: Usando STEP em um comando FOR.

2. Execute o programa. O QBasic mostrará a palavra *Caindo!* 15 vezes, cada vez numa cor diferente, com um som acompanhando.

Endentação de Laços FOR

Endentação é o termo que descreve a prática de colocar um laço dentro de outro. Veja um exemplo de um laço FOR endentado:

```
FOR i% = 1 TO 5
    FOR j% = 1 TO 5
        PRINT "Você verá esta mensagem 25 vezes."
    NEXT j%
NEXT i%
```

Veja exatamente onde a endentação o ajudará a entender o que está acontecendo. Em um laço FOR endentado, o laço FOR interno é, na realidade, um comando do laço externo. Ou seja, o laço FOR externo executa o laço interno tantas vezes quantas você especificar. Neste exemplo, o laço externo executa o laço interno num total de 5 vezes. Como o laço interno executa seus comandos 5 vezes, o QBasic acaba executando o comando PRINT num total de 25 vezes.

Prática: Trabalhando com laços FOR endentados

1. Digite o programa FOR-7.BAS (Figura 6-7).

```
' FOR-7.BAS
' Este programa demonstra laços FOR endentados.

CLS

PRINT "O laço interno mostrará números coloridos:"
PRINT
```

FIGURA 6-7. *(continua)*
FOR-7.BAS: Usando laços FOR endentados.

FIGURA 6-7. *continuação*

```
FOR i% = 1 TO 5
  COLOR 7      ' define cor branca no laço externo
  PRINT        ' mostra linha em branco
  PRINT "Laço externo, volta número"; i%
  FOR j% = 1 TO 10
     COLOR j%
     PRINT j%;
     SOUND (j% * 100), 1
  NEXT j%
NEXT i%
```

2. Execute o programa. Quando tiver terminado sua execução, a tela de saída ficará como esta:

O laço interno mostrará números coloridos:

```
Laço externo, volta número 1
 1 2 3 4 5 6 7 8 9 10
Laço externo, volta número 2
 1 2 3 4 5 6 7 8 9 10
Laço externo, volta número 3
 1 2 3 4 5 6 7 8 9 10
Laço externo, volta número 4
 1 2 3 4 5 6 7 8 9 10
Laço externo, volta número 5
 1 2 3 4 5 6 7 8 9 10
```

O comando SOUND dentro do laço interno criou um som depois que cada um dos números do laço interno foi mostrado. Se quiser, pressione uma tecla para retornar à janela View e mudar alguns dos valores. Tente antecipar o que o programa alterado fará antes de rodá-lo, e depois rode-o para ver se acertou.

O COMANDO WHILE

Para elaborar um laço que se repete enquanto uma condição específica for verdadeira, use o comando WHILE. O comando WHILE sempre termina com o comando WEND, como mostra a sintaxe a seguir:

```
WHILE condição
    comandos a serem executados repetidamente
WEND
```

condição pode ser qualquer expressão condicional (como *lucro! = 11500*, *ano% <= 1959* ou *temperatura% < 0*).

Quando o QBasic encontra um comando WHILE, ele avalia a *condição*:

- Se a *condição* for verdadeira, o QBasic executa o bloco de comandos entre os comandos WHILE e WEND e depois verifica novamente a *condição*.
- Se a *condição* for falsa, o QBasic salta para o comando WEND e prossegue para o restante do programa.

Para usar um laço WHILE com sucesso, você deve tomar providências *dentro* do laço para alterar a condição após o comando WHILE.

NOTA: Assegure-se de que a condição eventualmente se tornará falsa; se não o fizer, o seu laço nunca terminará, e o restante do programa nunca será alcançado. (Um laço que nunca termina é chamado laço infinito ou sem fim. Para terminar um laço infinito no QBasic, segure a tecla Ctrl e pressione Break.)

Prática: Trabalhando com o laço WHILE

No programa a seguir, toda a atividade ocorre dentro do laço. E como o QBasic incrementa o valor de *num%* toda vez que executa o laço, *num%* eventualmente se tornará maior do que 12. Isso cria uma condição falsa e faz com que o QBasic termine a execução do laço.

1. Digite o programa WHILE.BAS (Figura 6-8).

```
' WHILE.BAS
' Este programa demonstra o laço WHILE.

CLS

INPUT "Favor entrar com um número de 1 a 10: ", num%
PRINT

WHILE num% <= 12
  COLOR num%
  PRINT "O valor corrente de num% é"; num%
  SOUND ((num% * 30) + 300), .5
  num% = num% + 1
  PRINT
WEND
```

FIGURA 6-8.
WHILE.BAS: Usando um laço WHILE com um contador.

2. Execute o programa. A sua tela de saída deverá se parecer com esta:

```
Favor entrar com um número de 1 a 10: 9

O valor corrente de num% é 9
```

```
O valor corrente de num% é 10

O valor corrente de num% é 11

O valor corrente de num% é 12
```

Endentação de Laços WHILE

Assim como você pode endentar comandos FOR, também pode colocar um comando WHILE dentro de outro. Cada comando WHILE possui sua própria *condição* e comando WEND correspondente; as duas condições não precisam ter relação uma com a outra.

A sintaxe para um comando WHILE endentado é a seguinte:

```
WHILE condição
    WHILE condição
        comando para o WHILE interno
    WEND
WEND
```

O laço WHILE interno serve como comando do laço WHILE externo.

NOTA: Para evitar laços infinitos, lembre-se de que as condições interna e externa eventualmente deverão ser avaliadas como falsas.

O COMANDO DO

Use o comando DO juntamente com o comando LOOP para desenvolver dois tipos de laços:

- Um laço que se repete *enquanto* uma certa condição é verdadeira (DO WHILE)
- Um laço que se repete *até que* uma certa condição *se torne* verdadeira (DO UNTIL)

DO WHILE: Laço Enquanto uma Condição For Verdadeira

Para criar um laço que se repete enquanto uma condição é verdadeira, use um laço DO com a seguinte sintaxe:

```
DO WHILE condição
    bloco de comandos a serem executados
LOOP
```

Um laço DO deste tipo funciona exatamente como um comando WHILE. O QBasic examina a *condição*:

- Se a *condição* for verdadeira, o QBasic executa os comandos e continua a fazer isso enquanto a *condição* for verdadeira
- Se a *condição* for falsa, o QBasic salta para depois do comando LOOP e executa o restante do programa.

A Conexão Condição-Comando

Como vimos no Capítulo 2, o QBasic executa instruções na ordem em que aparecem nos seus programas. A importância da ordem é aparente sobretudo dentro do comando DO, que exige a colocação de uma condição em um dentre dois locais, com base no resultado desejado:

- Se quiser que os comandos sejam executados com base simplesmente no valor da *condição*, coloque a *condição* na linha com o comando DO:

```
DO {WHILE|UNTIL} condição
  bloco de comandos a serem executados
LOOP
```

Neste formato, os comandos interiores só serão executados se a *condição* for verdadeira.

- Se quiser que os comandos sejam executados pelo menos uma vez, não importa qual seja o valor da *condição*, coloque-a na linha com o comando LOOP:

```
DO
  bloco de comandos a serem executados
LOOP {WHILE|UNTIL} condição
```

Quando a *condição* for avaliada, o QBasic já terá executado os comandos uma vez.

Prática: Usando um laço DO WHILE com a condição em cima

1. Digite o programa DO-1.BAS (Figura 6-9).

```
' DO-1.BAS
' Este programa demonstra um laço DO WHILE.
'   Observe a condição em cima do laço.

CLS

INPUT "Favor entrar com um número de 10 a 15:", num%
PRINT

DO WHILE num% >= 10
  COLOR num%
  PRINT "O valor de num% é"; num%
  num% = num% - 1   ' decrementa num% em 1
LOOP
```

FIGURA 6-9.
DO-1.BAS: Usando um laço DO WHILE.

2. Rode o programa. A sua tela de saída deverá se parecer com esta:

```
Favor entrar com um número de 10 a 15: 13

O valor de num% é 13
O valor de num% é 12
O valor de num% é 11
O valor de num% é 10
```

3. Rode o programa novamente, mas desta vez entre com um valor menor ou igual a 9. A sua tela de saída se parecerá com esta:

```
Favor entrar com um número de 10 a 15: 6
```

O QBasic percebeu a condição falsa e, portanto, saltou os comandos dentro do laço.

Prática: Usando um laço DO WHILE com a condição embaixo

O programa DO-2.BAS é idêntico ao DO-1.BAS, mas a condição foi movida para baixo do laço.

1. Modifique o programa DO-1.BAS de modo que se iguale ao programa DO-2.BAS (Figura 6-10).

```
' DO-2.BAS
' Este programa demonstra um laço DO WHILE.
'   Observe a condição embaixo do laço.

CLS
```

FIGURA 6-10. (*continua*)
DO-2.BAS: Um laço DO WHILE com a condição embaixo.

FIGURA 6-10. *continuação*

```
INPUT "Favor entrar com um número de 10 a 15: ",
num%
PRINT

DO
  COLOR num%
  PRINT "O valor de num% é"; num%
  num% = num% - 1   ' decrementa num% em 1
LOOP WHILE num% >= 10
```

2. Rode o programa. A sua tela de saída deverá se parecer com esta:

```
Favor entrar com um número de 10 a 15: 13

O valor de num% é 13
O valor de num% é 12
O valor de num% é 11
O valor de num% é 10
```

Idêntica à saída de DO-1.BAS, não? Mas espere.

3. Rode o programa novamente, mas desta vez entre com um valor menor ou igual a 9. A sua tela de saída se parecerá com esta:

```
Favor entrar com um número de 10 a 15: 6

O valor de num% é 6
```

Agora você pode ver a diferença na prática. Embora o valor de *num%* fizesse com que a condição no laço LOOP fosse falsa, o QBasic não encontrou essa condição até *depois* de ter executado o corpo do laço DO uma vez.

DO UNTIL: Laço Até uma Condição Verdadeira

Para criar um laço que será executado até que uma condição *se torne* verdadeira, use um laço DO com a seguinte sintaxe:

```
DO UNTIL condição
    bloco de comandos a serem executados
LOOP
```

O QBasic examina a *condição*:

- Se a *condição* for verdadeira, o QBasic salta o comando LOOP e executa o restante do programa.
- Se a *condição* for falsa, o QBasic executa os comandos e continua a fazer isso até que a *condição* seja verdadeira.

Prática: Usando um laço DO UNTIL com a condição em cima

1. Digite o programa DO-3.BAS (Figura 6-11).

```
' DO-3.BAS
' Este programa demonstra um laço DO UNTIL.
'   Observe a condição em cima do laço.

CLS

INPUT "Favor entrar com um número de 1 a 10: ", num%
PRINT

DO UNTIL num% > 10
  COLOR num%
  PRINT "O valor de num% é"; num%
  num% = num% + 1   ' incrementa num% em 1
LOOP
```

FIGURA 6-11.
DO-3.BAS: Usando um laço DO UNTIL.

2. Rode o programa. A sua tela de saída deverá se parecer com esta:

```
Favor entrar com um número de 1 a 10: 7

O valor de num% é 7
O valor de num% é 8
O valor de num% é 9
O valor de num% é 10
```

Como a condição *num% > 10* era falsa quando o QBasic encontrou o laço inicialmente, ele continuou a executá-lo até que a condição se tornasse verdadeira.

3. Rode o programa novamente, desta vez entrando com um valor 11 ou maior. A sua tela de saída ficará como esta:

```
Favor entrar com um número de 1 a 10: 13
```

Desta vez, como a condição era verdadeira quando o QBasic encontrou o laço, ele saltou o laço e executou o restante do programa.

Prática: Usando um laço DO UNTIL com a condição embaixo

1. Modifique o programa DO-3.BAS de modo que se iguale ao programa DO-4.BAS (Figura 6-12).

```
' DO-4.BAS
' Este programa demonstra um laço DO UNTIL.
'   Observe a condição embaixo do laço.

CLS

INPUT "Favor entrar com um número de 1 a 10: ", num%
PRINT

DO
  COLOR num%
  PRINT "O valor de num% é"; num%
  num% = num% + 1   ' incrementa num% em 1
LOOP UNTIL num% > 10
```

FIGURA 6-12.
DO-4.BAS: Um laço DO UNTIL com a condição embaixo.

2. Rode o programa. A sua tela de saída deverá se parecer com esta:

```
Favor entrar com um número de 1 a 10: 7

O valor de num% é 7
O valor de num% é 8
O valor de num% é 9
O valor de num% é 10
```

A saída deste programa é a mesma do programa anterior.

3. Rode o programa novamente, mas desta vez entre com um valor 11 ou maior. A sua tela de saída deverá ser algo assim:

```
Favor entrar com um número de 1 a 10: 13

O valor de num% é 13
```

Dessa vez, como a condição estava embaixo do laço, o QBasic executou o conteúdo do laço uma vez, pois não sabia qual era a condição até que chegou ao término do laço.

Endentando Laços DO

Você pode endentar laços DO da mesma forma como endentou os laços FOR e WHILE.

Os laços não têm relação alguma em termos de como você os estabelece. Em outras palavras, se tiver dois laços DO endentados e colocar a condição em cima do laço externo, não precisará colocar a condição em cima do laço interno. Naturalmente, assim como em um único laço DO, você deverá assegurar-se de que os laços externo e endentado eventualmente terminarão.

ENDENTAÇÃO DE DIFERENTES TIPOS DE LAÇOS

Até aqui discutimos a endentação de laços do mesmo tipo — um laço FOR dentro de outro laço FOR, um laço WHILE dentro de outro laço WHILE etc.

Você também pode misturar os tipos de laços, sobretudo quando isso tornar o seu programa mais eficiente. O programa a seguir endenta um laço FOR dentro de um laço DO. O laço DO permite a execução do laço FOR quantas vezes quiser sem ter que iniciar o programa a cada vez.

Prática: Endentando diferentes tipos de laços

1. Digite o programa ENDENT.BAS (Figura 6-13).

```
' ENDENT.BAS
' Este programa demonstra a endentação
'   de vários tipos de laços.

CLS

DO
  PRINT "Quantos sons deseja ouvir?"
  INPUT "(Entre com 0 para terminar) ", numSons%
  PRINT

  FOR i% = 1 TO numSons%
    SOUND (i% * 100), 1
  NEXT i%

LOOP UNTIL numSons% = 0
```

FIGURA 6-13.
ENDENT.BAS: Endentando diferentes tipos de laços.

2. Rode o programa. A sua tela de saída deverá ficar como esta:

```
Quantos sons deseja ouvir?
(Entre com 0 para terminar) 5

Quantos sons deseja ouvir?
(Entre com 0 para terminar) 20

Quantos sons deseja ouvir?
(Entre com 0 para terminar) 0
```

UTILIDADES PRÁTICAS PARA OS LAÇOS DO QBASIC

Agora que você já conhece os laços, é hora de ver o que realmente podem fazer por você. Os exemplos a seguir apresentam algumas aplicações práticas e divertidas para os laços.

Uso de um Laço para Reunir Informações

Digamos que você queira automatizar o planejamento do seu orçamento mensal. A filosofia em que se baseia o orçamento é simples: subtraia despesas da receita mensal.

Usando as ferramentas que vimos até aqui, você poderia escrever um programa em QBasic para ajudá-lo a fazer isso. Você poderia usar vários comandos INPUT: um para obter o valor inicial e depois um para cada gasto. Após alguns cálculos, você poderia fazer com que o QBasic imprimisse quanto dinheiro você possui de saldo. O programa mostrado na Figura 6-14 é um exemplo desse tipo de programa.

```
' Este programa ajuda-o a descobrir quanto
'   dinheiro sobrou após pagar despesas mensais.

CLS

COLOR 2             ' primeiro plano verde
PRINT "Calculador de Orçamento"
COLOR 7             ' restaura cor branca default
PRINT
INPUT "Entre com a receita total para este mês: $",
total!
PRINT

INPUT "Entre com despesa 1: $", desp1!
total! = total! - desp1!
INPUT "Entre com despesa 2: $", desp2!
total! = total! - desp2!
INPUT "Entre com despesa 3: $", desp3!
total! = total! - desp3!
INPUT "Entre com despesa 4: $", desp4!
total! = total! - desp4!

PRINT
PRINT "Sobrou $"; total!; "este mês."
```

FIGURA 6-14.
Um exemplo de programa orçamentário.

Que Laço Você Deve Escolher?

Lembre-se dos seguintes aspectos quando tiver que decidir que tipo de laço usará em um programa:

- Use um laço FOR quando quiser executar um bloco de comandos por um número específico de vezes.
- Use um laço WHILE ou DO para executar comandos com base no valor de uma condição.
- Você pode fazer com que um laço DO faça qualquer coisa que um laço WHILE possa fazer. Adquira o hábito de usar o laço DO mais versátil sempre que possível. (O QBasic aceita o laço WHILE para que você possa rodar programas escritos para versões anteriores do BASIC.)

Este programa funcionaria bem, mas possui limitações:

- Como o seu número de despesas poderá mudar de um mês para o outro, seria preciso alterar o programa toda vez que o executasse.
- Cada despesa entrada deverá ser subtraída imediatamente do total acumulado. Isso não é um problema, mas gera muito cálculo repetitivo. Pense no tamanho deste programa se você tivesse vinte ou mais despesas todo mês!

Os programas com este grau de repetição são candidatos ideais para um laço. O programa ORCAM.BAS na Figura 6-15 faz a mesma coisa que o programa na Figura 6-14, mas oferece algumas vantagens:

- É um programa menor.
- Acomoda um número variável de despesas, perguntando o número total de contas que você pretende entrar.
- Usa um laço tanto para lhe pedir o valor da despesa quanto para subtrair o valor do total acumulado.

```
' ORCAM.BAS
' Este programa ajuda-o a descobrir quanto
'  dinheiro sobrou após pagar despesas mensais.

CLS

COLOR 2         primeiro plano verde
PRINT "Calculador de Orçamento"
```

FIGURA 6-15. *(continua)*
ORCAM.BAS: Um programa orçamentário melhor.

FIGURA 6-15. *continuação*

```
COLOR 7         ' restaura o branco default
PRINT
INPUT "Entre com a receita total para este mês: $",
total!
PRINT
INPUT "Quantas contas você terá este mês? ", contas%

FOR i% = 1 TO contas%
  PRINT "Entre com a despesa"; i%;
  INPUT ": $", estaDesp!
  total! = total! - estaDesp!
NEXT i%

PRINT
PRINT "Sobrou $"; total!; "este mês."
```

Uso de Laços com Números Randômicos

Dados. Roleta. Sena. Bingo. Loteria. Todos esses jogos utilizam *números randômicos* — números que ocorrem sem qualquer ordem previsível. Discutiremos três das ferramentas que o QBasic oferece para a criação de números randômicos: RND, RANDOMIZE TIMER e INT.

Criação de números randômicos

A função RND diz ao QBasic para gerar um número randômico para o seu programa. Este será um número de ponto flutuante com precisão simples entre 0 e 1. RND é uma função, de modo que deverá ser usada dentro de um comando do QBasic. Veja a sintaxe da forma mais simples da função RND:

```
RND
```

Por si só, RND retorna a mesma série de números sempre que você roda o seu programa. Para criar uma série *diferente* de números a cada vez, use RANDOMIZE TIMER como uma das primeiras instruções no seu programa. A sintaxe para o comando RANDOMIZE TIMER é tão simples quanto a função RND:

```
RANDOMIZE TIMER
```

Prática: Criando números randômicos

O programa RND-1.BAS (Figura 6-16) usa um laço FOR para criar seis números randômicos. (Para criar mais números randômicos, basta mudar o *6* no comando FOR por um valor mais alto.)

1. Digite o programa RND-1.BAS.

```
' RND-1.BAS
' Este programa gera números randômicos entre 0 e 1.

CLS

RANDOMIZE TIMER

FOR i% = 1 TO 6
  PRINT RND
NEXT i%
```

FIGURA 6-16.
RAND-1.BAS: Criando números randômicos.

1. Execute o programa. A sua tela de saída deverá se parecer com esta:

```
.2494013
.3081127
3.670245E-02
.574261
.4791026
.1495265
```

Os números que você obtém sem dúvida alguma serão diferentes destes — lembre-se de que são números randômicos [aleatórios].

Ajuste dos resultados de RND

Como os números entre 0 e 1 quase nunca atenderão às suas necessidades, você pode usar os operadores matemáticos do QBasic para mudar o tamanho dos resultados de RND.

Prática: Criando números randômicos maiores

O programa RND-2.BAS (Figura 6-17) é idêntico a RND-1.BAS, mas multiplica o resultado de RND por 100, criando valores na faixa de 0 a 100.

1. Modifique o programa RND-1.BAS para que se iguale ao programa RND-2.BAS.

2. Rode o programa. A sua tela de saída deverá ficar semelhante a esta:

```
85.67621
42.35302
87.795
48.80183
42.53306
24.83424
```

```
' RND-2.BAS
' Este programa gera números randômicos de 0 a 100.

CLS

RANDOMIZE TIMER
FOR i% = 1 TO 6
  PRINT RND * 100
NEXT i%
```

FIGURA 6-17.
RAND-2.BAS: Criando números randômicos maiores.

Números Muito Altos e Muito Baixos no QBasic

Os valores produzidos pela função RND do QBasic sempre serão maiores do que 0 e menores do que 1. Às vezes os valores são muito pequenos.

Quando o QBasic tenta imprimir um valor muito pequeno ou muito grande para que seja mostrado sem usar muitos dígitos, ele passa para a versão do QBasic da notação científica. Para simplificar as coisas, quando você encontrar um *E* ou um *D* em um número, o ponto decimal terá sido deslocado do seu local normal. O número após o *E* ou *D* indica em que direção e em quantas casas o ponto decimal foi movido. Se esse número for positivo, o ponto decimal realmente reside à *direita* do local mostrado; se o número for negativo, o ponto decimal pertence à *esquerda* do seu local apresentado.

Por exemplo, na saída de exemplo de RND-1.BAS você pode ver o número *3.670245E-02*. O *E-02* significa que o ponto decimal na realidade encontra-se duas casas à esquerda. Em outras palavras, você normalmente escreveria o número como 0.03670245.

Logo, por que não mostrar simplesmente o número como aparece de modo geral? Normalmente o número não aparecerá bem na tela. Considere o número *4.136111E28*. Se fosse impresso por inteiro, ele ficaria assim:

41361110000000000000000000000

Para aqueles que estão acostumados com a notação científica padrão, eis o mesmo número nesse formato:

4.136111×10^{28}

Criação de inteiros randômicos

Se você quiser criar números randômicos inteiros (números sem pontos decimais), use a função INT junto com a função RND. INT descarta a parte fracionária de um número de ponto flutuante, deixando apenas a parte inteira. Ele tem o seguinte formato:

INT(*número*)

número é o número de ponto flutuante que você deseja arredondar. Assim como todas as funções que retornam inteiros, um resultado inteiro de INT deverá ser atribuído a um comando que aceite valores inteiros ou a uma variável inteira.

A sessão de prática a seguir demonstra como usar a função INT com RND para criar números inteiros randômicos.

Prática: Criando um jogo de adivinhação

Os números randômicos são ideais para jogos de adivinhação. O programa a seguir usa um laço DO para demonstrar isso:

1. Digite o programa ADIVINHA.BAS (Figura 6-18).

```
' ADIVINHA.BAS
' Este programa é um jogo de adivinhação, que gera
'    um número randômico e pede ao usuário para
'    adivinhar. Quando conseguir, o programa
'    mostrará o número de tentativas feitas.

CLS

PRINT "Jogo de adivinhação"
PRINT
PRINT "Estou pensando em um número de 1 a 100."
PRINT "Pode adivinhar qual é?"
PRINT

RANDOMIZE TIMER

numRand% = INT(RND * 100) + 1      ' gera número
numTent% = 0                       ' zera tentativas

DO
  INPUT "Dê um chute: ", chute%
  IF chute% = numRand% THEN PRINT "Acertou!!!"
  IF chute% < numRand% THEN PRINT "Tente um número
maior!"
```

FIGURA 6-18. (*continua*)
ADIVINHA.BAS: Um jogo de adivinhação.

FIGURA 6-18. *continuação*

```
  IF chute% > numRand% THEN PRINT "Tente um número
menor!"

  numTent% = numTent% + 1     ' outra tentativa
  PRINT                       ' mostra linha em branco

LOOP UNTIL chute% = numRand%

PRINT "Você acertou o número em"; numTent%;
"tentativas!"
```

2. Rode o programa. A sua tela de saída deverá ficar semelhante a esta:

```
Jogo de adivinhação

Estou pensando em um número de 1 a 100.
Pode adivinhar qual é?

Dê um chute: 50
Tente um número maior!

Dê um chute: 99
Tente um número menor!

Dê um chute: 65
Tente um número maior!

Dê um chute: 71
Acertou!!!

Você acertou o número em 4 tentativas!
```

Prática: Escrevendo uma simulação por computador

Digamos que você queira escrever um programa que calcule quantas vezes o número 7 apareceu dentre 100 jogadas de 2 dados. O programa DADOS.BAS (Figura 6-19) faz exatamente isso, e a função RND novamente facilitará as coisas.

1. Digite o programa DADOS.BAS.
2. Rode o programa. Você deverá ver uma saída semelhante a esta:

```
Progama de Simulação de Dados

Quantas vezes devo rolar os dados? 50

Agindo...

Em 50 jogadas, o número 7
apareceu 10 vezes.
```

```
' DADOS.BAS
' Este programa lhe pedirá para entrar com quantas
'  vezes o QBasic deve "rolar" dois dados,
'  calculando quantas vezes o número 7 aparece.

CLS

RANDOMIZE TIMER

COLOR 4              ' cor vermelha para título
PRINT "Programa de Simulação de Dados"
COLOR 7              ' restaura branco default
PRINT
numSet% = 0          ' zera o contador

INPUT "Quantas vezes devo rolar os dados? ", rolar%
PRINT
PRINT "Agindo..."    ' o usuário deverá esperar

FOR i% = 1 TO rolar%
  dado1% = INT(RND * 6) + 1  ' "joga" o primeiro dado
  dado2% = INT(RND * 6) + 1  ' "joga" o segundo dado
  IF dado1% + dado2% = 7 THEN numSete% = numSete% + 1
NEXT i%
PRINT
PRINT "Em"; rolar%; "jogadas, o número 7"
PRINT "apareceu"; numSete%; "vezes."
```

FIGURA 6-19.
DADOS.BAS: Um programa de simulação de dados.

RESUMO

Os laços permitem a criação de poderosos programas que repetem uma tarefa por um número específico de vezes ou até que uma condição seja atendida. Neste capítulo você aprendeu sobre os laços FOR, WHILE e DO; junte esse aprendizado com os comandos COLOR e SOUND, a função RND e as habilidades adquiridas nos capítulos anteriores, e estará pronto a empreender alguns projetos de programação interessantes. Os capítulos seguintes darão a partida.

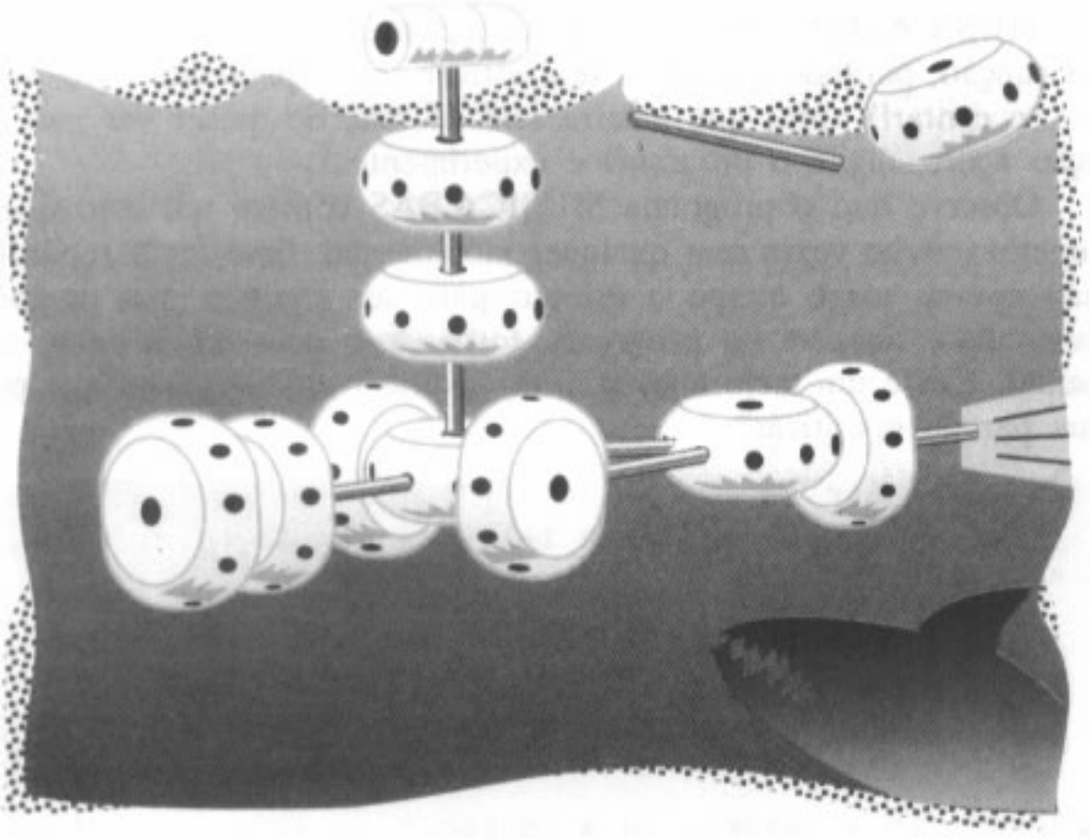
PERGUNTAS E EXERCÍCIOS

1. Qual a finalidade da variável contadora em um laço FOR?
2. Que tipos de valores podem ser atribuídos aos elementos *início* e *fim* de um laço FOR?

3. Que número você especificaria como argumento do comando COLOR se quisesse que a cor do primeiro plano fosse vermelha?
4. O que faz o comando SOUND?
5. O que é um laço endentado?
6. Qual é o efeito de colocar a condição de um laço DO em cima do laço? Qual é o efeito de colocá-la embaixo do laço?
7. Sob que circunstâncias você poderia usar um laço FOR? E um laço WHILE ou DO?
8. Escreva um programa que gerencie a quantidade de dinheiro gasta em combustível durante uma semana. Use um laço FOR para reunir os valores gastos, mantendo um total acumulado em uma variável de ponto flutuante com precisão simples.
9. Escreva um programa que peça ao usuário valores válidos de freqüência e duração, reproduzindo-os com o comando SOUND. Use um laço DO UNTIL para reunir a informação até que o usuário digite *-999* como freqüência.
10. Escreva um programa que jogue 10 vezes um dado simulado. Imprima o valor do dado após cada jogada, mostrando a mensagem *Boa Jogada!* em verde se o dado "cair" em 6.

CAPÍTULO 7

Criação de Subprogramas e Funções



Até o momento, você escreveu programas relativamente pequenos — nenhum teve mais de 30 linhas. Mas agora que você já aprendeu as bases do QBasic, está pronto para escrever programas maiores. Neste capítulo veremos algumas técnicas que lhe permitem escrever programas maiores com um mínimo de tempo e esforço adicionais.

Você aprenderá duas estruturas de programa: *subprogramas* e *funções*. (E verá como estas funções são diferentes das funções embutidas que já usamos.) Essas estruturas são usadas para separar um programa em unidades de fácil utilização chamadas *procedures*. Mostraremos como declarar os subprogramas e funções e como usá-los nos seus programas. Também veremos como o ambiente do QBasic ajuda na programação modular. Ao final do capítulo, você terá todas as ferramentas de que precisa para escrever programas compactos e bem organizados.

POR QUE PROCEDURES?

Suponha que queira escrever um programa para imprimir a letra da canção tradicional norte-americana "Clementine". Usando o que já sabemos, talvez você escrevesse um programa com inúmeros comandos PRINT, como no programa MUSICA.BAS mostrado na Figura 7-1.

MUSICA.BAS é muito simples: imprime cada verso e cada coro da canção, parando quando a tela estiver cheia, para que você possa ler (e cantar!) antes que a letra saia da tela. Se quiser ver isso em ação agora, digite o programa e experimente.

Observe que o programa MUSICA.BAS contém um coro que é repetido várias vezes sem qualquer modificação. Esse texto repetitivo não apenas exige tempo e esforço para ser digitado, mas também aumenta a listagem do programa, tornando-o mais difícil de se trabalhar. Existe um meio mais fácil de codificar um programa que possua partes repetitivas?

```
' MUSICA.BAS
' Este programa mostra a letra da canção
"Clementine."

CLS
PRINT "- - - - - - - - Clementine - - - - - - - - -"
PRINT

PRINT "In a cavern, in a canyon,"   ' primeira estrofe
PRINT "Excavating for a mine,"
PRINT "Lived a miner, forty-niner,"
```

FIGURA 7-1. (*continua*)
MUSICA.BAS: Um programa que imprime a letra da canção "Clementine" usando uma série de comandos PRINT.

FIGURA 7-1. *continuação*

```
PRINT "And his daughter, Clementine."
PRINT
PRINT "Oh my darling, oh my darling,"    ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT

PRINT "Light she was and like a fairy,"  ' segunda
                                         ' estrofe
PRINT "And her shoes were number nine;"
PRINT "Herring boxes without topses,"
PRINT "Sandals were for Clementine."

PRINT
PRINT "Oh my darling, oh my darling,"    ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT

INPUT "Pressione Enter para ver mais...", algo$
PRINT                                    ' pausa
                                         ' terceira estrofe
PRINT "Drove she ducklings to the water,"
PRINT "Ev'ry morning just at nine;"
PRINT "Hit her foot against a splinter,"
PRINT "Fell into the foaming brine."

PRINT
PRINT "Oh my darling, oh my darling,"    ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT

PRINT "Ruby lips above the water,"    ' quarta estrofe
PRINT "Blowing bubbles soft and fine;"
PRINT "But alas, he was no swimmer,"
PRINT "So he lost his Clementine."

PRINT
PRINT "Oh my darling, oh my darling,"    ' coro
PRINT "Oh my darling, Clementine,"
```

(*continua*)

FIGURA 7-1. *continuação*

```
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT

INPUT "Pressione Enter para ver mais...", algo$
                                                ' pausa
PRINT

PRINT "Then the miner, forty-niner,"  ' quinta estrofe
PRINT "Soon began to peak and pine;"
PRINT "Thought he oughter join his daughter,"
PRINT "Now he's with his Clementine."

PRINT
PRINT "Oh my darling, oh my darling,"    ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT
```

A Vantagem da Procedure

A resposta é *Sim!* O QBasic oferece uma estrutura de programação chamada *procedure*, que lhe permite digitar um bloco de comandos, atribuir-lhe um nome e depois chamá-lo pelo nome sempre que você quiser que o programa o execute. As procedures oferecem as seguintes vantagens:

- **Procedures eliminam linhas repetidas.** Você pode definir uma procedure uma vez e fazer com que o programa a execute qualquer número de vezes.
- **Procedures tornam os programas mais fáceis de se ler.** Um programa dividido em um grupo de pequenas partes é mais fácil de relacionar e entender.
- **Procedures simplificam o desenvolvimento do programa.** Os programas separados em unidades lógicas são mais fáceis de projetar, escrever e depurar. Além do mais, se estiver escrevendo um programa com um amigo, poderá trocar procedures em vez de programas inteiros.
- **Procedures podem ser reutilizadas em outros programas.** Você pode incorporar suas procedures de uso geral em outros projetos de programação.

- **Procedures estendem a linguagem QBasic.** As procedures normalmente podem executar tarefas que não podem ser realizadas diretamente por comandos embutidos e funções do QBasic.

Neste capítulo você aprenderá a criar e usar duas procedures do QBasic: *subprogramas* e *funções*. Subprogramas lhe permitem subdividir o seu programa em unidades menores, que podem ser chamadas uma ou mais vezes. Funções retornam valores que o seu programa pode usar.

CRIAÇÃO DE SUBPROGRAMAS

Subprogramas tornam o seu código mais fácil de ler e reduzem a repetição. Um subprograma é um bloco de código entre os comandos SUB e END SUB. Você pode fazer com que seu programa chame um subprograma quantas vezes desejar. Quando um subprograma termina, o controle retorna ao comando seguinte à chamada do subprograma no programa principal. Em QBasic, você cria e armazena subprogramas separadamente dos programas principais.

Sintaxe de um Subprograma

A sintaxe de um subprograma é a seguinte:

```
SUB NomeSubprograma (listaParâmetros)
    declarações de variáveis e constantes locais
    comandos do subprograma
END SUB
```

- SUB é o comando do QBasic que marca o início da definição do subprograma.

- *NomeSubprograma* é o nome do subprograma. Ele pode ter até 40 caracteres de extensão, sendo o nome que o programa principal usa para chamar o subprograma. Ele não pode ser uma palavra-chave do QBasic ou igual a qualquer nome de variável ou procedure no programa.

- (*listaParâmetros*) é uma lista opcional de variáveis. (Veja "Passagem de Argumentos a um Subprograma", mais adiante.) Se você usar *listaParâmetros*, deverá cercá-la com parênteses.

- *declarações de variáveis e constantes locais* é uma lista opcional de variáveis e constantes declaradas e usadas dentro — e somente dentro — do subprograma. Elas não possuem efeito sobre variáveis ou constantes com o mesmo nome em outro lugar no programa.

- *comandos do subprograma* é a parte do subprograma que gera ação. Você pode usar quase todos os comandos do BASIC em um subprograma.
- END SUB é o comando do QBasic que marca o final da definição do subprograma.

Para ver como estes elementos trabalham em conjunto, examine o seguinte subprograma, chamado *Coro*. Toda vez que *Coro* é executado, ele imprime uma linha em branco, o coro de "Clementine" em quatro linhas e outra linha em branco.

```
SUB Coro
' O subprograma Coro mostra o coro de "Clementine."
PRINT
PRINT "Oh my darling, oh my darling,"          ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT
END SUB
```

Criação de um Subprograma

No ambiente do QBasic, os menus o ajudarão a criar e trabalhar com subprogramas de modo rápido e fácil. O QBasic oferece dois comandos de menu para trabalhar com subprogramas:

- O comando New SUB (no menu Edit)
- O comando SUBs (no menu View)

O comando New SUB

O comando New SUB [novo subprograma] é o comando de menu do QBasic que você usa para incluir um subprograma ao seu programa. Para criar um subprograma, você executa os seguintes passos:

1. Selecione o comando New SUB. O quadro de diálogo New SUB aparecerá.
2. Entre com o nome do subprograma (se ainda não estiver lá).
3. Digite os parâmetros e o corpo do subprograma.

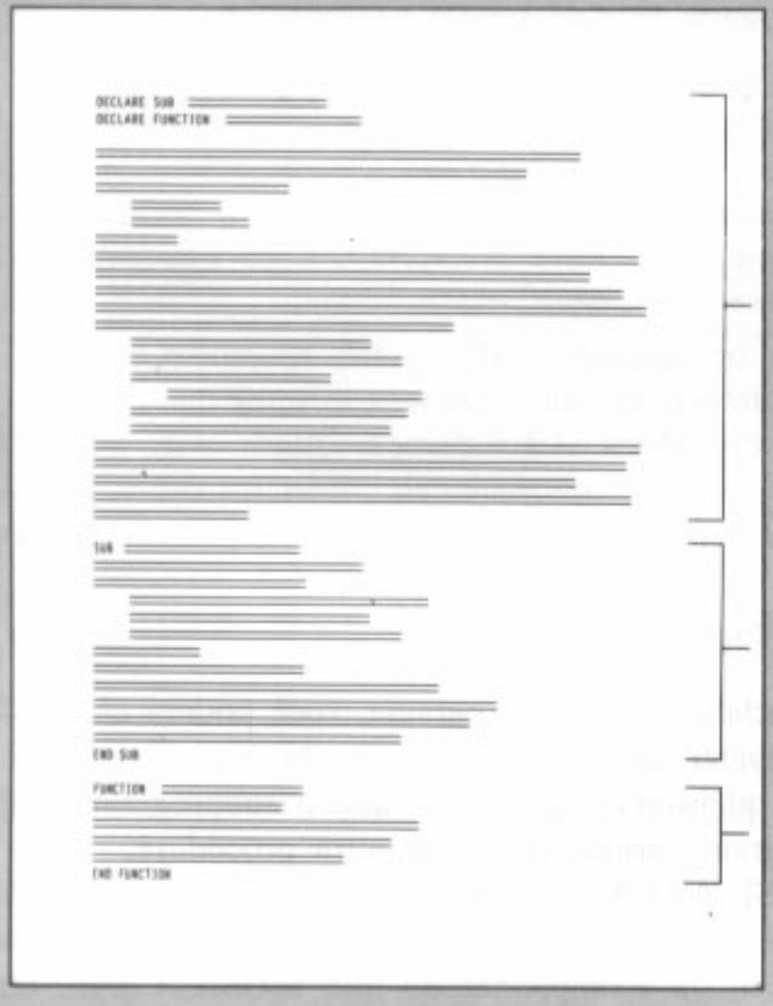
O comando SUBs

Você pode examinar e editar os seus subprogramas usando o comando SUBs [subprogramas]. A seleção do comando SUBs gera um quadro de diálogo, do qual você seleciona a parte do programa com que deseja trabalhar: o programa principal, um subprograma ou uma função.

NOTA: A tecla de atalho para o comando SUBs é F2 – pressionar esta tecla é o meio mais rápido de chegar ao quadro de diálogo SUBs.

Dividindo um Programa em Partes

Este capítulo descreve como organizar um programa em três blocos de código independentes: o *programa principal*, os *subprogramas* e as *funções*. Você pode ter qualquer quantidade de subprogramas e funções (chamadas, em conjunto, *procedures*) no seu programa, ou então nenhuma delas. Entretanto, todo programa deve ter uma seção de programa principal, que controla o fluxo geral do programa e chama os subprogramas e as funções.



Embora a princípio possa parecer um pouco de trabalho extra, aprender a dividir os seus programas nessas unidades organizacionais recompensará quando você começar a escrever programas maiores. Lembre-se de que não importa quantos subprogramas e funções um programa contenha, ele ainda será um único programa em QBasic.

O Comando DECLARE

Uma das vantagens do QBasic é que ele o ajuda a programar corretamente. Por exemplo, com base na informação fornecida quando você cria um subprograma, o QBasic inclui um comando DECLARE em cima do seu programa principal quando o programa é salvo no disco. Um comando DECLARE *declara* a existência de um subprograma ao programa principal. Todo subprograma deve ter um comando DECLARE correspondente no início do programa principal.

A sintaxe do comando DECLARE é a seguinte:

DECLARE SUB *NomeSubprograma (parâmetros)*

NomeSubprograma é o nome do subprograma, e *parâmetros* é uma lista de variáveis recebidas pelo subprograma.

NOTA: Embora o QBasic gere o comando DECLARE por conta própria, é responsabilidade sua manter a lista de parâmetros. Se você mudar a lista de parâmetros no subprograma, deverá mudar a lista de parâmetros no comando DECLARE da mesma forma. Veja maiores explicações na seção "Passagem de Argumentos a um Subprograma", mais adiante.

Por convenção, os comandos DECLARE aparecem após os comentários introdutórios em um programa e antes das declarações de constantes e variáveis. Além disso, o QBasic exige que os comandos DECLARE precedam quaisquer comandos executáveis.

Chamada de um Subprograma

Após ter criado e declarado um subprograma, você poderá *chamá-lo* (executá-lo). Para chamar um subprograma, basta especificar o seu nome — junto com quaisquer argumentos que o subprograma exija — no corpo do programa principal ou de outra procedure.

O programa MUSICA2.BAS (Figura 7-2) demonstra como declarar e chamar o subprograma *Coro*. MUSICA2.BAS tem nove linhas a menos e é mais fácil de acompanhar do que MUSICA.BAS. Observe como o subprograma *Coro* fica separado do programa, a partir da linha 57, pelos comandos SUB e END SUB. O programa principal contém cinco chamadas ao subprograma *Coro*. Observe também que o nome do subprograma *Coro* começa com uma letra maiúscula — adotaremos esta convenção em todo o livro para distinguir nomes de procedure dos nomes de variável e de outros comandos e funções do QBasic.

NOTA: A Figura 7-2 contém uma listagem do programa MUSICA2.BAS conforme seria gerada na impressora ou listada em um livro — não como aparece no ambiente do QBasic. Lembre-se de que dentro do QBasic todos os subprogramas são separados do programa principal e acessados por meio do comando SUBs no menu View. Para digitar este programa desde o início, siga as instruções na seção de prática a seguir.

```
1  ' MUSICA2.BAS
2  ' O programa mostra a letra da canção "Clementine."
3
4  DECLARE SUB Coro ()' declara Coro antes de usar
5
6  CLS
7  PRINT "- - - - - - - - - Clementine - - - - - - - - -"
8  PRINT
9
10 PRINT "In a cavern, in a canyon,"   ' primeira estofe
11 PRINT "Excavating for a mine,"
12 PRINT "Lived a miner, forty-niner,"
13 PRINT "And his daughter, Clementine."
14
15 Coro              ' chama o subprograma Coro
16
17 PRINT "Light she was and like a fairy,"
18                                     ' segunda estrofe
19 PRINT "And her shoes were number nine;"
20 PRINT "Herring boxes without topses,"
21 PRINT "Sandals were for Clementine."
22
23 Coro              ' chama o subprograma Coro
24
25 INPUT "Pressione Enter para ver mais...", algo$
26                                                  ' pausa
27 PRINT
28
29                                     ' terceira estrofe
30 PRINT "Drove she ducklings to the water,"
31 PRINT "Ev'ry morning just at nine;"
32 PRINT "Hit her foot against a splinter,"
33 PRINT "Fell into the foaming brine."
34
35 Coro              ' chama o subprograma Coro
36
```

FIGURA 7-2. *(continua)*
MUSICA2.BAS: Um programa que mostra a letra da canção "Clementine" usando um subprograma chamado Coro.

FIGURA 7-2. *continuação*

```
37  PRINT "Ruby lips above the water,"   ' quarta estrofe
38  PRINT "Blowing bubbles soft and fine;"
39  PRINT "But alas, he was no swimmer,"
40  PRINT "So he lost his Clementine."
41
42  Coro                        ' chama o subprograma Coro
43
44  INPUT "Pressione Enter para ver mais...", algo$
45                                                ' pausa
46  PRINT
47
48  PRINT "Then the miner, forty-niner,"  ' quinta estrofe
49  PRINT "Soon began to peak and pine;"
50  PRINT "Thought he oughter join his daughter,"
51  PRINT "Now he's with his Clementine."
52
53  Coro                        ' chama o subprograma Coro
54
55  END
56
57  SUB Coro
58  '
59  ' O subprograma Coro imprime o coro de "Clementine."
60
61  PRINT
62  PRINT "Oh my darling, oh my darling,"' coro
63  PRINT "Oh my darling, Clementine,"
64  PRINT "You are lost and gone forever,"
65  PRINT "Dreadful sorry, Clementine."
66  PRINT
67
68  END SUB
```

A Figura 7-3 dá uma visão detalhada da definição e chamada do subprograma *Coro*.

Declaração do subprograma	DECLARE SUB Coro () ' declara subprograma Coro
	...
Chamada do subprograma	Coro ' chama subprograma Coro
	...
Final do prog. principal	END

FIGURA 7-3. (*continua*)
Os componentes da definição e chamada de um subprograma típico.

FIGURA 7-3. *continuação*

Início do subprograma

```
SUB Coro

PRINT
PRINT "Oh my darling, oh my darling,"          ' coro
PRINT "Oh my darling, Clementine,"
PRINT "You are lost and gone forever,"
PRINT "Dreadful sorry, Clementine."
PRINT

END SUB
```

Corpo do subprograma

Fim do subprograma

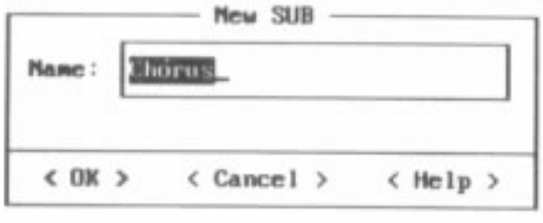
Prática: Entrando com o programa MUSICA2.BAS

O exercício a seguir o guiará pelos passos necessários para entrar com o programa MUSICA2.BAS (Figura 7-2). Você deverá adaptar e seguir estes passos sempre que incluir um novo subprograma ao seu programa.

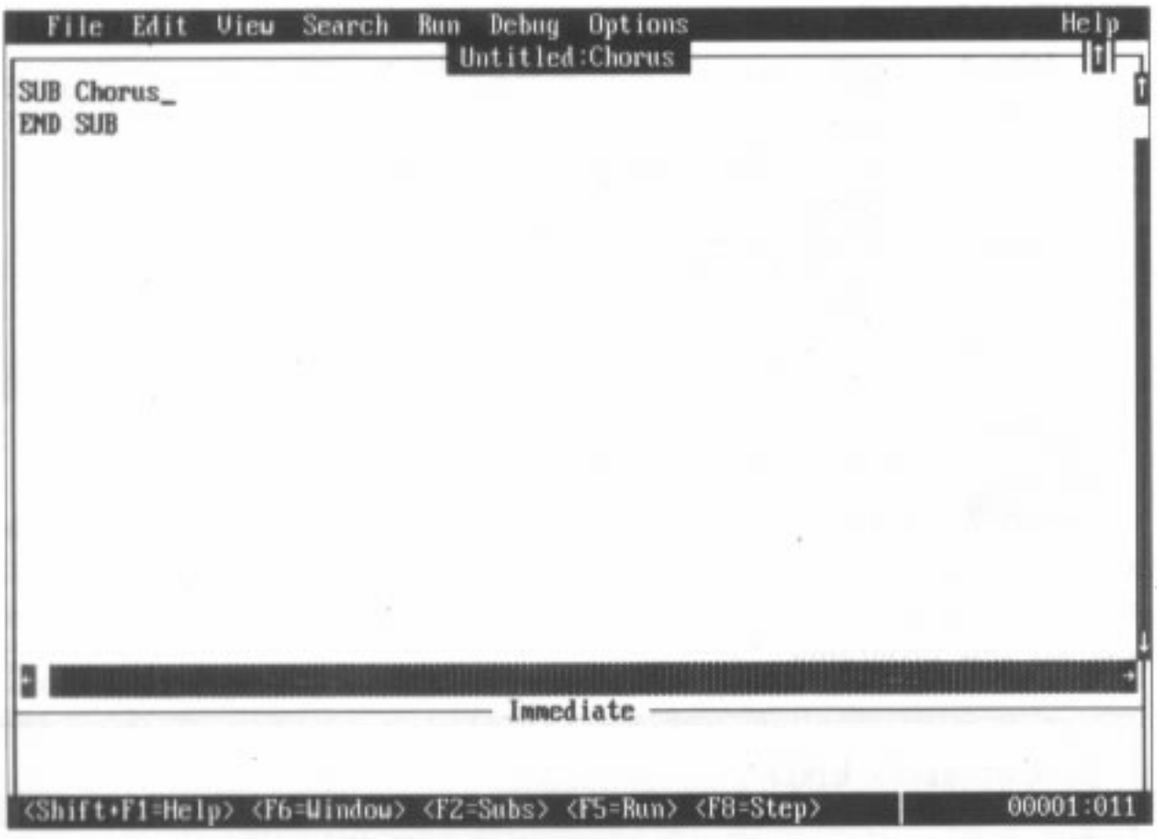
O Comando END

Em todo o livro você verá um comando END ao final da seção do programa principal quando o programa tiver subprogramas ou funções. END é uma instrução opcional que diz ao QBasic que o programa alcançou a última instrução e que a execução deve parar. Embora o QBasic não exija o comando END, ele é uma indicação visual útil de onde termina a seção do programa principal.

1. Escolha New pelo menu File para iniciar um novo programa e digite as linhas de 6 a 15 da Figura 7-2.
2. Coloque o cursor na chamada ao subprograma *Coro* (a palavra *Coro*) e selecione New SUB pelo menu Edit. O seguinte quadro de diálogo aparecerá em seguida:

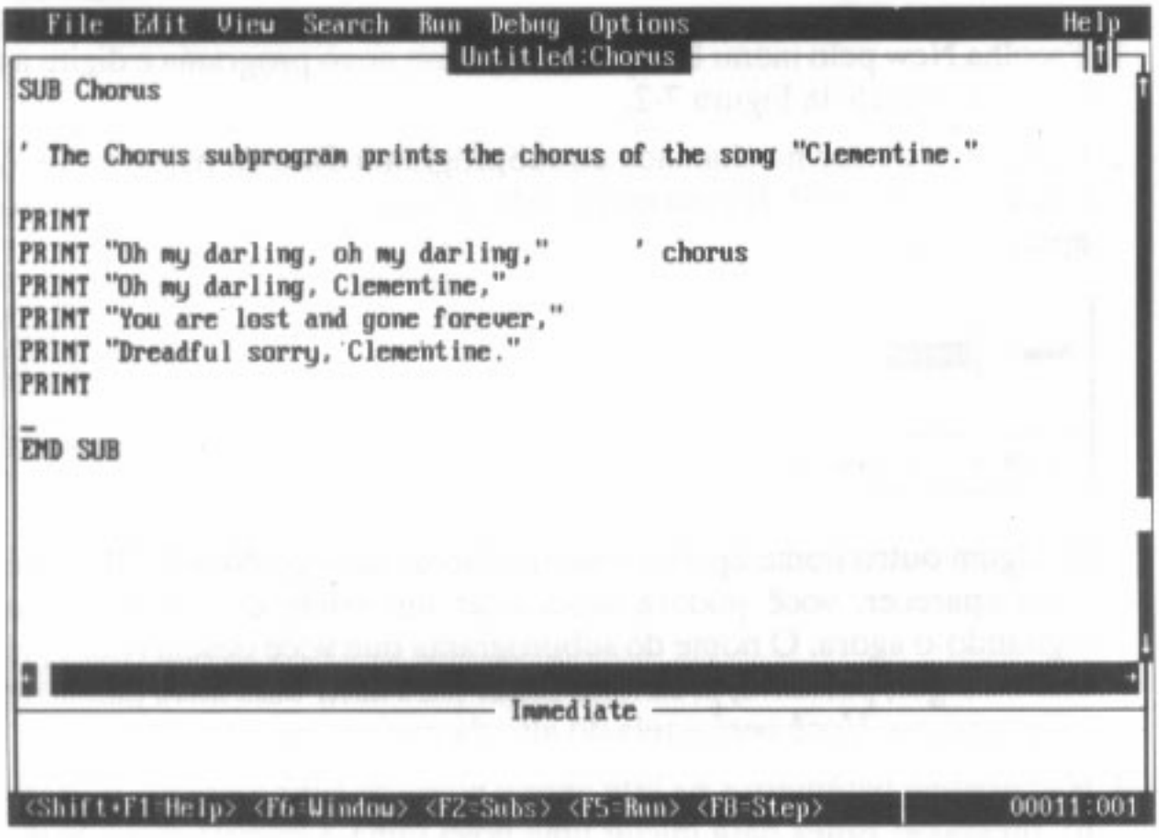


3. Se algum outro nome aparecer no quadro de diálogo New SUB, ou se nada aparecer, você poderá especificar um nome de subprograma digitando-o agora. O nome do subprograma que você deseja (*Coro*) já está no lugar; portanto, pressione Enter para abrir uma nova janela de subprograma. Você verá a tela do alto da página seguinte.
4. Não existem parâmetros na lista após o nome do subprograma; portanto, pressione Enter para iniciar uma nova linha. Observe que a barra

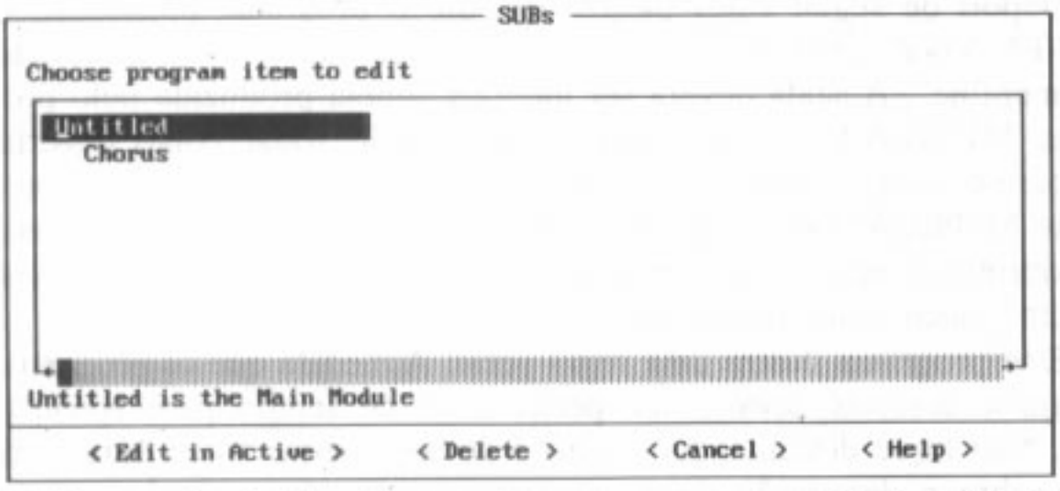


de título da janela mostra agora o nome do programa e o nome da procedure, separados por dois pontos.

5. Digite o corpo do subprograma (linhas 58 a 67 da Figura 7-2). Observe que o QBasic adicionou a linha 68 (o comando END SUB) por você. A sua tela deverá se parecer com esta:

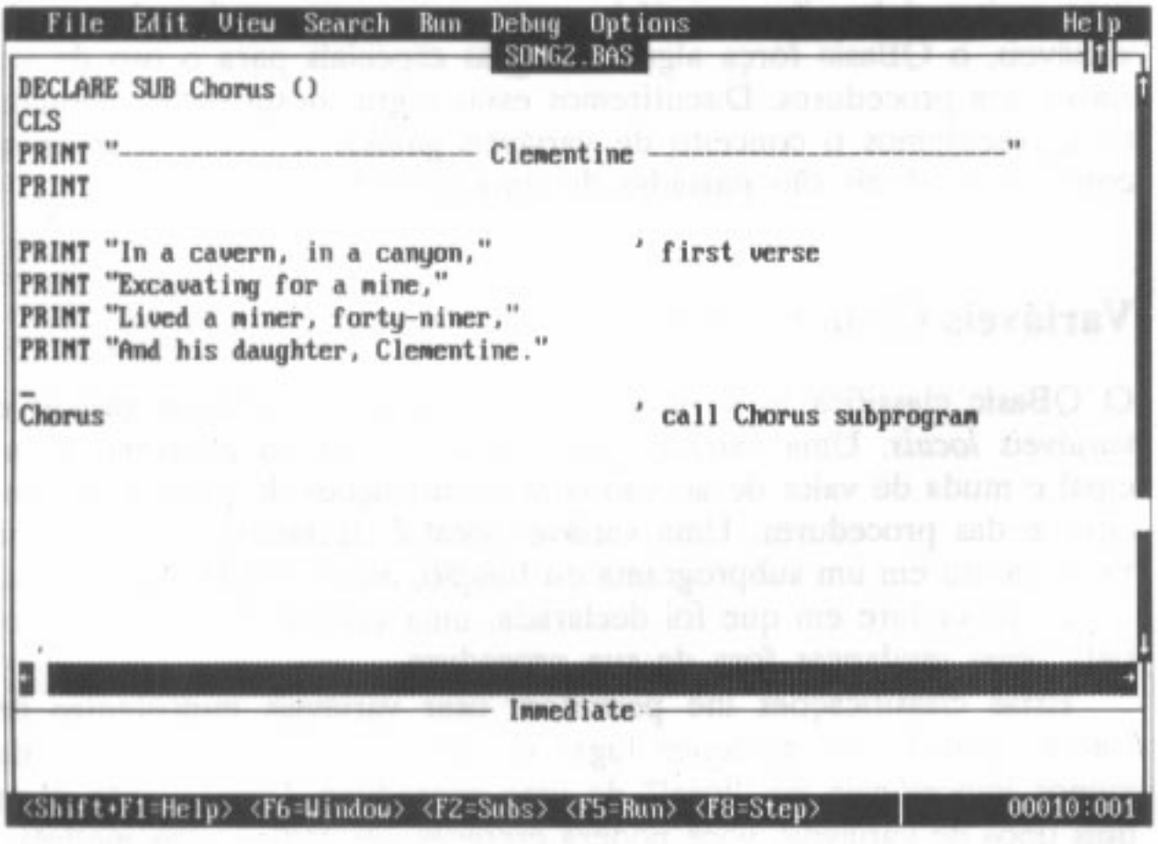


6. Selecione o comando SUBs pelo menu View para ver um quadro de diálogo mostrando o programa principal, subprogramas e funções no seu programa.



7. Use as teclas de direção para destacar o programa principal (atualmente *Untitled*) e pressione Enter. O QBasic mostra o programa principal na janela View e coloca o cursor na linha em que você o deixou.

8. Salve o seu programa com o nome MUSICA2.BAS. Antes do QBasic gravar o seu programa no disco, ele coloca um comando DECLARE no início do programa (linha 4 na Figura 7-2). Agora, sua tela deverá se parecer com esta:



9. Mova o cursor para a linha imediatamente após a chamada ao subprograma *Coro* e digite o restante do programa (linhas 16 a 55 da Figura 7-2).

10. Inclua as linhas 1, 2, 3 e 5 no início do programa e depois salve-o novamente.

Depois de seguir estes passos, execute o programa pressionando Shift-F5. Você deverá ver três telas sucessivas com a letra da canção "Clementine". A saída deverá ser idêntica àquela produzida pelo programa MUSICA.BAS. Se o seu programa não rodar como deveria, compare-o com a Figura 7-2 e conserte quaisquer diferenças. O programa MUSICA2.BAS é um pouco maior do que aqueles que vimos anteriormente neste livro, de modo que os erros de digitação podem ser um pouco mais difíceis de achar.

Este programa ilustra uma regra geral: À medida que os programas tornam-se maiores, leva-se um pouco mais de tempo para aprontar tudo. Não fique desanimado — isto é de se esperar. Você aprenderá mais sobre a depuração de programa e reparo de erros de programação comuns no Capítulo 12.

USO DE VARIÁVEIS COM PROCEDURES

Os programas que você escreveu até aqui usavam apenas algumas variáveis, mas, quando escrever programas maiores, poderá ter que usar muitas delas. Para ajudá-lo a gerenciar um grande número de variáveis, o QBasic força algumas regras especiais para o uso de variáveis em procedures. Discutiremos essas regras nesta seção, enquanto apresentamos o conceito de variáveis *globais* e *locais*, e veremos como as variáveis são passadas de uma procedure para outra.

Variáveis Globais e Locais

O QBasic classifica as variáveis em dois grupos: *variáveis globais* e *variáveis locais*. Uma variável global é declarada no programa principal e muda de valor de acordo com as instruções do programa principal e das procedures. Uma variável local é declarada no programa principal ou em um subprograma ou função, mas é válida apenas dentro da procedure em que foi declarada; uma variável local não é afetada pelas mudanças fora de sua procedure.

Estas classificações lhe permitem usar variáveis importantes de forma "global" em qualquer lugar de um programa e variáveis de menor importância no "local" de uma procedure. Distinguindo esses dois tipos de variáveis, você poderá escrever um código mais modular. Como as variáveis locais estão confinadas às suas próprias procedures, os programas que usam variáveis locais possuem uma finalidade mais geral do que os programas que usam variáveis globais.

Declaração de variáveis globais

Para declarar uma variável global, você coloca um comando COMMON SHARED próximo ao início do programa principal. Esta declaração permite que uma variável seja usada e modificada em qualquer lugar do programa principal ou de suas procedures. Entretanto, ele não atribui um valor à variável. Os valores são atribuídos mais tarde no programa com os comandos de atribuição. A sintaxe do comando COMMON SHARED é a seguinte:

COMMON SHARED *listaVariáveis*

onde *listaVariáveis* é uma lista de uma ou mais variáveis separadas por vírgulas. Você pode declarar qualquer número de variáveis em um comando COMMON SHARED.

Prática: Usando variáveis globais

O programa GLOBAL.BAS (Figura 7-4) mostra como a variável global *jogo$* é declarada no programa principal e modificada pelo subprograma *OutroJogo*.

Digite o programa GLOBAL.BAS e execute-o. Você verá a seguinte saída:

```
No programa principal, jogo$ = Xadrez
No subprograma OutroJogo, jogo$ = Xadrez e gamão
De volta ao programa principal, jogo$ = Xadrez e gamão
```

O programa mostra que quaisquer mudanças feitas na variável global *jogo$* serão refletidas em todo o programa.

```
' GLOBAL.BAS
' O programa demonstra o uso de uma variável global.

DECLARE SUB OutroJogo()   ' declara subprograma

COMMON SHARED jogo$
                ' declara jogo$ como variável global

jogo$ = "Xadrez"    ' inicializa jogo$

CLS

PRINT "No programa principal, jogo$ = "; jogo$
                ' mostra no programa principal
OutroJogo
                ' mostra no subprograma
```

FIGURA 7-4. (*continua*)
GLOBAL.BAS: Um programa demonstrando o uso de variáveis globais.

FIGURA 7-4. *continuação*

```
        ' mostra no programa principal
PRINT "De volta ao programa principal, jogo$ = ";
jogo$

END

SUB OutroJogo

jogo$ = jogo$ + " e gamão"

PRINT "No subprograma OutroJogo, jogo$ = "; jogo$

END SUB
```

Declaração de variáveis locais

As variáveis são locais por omissão, de modo que não será preciso usar qualquer comando especial para declará-las. Na realidade, você já tem prática em declarar variáveis locais — fez isso o tempo todo!

Prática: Usando variáveis locais

O programa LOCAL.BAS (Figura 7-5) remove o comando COMMON SHARED do programa GLOBAL.BAS para demonstrar que uma variável local no programa principal e uma variável local em um subprograma não interagem uma com a outra. As duas variáveis são chamadas *jogo$*, e ambas são variáveis de cadeia. Observe que, embora tendo o mesmo nome, as duas variáveis locais são completamente independentes uma da outra.

Modifique o programa GLOBAL.BAS para que se iguale ao programa LOCAL.BAS e execute-o. Você verá a seguinte saída:

```
No programa principal, jogo$ = Xadrez
No subprograma OutroJogo, jogo$ = e gamão
De volta ao programa principal, jogo$ = Xadrez
```

Como podemos ver, a modificação de *jogo$* no subprograma *OutroJogo* não afeta a variável *jogo$* no programa principal.

```
' LOCAL.BAS
' Este programa demonstra o uso de uma variável
'  local.

DECLARE SUB OutroJogo()      ' declara subprograma
```

FIGURA 7-5. (*continua*)
LOCAL.BAS: Um programa demonstrando o uso de uma variável local.

FIGURA 7-5. *continuação*

```
jogo$ = "Xadrez"    ' inicializa jogo$

CLS

PRINT "No programa principal, jogo$ = "; jogo$
      ' mostra no programa principal
OutroJogo
      ' mostra no subprograma
PRINT "De volta ao programa principal, jogo$ = ";
jogo$
      ' mostra no programa principal
END

SUB OutroJogo

jogo$ = jogo$ + " e gamão"

PRINT "No subprograma OutroJogo, jogo$ = "; jogo$

END SUB
```

Passagem de Argumentos a um Subprograma

Você pode tornar acessável qualquer quantidade de variáveis declarando-as globalmente. Embora seja conveniente, este método aumenta as chances da sua variável ser modificada acidentalmente em algum local no programa.

Assim, o QBasic oferece um meio de compartilhar argumentos (como variáveis, constantes e resultados de expressões e funções) com um número limitado de subprogramas em vez de um programa inteiro. Este método de compartilhar é chamado *passagem de argumentos.*

Os argumentos passados a um subprograma são recebidos por *parâmetros*, que são variáveis locais dentro de um subprograma. Você pode usar essas variáveis exatamente como quaisquer outras.

Argumentos versus parâmetros

Vamos formalizar a diferença entre os termos *argumento* e *parâmetro*:

- **Um argumento é uma constante, uma variável ou uma expressão que é *passada* a um subprograma.** Os nomes de argumento aparecem após o nome do subprograma em uma chamada de procedure. Um grupo de nomes de argumento é chamado *lista de argumentos.*

- **Um parâmetro é uma variável que *recebe* um valor passado a um subprograma.** Os nomes de parâmetro aparecem em comandos SUB e DECLARE e seguem as regras que se aplicam aos tipos de dados padrões. Um grupo de nomes de parâmetro é chamado *lista de parâmetros*.

Cada argumento deve ter um parâmetro correspondente do mesmo tipo (mas não necessariamente com o mesmo nome). A Figura 7-6 mostra a relação entre uma lista de argumentos e uma lista de parâmetros.

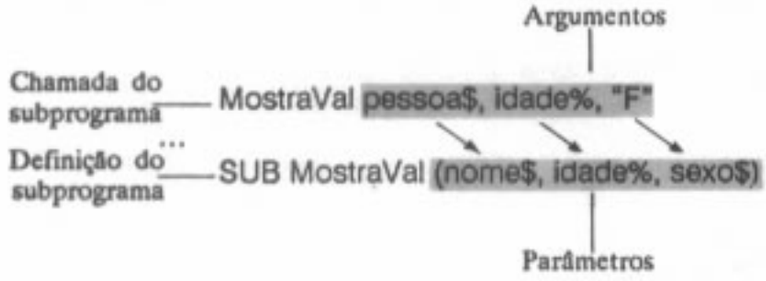


FIGURA 7-6.
A relação entre argumentos e parâmetros.

A lista a seguir mostra alguns pares argumento-parâmetro válidos:

```
PegaEntrada nome$, numero$, endereco$
...
SUB PegaEntrada (pessoa$, fone$, endereco$)

Traduz pig$, dog$
...
SUB Traduz (porco$, cachorro$)

ImprCabec TIME$, DATE$, titulo$, pagina%
...
SUB ImprCabec (horaAtual$, dataAtual$, titulo$, num%)

ChamadaMista intTeste%, cadeiaTeste$, VALOR!, "Olá", 747
...
SUB ChamadaMista (a%, b$, c!, d$, e%)
```

Modificando argumentos

A passagem de argumentos a um subprograma não é uma rua de mão única. Qualquer valor que um subprograma atribui a um parâmetro é passado de volta ao argumento correspondente quando um subprograma termina.

Vamos usar a Figura 7-6 como exemplo. Se *pessoa$* tivesse o valor *Elisabete* na hora em que o programa chamou *MostraVal*, o parâmetro *nome$* receberia esse valor.

Se o subprograma em seguida atribuísse o valor *Viviane* a *nome$*, esse valor seria atribuído a *pessoa$* após o término do subprograma.

No entanto, não se esqueça de que as constantes e as expressões são definitivamente constantes: seus valores não podem mudar. Na Figura 7-6, por exemplo, mesmo que *MostraVal* alterasse o valor de *sexo$*, nada aconteceria com o argumento correspondente ("*F*").

Prática: Passando argumentos a um subprograma

O programa ARGUMENT.BAS (Figura 7-7) demonstra como dois argumentos são passados ao subprograma *SomaJuro*. O parâmetro *nomeMes$* recebe o argumento *mes$*, e o parâmetro *valor!* recebe o argumento *saldo!*. O subprograma *SomaJuro* depois muda o mês, multiplica o valor do parâmetro *valor!* por 1.05 e retorna os dois valores ao programa principal.

```
' ARGUMENT.BAS
' Este programa demonstra a passagem de argumentos
'  para um subprograma.

DECLARE SUB SomaJuro (nomeMes$, valor!)

mes$ = "Janeiro"    ' inicializa mês$ com "Janeiro"
saldo! = 1500       ' inicializa saldo! com 1500

CLS

PRINT "Antes do subprograma:"    ' mostra originais
PRINT " mes$ = "; mes$
PRINT " saldo! ="; saldo!
PRINT

SomaJuro mes$, saldo!      ' chama subprograma SomaJuro

PRINT "Após o subprograma:"  ' mostra valores
                             ' modificados
PRINT " mes$ = "; mes$
PRINT " saldo! ="; saldo!

END

SUB SomaJuro (nomeMes$, valor!)

nomeMes$ = "Fevereiro"    ' muda nome do mês
valor! = valor! * 1.05    ' soma 5% ao valor

END SUB
```

FIGURA 7-7.
ARGUMENT.BAS: Programa para passar argumentos a um subprograma.

Digite o programa ARGUMENT.BAS e execute-o. Você verá a seguinte saída:

```
Antes do subprograma:
  mes$ = Janeiro
  saldo! = 1500

Após o subprograma:
  mes$ = Fevereiro
  saldo! = 1575
```

Tanto *mes$* quanto *saldo!* foram modificados sem o uso de variáveis globais.

CRIAÇÃO DE FUNÇÕES

Uma função é um bloco de código colocado entre os comandos FUNCTION e END FUNCTION. As funções seguem as mesmas regras gerais dos subprogramas, com uma exceção importante: uma função realiza uma tarefa e retorna um valor ao programa principal ou à procedure que a chama. Você usa uma função da mesma forma como usa uma função embutida do QBasic — incluindo-a em um comando do QBasic. Nesta seção você aprenderá como criar suas próprias funções — chamadas *funções definidas pelo usuário* — com o QBasic.

Sintaxe de uma Função

A sintaxe de uma função é a seguinte:

```
FUNCTION NomeFunção (listaParâmetros)
    declarações de variáveis e constantes locais
    comandos da função
    NomeFunção = valorRetorno
END FUNCTION
```

- FUNCTION é o comando do QBasic que marca o início da definição de função.
- *NomeFunção* é o nome da função, e deve terminar com um caracter de declaração de tipo (assim como um nome de variável também pode). O nome da função pode ter até 40 caracteres e é usado para chamar a função. O nome da função não pode ser uma palavra-chave do QBasic ou igual a qualquer nome de variável ou procedure no seu programa.
- (*listaParâmetros*) é uma lista opcional de variáveis.
- *declarações de variáveis e constantes locais* é uma lista opcional de variáveis e constantes declaradas e usadas dentro da função. Elas não possuem efeito sobre as variáveis ou constantes com os mesmos nomes em outro lugar no programa.

- *comandos da função* é a parte da função que realiza o trabalho. Você pode usar praticamente todos os comandos do QBasic em uma função.
- *NomeFunção = valorRetorno* é um comando de atribuição no corpo da função (geralmente próximo ao final) que define o valor de retorno da função. *valorRetorno* pode ser uma variável simples ou o resultado de uma expressão ou de outra função. *NomeFunção* e *valorRetorno* devem ser do mesmo tipo.
- END FUNCTION é o comando do QBasic que marca o final da definição de função.

Para ver como esses elementos trabalham em conjunto, examine a seguinte chamada, que passa três argumentos inteiros e retorna um valor inteiro atribuído à variável *numero%*:

```
numero% = SomaDosTermos%(a%, b%, c%)
```

Observe que o nome da função, *SomaDosTermos%*, contém um caracter de declaração de tipo inteiro, correspondente ao caracter de declaração de tipo em *numero%*. Uma função definida pelo usuário retorna um valor em um dos cinco tipos de dados do QBasic: cadeia, inteiro, inteiro longo, ponto flutuante de precisão simples ou ponto flutuante de dupla precisão.

Criação de uma Função

A criação de uma função definida pelo usuário é muito semelhante à criação de um subprograma:

1. Entre com a função usando o comando New Function [nova função] do menu Edit, assim como usou o comando New SUB para entrar com um subprograma em outro ponto deste capítulo.
2. Examine e edite as suas funções usando o comando SUBs pelo menu View. Todos os subprogramas e funções serão listados juntos no quadro de diálogo SUBs.

Prática: Usando uma função definida pelo usuário

O programa MEDIA.BAS (Figura 7-8) demonstra vários elementos das funções definidas pelo usuário na declaração de uma função definida pelo usuário chamada *Media!*. *Media!* retorna a média de três parâmetros inteiros que são passados à função. Depois que o QBasic executar a função *Media!*, o nome *Media!* representará um valor de ponto flutuante de precisão simples no programa principal, que pode ser atribuído a uma variável de ponto flutuante ou exibido com um comando PRINT.

```
' MEDIA.BAS
' Este programa imprime a média de três números.

DECLARE FUNCTION Media! (int1%, int2%, int3%)
                                    ' declara função

CLS

PRINT "Entre com três inteiros para tirar a média."
PRINT
INPUT "  Primeiro inteiro: ", primeiro$
INPUT "  Segundo inteiro: ", segundo$
INPUT "  Terceiro inteiro: ", terceiro$

PRINT
PRINT "A média é"; Media!(primeiro%, segundo%,
terceiro%)

END

FUNCTION Media! (int1%, int2%, int3%)

soma% = int1% + int2% + int3%    ' soma dos argumentos

Media! = soma% / 3
         ' valor de retorno é a média dos argumentos

END FUNCTION
```

FIGURA 7-8.
MEDIA.BAS: Um programa que imprime o valor retornado por uma função.

Digite o programa MEDIA.BAS e execute-o. Você verá uma saída semelhante a esta:

```
Entre com três inteiros para tirar a média.

  Primeiro inteiro: 10
  Segundo inteiro: 20
  Terceiro inteiro: 50

A média é 26.66667
```

Semelhanças com Subprogramas

As funções têm uma série de características em comum com os subprogramas:

- Uma chamada de função contém o mesmo número de argumentos presentes na lista de parâmetros da função.
- Uma lista de parâmetros de função recebe valores passados pelo programa principal ou procedure que a chama, atribuindo-lhes nomes de variáveis para que possam ser usados no corpo da função.
- Uma função pode declarar e usar variáveis e constantes locais.
- Uma função é declarada próxima ao início do programa principal com um comando DECLARE, que é gerado automaticamente pelo QBasic.
- Uma função é armazenada separadamente no ambiente do QBasic e tem acesso por meio do comando SUBs no menu View.
- As funções podem ajudá-lo a modularizar seus programas, evitando um código repetitivo; isso é obtido colocando-se as rotinas mais usadas em módulos fáceis de usar.

QUE TIPO DEVE SER USADO?

Por enquanto você pode até pensar que subprogramas e funções são a mesma coisa. Afinal, tanto subprogramas quanto funções são estruturas de programação flexíveis, que podem cuidar de uma série de tarefas e retornar um ou mais valores ao módulo que faz a chamada. Então, quais são as diferenças reais entre subprogramas e funções, e qual deverá ser usado em uma determinada situação na programação?

Um Subprograma É um Miniprograma

Pense em um subprograma como um pequeno programa completo. Um subprograma deverá completar uma tarefa importante em um programa e ainda ter uma finalidade geral suficiente para que seja usado em outros projetos de programação. Os subprogramas normalmente são maiores do que as funções, e geralmente são usados quando a informação é mostrada na tela. Em geral, as tarefas a seguir são mais adequadas a subprogramas:

- Apanhar entrada do usuário
- Mostrar informações na tela
- Processar vários valores numéricos ou cadeias
- Desenhar formas e estilos gráficos
- Tocar notas musicais ou canções
- Retornar valores múltiplos ao módulo de chamada

Uso do Comando Split

O comando Split [partir] no menu View lhe permite ver dois locais no seu código ao mesmo tempo. Isso é útil sobretudo quando o seu programa possui muitos subprogramas e funções, e você deseja editar uma procedure enquanto vê outra. A seguir mostramos uma lista de passos que você deverá seguir quando usar o comando Split. Como este comando pode ser usado de muitas maneiras diferentes, experimente algumas para ver qual a que lhe atende melhor.

1. Selecione o comando Split pelo menu View. A janela View será partida horizontalmente em duas janelas. A janela superior, contendo o cursor, será a janela ativa.
2. Se precisar fazer edições na janela superior, passe ao local correto e edite. Se as duas janelas View tiverem a mesma parte do programa, as edições feitas na janela superior serão refletidas na janela View inferior assim que o cursor sair da linha que está sendo editada.
3. Pressione F6 para ativar a janela View inferior (ou clique sobre ela com o mouse). Se quiser voltar à janela superior novamente, pressione F6 duas vezes (ou clique na janela com o mouse).
4. Se quiser colocar o conteúdo de uma procedure em uma das janelas divididas, selecione a janela onde deseja colocar a procedure, pressione F2 para ver a lista de procedures e selecione a procedure que deseja carregar.
5. Quando tiver terminado de usar as janelas, escolha o comando Split pelo menu View para voltar ao modo de única janela. A janela que estiver ativa no momento tornar-se-á a única janela

Uma Função Retorna um Valor

O ponto forte de uma função é calcular e retornar um único valor ao programa principal; as funções não poderão retornar múltiplos valores ou executar tarefas gerais. Em geral, as tarefas a seguir são mais adequadas a funções:

- Execução de um cálculo numérico
- Retorno de um valor de cadeia
- Geração de um número randômico
- Conversão de um valor para outro

- Avaliação de uma expressão lógica e retorno de um valor verdadeiro ou falso
- Cálculo de um resultado a partir de vários argumentos

O Programa Principal Cuida da Declaração e do Controle

Por quais tarefas um programa principal é responsável? Na verdade, não por muitas, se você utilizar muitos subprogramas e funções. As tarefas a seguir são mais adequadas ao programa principal:

- Comentários e explicações introdutórias.
- Inicialização das principais variáveis e estruturas
- Código do programa que é executado apenas uma vez
- Estruturas de controle de fluxo que determinam o caminho da execução do programa

As declarações a seguir, se aparecerem no programa, *devem* ser incluídas no programa principal:

- Declarações de variáveis e constantes globais
- Declarações de subprograma e função

Agora que você já aprendeu o programa principal, subprogramas e funções, deverá estar pronto para começar a escrever programas modulares e com boa organização.

RESUMO

Neste capítulo você aprendeu a respeito de subprogramas e funções — duas estruturas de programação que evitam a repetição e estendem a linguagem QBasic. Os subprogramas são definidos entre os comandos SUB e END SUB, e as funções entre FUNCTION e END. Os dois tipos de procedures são entrados e acompanhados por meio do sistema de menu do QBasic. Ambos aceitam variáveis locais e a troca de argumentos com o módulo de chamada. Os subprogramas e as funções, combinados com as estruturas de laço e controle de fluxo, discutidas nos Capítulos 5 e 6, compõem um conjunto completo de ferramentas que você poderá usar para escrever programas estruturados e bem organizados.

PERGUNTAS E EXERCÍCIOS

1. Quais dos seguintes itens são vantagens da programação com procedures?

a. Procedures podem ser chamadas inúmeras vezes.

b. Procedures lhe permitem criar seus próprios comandos e funções.

c. Procedures facilitam o desenvolvimento do programa.

d. Procedures de uso geral podem ser incorporadas a outros projetos de programação.

2. O que há de errado com o seguinte comando SUB?

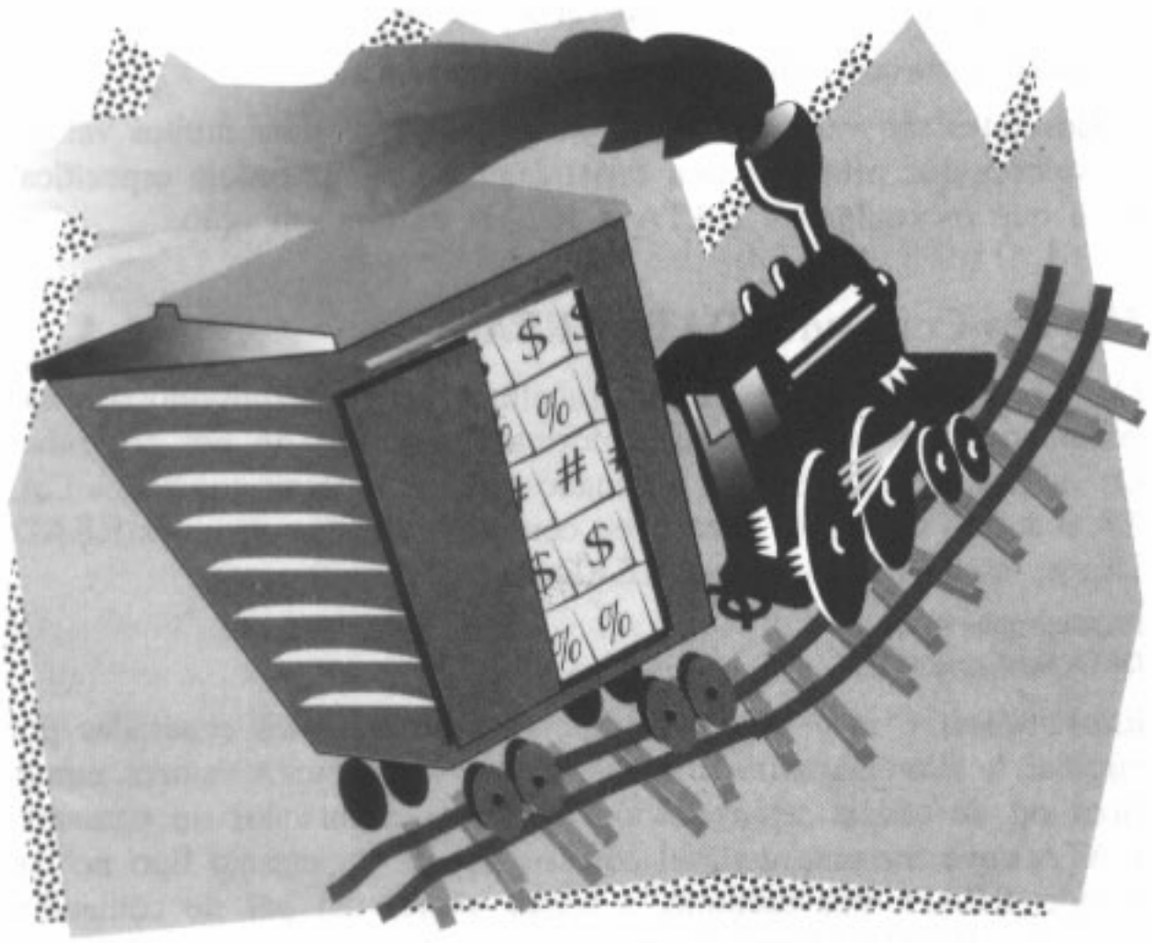
```
SUB EntraNome$ (nome$, sobrenome$)
```

3. Para se chegar ao comando SUBs do menu View, que tecla de função pode ser usada?
4. Que comando de menu pode ser usado para executar uma procedure inteira em um só passo?
5. Escreva um comando do QBasic que declare uma variável global chamada *total%*.
6. Qual a diferença entre um subprograma e uma função?
7. Escreva um subprograma chamado *PegaDadosCarro* que peça ao usuário informações sobre marca, modelo, ano e cor de um automóvel e retorne esta informação ao programa principal em quatro variáveis. Quando terminar, escreva um comando que chame o subprograma *PegaDadosCarro*.
8. Escreva uma função chamada *AchaMaior%* que aceite dois argumentos inteiros e retorne o maior deles ao módulo de chamada. Quando terminar, escreva um comando que chame a função *AchaMaior%*.
9. Escreva um programa que use subprogramas para imprimir um grupo de caracteres arrumados de modo a formarem uma dentre três formas: uma linha, um retângulo ou um triângulo. Quando rodar o programa, ele deverá produzir uma saída semelhante à que mostramos na tela seguinte.

```
Este programa imprime um grupo de caracteres na forma especificada.

Que caracter gostaria de usar? ^

Que forma gostaria de ver?

    1) Triângulo
    2) Retângulo
    3) Linha

Forma (1, 2 ou 3): 1

          ^
         ^^^
        ^^^^^
       ^^^^^^^
      ^^^^^^^^^
     ^^^^^^^^^^^
    ^^^^^^^^^^^^^
   ^^^^^^^^^^^^^^^
  ^^^^^^^^^^^^^^^^^
 ^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^

Pressione qualquer tecla para continuar
```

CAPÍTULO 8

Uso de Grandes Volumes de Dados



Agora que você já aprendeu como trabalhar com tipos de dados e variáveis simples, é hora de expandir o seu conhecimento — para aprender como o QBasic trata de grandes quantidades de dados em um programa. Usando o conhecimento adquirido neste capítulo, você poderá criar estruturas de dados — conjuntos de itens de dados individuais — que o ajudarão a organizar grandes quantidades de informação e a agilizar operações envolvendo muitas variáveis.

ARMAZENAMENTO E RECUPERAÇÃO DE INFORMAÇÕES

Você já aprendeu três formas de armazenar valores em um programa:

- Atribuindo um valor a uma constante:

```
CONST VALOR! = 12.95
```

- Atribuindo um valor a uma variável:

```
nome$ = "Davi"
```

- Atribuindo um valor a uma variável com INPUT:

```
INPUT "Entre com número de rodadas: ", rodadas%
```

Todos estes são valores simples. Mas, e se você tiver muitos valores — valores que pretende usar repetidamente, numa ordem específica? É aí que os comandos DATA e READ entram em ação.

Uso dos Comandos DATA e READ

Os comandos DATA e READ trabalham juntos permitindo o armazenamento e recuperação de informações dentro de um programa. Os valores são inicialmente ordenados em um ou mais comandos DATA e depois atribuídos a variáveis com um ou mais comandos READ. Use os comandos DATA e READ da seguinte forma:

```
READ listaVariáveis
DATA listaConstantes
```

listaVariáveis é uma lista com uma ou mais variáveis separadas por vírgulas, e *listaConstantes* é uma lista de um ou mais valores numéricos ou de cadeia separados por vírgulas. Cada valor no comando DATA deve ter uma variável correspondente do mesmo tipo no comando READ. Por exemplo, a seguir vemos um par de comandos DATA e READ. Observe como os valores aparecem na mesma ordem das variáveis.

```
READ nome$, idade%
DATA João da Silva, 9
```

Um exemplo simples demonstra melhor como DATA e READ trabalham em conjunto. No programa a seguir, o comando READ atribui o primeiro valor DATA à variável de cadeia *nome$* e o segundo valor DATA à variável inteira *idade%*. O comando PRINT, em seguida, utiliza as variáveis.

```
READ nome$, idade%
PRINT nome$; " tem"; idade%; "anos de idade."
DATA João da Silva, 9
```

Quando rodar este programa, a sua tela de saída deverá mostrar o seguinte resultado:

```
João da Silva tem 9 anos de idade.
```

A Figura 8-1 mostra como os valores de dados são atribuídos às variáveis.

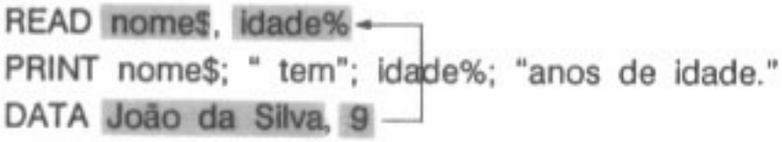


FIGURA 8-1.
Valores armazenados em um comando DATA são atribuídos a variáveis com um comando READ.

Comandos DATA e READ: Dicas úteis

Tenha em mente os seguintes fatos quando começar a usar os comandos DATA e READ para armazenar e recuperar informações:

- Você pode criar múltiplos comandos DATA e READ; basta assegurar-se de que cada variável de READ possui um valor correspondente em DATA, como em:

```
READ mes$, numeroDeDias%
READ feriados%

DATA novembro
DATA 30, 3
```

- Se um valor no comando DATA tiver uma vírgula, dois pontos ou espaços iniciais ou finais, o valor inteiro deverá ficar entre aspas, como vemos a seguir:

```
DATA "    Total    ",    "Silva Jardim, RJ"
```

Espaços iniciais — Espaços finais — Vírgula incluída

- Comandos DATA devem aparecer dentro do programa principal. (Eles normalmente são colocados no final do programa principal.) Comandos READ devem estar localizados *acima* dos comandos

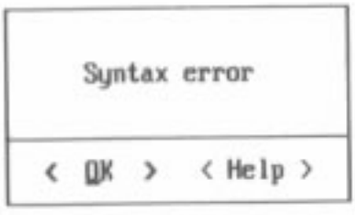
DATA no programa principal, mas podem aparecer em qualquer lugar nas procedures do programa.

E se os tipos não combinarem?

Cada valor em um comando DATA deverá ser atribuído a uma variável correspondente do mesmo tipo em um comando READ. Se os tipos não combinarem, o QBasic fará as seguintes conversões:

Se DATA for	*Mas READ for*	*O resultado da conversão é*
cadeia	inteiro/ponto flutuante	*erro de sintaxe*
inteiro/ponto flutuante	cadeia	cadeia
inteiro	ponto flutuante	ponto flutuante
ponto flutuante	inteiro	arredondado para inteiro

Um erro de sintaxe é indicado pelo seguinte quadro:



O programa a seguir demonstra uma discordância de tipo entre um valor de DATA de cadeia e uma variável READ inteira.

```
READ nome$
PRINT "Nome: "; nome$
READ endereco$
PRINT "Endereço: "; endereco$
READ idade%
PRINT "Idade: "; idade%

DATA João da Silva, "R. das Margaridas, 211, São Paulo", nove
```

Ao rodar o programa, a sua tela deverá se parecer com a seguinte:

```
Nome: João da Silva
Endereço: R. das Margaridas, 211, São Paulo
```

A mensagem *Sintaxe error* aparecerá em seguida, já que o comando READ não pode atribuir um valor de cadeia (*nove*) a uma variável inteira (*idade%*). Para consertar o programa, mude *nove* para *9*, a fim de ajustá-lo à variável inteira *idade%*, ou então mude *idade%* para *idade$*, ajustando-a ao valor de cadeia *nove*.

Valores Típicos do Comando DATA

Os comandos DATA e READ são mais apropriados quando você sabe de antemão os valores das variáveis e se os valores sempre aparecerem na mesma ordem. Os valores a seguir são sempre adequados ao método DATA e READ de armazenamento e recuperação:

- Dias da semana (domingo a sábado)
- Meses do ano (janeiro a dezembro)
- Nomes de pessoas, lugares e organizações
- Dados numéricos para cálculo ou análise
- Valores numéricos para notas musicais

Prática: Armazenando diversos valores

O programa SOMA.BAS (Figura 8-2) mostra como usar um comando READ sem um laço FOR para atribuir vários valores de DATA às variáveis. Observe que a constante inteira *ITENS%* controla o número de itens que são lidos dos comandos DATA. Alterando *ITENS%*, você pode alterar o número de itens a serem lidos.

1. Digite o programa SOMA.BAS e execute-o.

```
' SOMA.BAS
' Este programa lê e soma valores de comandos DATA.

CONST ITENS% = 20       ' define número de itens a ler

CLS

FOR i% = 1 TO ITENS%    ' para cada item a ler
  READ numero!          ' atribui o próximo item a numero!
  soma! = soma! + numero! ' soma o item ao total
NEXT i%

PRINT "A soma dos"; ITENS%; "números é"; soma!

DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
DATA 88.2, 25, 3.3, 100, -74.2, 0, 20, 0.34, -89,
5.4567
```

FIGURA 8-2.
SOMA.BAS: Um programa que usa READ para atribuir 20 itens de dados a variáveis.

A sua tela de saída deverá mostrar o seguinte resultado:

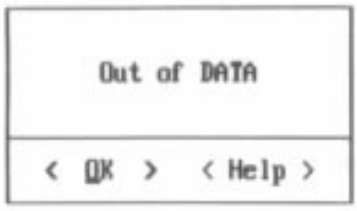
```
A soma dos 20 números é 134.0967
```

2. Mude o valor da constante *ITENS%* para *4* e rode o programa novamente. A sua tela de saída deverá mostrar o seguinte resultado:

```
A soma dos 4 números é 10
```

O marcador de fim-de-dados

A menos que o seu programa possa identificar o último valor de DATA, ele pode continuar executando comandos READ — tentando atribuir um valor a uma variável. O resultado é a seguinte mensagem de erro, que indica que não há mais dados disponíveis para atribuição:



Para evitar este erro, use um *marcador de fim-de-dados* como entrada final no seu comando DATA. O marcador de fim-de-dados é um valor que o seu programa compara a fim de determinar o final dos valores de DATA. Quando ler este valor final, ele continuará executando o restante do programa. Esta técnica pode ser útil quando você possuir uma longa lista de valores em comandos DATA, que serão processados uma única vez. O programa SOMA.BAS resolve este problema em potencial usando uma constante contendo o número exato de valores como limite superior para o laço FOR. Muitas vezes você não terá esse luxo. O programa a seguir, SOMA2.BAS, foi modificado para verificar um marcador de fim-de-dados.

Acompanhando Valores de DATA: O Apontador de Dados

Para ajudar o comando READ a atribuir valores às variáveis, o QBasic usa um *apontador de dados* para apontar para o próximo valor do comando DATA a ser atribuído. Quando um programa começa, o apontador de dados aponta para o primeiro valor no primeiro comando DATA. À medida que a execução do programa prossegue e valores de DATA são atribuídos, o apontador de dados aponta para o próximo valor sem atribuição do comando DATA.

Prática: Checando um marcador de fim-de-dados

O programa SOMA2.BAS (Figura 8-3) usa um marcador de fim-de-dados (-9999) como última entrada de DATA no programa.

Digite o programa SOMA2.BAS e execute-o.

```
' SOMA2.BAS
' Este programa lê e soma valores de em comandos
' DATA até achar o marcador de fim-de-dados (-9999).

CLS

DO WHILE (numero! <> -9999)  ' laço até ler marcador
  READ numero!               ' atribui próximo item
  IF (numero! <> -9999) THEN ' se não for fim-de-dados
     soma! = soma! + numero! '  mantém uma contagem
     itens% = itens% + 1     '  conta valores lidos
  END IF
LOOP

PRINT "A soma dos"; itens%; "números é"; soma!

DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
DATA 88.2, 25, 3.3, 100, -74.2, 0, 20, 0.34, -89,
5.4567
DATA -9999
```

FIGURA 8-3.
SOMA2.BAS: Um programa que lê dados até chegar a um marcador de fim-de-dados.

A sua tela de saída deverá mostrar o seguinte resultado:

```
A soma dos 20 números é 134.0967
```

Relendo valores de DATA com o comando RESTORE

Às vezes você pode querer ler uma lista de valores de DATA mais de uma vez em um programa. Por exemplo, você pode querer executar uma série de cálculos diferentes com o mesmo conjunto de valores. No ponto em que desejar voltar ao primeiro valor de DATA, use o comando RESTORE, que zera o contador de dados para o primeiro valor do comando DATA no programa. Você pode usar RESTORE quantas vezes quiser em um programa.

O comando RESTORE tem a seguinte sintaxe:

```
RESTORE
```

Prática: Usando RESTORE para repetir uma lista de valores

O programa HORASTV.BAS (Figura 8-4) usa RESTORE, DATA e READ para verificar quantas horas uma pessoa vê televisão durante um período de três semanas. Um par de comandos FOR endentados percorre os dias da semana três vezes (para simular três semanas), acompanhando o número total de horas na TV. O comando RESTORE, ao final de cada ciclo, volta o apontador de dados para Segunda.

NOTA: Para melhorar a tela de saída, o comando COLOR mostra o número total de horas na TV em vermelho piscante. Você só verá isso se tiver um monitor colorido.

1. Digite o programa HORASTV.BAS e execute-o. Você receberá três grupos de pedidos semelhantes a este:

```
Quantas horas de TV você viu nas últimas 3 semanas?

Segunda, Semana 1 —> 0
Terça, Semana 1 —> 1
Quarta, Semana 1 —> 0
Quinta, Semana 1 —> 2
Sexta, Semana 1 —> 0
Sábado, Semana 1 —> 2.5
Domingo, Semana 1 —> 2
```

2. Responda aos pedidos. Ao entrar com todos os 21 valores, a tela de saída mostrará o resultado final, que aparecerá da seguinte forma:

```
Você viu 23.5 horas de TV durante as três últimas semanas!
```

```
' HORASTV.BAS
' Este programa usa DATA, READ e RESTORE para
'  acompanhar o número de horas em que a TV foi
'  vista durante as três últimas semanas.

CLS

PRINT "Quantas horas de TV você viu nas últimas 3
semanas?"
PRINT

FOR i% = 1 TO 3      ' para cada uma das 3 semanas
  FOR j% = 1 TO 7    '  e para cada dia da semana
    READ dia$        '    lê nome do dia na lista DATA
```

FIGURA 8-4. (*continua*)
HORASTV.BAS: Um programa demonstrando o comando RESTORE.

FIGURA 8-4. *continuação*

```
    PRINT dia$; ", Semana"; i%; ' mostra dia e semana
    INPUT "- -> ", horas!
          ' pede número de horas para esse dia
    totHoras! = totHoras! + horas!    ' acumula
  NEXT j%
  PRINT         ' linha em branco após cada semana
  RESTORE       ' move apontador para Segunda para
NEXT i%         ' a próxima iteração

PRINT "Você viu";       ' mostra total de horas
COLOR 20                ' em vermelho piscante
PRINT totHoras!; "horas";
COLOR 7                 ' volta à cor branca
PRINT "horas de TV durante as três últimas semanas!"

DATA Segunda, Terça, Quarta, Quinta, Sexta, Sábado,
Domingo
```

Agora que já viu como armazenar e recuperar grandes quantidades de valores no seu programa, é hora de aprender um meio eficiente de trabalhar com eles. A seção a seguir apresenta o *vetor*, uma poderosa estrutura de dados que poderá ajudá-lo a manipular grandes quantidades de dados do mesmo tipo.

TRABALHO COM VETORES

Quando quiser organizar variáveis do mesmo tipo sob um único nome, use um vetor. Assim como uma embalagem de ovos organiza uma dúzia de ovos individuais, um vetor lhe permite agrupar muitos valores sob um único nome.

Um vetor pode conter qualquer um dos seguintes tipos de dados, com os quais você já trabalhou até aqui; ou seja, você pode ter

- Vetores de cadeia
- Vetores de inteiros
- Vetores de inteiros longos
- Vetores de ponto flutuante com precisão simples
- Vetores de ponto flutuante com precisão dupla

Você não pode misturar os tipos dentro de um vetor; ou seja, um vetor de cadeia só poderá conter cadeias de caracteres, um vetor de inteiros só poderá conter números inteiros etc.

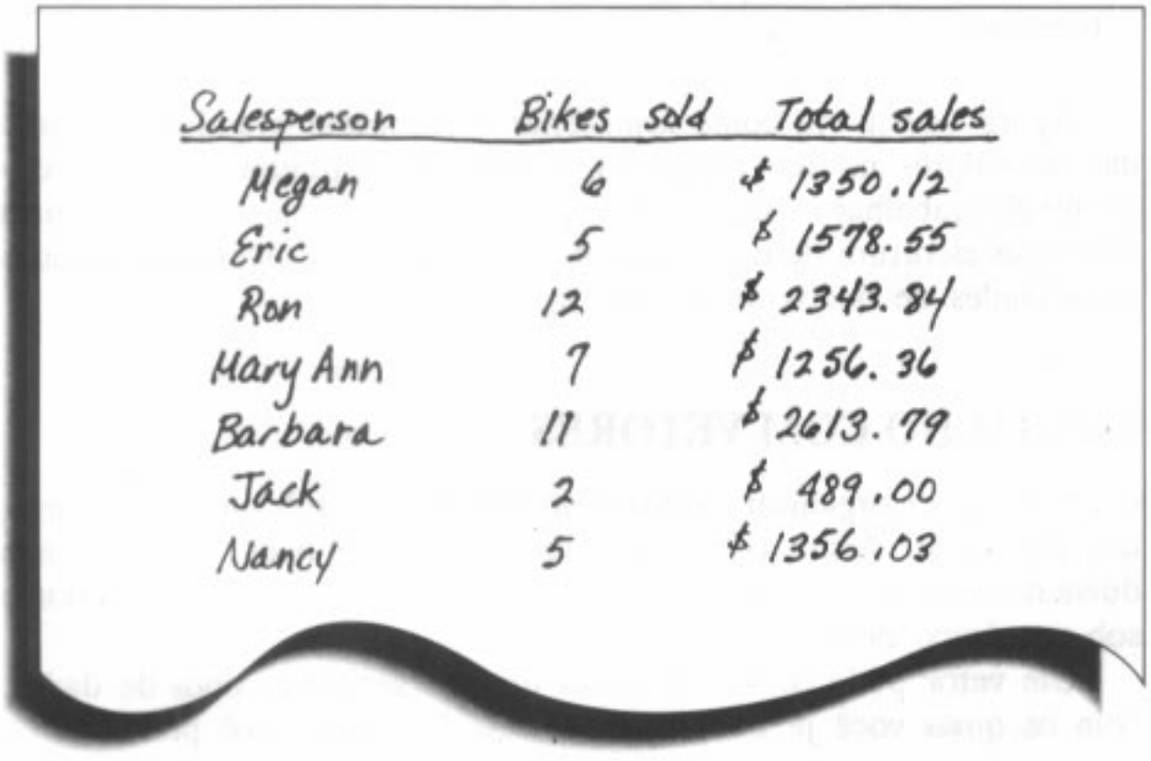
Vejamos um exemplo que organiza os dados em três tipos de vetores: vetores de cadeia, vetores de inteiros e vetores de ponto flutuante com precisão simples.

Acompanhando a Informação com Vetores: Mercado de Motos do Ernani

O Mercado de Motos do Ernani, uma grande loja de motos no centro da cidade, possui sete vendedores. Ernani deseja acompanhar dois valores a cada mês:

- O número de motos vendidas por cada vendedor
- O valor das vendas por cada vendedor

Ao escrever esta informação em papel, ela certamente ficará disposta em três listas, como mostra a Figura 8-5:



Salesperson	Bikes sold	Total sales
Megan	6	$ 1350.12
Eric	5	$ 1578.55
Ron	12	$ 2343.84
Mary Ann	7	$ 1256.36
Barbara	11	$ 2613.79
Jack	2	$ 489.00
Nancy	5	$ 1356.03

FIGURA 8-5.
Um exemplo de dados de vendedor do Mercado de Motos do Ernani.

- Uma lista de valores de cadeia (nomes dos vendedores)
- Uma lista de valores inteiros (o número de motos vendidas por cada vendedor)
- Uma lista de valores de ponto flutuante (o total de vendas para cada vendedor)

Criando essas listas, Ernani já terá trabalhado com vetores. Na verdade, ele criou três vetores: um vetor de cadeias para os nomes dos

vendedores, um vetor de inteiros para o número de motos vendidas por cada vendedor, e um vetor de ponto flutuante com precisão simples para o valor total das vendas. Cada lista, com seu tipo de informação distinto, qualifica um vetor. E cada vetor contém sete *elementos*, ou valores individuais. Agora que Ernani organizou seus vetores no papel, poderá começar a convertê-los para um programa com a ajuda do comando DIM.

O Comando DIM: Reservas para o Vetor

Quando você faz uma reserva em um restaurante, presta informações particulares em troca de uma mesa garantida. Ou seja, informando seu nome, o tipo de refeição que será servida (café, almoço ou jantar), e o número de pessoas convidadas, você garantirá um espaço. No mundo dos vetores, o comando DIM tem a mesma finalidade de uma reserva desse tipo. Ele oferece espaço na memória para um vetor — e tudo que ele pede em troca são três informações:

- O nome selecionado para o vetor
- O tipo de dado que você pretende armazenar no vetor
- O número máximo de elementos que o vetor deverá acomodar

Este processo de alocação de espaço é chamado *dimensionamento.*

O comando DIM possui a seguinte sintaxe:

DIM *nomeVetor(subscrito)*

nomeVetor é o nome que você seleciona para o vetor e *subscrito* é o elemento com a posição mais alta no vetor. O caracter final do *nomeVetor* deve ser um caracter de declaração de tipo identificando o tipo de dado do vetor: *$* para cadeias, *%* para inteiros, *&* para inteiros longos, *!* para números de ponto flutuante com precisão simples ou *#* para números de ponto flutuante com precisão dupla.

O comando DIM a seguir, por exemplo, dimensiona um vetor de cadeia de caracteres que pode conter até 50 elementos numerados de 0 a 49:

```
DIM capEstados$(49)
```

O elemento com maior número no seu vetor
O tipo de dado do seu vetor ($, %, !, & ou #)
O nome do seu vetor

Quando você executa este comando DIM, o QBasic reserva espaço na memória para um vetor de 50 elementos para conter os nomes das capitais de estado.

NOTA: Um computador típico tem espaço para muitos vetores, cada um contendo milhares de elementos. Mas como a memória do computador é limitada, você não deve reservar espaço de memória para mais elementos do que a quantidade que realmente usará.

Ernani criaria os três vetores de que precisa usando esses comandos DIM:

- Para os nomes dos vendedores, um vetor de cadeia para sete elementos chamado *grupoVendas$*:

```
DIM grupoVendas$(6)
```

- Para o número de motos que cada um vendeu, um vetor de inteiros para sete elementos chamado *motosVend%*:

```
DIM motosVend%(6)
```

- Para o valor total de vendas de cada vendedor, um vetor de ponto flutuante com precisão simples para sete elementos chamado *totVendas!*:

```
DIM totVendas!(6)
```

Observe que em todos os três vetores o número *6* reserva espaço de memória para sete elementos numerados de 0 a 6.

Cada elemento no vetor é associado a um número. Por default, o QBasic associa o primeiro elemento de um vetor ao número 0, o segundo elemento ao número 1, e assim por diante para cada elemento do vetor, como vemos na Figura 8-6.

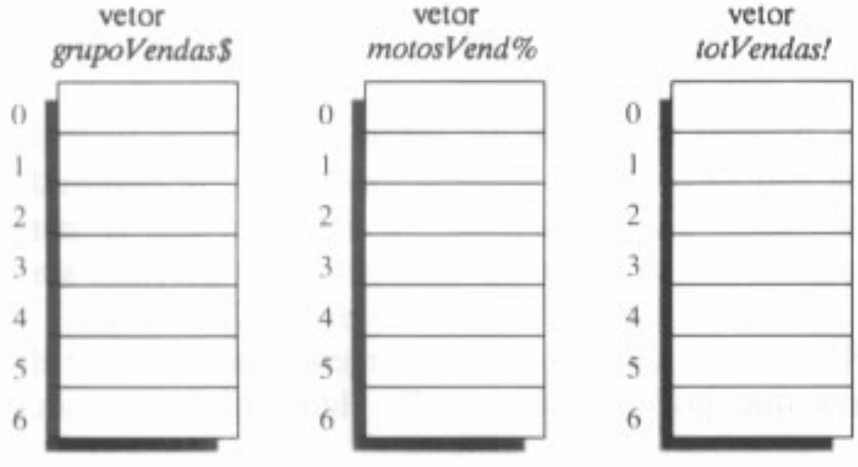


FIGURA 8-6.
Cada elemento de um vetor dimensionado é associado a um número.

Uso de Elementos do Vetor

Depois que dimensionar um vetor com o comando DIM, será fácil referir-se a qualquer um dos elementos dentro dele. Para referir-se a um elemento de um vetor, use o nome do vetor e um *índice* do vetor entre parênteses. O índice deve ser um valor inteiro; por exemplo,

ele pode ser um número simples ou uma variável inteira. O comando a seguir atribui o valor de cadeia *Tomate* ao elemento 3 no vetor *listaCompras$*:

```
listaCompras$(3) = "Tomate"
```

O Comando OPTION BASE

Se você acha que o seu programa ficaria conceitualmente mais nítido se o primeiro elemento em cada vetor fosse 1 em vez de 0, use o comando OPTION BASE. O comando OPTION BASE associa o primeiro elemento — ou *base* — de todos os vetores em um programa ao número 1. Para usar OPTION BASE, basta colocar o seguinte comando próximo ao início do programa, antes de qualquer comando DIM:

```
OPTION BASE 1
```

Neste capítulo os programas usam o comando OPTION BASE desta forma.

Prática: Armazenando valores em um vetor

O programa PEGANOME.BAS (Figura 8-7) demonstra como a informação é armazenada em um vetor e impressa usando dois laços FOR. PEGANOME usa a constante PESSOAS% para guardar o número de vendedores no mercado de motos do Ernani e OPTION BASE para fazer com que o primeiro elemento de todos os vetores no programa sejam 1 em vez de 0.

Digite o programa PEGANOME.BAS e execute-o.

```
' PEGANOME.BAS
' Este programa lê informação de cadeia para um
' vetor.

CONST PESSOAS% = 7          ' número de vendedores
OPTION BASE 1               ' base dos vetores será 1

DIM grupoVendas$(PESSOAS%)   ' dimensiona vetor
grupoVend$

CLS

FOR i% = 1 TO PESSOAS%     'i% acessa os elementos
```

FIGURA 8-7. (*continua*)
PEGANOME.BAS: Um programa demonstrando a carga e impressão de um vetor.

FIGURA 8-7. *continuação*

```
INPUT "Entre com nome do vendedor:", grupoVendas$(i%)
NEXT i%

PRINT
PRINT "Você entrou com os seguintes nomes:"
PRINT

FOR i% = 1 TO PESSOAS%    ' imprime todo o vetor
  PRINT grupoVendas$(i%)
NEXT i%
```

Você deverá fornecer os nomes para os vendedores; verá uma saída parecida com a seguinte:

```
Entre com nome do vendedor: Marcos
Entre com nome do vendedor: Eric
Entre com nome do vendedor: Roni
Entre com nome do vendedor: Mariana
Entre com nome do vendedor: Bárbara
Entre com nome do vendedor: Jaques
Entre com nome do vendedor: Nanci

Você entrou com os seguintes nomes:

Marcos
Eric
Roni
Mariana
Bárbara
Jaques
Nanci
```

Prática: Usando múltiplos vetores em um programa

O programa INFOMOTO (Figura 8-8) demonstra como uma série de vetores pode ser usada em um programa para gerenciar informações relacionadas. INFOMOTO é uma revisão do programa PEGANOME. Esta versão relaciona os nomes dos vendedores ao vetor *grupoVend$*, o número de motos que cada um vendeu ao vetor *motosVend%* e o valor total das vendas de cada pessoa ao vetor *totVendas!*. Os três vetores foram elaborados para uso em conjunto — cada um contém uma informação sobre os vendedores na loja, como mostra a Figura 8-6. Se você se referir a um mesmo índice do vetor em um laço FOR, seu programa poderá acessar itens relacionados

Formatação da Sua Saída: O Comando PRINT USING

O comando PRINT USING lhe permite formatar a aparência da sua saída na tela. PRINT USING é útil sobretudo se você precisar mostrar grandes quantidades de dados de forma colunar. Para criar o projeto que deseja, você usa uma cadeia de *gabarito*. A sintaxe para o comando PRINT USING é a seguinte:

```
PRINT USING gabarito; listaArgumento
```

gabarito é uma cadeia e *listaArgumento* é um grupo de um ou mais valores a serem mostrados, separados por ponto-e-vírgulas. *gabarito* especifica como os valores na *listaArgumento* deverão ser mostrados. Os caracteres de formatação no *gabarito* deverão corresponder um a um aos caracteres dos valores em *listaArgumento*. A tabela a seguir descreve alguns dos caracteres de formatação úteis, que poderão ser incluídos em *gabarito*:

Caracter(es)	*Descrição*
#	Representa um dígito de um valor numérico
.	Representa o ponto decimal em um valor numérico
$$	Faz com que um cifrão seja mostrado com um número
\\	Representa um ou mais espaços que podem ser preenchidos com dados de cadeia

O programa a seguir cria um gabarito de formatação chamado *gab$*, utilizando-o em um comando PRINT USING para mostrar uma variável de cadeia e um valor monetário de ponto flutuante com precisão simples:

```
gab$ = "Nome: \          \ Pagamento: $$#####.##"
NOME$ = "João da Silva"
pagamento! = 2496.33
PRINT USING gab$; nome$; pagamento!
```

Ao executar os comandos, verá esta saída:

```
Nome: João da Silva  Pagamento: $2496.33
```

ao mesmo tempo. O comando PRINT USING e o gabarito *gab$* mostram os dados em cada vetor.

Digite o programa INFOMOTO.BAS e execute-o.

```
' INFOMOTO.BAS
' Este programa lê e imprime dados de 3 vetores.

CONST PESSOAS% = 7      ' número de vendedores
OPTION BASE 1           ' define base de vetores em 1

DIM grupoVend$(PESSOAS%)  ' dimensiona vetor de cadeia
DIM motosVend%(PESSOAS%)  ' dimensiona vetor inteiro
DIM totVendas!(PESSOAS%)  ' dimensiona vetor
                          ' fracionário

CLS

FOR i% = 1 TO PESSOAS% ' pega nome e dados de vendas
  INPUT "Entre com nome do vendedor: "; grupoVend$(i%)
  INPUT "  Motos vendidas: ", motosVend%(i%)
  INPUT "  Total de vendas: $", totVendas!(i%)
  PRINT
NEXT i%

PRINT "Você entrou com os seguintes dados de vendas:"
PRINT
PRINT "Vendedor    Motos vendidas    Total de vendas"
PRINT  "- - - - - - - - - - - - - - - - - - - - - - -"
PRINT

' inicializa gab$, um gabarito para PRINT USING
gab$ = "\              \ ###          $$#####.##"

FOR i% = 1 TO PESSOAS% ' imprime conteúdo de cada vetor
  PRINT USING gab$; grupoVend$(i%); motosVend%(i%);
totVendas!(i%)
NEXT i%
```

FIGURA 8-8.
INFOMOTO.BAS: Um programa demonstrando o uso de três vetores em um laço FOR.

Você verá uma saída semelhante a esta:

```
Entre com nome do vendedor: Marcos
  Motos vendidas: 6
  Total de vendas: $1350.12

Entre com nome do vendedor: Eric
  Motos vendidas: 5
  Total de vendas: $1578.55
```

```
Entre com nome do vendedor: Roni
  Motos vendidas: 12
  Total de vendas: $2343.84

Entre com nome do vendedor: Mariana
  Motos vendidas: 7
  Total de vendas: $1256.36

Entre com nome do vendedor: Bárbara
  Motos vendidas: 11
  Total de vendas: $2613.79

Entre com nome do vendedor: Jaques
  Motos vendidas: 2
  Total de vendas: $489

Entre com nome do vendedor: Nanci
  Motos vendidas: 5
  Total de vendas: $1356.03

Você entrou com os seguintes dados de vendas:

Vendedor        Motos vendidas        Total de vendas
- - - - - - - - - - - - - - - - - - - - - - - - - - -

Marcos              6                  $1350.12
Eric                5                  $1578.55
Roni               12                  $2343.84
Mariana             7                  $1256.36
Bárbara            11                  $2613.79
Jaques              2                   $489.00
Nanci               5                  $1356.03
```

Preenchimento de Parte de um Vetor

Você não precisa encher um vetor até a borda. Na verdade, em geral é uma boa idéia permitir algum espaço extra para inclusão futura de novos elementos. No entanto, se pretente fazer isso, precisa de um meio de indicar ao QBasic que o usuário acabou de entrar com os dados. Um meio de fazer isso é usando um marcador de fim-de-dados, como já vimos neste capítulo. Você pode fazer com que o usuário inclua elementos no vetor como parte de um laço WHILE que verifica continuamente o marcador de fim-de-dados. Assim que o QBasic vê o marcador, ele sai do laço e executa o restante do programa. O marcador de fim-de-dados deve ter o mesmo tipo de dado do vetor preenchido, e deverá ser um valor que o usuário nunca deverá digitar durante a execução normal do programa. Um marcador de fim-de-dados típico para um vetor de cadeia pode ser *FIM* ou *TERMINA*. Para um vetor numérico, use *-9999*.

Prática: Usando um marcador de fim-de-dados

O programa FIMDADOS.BAS (Figura 8-9) demonstra como preencher um vetor com diferentes quantidades de dados. O programa preenche e imprime três vetores novamente, mas desta vez dimensiona os vetores com 50 elementos cada um e usa um marcador de fim-de-dados *FIM* para indicar que o usuário terminou a entrada de dados. (Observe que *FIM* deverá ser entrado com letras maiúsculas.)

Digite o programa FIMDADOS.BAS e execute-o.

```
' FIMDADOS.BAS
' Este programa lê dados em três vetores e os
'  imprime. O máximo de nomes a ser entrado é 50;
'  pode-se entrar menos que isso digitando "FIM"
'  para o nome do vendedor.

OPTION BASE 1        ' define base de vetores em 1

DIM grupoVend$(50)     ' dimensiona vetor de cadeia
DIM motosVend%(50)     ' dimensiona vetor inteiro
DIM totVendas!(50)     ' dimensiona vetor fracionário

CLS

PRINT "Siga os pedidos para entrar com dados de
vendas."
PRINT "Digite END para terminar."
PRINT

cont% = 1            ' inicializa contador do vetor

                     ' continua até nome="FIM"
WHILE (grupoVend$(conta%) <> "FIM")
  INPUT "Entre com nome do vendedor: ";
grupoVend$(cont%)
  IF (grupoVend$(cont%) <> "FIM") THEN
     INPUT "  Motos vendidas: ", motosVend%(cont%)
     INPUT "  Total de vendas: $", totVendas!(cont%)
     PRINT
     cont% = cont% + 1 ' incrementa contador do vetor
  END IF
WEND
```

FIGURA 8-9. (*continua*)
FIMDADOS.BAS: Um programa que usa um marcador de fim-de-dados para determinar quando a entrada termina.

FIGURA 8-9. *continuação*

```
PRINT
PRINT "Você entrou com os seguintes dados de
vendas:"
PRINT
PRINT "Vendedor    Motos vendidas    Total de vendas"
PRINT "- - - - - - - - - - - - - - - - - - - - - - - "
PRINT

' inicializa gab$, um gabarito para PRINT USING
gab$ = "\              \ ###          $$####.##"

' imprime conteúdo de cada vetor
FOR i% = 1 TO cont% - 1
  PRINT USING gab$; grupoVend$(i%); motosVend%(i%);
totVendas!(i%)
NEXT i%
```

Você verá uma saída semelhante a esta:

```
Siga os pedidos para entrar com dados de vendas.
Digite FIM para terminar.

Entre com nome do vendedor: Nanci
  Motos vendidas: 5
  Total de vendas: $1356.03

Entre com nome do vendedor: Bárbara
  Motos vendidas: 11
  Total de vendas: $2613.79

Entre com nome do vendedor: FIM

Você entrou com os seguintes dados de vendas:

Vendedor        Motos vendidas          Total de vendas
- - - - - - - - - - - - - - - - - - - - - - - - - - - -

Nanci             5                     $1356.03
Bárbara           11                    $2613.79
```

Criação de Vetores Flexíveis

Como já vimos, quando você usa o comando DIM deve informar ao QBasic o número de elementos no seu vetor para que possa reservar uma quantidade adequada de espaço na memória. E se você não souber com certeza quantos elementos o seu vetor terá? Por exemplo,

se o seu programa depender da entrada do usuário para o conteúdo do vetor, o número de elementos entrados pode variar a cada execução do programa. Como você pode dizer ao QBasic quanta memória deverá ser reservada?

O comando DIM é muito flexível. Na verdade, ele pode criar dois tipos de vetores, dependendo do tipo de informação que você oferece: *vetores estáticos* e *vetores dinâmicos*.

Vetores estáticos

Um vetor estático tem um tamanho específico — que você já conhece antes da execução do programa. Por exemplo, no programa INFOMOTO.BAS, o número de vendedores é conhecido de antemão e colocado no programa como uma constante:

```
CONST PESSOAS% = 7
```

O comando DIM em seguida usa a constante PESSOAS% como número de elementos no vetor:

```
DIM grupoVend$(PESSOAS%)
```

Vetores dinâmicos

Um vetor dinâmico é um vetor cujo tamanho pode mudar. Um vetor dinâmico é dimensionado apenas quando o usuário do programa define o número de elementos que o vetor terá. Para criar um vetor dinâmico, você deve seguir estes passos:

1. Use o comando INPUT para pedir ao usuário o número de elementos.
2. Atribua o valor entrado pelo usuário a uma variável inteira.
3. Use a variável inteira com o comando DIM para dimensionar o vetor.

Use um vetor estático quando souber de antemão o tamanho que o vetor deverá ter. Use um vetor dinâmico quando o tamanho do vetor for determinado novamente toda vez que o programa é executado.

Prática: Usando uma variável para definir o tamanho de um vetor.

O programa DINAMICO.BAS (Figura 8-10) usa a variável *pessoa%* para dimensionar os três vetores dinâmicos para vendas. Observe o comando IF, que verifica o valor de *pessoas%*: se *pessoas%* for menor ou igual a 0, nenhum vetor é dimensionado.

Digite o programa DINAMICO.BAS e execute-o.

```
' DINAMICO.BAS
' Este programa lê dados em três vetores
'  dinâmicos e os imprime.

OPTION BASE 1                ' define base de vetores em 1

CLS

INPUT "Quantos nomes de vendedor você deseja
entrar? ", pessoas%
IF (pessoas% > 0) THEN     ' pelo menos um vendedor

  DIM grupoVend$(pessoas%)' dimensiona vetor cadeia
  DIM motosVend%(pessoas%)' dimensiona vetor inteiro
  DIM totVendas!(pessoas%)' dimensiona vetor decimal

  PRINT

  ' pega nome do vendedor e dados de venda
  FOR i% = 1 TO pessoas%
     INPUT "Entre com nome do vendedor: "; grupoVend$(i%)
     INPUT "  Motos vendidas: ", motosVend%(i%)
     INPUT "  Total de vendas: $", totVendas!(i%)
     PRINT
  NEXT i%

  PRINT "Você entrou com os seguintes dados de
vendas:"
  PRINT
  PRINT "Vendedor   Motos vendidas    Total de vendas"
  PRINT "- - - - - - - - - - - - - - - - - - - - - - - "
  PRINT

  ' inicializa gab$, um gabarito para PRINT USING
  gab$ = "\              \ ###          $$####.##"

  FOR i% = 1 TO pessoas%  ' imprime conteúdo de cada
                          ' vetor
     PRINT USING gab$; grupoVend$(i%); motosVend%(i%);
totVendas!(i%)
  NEXT i%

END IF
```

FIGURA 8-10.
DINAMICO.BAS: Um programa demonstrando o uso de três vetores dinâmicos.

Você verá uma saída semelhante a esta:

```
Quantos nomes de vendedor você deseja entrar? 2

Entre com nome do vendedor: Marcos
  Motos vendidas: 6
  Total de vendas: $1350.12

Entre com nome do vendedor: Eric
  Motos vendidas: 5
  Total de vendas: $1578.55

Você entrou com os seguintes dados de vendas:

Vendedor        Motos vendidas        Total de vendas
- - - - - - - - - - - - - - - - - - - - - - - - - - -

Marcos          6                     $1350.12
Eric            5                     $1578.55
```

Procura de um Elemento em um Vetor

Às vezes você pode querer realizar uma pesquisa dentro de um vetor. Em geral isso é feito para encontrar um elemento específico no vetor ou para encontrar um elemento com base em uma comparação. As duas operações envolvem o rastreamento do vetor um elemento de cada vez, comparando cada elemento com uma cadeia sendo pesquisada.

- Na pesquisa por um elemento específico do vetor, o QBasic compara uma cadeia de pesquisa com cada elemento do vetor até igualar ou até que todos os elementos tenham sido examinados.
- Na pesquisa por comparação, você usa uma ou mais variáveis temporárias para acompanhar o progresso da comparação. As comparações geralmente tomam uma das seguintes formas:
  - Encontrar o maior número no vetor.
  - Encontrar o menor número no vetor.

O exercício a seguir usa métodos típicos para encontrar um elemento específico no vetor e para descobrir o maior número em um vetor.

Prática: Achando um elemento do vetor

O programa PESQUISA.BAS (Figura 8-11) demonstra como pesquisar um vetor, procurando um elemento específico. O programa pede ao usuário os dados do vendedor e depois pergunta que dado o usuário gostaria de examinar. Um laço FOR percorre cada

elemento do vetor *grupoVend$* até encontrar uma correspondência ou até que todas as entradas tenham sido examinadas:

- Se for achada uma correspondência, os elementos dos vetores *motosVend%* e *totVendas!* serão mostrados, e depois o laço termina com um comando EXIT FOR.

O Comando EXIT FOR

O comando EXIT FOR lhe permite sair de um laço FOR mais cedo — antes que o laço termine por conta própria; ele é útil quando você deseja repetir um número específico de vezes, a menos que uma certa condição seja atendida. O programa a seguir pede os nomes até que o usuário entre com 10 nomes ou até que entre com *FIM*, o que acontecer primeiro:

```
FOR i% = 1 TO 10
    INPUT "Entre com um nome: ", nome$
    IF (nome$ = "FIM") THEN EXIT FOR
    PRINT nome$
NEXT i%
```

- Se nenhuma correspondência for achada, a mensagem *Nome não encontrado* será mostrada.

Digite o programa PESQUISA.BAS e execute-o.

```
' PESQUISA.BAS
' Este programa lê dados para três vetores e procura
'  um nome. O máximo de entradas é 50; menos poderá
'  ser entrado digitando "FIM" no nome do vendedor.

OPTION BASE 1           ' define base de vetores em 1
DIM grupoVend$(50)      ' dimensiona vetor de cadeia
DIM motosVend%(50)      ' dimensiona vetor inteiro
DIM totVendas!(50)      ' dimensiona vetor fracionário

CLS
PRINT "Siga os pedidos para entrar com dados de
vendas."
PRINT "Digite END para terminar."
PRINT

cont% = 1               ' inicializa contador do vetor
                        ' continua até nome="FIM"
WHILE (grupoVend$(conta%) <> "FIM")
  INPUT "Entre com nome do vendedor: "; grupoVend$(cont%)
```

FIGURA 8-11. (*continua*)
PESQUISA.BAS: Um programa para pesquisar um elemento específico em um vetor.

FIGURA 8-11. *continuação*

```
  IF (grupoVend$(cont%) <> "FIM") THEN
     INPUT "  Motos vendidas: ", motosVend%(cont%)
     INPUT "  Total de vendas: $", totVendas!(cont%)
     PRINT
     cont% = cont% + 1 ' incrementa contador do vetor
  END IF
WEND

PRINT              ' pede cadeia de pesquisa ao usuário
PRINT "Que nome deseja pesquisar? ", pesq$
PRINT

' inicializa gab$, um gabarito para PRINT USING
gab$ = "\              \ ###          $$####.##"

' compara cada elemento com a cadeia até achar uma
'  igualdade, depois mostra o registro e sai do laço;
'  mostra mensagem se não achar a cadeia de pesquisa.

FOR i% = 1 TO cont% - 1   ' até o último elemento
                          ' do vetor
  IF (grupVendas$(i%) = pesq$) THEN
     PRINT "Vendedor  Motos vendidas  Total de vendas"
     PRINT "- - - - - - - - - - - - - - - - - - - - -"
     PRINT
     PRINT USING gab$; grupoVend$(i%); motosVend%(i%);
totVendas!(i%)
     EXIT FOR             ' interrompe um laço FOR
  END IF
  IF (i% = cont% - 1) THEN PRINT "** Nome não
encontrado **"
NEXT i%
```

Você verá uma saída semelhante a esta:

```
Siga os pedidos para entrar com dados de vendas.
Digite FIM para terminar.

Entre com nome do vendedor: Jaques
  Motos vendidas: 2
  Total de vendas: $489.00

Entre com nome do vendedor: Roni
  Motos vendidas: 12
  Total de vendas: $2343.84

Entre com nome do vendedor: Mariana
  Motos vendidas: 7
  Total de vendas: $1256.36
```

```
Entre com nome do vendedor: FIM
Que nome deseja pesquisar? Roni
Vendedor        Motos vendidas          Total de vendas
- - - - - - - - - - - - - - - - - - - - - - - - - - -
Roni            12                      $2343.84
```

Prática: Encontrando o maior número em um vetor

O programa MAXVENDA.BAS (Figura 8-12) demonstra como o maior elemento em um vetor pode ser extraído pelo uso de um laço FOR. O programa pede ao usuário os dados de vendedor e depois examina cada elemento do vetor *totVendas!*. O maior valor de vendas é armazenado na variável *maior!* e comparado com cada elemento de *totVendas!*. Se algum elemento for maior do que *maior!*, esse elemento do vetor torna-se o novo *maior!* (A variável *m%* armazena o índice de vetor associado a *maior!*.) Depois de examinar todos os elementos do vetor, o programa mostra o maior valor de vendas e os elementos relacionados nos vetores *grupoVend$* e *motosVend%*.

Digite o programa MAXVENDA.BAS e execute-o.

```
' MAXVENDA.BAS
' O programa lê e mostra dados de vendas em 3 vetores
'  dados do vendedor com o maior total. O número máximo
'  de nomes a entrar é 50; menos pode ser entrado
'  digitando-se "FIM" para o nome do vendedor.

OPTION BASE 1           ' define base de vetores em 1

DIM grupoVend$(50)      ' dimensiona vetor de cadeia
DIM motosVend%(50)      ' dimensiona vetor inteiro
DIM totVendas!(50)      ' dimensiona vetor fracionário

CLS
PRINT "Siga os pedidos para entrar com dados de
vendas."
PRINT "Digite END para terminar."
PRINT

cont% = 1         ' inicializa contador do vetor
' continua até nome="FIM"
WHILE (grupoVend$(conta%) <> "FIM")
  INPUT "Entre com nome do vendedor: ";
grupoVend$(cont%)
```

FIGURA 8-12.
MAXVENDA.BAS: Um programa demonstrando a extração do maior elemento em um vetor.

FIGURA 8-12. *continuação*

```
  IF (grupoVend$(cont%) <> "FIM") THEN
    INPUT "  Motos vendidas: ", motosVend%(cont%)
    INPUT "  Total de vendas: $", totVendas!(cont%)
    PRINT
    cont% = cont% + 1 ' incrementa contador do vetor
  END IF
WEND

maior! = totVendas!(1)' por enquanto, elemento 1
                      ' é o maior
m% = 1           ' salva índice do vetor

' compara outros elementos procurando algo maior -
'   se achar, atribui a maior! e salva o índice em
'   m%; se houver um empate, retorna o primeiro
'   elemento achado

FOR i% = 2 TO cont% - 1
  IF (totVendas!(i%) > maior!) THEN
    maior! = totVendas(i)    ' salva novo valor maior
    m% = i%                  ' salva índice do vetor
  END IF
NEXT i%

' inicializa gab$, um gabarito para PRINT USING
  gab$ = "\              \ ###          $$####.##"

PRINT
PRINT "** "; grupoVend$(m%); " tem maiores vendas **"
PRINT
PRINT "Vendedor     Motos vendidas     Total de vendas"
PRINT "- - - - - - - - - - - - - - - - - - - - - - - -"
PRINT
PRINT USING gab$; grupoVend$(m%); motosVend%(m%);
totVendas!(m%)
```

Você verá uma saída semelhante a esta:

```
Siga os pedidos para entrar com dados de vendas.
Digite FIM para terminar.

Entre com nome do vendedor: Marcos
  Motos vendidas: 6
  Total de vendas: $1350.12

Entre com nome do vendedor: Eric
  Motos vendidas: 5
  Total de vendas: $1578.55
```

```
Entre com nome do vendedor: Nanci
  Motos vendidas: 5
  Total de vendas: $1356.03

Entre com nome do vendedor: Mariana
  Motos vendidas: 7
  Total de vendas: $1256.36

Entre com nome do vendedor: FIM

** Eric tem maiores vendas **

Vendedor        Motos vendidas          Total de vendas
- - - - - - - - - - - - - - - - - - - - - - - - - - - -

Eric            5                       $1578.55
```

Vetores Bidimensionais

O QBasic também lhe permite declarar vetores com duas dimensões. Um vetor bidimensional representa uma tabela de valores com linhas e colunas, como um placar, um livro contábil ou um tabuleiro de jogo. Nesta seção discutiremos como declarar e usar um vetor bidimensional num programa em QBasic.

Vamos começar com um exemplo. Samuel, um distribuidor de refrigerantes da região, deseja acompanhar as vendas das principais marcas nos últimos 12 meses. Ele também deseja colocar a informação em uma tabela que mostre os nomes das marcas como títulos das linhas e os meses do ano como títulos das colunas, conforme mostramos na Figura 8-13.

	Jan	*Fev*	*Mar*	*Abr*	*Mai*	*Jun*	*Jul*	*Ago*	*Set*	*Out*	*Nov*	*Dez*
Coca-cola	64	63	58	45	36	32	41	39	50	67	69	103
Fanta	35	41	60	57	38	29	25	19	26	37	43	36
Guaraná	15	9	12	21	24	32	46	42	37	22	18	13
Pepsi	30	30	30	35	42	44	49	48	38	35	31	30

FIGURA 8-13.
Uma tabela de valores mostrando vendas de refrigerantes nos últimos 12 meses.

A informação tabular é perfeitamente adequada a um vetor bidimensional. Neste exemplo, uma dimensão do vetor corresponde às linhas de marca e a outra dimensão corresponde às colunas de mês. Os itens em um vetor bidimensional são identificadas com subscritos de linha *e* coluna. A Figura 8-14 mostra como os subscritos de linha e coluna seriam atribuídos a cada dimensão se a base do vetor fosse

Uso de Vetores com Procedures

No Capítulo 7 você aprendeu como as variáveis podem ser declaradas de modo local ou global e como argumentos variáveis são passados a subprogramas e funções. Basicamente as mesmas regras se aplicam a vetores usados em programa que contêm procedures:

- Por default, um vetor é local ao programa principal ou à procedure em que foi declarado.
- Um vetor é declarado como global por meio do comando COMMON SHARED no programa principal; por exemplo:

```
DIM lojas$(20)
COMMON SHARED lojas$( )
```

- Um único elemento do vetor pode ser passado a uma procedure; por exemplo:

```
PegaEntrada lojas$(12)
```

- Um vetor inteiro pode ser passado a uma procedure quando nenhum subscrito é indicado; por exemplo:

```
ImprimeTudo lojas$( )
```

Para manter o código organizado e eficiente, à medida que os seus programas se tornarem maiores você desejará usar vetores junto com subprogramas e funções.

definida em 1. Um vetor unidimensional precisa de um subscrito para identificar um elemento do vetor, e um vetor bidimensional precisa de dois subscritos para identificar um elemento do vetor. Na Figura 8-14, por exemplo, o número de caixas de Coca-cola vendidas em maio seria identificado pela linha 1, coluna 5.

Subscritos de linha

Subscritos de coluna

		1	2	3	4	5	6	7	8	9	10	11	12
1	**Coca-cola**	64	63	58	45	36	32	41	39	50	67	69	103
2	**Fanta**	35	41	60	57	38	29	25	19	26	37	43	36
3	**Guaraná**	15	9	12	21	24	32	46	42	37	22	18	13
4	**Pepsi**	30	30	30	35	42	44	49	48	38	35	31	30

FIGURA 8-14.
Atribuindo subscritos de vetor bidimensional à tabela de vendas de refrigerantes.

Declaração de um vetor bidimensional

Para dimensionar um vetor bidimensional, use o comando DIM da seguinte maneira:

```
DIM nomeVetor(linhas, colunas)
```

nomeVetor é o nome do vetor, *linhas* é o número de linhas no vetor (a primeira dimensão) e *colunas* é o número de colunas no vetor (a segunda dimensão). *linhas* e *colunas* devem ser inteiros e podem ser expressos como números, constantes ou variáveis. O caracter final de *nomeVetor* deve ser o caracter de declaração de tipo que identifica o tipo de dado do vetor: *$* para uma cadeia, *%* para um inteiro, *&* para um inteiro longo, *!* para um vetor de ponto flutuante com precisão simples ou *#* para um vetor de ponto flutuante com precisão dupla. Assim como em um vetor unidimensional, cada elemento em um vetor bidimensional deve ser do mesmo tipo dos outros.

NOTA: O primeiro elemento em cada dimensão do vetor é o de número 0, a não ser que você use o comando OPTION BASE 1 antes de dimensionar o vetor.

Os comandos a seguir dimensionam um vetor chamado *vendasRefrig%* com 4 linhas e 12 colunas:

```
OPTION BASE 1
DIM vendasRefrig%(4, 12)
```

Esses comandos pedem ao usuário para entrar com as dimensões do vetor bidimensional dinâmico chamado *vendasRefrig%*:

```
OPTION BASE 1
INPUT "Entre com o número de marcas de refrigerantes: ", marcas%
INPUT "Entre com o número de meses a acompanhar: ", meses%
DIM vendasRefrig%(marcas%, meses%)
```

Prática: Montando uma tabela de valores

O programa VENDAS.BAS (Figura 8-15) demonstra como o vetor bidimensional *vendasRefrig%* é preenchido e apresentado na tela. VENDAS pede ao usuário o número de linhas e colunas na tabela de vendas, armazena esses números nas variáveis inteiras *marcas%* e *meses%* e os utiliza para dimensionar o vetor *vendasRefrig%*. VENDAS usa dois laços FOR endentados para preencher e mostrar o vetor, usando comandos DATA e READ para armazenar e recuperar os meses do ano. Use este programa como guia para o preenchimento e impressão de qualquer vetor bidimensional dinâmico.

Digite o programa VENDAS.BAS.

```
' VENDAS.BAS
' Este programa acompanha as vendas de refrigerantes
'  por um determinado número de meses.

CLS

PRINT "** Programa de acompanhamento de vendas **"
PRINT
DO
  INPUT "Quantas marcas de refrigerantes você vende?
", marcas%
LOOP WHILE (marcas% < 1)

DO
  INPUT "Quantos meses você deseja registrar (1-12)?
", meses%
LOOP WHILE (meses% < 1) OR (meses% > 12)
PRINT

OPTION BASE 1        ' define primeiro elemento como 1
' dimensiona vetores de vendas e nomes de marca
DIM vendasRefrig%(marcas%, meses%)
DIM nomesMarcas$(marcas%)


' apanha nomes de marcas de refrigerantes

PRINT "Entre com as"; marcas%; "marcas de
refrigerantes"
PRINT
FOR i% = 1 TO marcas%
  INPUT "Nome da marca: ", nomesMarcas$(i%)
NEXT i%

' apanha vendas de refrigerantes para cada mês

PRINT
PRINT "Entre com o número de caixas de refrigerantes
vendidas"
PRINT

FOR i% = 1 TO marcas%      ' para cada marca...
  PRINT "** "; nomesMarcas$(i%); " **"
```

FIGURA 8-15. (*continua*)
VENDAS.BAS: Um programa que usa um vetor bidimensional dinâmico para acompanhar as vendas de refrigerantes.

FIGURA 8-15. *continuação*

```
  PRINT                    ' imprime nome da marca
  FOR j% = 1 TO meses%     ' para cada mês...
    READ mês$              '  lê nome do mês em DATA
    ' imprime nome e pede entrada da quantidade,
    ' armazenando-a no vetor vendasRefrig%
    PRINT "  "; mês;
    INPUT ": ", vendasRefrig%(i%, j%)
  NEXT j%
  PRINT
  RESTORE      ' volta a lista DATA ao primeiro mês
NEXT i%

' imprime tabela de vendas de refrigerantes

CLS

PRINT "Vendas em caixas durante"; meses%; "meses"
PRINT
PRINT "- - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -"
PRINT "Marca/Mês    ";

FOR i% = 1 TO meses%
  PRINT " ";
  READ mês$        ' lê nome do mês na lista DATA
  PRINT mês$;      ' imprime nomes no alto da tabela
NEXT i%

PRINT
PRINT "- - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -"

' gabaritos para PRINT USING

gabNome$ = "\           \"' para marcas até 12 letras
gabVendas$ = " ###"        ' para caixas até 3 dígitos

FOR i% = 1 TO marcas%      ' preenche a tabela
  PRINT USING gabNome$; nomesMarcas$(i%);
  FOR j% = 1 TO meses%
    PRINT USING gabVendas$; vendasRefrig%(i%, j%);
  NEXT j%
```

(continua)

FIGURA 8-15. *continuação*

```
  PRINT
NEXT i%


PRINT "- - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - -"

' dados para comandos READ acima

DATA Jan, Fev, Mar, Abr, Mai, Jun, Jul, Ago, Set,
Out, Nov, Dez
```

Quando você rodar VENDAS.BAS, as entradas lhe serão pedidas:

```
** Programa de acompanhamento de vendas **

Quantas marcas de refrigerantes você vende? 4
Quantos meses você deseja registrar (1-12)? 12

Entre com as 4 marcas de refrigerantes

Nome da marca: Coca-cola
Nome da marca: Fanta
Nome da marca: Guaraná
Nome da marca: Pepsi

Entre com o número de caixas de refrigerantes vendidas

** Coca-cola **

 Jan: 64
 Fev: 63
 Mar: 58
 ...
```

Depois que os dados para cada marca e mês forem entrados, você verá uma saída semelhante à seguinte:

Vendas em caixas durante 12 meses

Marca/Mês	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Coca-cola	64	63	58	45	36	32	41	39	50	67	69	103
Fanta	35	41	60	57	38	29	25	19	26	37	43	36
Guaraná	15	9	12	21	24	32	46	42	37	22	18	13
Pepsi	30	30	30	35	42	44	49	48	38	35	31	30

NOTA: Para imprimir uma cópia da tabela final na sua impressora, assegure-se de que esteja ligada e depois segure a tecla Shift e pressione a tecla PrtSc. A combinação de teclas Shift-PrtSc envia todo o texto na tela para a sua impressora.

Solução de Problemas com Vetores

Embora os vetores sejam uma grande dádiva para os seus programas, eles também aumentam o potencial para erros. Vejamos alguns erros de programação típicos associados com vetores e com o processamento de grandes quantidades de dados e depois alguns meios de evitar os erros.

Erro 1: Não usar um inteiro para definir um subscrito

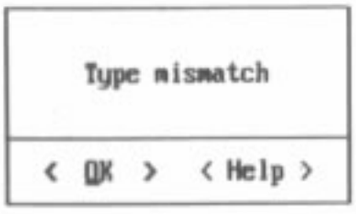
O número de elementos e dimensões em um vetor é determinado pelos subscritos inteiros no comando DIM. *Assegure-se de que os subscritos do vetor sejam valores inteiros ou variáveis inteiras.* A declaração de vetor a seguir, por exemplo, é válida:

```
DIM notas%(estudantes%, 15)
```

Os subscritos *estudantes%* e *15* são valores inteiros. A declaração a seguir não é válida, pois *itens$* e *quantidade$* são valores de cadeia:

```
DIM tabCustos!(itens$, quantidade$)
```

Quando você tentar dimensionar um vetor com subscritos inválidos, receberá a seguinte mensagem de erro:



Se indicar um número de ponto flutuante como subscrito, o QBasic arredondará o valor para o valor inteiro mais próximo e depois dimensionará o vetor. Por exemplo, o comando a seguir consegue dimensionar um vetor com 6 elementos (de 0 a 5):

```
DIM valores%(4.6)
```

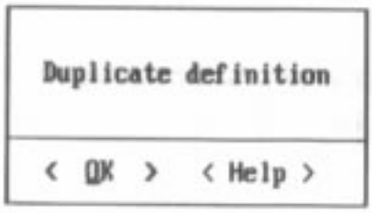
NOTA: O uso de um valor de ponto flutuante como subscrito de um vetor ficará confuso para uma pessoa lendo o programa. Acostume-se a usar subscritos inteiros.

Erro 2: Usar um comando DIM em um laço

Assegure-se de que os comandos DIM estejam fora de quaisquer laços no seu programa. O trecho de programa a seguir demonstra o uso incorreto de um comando DIM para declarar um vetor dinâmico:

```
OPTION BASE 1
tamanho% = 5
CLS

FOR i% = 1 TO tamanho%
    DIM nome$(tamanho%)
    INPUT "Entre com um nome: ", nome$(i%)
NEXT i%

PRINT
FOR i% = 1 TO tamanho%
    PRINT nome$(i%)
NEXT i%
```

Se você executar o programa, receberá a seguinte mensagem de erro quando o comando DIM for executado mais uma vez:



Para evitar este erro, dimensione o vetor *nome$* antes do laço, da seguinte forma:

```
OPTION BASE 1
tamanho% = 5
DIM nome$(tamanho%)
CLS

FOR i% = 1 TO tamanho%
    INPUT "Entre com um nome: ", nome$(i%)
NEXT i%

PRINT
FOR i% = 1 TO tamanho%
    PRINT nome$(i%)
NEXT i%
```

Erro 3: Confundir índice do vetor com valor do vetor

É fácil confundir o valor usado para *índice* de um elemento do vetor com o valor *no* elemento do vetor. Lembre-se de que o índice do vetor está sempre entre parênteses e segue o nome do vetor, descrevendo o local do valor (Figura 8-16).

```
DIM colega!(estilos%, diasAlug%)
...
colega!(5, 3) = 45.85
```

Índice do vetor — Valor do vetor

FIGURA 8-16.
Índice do vetor e valor do vetor.

O comando a seguir tenta atribuir o valor de cadeia "Hotel Orcas" ao quinto elemento no vetor *locaisVagos$*, mas falha devido ao posicionamento errado do índice e do valor do vetor:

```
locaisVagos$("Hotel Orcas") = 5
```

A atribuição correta do vetor seria:

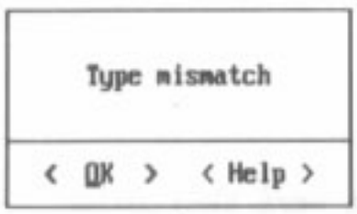
```
locaisVagos$(5) = "Hotel Orcas"
```

Erro 4: Falta de correspondência nos tipos do vetor

Cada elemento em um vetor deve ser do mesmo tipo de dados. Se você tentar atribuir uma variável ou valor a um vetor que não combine com o tipo do vetor, especificado no comando DIM, verá uma mensagem de erro. Por exemplo, o seguinte comando de atribuição gera uma mensagem de erro porque *custo!* é um vetor de ponto flutuante com precisão simples e o valor *"57.36"* é um tipo de dado de cadeia. (Uma atribuição correta seria *57.36* sem aspas.)

```
custo!(i%) = "57.36"
```

Este tipo de erro de atribuição gera a seguinte mensagem:



Uma discordância de tipo semelhante ocorre enquanto o programa está rodando e o usuário entra com um valor que não combina com o elemento do vetor que recebe esse valor. O laço a seguir, por exemplo, pede ao usuário que entre com vários inteiros.

```
DIM valores%(5)
FOR i% = 1 TO 5
    INPUT "Entre com um número: ", valores%(i%)
NEXT i%
```

Se o usuário entrar com um valor de cadeia, aparecerá a seguinte mensagem:

```
Entre com um número: dez

Redo from start
Entre com um número:
```

Redo from start [refazer do início] é o modo como o QBasic pede ao usuário dados de tipo correto. Em geral, esta mensagem será suficiente para que o usuário acerte o caminho. No entanto, como já dissemos, a melhor forma de evitar problemas de entrada é explicar exatamente o que você deseja na mensagem de pedido.

Erro 5: Erros de valor fora da faixa permitida

A tentativa de referenciar um elemento que não existe em um vetor gera uma mensagem de erro *out-of-range* [fora da faixa] quando o programa é executado. Por exemplo, o QBasic gera uma mensagem de erro quando o terceiro comando a seguir é executado, pois *notasTestes%* só pode conter 25 elementos, e é feita uma referência ao trigésimo elemento.

```
OPTION BASE 1
DIM notasTestes%(25)
notasTestes(30) = 92
```

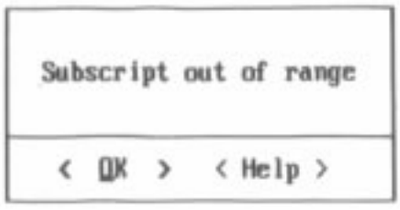
O erro "fora da faixa" é mais comum em uma estrutura de laço, como vemos no trecho de programa a seguir:

```
OPTION BASE 1
DIM cores$(4)
conta% = 0
CLS

DO
    conta% = conta% + 1
    INPUT "Entre com nome da cor: ", cores$(conta%)
LOOP UNTIL (cores$(conta%) = "FIM")
```

Quando você executar este trecho de programa, verá algo assim:

```
Entre com nome da cor: amarelo
Entre com nome da cor: vermelho
Entre com nome da cor: azul
Entre com nome da cor: verde
Entre com nome da cor: branco
```

Após a entrada da quinta cor (preenchendo o último elemento do vetor), o contador do laço *conta%* é incrementado e o comando INPUT (agora referenciando um elemento além do final do vetor) dispara a mensagem de erro *Subscript out of range* [subscrito fora da faixa].



A solução para problemas "fora da faixa" é determinar os limites superior e inferior do vetor e não ultrapassá-los. O QBasic oferece as funções UBOUND e LBOUND para retornar os limites de qualquer vetor no seu programa, de modo que você possa enfrentar este problema em potencial.

As funções UBOUND e LBOUND

As funções UBOUND e LBOUND retornam valores inteiros correspondentes aos limites superior e inferior de um vetor [respectivamente]. Veja em seguida a sintaxe das funções UBOUND e LBOUND:

UBOUND(*nomeVetor*, *dimensão*)
LBOUND(*nomeVetor*, *dimensão*)

nomeVetor é o nome do vetor para o qual você deseja determinar os limites, e *dimensão* é a dimensão que você quer checar (opcional se você estiver avaliando um vetor unidimensional). As duas funções retornam valores inteiros que podem ser atribuídos a variáveis ou usados em expressões.

O trecho de programa a seguir usa as funções UBOUND e LBOUND para checar um vetor unidimensional chamado *jogadores$*.

```
OPTION BASE 1
DIM jogadores$(9)
PRINT "Limite superior é"; UBOUND(jogadores$)
PRINT "Limite inferior é"; LBOUND(jogadores$)
```

Ao executar este trecho, verá a seguinte saída:

```
Limite superior é 9
Limite inferior é 1
```

Ao checar os limites de um vetor bidimensional, você deverá fornecer um número de dimensão: 1 para a primeira dimensão ou 2 para a segunda. O trecho de programa a seguir usa UBOUND e LBOUND para verificar os limites das duas dimensões de um vetor bidimensional chamado *vendasJan%*:

```
OPTION BASE 1
DIM vendasJan%(3, 4)
PRINT "Limite superior da primeira dimensão é"; UBOUND(vendasJan%, 1)
PRINT "Limite inferior da primeira dimensão é"; LBOUND(vendasJan%, 1)
PRINT "Limite superior da segunda dimensão é"; UBOUND(vendasJan%, 2)
PRINT "Limite inferior da segunda dimensão é"; LBOUND(vendasJan%, 2)
```

Após executar este trecho, verá a seguinte saída:

```
Limite superior da primeira dimensão é 3
Limite inferior da primeira dimensão é 1
Limite superior da segunda dimensão é 4
Limite inferior da segunda dimensão é 1
```

Prática: Evitando a mensagem Subscript out of range

O programa LIMITES.BAS (Figura 8-17) demonstra como usar UBOUND e LBOUND para checar os limites de um vetor em um laço DO, evitando, assim, erros *out-of-range* [fora da faixa]. O programa dimensiona um vetor de cadeia unidimensional chamado *festa$*, que contém idéias para brincadeiras em festas. Em seguida, LIMITES.BAS preenche o vetor e o imprime.

NOTA: O laço DO no programa LIMITES utiliza o comando EXIT DO, que sai do laço se o usuário entrar com FIM antes que todos os elementos sejam preenchidos. Use EXIT DO nos seus programas quando precisar terminar um laço DO antes que a condição de WHILE ou UNTIL seja atendida.

Digite o programa LIMITES.BAS e execute-o.

```
' LIMITES.BAS
' Este programa carrega dados para um vetor até que
'  seja digitado FIM ou até que sejam excedidos os
'  limites do vetor.

OPTION BASE 1          ' define base do vetor em 1
DIM festa$(5)          ' vetor com 5 elementos

conta% = 1             ' contador do laço

CLS                    ' mostra pedido e nota sobre FIM
PRINT "Entre com idéias para festa; digite FIM para
parar"
PRINT
' laço enquanto conta% estiver nos limites do vetor
DO WHILE (conta% >= LBOUND(festa$)) AND (conta% <=
UBOUND(festa$))
  INPUT "Brincadeira: ", festa$(conta%)  ' lê entrada
  ' se o usuário digitar FIM, sai do laço
  IF (festa$(conta%) = "FIM") THEN EXIT DO
  conta% = conta% + 1
```

(*continua*)

FIGURA 8-17.
LIMITES.BAS: Um programa que usa LBOUND e UBOUND para evitar referências a um elemento fora da faixa do vetor.

FIGURA 8-17. *continuação*

```
LOOP

PRINT
PRINT "Você entrou com as seguintes brincadeiras:"
PRINT

FOR i% = 1 TO conta% - 1 ' imprime conteúdo do vetor
  PRINT festa$(i%)       ' conta% -1 possui o último
                         '  valor entrado
NEXT i%
```

Você receberá uma saída semelhante à seguinte:

```
Entre com idéias para festa; digite FIM para parar

Brincadeira: Cabra cega
Brincadeira: Bingo
Brincadeira: Colocar o rabo no burro
Brincadeira: Pique
Brincadeira: Roda

Você entrou com as seguintes brincadeiras:

Cabra cega
Bingo
Colocar o rabo no burro
Pique
Roda
```

RESUMO

Neste capítulo você trabalhou com as importantes estruturas, funções, comandos e técnicas que o QBasic oferece para trabalhar com grandes quantidades de dados em um programa:

- Uso de DATA, READ e RESTORE para atribuir dados armazenados em um programa a variáveis
- Criação e uso de um vetor unidimensional
- Pesquisa de elementos em um vetor
- Formatação de dados colunares com PRINT USING
- Criação e uso de um vetor bidimensional
- Solução para erros de programação comuns e relacionados com vetores

No próximo capítulo você aprenderá as diversas funções do QBasic e as técnicas elaboradas para o trabalho com cadeias.

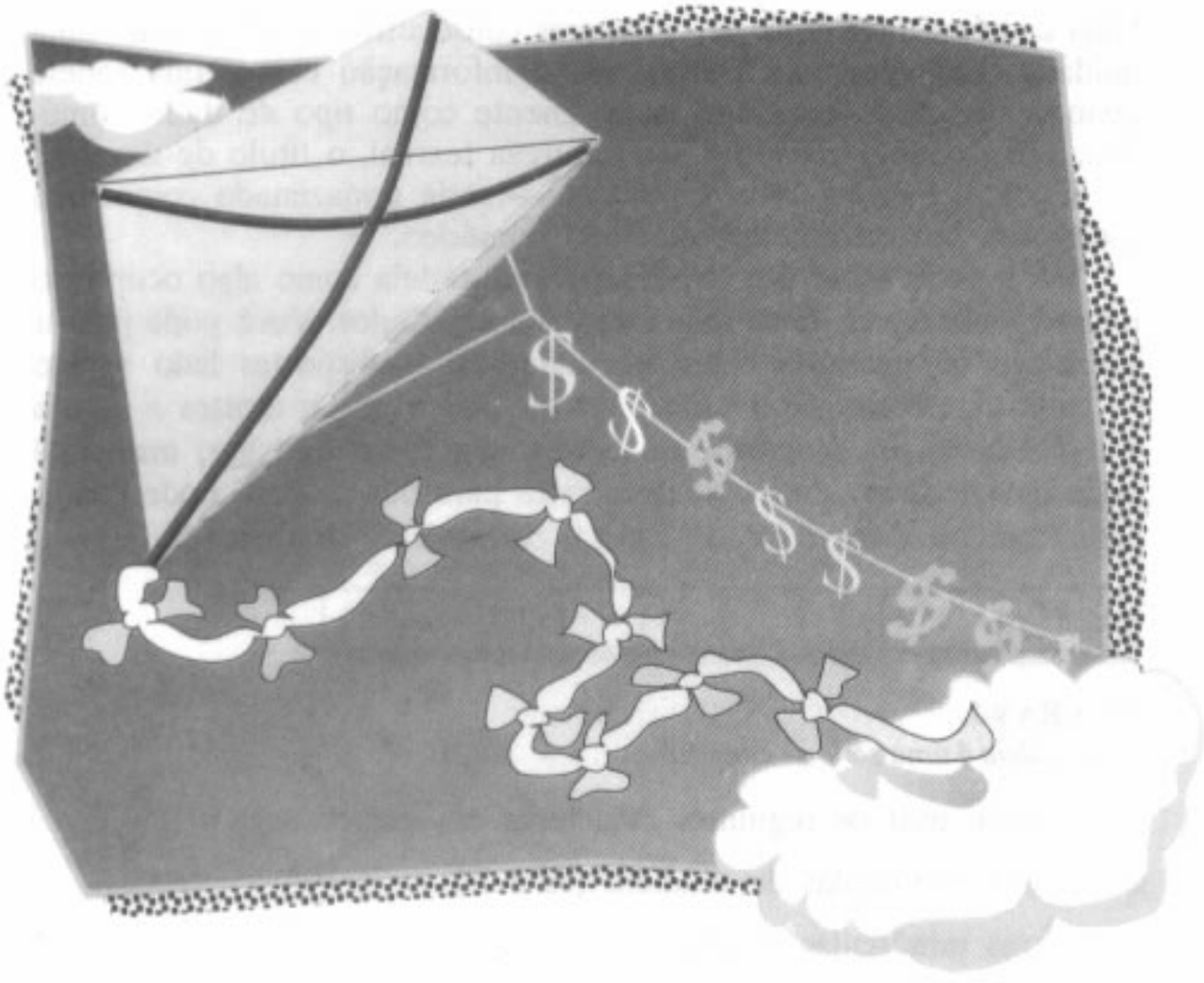
PERGUNTAS E EXERCÍCIOS

1. O que vem primeiro em um programa, um comando DATA ou um comando READ?
2. Que tipo de informação é mais adequada para o método DATA-READ-RESTORE de trabalho com dados?
3. Escreva um programa que mostre cada um dos valores no comando DATA a seguir dentro de um laço FOR:

   DATA Beto, Walter, Lúcia, Wander, Gustavo, Edu, Laura
4. Falso ou Verdadeiro: Um vetor pode armazenar mais de um tipo de dados.
5. Escreva um comando que reserve memória para um vetor de 100 valores de ponto flutuante com precisão simples.
6. O que é um marcador de fim-de-dados?
7. Escreva um programa que declare, preencha e depois imprima um vetor unidimensional dinâmico contendo os principais personagens da sua novela ou filme de televisão preferido?
8. O que é um erro do tipo "fora da faixa"?
9. Escreva um programa que use um vetor bidimensional para acompanhar o marcador de um jogo de tênis. O seu programa deverá inicializar o vetor, pedir ao usuário os nomes dos jogadores e suas nacionalidades, indicar os escores de cada *set*, determinar o escore final e o vencedor do jogo, mostrando todos os resultados.

CAPÍTULO 9

Uso de Cadeias



Os Capítulos 3 e 4 apresentaram o tipo de dado de cadeia e lhe permitiram praticar a criação e apresentação de cadeias na tela. Este capítulo continua essa discussão e introduz muitas das funções disponíveis no QBasic para o trabalho com cadeias. Iremos descrever

- A atribuição da entrada do usuário a uma cadeia
- A combinação de cadeias
- A seleção de caracteres de uma cadeia
- A comparação de cadeias
- A ordenação de cadeias

Combinados com as habilidades desenvolvidas nos capítulos anteriores, essas funções poderão ajudá-lo a realizar uma grande variedade de serviços.

CADEIAS: UMA VISÃO GERAL

Uma cadeia é uma série de caracteres consecutivos usados como uma unidade. Em geral, você armazena a informação como uma cadeia quando não pode armazená-la facilmente como tipo de dado numérico. Por exemplo, devido a sua natureza textual, o título de um livro — digamos, *Fernão Capelo Gaivota* — seria armazenado como uma cadeia em vez de um tipo de dado numérico.

Você pode achar útil pensar em uma cadeia como algo ocupando uma série de locais da memória de um computador. Você pode pensar em locais da memória como pequenas caixas dispostas lado a lado — cada uma contendo um caracter da cadeia, como mostra a Figura 9-1. Os locais da memória, ou caixas, são fixos no lugar, mas você pode mover os caracteres de uma caixa para outra. Você pode incluir caracteres ou removê-los da cadeia, conforme a necessidade.

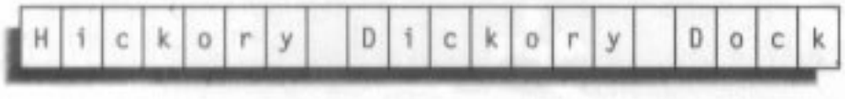


FIGURA 9-1.
Uma cadeia é uma série de caracteres.

Você pode usar os seguintes caracteres em uma cadeia:

- Letras maiúsculas do alfabeto (A a Z)
- Letras minúsculas do alfabeto (a a z)
- Dígitos numéricos (0 a 9)
- Sinais de pontuação (. , ; : ' ' ? !)
- Símbolos matemáticos (# % ( ) - + = \ / < >)

- Símbolos diversos (~ @ $ ^ & * _ | { } [ ])
- Caracteres no conjunto estendido da IBM

DOIS TIPOS DE CADEIAS

O QBasic aceita dois tipos de cadeias nos seus programas:

- *Constantes de cadeia* são declaradas dentro do seu programa. Como o nome indica, os caracteres em uma constante de cadeia nunca mudam.
- *Variáveis de cadeia* também são declaradas dentro do seu programa. Os caracteres em uma variável de cadeia podem mudar a qualquer hora.

Vamos examinar por que e como usar esses tipos de cadeias no QBasic.

Constantes de Cadeia

As constantes de cadeia são cadeias que não mudam enquanto um programa está sendo executado. O QBasic separa as constantes de cadeia em dois grupos: *constantes de cadeia literais* e *constantes de cadeia simbólicas*.

Constantes de cadeia literais

Uma cadeia literal é uma série de caracteres consecutivos cercados por aspas. Você geralmente atribui cadeias literais a uma variável ou então as utiliza em um comando ou uma função. Por exemplo, os comandos a seguir contêm cadeias literais.

```
ânimo$ = "Vai em frente!"

PRINT "Rua do Terror, 1313"

ideal$ = CadeiaCentro$("Hoje soltarei pipa")

dataNasc$ = "19/11/63"
```

As cadeias literais também são conhecidas como *valores de cadeia*.

Constantes de cadeia simbólicas

Uma constante de cadeia simbólica é um nome atribuído a uma cadeia literal:

```
CONST DI$ = "Sua Alteza Real, Diana, Princesa de Gales"
```

Constante — Cadeia literal

Após ter atribuído uma constante, você poderá usá-la em qualquer lugar onde, de outra forma, teria que usar a cadeia literal (assim como você pode chamar alguém pelo seu apelido no lugar do nome completo).

Como vimos no Capítulo 4, você atribui uma constante em um comando CONST, em geral no início de um programa. Neste livro, as constantes são escritas com letras maiúsculas para que sejam diferenciadas das variáveis:

```
CONST DI$ = "Sua Alteza Real, Diana, Princesa de Gales"

PRINT "A Esposa de Charles agora se chama "; DI$
```

Após declarar uma constante, você não pode mudar o seu valor. Por exemplo, um programa contendo os dois comandos a seguir geraria uma mensagem de erro *Duplicate definition* [definição duplicada] quando o programa fosse rodado, pois o valor de uma constante não pode mudar durante a execução de um programa:

```
CONST DI$ = "Sua Alteza Real, Diana, Princesa de Gales"
DI$ = "Lady Diana Spencer"
```

Esta é a diferença entre uma constante e uma variável: o valor de uma variável pode mudar durante a execução de um programa; o valor de uma constante nunca pode mudar.

Prática: Usando constantes de cadeia

As constantes são úteis sobretudo quando você possui padrões repetitivos em um programa. O programa GURI.BAS (Figura 9-2) declara a constante *GAROTO$*, atribuindo-a a literal *O pequeno garoto.*

Digite o programa GAROTO.BAS e execute-o.

Você verá a seguinte saída:

```
Este pequeno garoto foi ao mercado
Este pequeno garoto ficou em casa
Este pequeno garoto comeu bife
Este pequeno garoto não comeu nada
Este pequeno garoto chorou o tempo todo no caminho
```

Variáveis de Cadeia

Diferente das constantes de cadeia, as variáveis de cadeia podem mudar a qualquer hora durante a execução de um programa. O QBasic aceita dois tipos de variáveis de cadeia: *cadeias de tamanho variável* e *cadeias de tamanho fixo.*

```
' GAROTO.BAS
' Este programa demonstra o uso de constantes de
' cadeia.

CLS

CONST GAROTO$ = "Este pequeno garoto "

PRINT GAROTO$; "foi ao mercado"
PRINT GAROTO$; "ficou em casa"
PRINT GAROTO$; "comeu bife"
PRINT GAROTO$; "não comeu nada"
PRINT GAROTO$; "chorou o tempo todo no caminho"
```

FIGURA 9-2.
GAROTO.BAS: Uma demonstração simples de constantes de cadeia literais e simbólicas.

Neste livro, usamos a convenção de iniciar as cadeias de tamanho fixo e variável com uma letra minúscula, diferenciando-as de subprogramas, funções, comandos e constantes.

Cadeias de tamanho variável

As cadeias de tamanho variável lhe permitem obter e armazenar informações do usuário de um programa. Uma cadeia de tamanho variável pode conter de 0 a 32.767 caracteres (embora a faixa típica seja de 0 a 80, devido à largura da tela), e seu tamanho pode aumentar ou diminuir durante a execução do programa. Você pode declarar uma cadeia de tamanho variável de três maneiras:

- Colocando o caracter de declaração de tipo (*$*) ao final do nome da variável. Por exemplo, o comando a seguir pede uma cadeia ao usuário, atribuindo-a à variável de cadeia *nome$*.

  ```
  INPUT "Entre com seu nome: ", nome$
  ```

  Ao examinar os exemplos neste livro, você verá muitas ocorrências em que os comandos INPUT usam cadeias de tamanho variável para atribuir nomes significativos em palavras ou frases em particular. Também usamos essas cadeias de tamanho variável como argumentos para comandos e funções, como PRINT.

- Usando o comando DEFSTR. Por exemplo, o comando a seguir instrui o QBasic a considerar como cadeia de tamanho variável qualquer variável começando com a letra *S* e que não tenha um caracter de declaração de tipo (*%*, *&*, *!*, *#* ou *$*):

  ```
  DEFSTR S
  ```

- Usando AS STRING com o comando DIM. Por exemplo, o comando a seguir declara a variável *nomeTodo* como uma cadeia de tamanho variável:

```
DIM nomeTodo AS STRING
```

Observe que *nomeTodo* não termina com um *$*. Sempre que você declara uma cadeia usando as palavras-chave AS STRING, não pode usar o caracter de declaração de tipo de cadeia.

Como vimos no capítulo anterior, você também pode usar o comando DIM para declarar um *vetor* de cadeias de tamanho variável. Por exemplo, o comando a seguir declara um vetor de 10 cadeias de tamanho variável (supondo que o comando OPTION BASE 1 apareça antes dele):

```
DIM nomes$(10)
```

Prática: Usando cadeias de tamanho variável

O programa FONE.BAS (Figura 9-3) usa comandos INPUT e um vetor de cadeia de tamanho variável chamado *contatos$* para armazenar uma lista de amigos e seus números de telefone. (Como os números de telefone normalmente contêm caracteres não numéricos, como traços e parênteses, é aconselhável armazená-los como cadeias.)

Digite o programa FONE.BAS e execute-o.

```
' FONE.BAS
' Este programa usa um vetor de cadeia com tamanho
'  variável para registrar nomes e números de
'  telefone.

OPTION BASE 1    ' limite inferior do vetor é 1

CLS

INPUT "Quantos nomes gostaria de incluir? ", nomes%
PRINT

DIM contatos$(nomes%, 2)     ' declara o vetor

FOR i% = 1 TO nomes%         ' lê nomes para o vetor
```

(*continua*)

FIGURA 9-3.
FONE.BAS: Uma demonstração de cadeias de tamanho variável.

FIGURA 9-3. *continuação*

```
  INPUT "Entre com o nome: ", contatos$(i%, 1)
  INPUT "Entre com o telefone: ", contatos$(i%, 2)
  PRINT
NEXT i%

PRINT "Você entrou com esta lista de contatos:"
PRINT

FOR i% = 1 TO nomes%   ' imprime conteúdo do vetor
  PRINT "Nome: "; contatos$(i%, 1), "Fone: ";
contatos$(i%, 2)
NEXT i%
```

Você receberá uma saída semelhante a esta:

```
Quantos nomes gostaria de incluir? 3

Entre com o nome: Marcio Vieira
Entre com o telefone: 722-2875

Entre com o nome: Bruna Glaser
Entre com o telefone: (0246) 68-1488

Entre com o nome: Roseni Teixeira
Entre com o telefone: 580-8236

Você entrou com esta lista de contatos:

Nome: Marcio Vieira       Fone: 722-2875
Nome: Bruna Glaser        Fone: (0246) 68-1480
Nome: Roseni Teixeira     Fone: 580-8236
```

Observe que você obtém o alinhamento da coluna de número de telefone usando uma vírgula no comando PRINT para formar duas colunas. Neste caso, avançar até a próxima zona de impressão é suficiente para alinhar os itens. No entanto, se os valores de cadeia forem muito diferentes em tamanho, seria preciso usar uma outra solução. Tente entrar com alguns nomes de tamanhos muito diferentes para ver o motivo.

Cadeias de tamanho fixo

As cadeias de tamanho fixo lhe permitem declarar uma variável de cadeia com um certo tamanho. Embora o conteúdo de uma cadeia de tamanho fixo possa mudar a qualquer momento, o tamanho permanece constante. Em geral, é bom usar cadeias de tamanho fixo

sempre que você souber com certeza o tamanho de uma cadeia de caracteres ou quando precisar alinhar grupos de cadeias.

Para declarar uma cadeia de tamanho fixo, você inclui um nome de variável de cadeia e o tamanho da cadeia no comando DIM, da seguinte forma:

```
DIM nomeCadeia AS STRING * n
```

nomeCadeia é o nome da cadeia de tamanho fixo, e *n* é o tamanho da cadeia. Por exemplo, o comando a seguir declara uma cadeia de tamanho fixo chamada *endereço*, contendo 25 caracteres:

```
DIM endereço AS STRING * 25
```

Você também pode dimensionar um *vetor* de cadeias de tamanho fixo incluindo o número de elementos de vetor e uma dimensão. Por exemplo, o comando a seguir declara um vetor de cadeias com tamanho fixo chamado *nomes*:

```
DIM nomes(25) AS STRING * 25
```

Cadeias de tamanho fixo são alinhadas à esquerda por default e preenchidas com espaços em branco à direita se o tamanho atribuído for menor do que o tamanho total da cadeia (Figura 9-4). Observe que não se pode usar o caracter de declaração de tipo (*$*) quando uma cadeia de tamanho fixo é declarada.

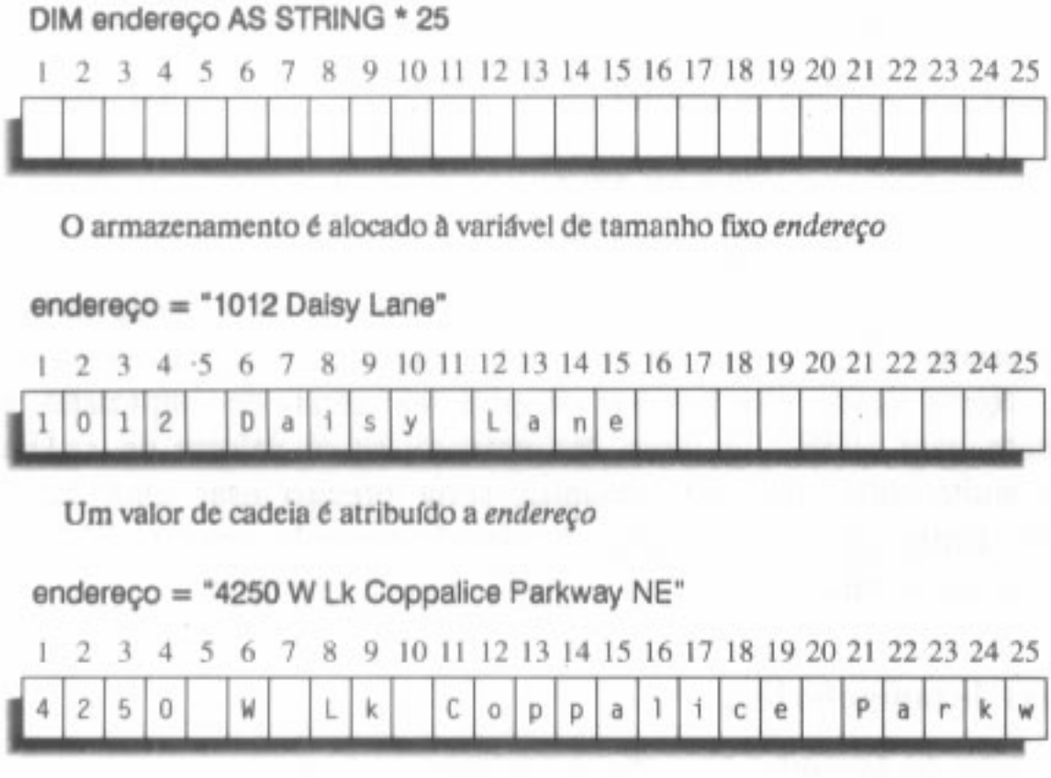


FIGURA 9-4.
Alinhamento de caracteres em uma cadeia de tamanho fixo.

Prática: Cadeias de tamanho fixo

O programa FONEFIXO.BAS (Figura 9-5) é uma outra versão do programa FONE.BAS, incluindo dois vetores unidimensionais de cadeia com tamanho fixo para acompanhar a lista de telefones. São necessários dois vetores separados porque o tamanho da cadeia em cada vetor é diferente. Se o tamanho da cadeia fosse igual, você poderia usar um vetor de cadeia bidimensional com tamanho fixo.

Digite o programa FONEFIXO.BAS e execute-o.

```
' FONEFIXO.BAS
' Este programa usa dois vetores de cadeia com tamanho
'  fixo para registrar nomes e números de telefone.

OPTION BASE 1    ' limite inferior do vetor é 1

CLS
                 ' apanha quantidade de nomes
INPUT "Quantos nomes gostaria de incluir? ", nomes%
PRINT

DIM nomes(nomes%) AS STRING * 18 ' nomes- 18 letras
DIM fones(nomes%) AS STRING * 14 ' fones- 14 dígitos

FOR i% = 1 TO nomes%      ' lê valores para vetores
  INPUT "Entre com o nome: ", nomes(i%)
  INPUT "Entre com o telefone: ", fones(i%)
  PRINT
NEXT i%

PRINT "Nome                Fone"
PRINT "- - - - - - - - - - - - - - - - - - - -"

FOR i% = 1 TO nomes%   ' mostra conteúdo dos vetores
  PRINT nomes(i%); "  "; fones(i%)
NEXT i%
```

FIGURA 9-5.
FONEFIXO.BAS: Uma demonstração de cadeias de tamanho fixo.

Você receberá uma saída semelhante à seguinte:

```
Quantos nomes gostaria de incluir? 3

Entre com o nome: Marcio Vieira
Entre com o telefone: 722-2875

Entre com o nome: Bruna Glaser
Entre com o telefone: (0246) 68-1480
```

```
Entre com o nome: Roseni Teixeira
Entre com o telefone: 580-8236

Nome              Fone
- - - - - - - - - - - - - - - - - -
Marcio Vieira     722-2875
Bruna Glaser      (0246) 68-1480
Roseni Teixeira   580-8236
```

Observe que o alinhamento da coluna de número de telefone agora é previsível. Se um dos nomes ou números de telefone fossem maiores do que o número de caracteres alocados, eles seriam truncados.

Combinando Cadeias

Uma das coisas mais simples que você pode fazer com cadeias é combiná-las para formar cadeias maiores. Este processo é chamado *concatenação*. Você concatena as cadeias principalmente para preparar um texto para saída na tela ou na impressora ou para preparar as cadeias para armazenamento em uma estrutura de dados ou em um arquivo. Uma cadeia concatenada geralmente deverá conter 80 caracteres ou menos. Esta limitação evita que a cadeia retorne (passando para a linha seguinte).

Você pode concatenar constantes e variáveis de cadeia em qualquer tipo de combinação e depois atribuir o resultado a uma variável de cadeia ou fornecer o resultado como argumento para um comando ou função que aceite valores de cadeia (como PRINT).

Por exemplo, o comando a seguir usa o operador de concatenação (+) para combinar as cadeias literais *Programar*, *é* e *divertido* e atribui o resultado à variável de cadeia *frase$*:

```
frase$ = "Programar" + "é" + "divertido"
```

O operador de concatenação combina as três cadeias literais para formar uma cadeia. O operador de atribuição (=), então, atribui o resultado à variável de cadeia *frase$*. Observe que não existem espaços nas cadeias literais neste comando. (Os espaços em volta dos operadores não contam.) Se você tivesse que imprimir o valor de *frase$* por meio do comando PRINT, a seguinte saída seria mostrada:

```
Programarédivertido
```

O operador de concatenação combina as cadeias exatamente como estão, sem espaços. Para incluí-los, você deverá inserir os espaços às próprias cadeias literias; ou seja, um espaço precisa aparecer *dentro* das aspas. Cada um dos comandos a seguir faria isso:

```
■ PRINT "Programar" + " é" + " divertido"
```

- PRINT "Programar " + "é " + "divertido"
- PRINT "Programar" + " " + "é" + " " + "divertido"

Você também pode fornecer o resultado de uma concatenação como um argumento de alguns comandos do QBasic diretamente — sem atribuir o resultado a uma variável intermediária, como *frase$*:

```
PRINT "Programar " + "é " + "divertido"
```

O resultado do comando anterior é o mesmo de

```
frase$ = "Programar " + "é " + "divertido"
PRINT frase$
```

Prática: Concatenando cadeias

O programa NOTICIAS.BAS (Figura 9-6) demonstra uma série de opções disponíveis por meio da concatenação de cadeias.

Digite o programa NOTICIAS.BAS e execute-o.

```
' NOTICIAS.BAS
' Este programa demonstra a concatenação.

CONST DOENCA$ = "Dengue"

DIM ação AS STRING * 10
ação = "avança"

direção$ = "para o interior"

CLS

INPUT "Favor entrar com o nome de uma cidade: ",
cidade$

PRINT
PRINT "Notícia em Destaque: ";
PRINT cidade + " " + DOENCA$ + " " + ação + " " +
direcao$ + "!"
```

FIGURA 9-6.
NOTICIAS.BAS: Um programa demonstrando a concatenação de cadeias.

Se você responder ao pedido de uma cidade com *Niterói*, receberá a seguinte saída:

```
Favor entrar com o nome de uma cidade: Niterói
Notícia em Destaque: Niterói. Dengue avança para o interior!
```

USO DE FUNÇÕES DE CADEIA

Até aqui você declarou constantes e variáveis de cadeia, combinou cadeias em um processo chamado concatenação e usou cadeias como argumentos em comandos INPUT e PRINT. Nesta seção você conhecerá as funções do QBasic criadas especificamente para manipular e retornar valores de cadeias literais e variáveis de cadeia. Você aprenderá a

- Mudar o tipo de letra de uma cadeia
- Determinar o tamanho de uma cadeia
- Separar cadeias

Alteração do Tipo de Letra de uma Cadeia

É fácil mudar as letras de uma cadeia para maiúsculas ou para minúsculas. Simplesmente use a função UCASE$ ou LCASE$. Essas funções são práticas se você precisar que todos os dados no seu programa estejam no mesmo formato, e não tiver certeza se o usuário entrará com o texto em maiúsculas ou minúsculas. Quando você aprender ao final deste capítulo como as cadeias são comparadas, entenderá a importância desta informação.

Uso da função UCASE$

Para mudar as letras de uma cadeia para maiúsculas, use a função UCASE$, da seguinte forma:

UCASE$(*expressão-cadeia*)

expressão-cadeia é qualquer variedade de cadeia. O valor retornado por UCASE$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento a um comando ou função que aceite valores de cadeia. UCASE$ só afeta as letras minúsculas de *expressão-cadeia*.

Prática: Usando a função UCASE$

O programa MAIUSC.BAS (Figura 9-7) demonstra como funciona a função UCASE$. MAIUSC.BAS declara uma constante de cadeia e duas variáveis de cadeia e depois usa a função UCASE$ para mostrá-las em maiúsculas. Observe que a função UCASE$ só afeta a saída do comando PRINT — não muda o conteúdo da constante *NOME$* ou a variável *endereço$*. (Isso também é bom, pois — como já vimos — atribuir um novo valor a uma constante seria um erro.) Quando as cadeias são mostradas novamente ao final do pro-

grama, somente a variável *bairro* permanece em maiúsculas, em virtude da sua atribuição de cadeia original.

Digite o programa MAIUSC.BAS e execute-o.

```
' MAIUSC.BAS
' Este programa demonstra a função UCASE$.

CONST NOME$ = "Sr. Artur Conrado"
endereço$ = "Av. Copacabana, 1326"
bairro$ = UCASE$("Copacabana")

CLS

PRINT UCASE$(NOME$)
PRINT UCASE$(endereco$) + ", " + bairro$
PRINT
PRINT NOME$
PRINT endereco$ + ", " + bairro$
```

FIGURA 9-7.
MAIUSC.BAS: Uma demonstração da função UCASE$.

Você verá a seguinte saída:

```
SR. ARTUR CONRADO
AV. COPACABANA, 1326, COPACABANA

Sr. Artur Conrado
Av. Copacabana, 1326, COPACABANA
```

Uso da função LCASE$

Para mudar as letras de uma cadeia para minúsculas, use a função LCASE$, da seguinte forma:

LCASE$(*expressão-cadeia*)

expressão-cadeia é qualquer variedade de cadeia. O valor retornado por LCASE$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento a um comando ou função que aceite valores de cadeia. LCASE$ só afeta as letras maiúsculas de *expressão-cadeia*.

Prática: Usando a função LCASE$

O programa MINUSC.BAS (Figura 9-8) demonstra como funciona a função LCASE$. MINUSC.BAS converte as letras maiúsculas de uma cadeia para minúsculas e atribui o resultado a uma variável e ao comando PRINT.

Digite o programa MINUSC.BAS e execute-o.

```
' MINUSC.BAS
' Este programa demonstra as funções UCASE$ e
'  LCASE$.

CLS

personagem$ = "Sherlock Holmes"
PRINT personagem$

personagem$ = UCASE$(personagem$)
PRINT personagem$

personagem$ = LCASE$(personagem$)
PRINT personagem$
```

FIGURA 9-8.
MINUSC.BAS: Uma demonstração das funções LCASE$ e UCASE$.

Você verá a seguinte saída:

```
Sherlock Holmes
SHERLOCK HOLMES
sherlock holmes
```

Determinação do Tamanho de uma Cadeia

Em geral, você desejará saber quantos caracteres existem em uma cadeia. Este conhecimento pode ser muito prático com cadeias de tamanho variável, sobretudo aquelas entradas pelo teclado.

Para determinar quantos caracteres (incluindo espaços) existem em uma cadeia, use a função LEN da seguinte forma:

```
LEN(expressão-cadeia)
```

expressão-cadeia mais uma vez é qualquer tipo de cadeia. O valor retornado por LEN pode ser atribuído a uma variável inteira ou fornecido como argumento para um comando ou função que aceite valores inteiros.

A rotina a seguir mostra como a função LEN determina o número de caracteres em uma cadeia e atribui o número a uma variável inteira:

```
nomeTodo$ = "Manuel da Silva Oliveira"
tamanhoNome% = LEN(nomeTodo$)
PRINT nomeTodo$; " tem"; tamanhoNome%; "caracteres"
```

Ao executar esta rotina, verá a seguinte saída:

```
Manuel da Silva Oliveira tem 24 caracteres
```

Prática: Usando a função LEN

O programa MENU.BAS (Figura 9-9) mostra como o valor retornado pela função LEN pode ser usado como argumento de um comando PRINT. MENU.BAS declara três cadeias de tamanho variável contendo seleções de pratos em potencial para o usuário. Essas cadeias são mostradas com comandos PRINT. Um comando INPUT pede ao usuário uma seleção de pratos. A seleção do usuário é atribuída à variável *escolha$* e convertida para maiúsculas com a função UCASE$. O comando SELECT CASE, em seguida, compara a variável *escolha$* com as três variáveis de pratos. Se houver alguma correspondência, a escolha de prato e o seu tamanho (retornado pela função LEN) serão impressos. Se não houver uma igualdade, o comando CASE ELSE imprimirá a mensagem *Não reconheço esse prato!*

Digite o programa MENU.BAS e execute-o.

```
' MENU.BAS
' Este programa demonstra a função LEN.

pratoSimples$ = "Bife à Milanesa"   ' declara cadeias
especialCasa$ = "Filé de Dourado"
escolhaMassa$ = "Lasanha Surpresa"

CLS

PRINT "Que prato deseja?" ' mostra o pedido
PRINT

PRINT pratoSimples$       ' mostra as escolhas
PRINT especialCasa$
PRINT escolhaMassa$
PRINT

INPUT "Seleção: ", escolha$ ' pede escolha do usuário
escolha$ = UCASE$(escolha$) ' converte para
                            ' maiúsculas
PRINT

SELECT CASE escolha$        ' acha variável combinando
  CASE IS = UCASE$(pratoSimples$) ' e imprime tamanho
    PRINT pratoSimples$; " tem"; LEN(pratoSimples$);
"caracteres"
  CASE IS = UCASE$(especialCasa$)
    PRINT especialCasa$; " tem"; LEN(especialCasa$);
"caracteres"
```

FIGURA 9-9. (*continua*)
MENU.BAS: Um programa demonstrando a função LEN.

FIGURA 9-9. *continuação*

```
  CASE IS = UCASE$(escolhaMassa$)
    PRINT escolhaMassa$; " tem"; LEN(escolhaMassa$);
"caracteres"
  CASE ELSE          ' mostra mensagem se não combinar
    PRINT "Não reconheço esse prato!"
END SELECT
```

Você verá uma saída semelhante a esta:

```
Que prato deseja?

Bife à Milanesa
Filé de Dourado
Lasanha Surpresa

Seleção: Bife à Milanesa

Bife à Milanesa tem 15 caracteres
```

Separação de Cadeias

Anteriormente neste capítulo você aprendeu que o QBasic lhe permite combinar cadeias por meio da concatenação. Mas às vezes você deseja separar uma cadeia — pegar apenas o sobrenome de uma pessoa, por exemplo. O QBasic oferece seis funções que lhe permitem trabalhar com partes de cadeias. As seções a seguir descrevem como usar essas funções para

- Obter o extremo direito de uma cadeia (RIGHT$)
- Obter o extremo esquerdo de uma cadeia (LEFT$)
- Obter o meio de uma cadeia (MID$)
- Recortar o extremo direito de uma cadeia (RTRIM$)
- Recortar o extremo esquerdo de uma cadeia (LTRIM$)
- Encontrar uma cadeia dentro de outra cadeia (INSTR)

Você também verá os comandos que lhe permitem

- Apanhar uma linha inteira da entrada (LINE INPUT$)
- Imprimir caracteres repetidos (SPACE$, STRING$)

Extremos de uma cadeia

As funções RIGHT$ e LEFT$ lhe permitem recuperar um ou mais caracteres a partir de um extremo de uma cadeia. Isso pode ser útil

quando você quiser mostrar apenas parte de uma cadeia ou quando quiser remover parte de uma cadeia.

A sintaxe da função RIGHT$ é a seguinte:

RIGHT$(*expressão-cadeia, n*)

A sintaxe para a função LEFT$ é a seguinte:

LEFT$(*expressão-cadeia, n*)

expressão-cadeia é qualquer variedade de cadeia, e *n* é um valor inteiro (variando de 0 até o tamanho da cadeia) indicando o número de caracteres a serem retornados por RIGHT$ e LEFT$. O valor retornado pode ser atribuído a uma variável de cadeia ou fornecido como argumento para um comando ou função que aceite valores de cadeia.

Prática: Usando a função RIGHT$

O programa PEGADIR.BAS (Figura 9-10) usa a função RIGHT$ para recuperar caracteres de uma variável chamada *alfabeto$*, que contém as 26 letras do alfabeto. PEGADIR.BAS extrai o número de caracteres solicitado, mostrando-os com uma contagem de caracteres.

Digite o programa PEGADIR.BAS e execute-o.

```
' PEGADIR.BAS
' Este programa demonstra a função RIGHT$.

CLS
' declara cadeia de teste
alfabeto$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

PRINT "Quantos caracteres da cadeia a seguir (a "
PRINT "partir da direita) você deseja mostrar?
PRINT
PRINT alfabeto$     ' mostra cadeia de teste
PRINT

     ' apanha o número de caracteres do usuário
DO   ' repete até o número estar na faixa de 1 a 26
  INPUT "   Número (1-26): ", numDireita%
LOOP WHILE (numDireita% < 1) OR (numDireita% > 26)

PRINT                ' mostra os caracteres
carDireita$ = RIGHT$(alfabeto$, numDireita%)
PRINT "Você indicou"; LEN(carDireita$);
"caracteres: "; carDireita$
```

FIGURA 9-10.
PEGADIR.BAS: Um programa demonstrando a função RIGHT$.

Você verá uma saída semelhante a esta:

```
Quantos caracteres da cadeia a seguir (a
 partir da direita) você deseja mostrar?

ABCDEFGHIJKLMNOPQRSTUVWXYZ

  Número (1-26): 14

Você indicou 14 caracteres: MNOPQRSTUVWXYZ
```

Prática: Usando a função LEFT$

O programa PEGAESQ.BAS (Figura 9-11) é uma nova versão do programa PEGADIR.BAS para extrair caracteres à esquerda de uma cadeia com a função LEFT$. Observe que os nomes da variável foram ligeiramente mudados (*numDireita%* tornou-se *numEsquerda%*, e *carDireita$* tornou-se *carEsquerda$*) e que a função RIGHT$ foi mudada para LEFT$. Além dessas mudanças (e algumas nas mensagens de pedido e comentários do programa), PEGAESQ.BAS é idêntico a PEGADIR.BAS. Como as operações das funções RIGHT$ e LEFT$ são muito semelhantes, é muito fácil mudar um programa para que modifique uma cadeia do lado oposto.

Modifique PEGADIR.BAS para PEGAESQ.BAS e execute-o.

```
' PEGAESQ.BAS
' Este programa demonstra a função LEFT$.

CLS
' declara cadeia de teste
alfabeto$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

PRINT "Quantos caracteres da cadeia a seguir (a "
PRINT "partir da esquerda) você deseja mostrar?
PRINT
PRINT alfabeto$     ' mostra cadeia de teste
PRINT
    ' apanha o número de caracteres do usuário
DO  ' repete até o número estar na faixa de 1 a 26
  INPUT "   Número (1-26): ", numEsquerda%
LOOP WHILE (numEsquerda% < 1) OR (numEsquerda% > 26)

PRINT             ' mostra os caracteres
carEsquerda$ = LEFT$(alfabeto$, numEsquerda%)
PRINT "Você indicou"; LEN(carEsquerda$);
"caracteres: "; carEsquerda$
```

FIGURA 9-11.
PEGAESQ.BAS: Um programa demonstrando a função LEFT$.

Você verá uma saída semelhante a esta:

```
Quantos caracteres da cadeia a seguir (a
partir da esquerda) você desejar mostrar?

ABCDEFGHIJKLMNOPQRSTUVWXYZ

  Número (1-26): 14

Você indicou 14 caracteres: ABCDEFGHIJKLMN
```

Extraindo o meio de uma cadeia

A função MID$ lhe permite apanhar um ou mais caracteres de qualquer lugar dentro de uma cadeia — a partir da esquerda, do meio ou (com a ajuda da função LEN) da direita. Sua versatilidade torna a função MID$ uma das funções de cadeia mais úteis. Além disso, como veremos adiante, ela oferece poder de processamento para solucionar muitos problemas relacionados à cadeia.

A sintaxe da função MID$ é a seguinte:

MID$(*expressão-cadeia, início, tamanho*)

expressão-cadeia é qualquer tipo de cadeia, *início* é um valor inteiro entre 1 e o tamanho da cadeia (indicando o primeiro caracter a ser retornado), e *tamanho* é um valor inteiro indicando o número de caracteres a serem retornados. O valor retornado por MID$ pode ser atribuído a uma variável de cadeia ou retornado como argumento para um comando ou função que aceite valores de cadeia. A Figura 9-12 mostra os elementos da sintaxe da função MID$.

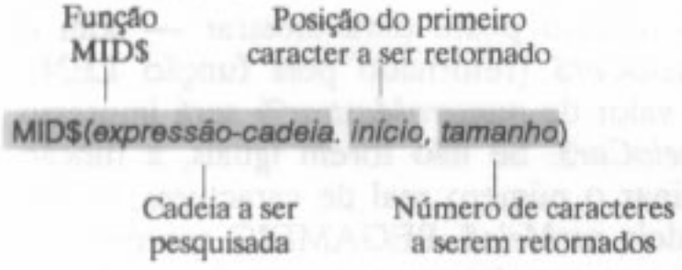


FIGURA 9-12.
Os componentes da função MID$.

Os comandos a seguir mostram algumas aplicações válidas da função MID$. Observe as poderosas possibilidades que surgem quando você usa o valor retornado por uma função como argumento para MID$ ou quando atribui o valor retornado por MID$ a outro comando ou função.

```
nomeMeio$ = MID$("Rainha Vitória da Inglaterra", 8, 7)
```
Resultado: *nomeMeio$* contém Vitória

```
endereço$ = "Rua das Ameixas, 1521"
inícioNúmero% = 18
```

```
tamanho% = 4
PRINT UCASE$(MID$(endereco$, inicioNumero%, tamanho%))
```
Resultado: 1521

```
cadeia$ = "Tudo faz sentido"
palavraDireita$ = MID$(cadeia$, LEN(cadeia) - 6, 7)
```
Resultado: *palavraDireita$* contém sentido

```
PRINT "O ano corrente é "; MID$(DATE$, 7, 4)
```
Resultado: O ano corrente é 1991

Prática: Usando a função MID$

O programa PEGAMEIO.BAS (Figura 9-13) mostra como recuperar caracteres do meio de uma cadeia com a função MID$. PEGAMEIO modifica os programas PEGADIR e PEGAESQ a fim de incluir um ponto de partida junto com o número de caracteres na cadeia *alfabeto$* a serem mostrados. PEGAMEIO usa dois laços DO para obter valores inteiros na faixa apropriada e, em seguida, usa a função MID$ para atribuir os caracteres selecionados à variável *carMeio$*. Os resultados da seleção são impressos com o seguinte comando IF, que aparece quase no fim do programa:

```
IF (numeroMostrar% = LEN(carMeio$)) THEN
    PRINT numeroMostrar%; "caracteres mostrados: "; carMeio$
ELSE
    PRINT numeroMostrar%; "caracteres solicitados,";
    PRINT LEN(carMeio$); "mostrados: "; carMeio$
END IF
```

O comando IF compara *numeroMostrar%* — a variável contendo o número de caracteres que o usuário pediu para mostrar — com o número de caracteres em *meioCar$* (retornado pela função LEN). Se os valores forem iguais, o valor de *numeroMostrar%* será impresso junto com o conteúdo de *meioCar$*. Se não forem iguais, a função LEN será usada para determinar o número real de caracteres, e este valor será mostrado com a cadeia *carMeio$*. PEGAMEIO contém esta mensagem adicional para indicar ao usuário que o tamanho solicitado excedeu o número de caracteres restantes na cadeia.

Modifique o programa PEGAESQ.BAS para que combine com o programa PEGAMEIO.BAS e execute-o. Você verá a seguinte saída:

```
Quantos caracteres da cadeia a seguir (a
partir da esquerda) você deseja mostrar?

ABCDEFGHIJKLMNOPQRSTUVWXYZ

  Número (1-26): 14

Em que posição gostaria de começar?
```

```
Número inicial (1-26): 4

14 caracteres mostrados: DEFGHIJKLMNOPQ
```

```
' PEGAMEIO.BAS
' Este programa demonstra a função MID$.

CLS

' declara cadeia de teste
alfabeto$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

PRINT "Quantos caracteres da cadeia a seguir (a "
PRINT "partir da esquerda) você deseja mostrar?
PRINT
PRINT alfabeto$          ' mostra cadeia de teste
PRINT

    ' apanha o número de caracteres do usuário
DO  ' repete até o número estar na faixa de 1 a 26
  INPUT "   Número (1-26): ", númeroMostrar%
LOOP WHILE (numeroMostrar% < 1) OR (numeroMostrar%
> 26)
PRINT                    ' pega número inicial...
PRINT "Em que posição gostaria de começar?"
PRINT

DO                       ' na faixa apropriada
  INPUT "   Número inicial (1-26): ", inicio%
LOOP WHILE (inicio% <1) OR (inicio% > 26)
PRINT                    ' apanha caracteres
carMeio$ = MID$(alfabeto$, inicio%, numeroMostrar%)

  ' compara caracteres solicitados com reais
  '   e imprime uma mensagem apropriada com a cadeia
IF (numeroMostrar% = LEN(carMeio$)) THEN
  PRINT numeroMostrar%; "caracteres mostrados: ";
carMeio$
ELSE
  PRINT numeroMostrar%; "caracteres solicitados,";
  PRINT LEN(carMeio$); "mostrados: "; carMeio$
END IF
```

FIGURA 9-13.
PEGAMEIO.BAS: Um programa demonstrando a função MID$.

Se você especificar um número fora da faixa permitida por PEGA-MEIO, verá uma saída semelhante a esta:

```
Quantos caracteres da cadeia a seguir (a
partir da esquerda) você deseja mostrar?

ABCDEFGHIJKLMNOPQRSTUVWXYZ

  Número (1-26): 0
  Número (1-26): 30
  Número (1-26): 15

Em que posição gostaria de começar?

  Número inicial (1-26): w

Redo from start
  Número inicial (1-26): 20

 15 caracteres solicitados, 7 mostrados: TUVWXYZ
```

Se você digitar um valor não-numérico (como uma letra) em um dos pedidos, o QBasic imprime a mensagem *Redo from start* [refazer do início] e reapresenta o pedido. É importante prever este tipo de resposta do usuário ao tentar desenvolver programas "à prova de bala" que outras pessoas usarão.

Recortando uma cadeia

Quando você deseja alinhar cadeias ou eliminar espaços desnecessários nos extremos das cadeias, pode usar as funções de recorte do QBasic, LTRIM$ e RTRIM$.

A função LTRIM$ lhe permite recortar o lado esquerdo de uma cadeia; ou seja, LTRIM$ retorna uma cadeia sem espaços ou tabulações iniciais. A função RTRIM$ faz um trabalho semelhante no lado direito da cadeia.

A sintaxe da função LTRIM$ é a seguinte:

```
LTRIM$(expressão-cadeia)
```

A sintaxe da função RTRIM$ é a seguinte:

```
RTRIM$(expressão-cadeia)
```

expressão-cadeia pode ser qualquer variedade de cadeia. O valor retornado por LTRIM$ ou RTRIM$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento a um comando ou função que aceite valores de cadeia.

A rotina a seguir usa a função LTRIM$ para remover os três espaços iniciais da variável de cadeia *saudação$* antes de imprimir:

```
saudacao$ = "  Boa noite"
saudacao$ = LTRIM(saudacao$)

PRINT saudacao$; "!"
```

Esta rotina resulta na seguinte saída:

```
Boa noite!
```

A rotina a seguir usa a função RTRIM$ para remover os três espaços finais da variável *alemao$* antes de imprimir:

```
alemao$ = "Guten Abend  "
alemao$ = RTRIM(alemao$)

PRINT alemao$; "!"
```

Esta rotina resulta na seguinte saída:

```
Guten Abend!
```

Prática: Usando LTRIM$ e RTRIM$

O programa SAUDACAO.BAS (Figura 9-14) mostra como usar as funções LTRIM$ e RTRIM$ juntas para retirar os espaços em branco dos dois extremos de uma cadeia de tamanho fixo.

SAUDACAO declara duas variáveis de cadeia com tamanho fixo, chamadas *nome* e *sobrenome*, e usa comandos INPUT para obter os valores de cadeia para essas variáveis. O programa SAUDACAO mostra duas mensagens para o usuário do programa. A primeira mensagem não é recortada; a segunda é recortada pelas funções LTRIM$ e RTRIM$. Observe que as próprias cadeias de tamanho fixo não são modificadas — você não pode mudar o tamanho de uma cadeia de tamanho fixo.

Observe como um argumento de cadeia é passado de RTRIM$ para LTRIM$ para PRINT no seguinte comando de SAUDACAO:

```
PRINT "Acho que "; LTRIM$(RTRIM$(nome)); " ";
```

A função RTRIM$ processa a variável *nome* e passa o valor de cadeia resultante à função LTRIM$. LTRIM$, então, processa a cadeia e passa o valor de cadeia ao comando PRINT. O comando PRINT combina o valor de cadeia com dois outros valores de cadeia, mostrando-os na tela.

Digite o programa SAUDACAO.BAS (Figura 9-14) e execute-o. Você verá uma saída semelhante a esta:

```
Entre com seu nome: Jaques
Entre com seu sobrenome: Silva

Muito prazer, Jaques      Silva      !
Acho que Jaques Silva tem uma ligação esperando.
```

```
' SAUDACAO.BAS
' Este programa ilustra as funções LTRIM$ e RTRIM$.

DIM nome AS STRING * 12   ' declara cadeias de tamanho
                          '  fixo
DIM sobrenome AS STRING * 15

CLS
                          ' pega nomes e sobrenomes
INPUT "Entre com seu nome: ", nome
INPUT "Entre com seu sobrenome: "; sobrenome

PRINT              ' imprime saudação sem recortar
PRINT "Muito prazer, "; nome; " "; sobrenome; "!"
                   ' imprime recortando
PRINT "Acho que "; LTRIM$(RTRIM$(nome)); " ";
PRINT LTRIM$(RTRIM$(sobrenome)); " tem uma ligação
esperando."
```

FIGURA 9-14.
SAUDACAO.BAS: Um programa demonstrando as funções LTRIM$ e RTRIM$.

Pegando uma linha inteira do usuário

Durante todo este capítulo usamos o comando INPUT para apanhar a entrada do usuário. O comando INPUT é muito versátil — ele pode atribuir a entrada a uma ou diversas variáveis de diferentes tipos e fornecer um pedido opcional para indicar exatamente o que o usuário deverá entrar. No entanto, uma coisa que ainda não vimos é como o comando INPUT processa uma vírgula na linha de entrada. Pense no comando a seguir, que pede ao usuário para entrar com um nome e um endereço:

```
INPUT "Entre com nome e endereço: ", corresp$
```

Se o usuário responder ao pedido com uma cadeia contendo vírgulas, a mensagem de erro *Redo from start* será mostrada, como vemos no seguinte diálogo:

```
Entre com nome e endereço: João Vítor, Rua 12 1118, Rio, RJ,
20000

Redo from start
Entre com nome e endereço:
```

Como vimos no Capítulo 4, o comando INPUT tem um uso especial para o caracter vírgula na linha de entrada: a vírgula separa os valores atribuídos às variáveis. Mas o que acontece quando vírgulas inesperadas aparecem na entrada — como vimos aqui? Neste caso, você poderia resolver o problema atribuindo partes da cadeia de entrada

a diferentes variáveis. Por exemplo, o comando INPUT a seguir atribui o valor de cadeia entrado pelo usuário a cinco variáveis de cadeia:

```
INPUT "Entre com nome e endereço: ", nome$, rua$, cidade$, estado$, cep$
```

Mas existem ocasiões em que seria muito mais simples ter apenas um nome de variável associado com uma linha de entrada. O QBasic possui uma solução para este problema com o comando LINE INPUT. O comando LINE INPUT pega uma linha de texto inteira do teclado, atribuindo-a a uma variável de cadeia, mesmo que existam vírgulas.

A sintaxe do comando LINE INPUT é a seguinte:

```
LINE INPUT [;] ["mensagem";] variável-cadeia
```

mensagem é uma cadeia literal que pede a entrada ao usuário, e *variável-cadeia* é qualquer variável do tipo cadeia.

- Colocar um ponto-e-vírgula logo após LINE INPUT mantém o cursor na mesma linha após o usuário teclar ENTER.
- Se a *mensagem* for incluída no comando, um ponto-e-vírgula é obrigatório para separar *mensagem* de *variável-cadeia.*
- Diferente do comando INPUT, o comando LINE INPUT não mostra um ponto de interrogação junto com a *mensagem.*

A rotina a seguir demonstra a utilidade de LINE INPUT para linhas longas contendo o caracter vírgula:

```
LINE INPUT "Entre com nome e endereço: "; corresp$
PRINT corresp$
```

Quando você executar a rotina e entrar com a informação que tentamos entrar antes, verá a seguinte saída:

```
Entre com nome e endereço: João Vítor, Rua 12 1118, Rio, RJ,
20000
João Vítor, Rua 12 1118, Rio, RJ, 20000
```

Você verá o comando LINE INPUT ocasionalmente nos próximos capítulos.

Impressão de caracteres repetidos

O QBasic oferece duas funções úteis para gerar cadeias de caracteres repetidos: a função SPACE$, que retorna uma cadeia de espaços, e a função STRING$, que retorna uma cadeia de caracteres. As duas funções oferecem meios rápidos para montar cadeias para uso na formatação e alinhamento da saída do seu programa.

A sintaxe da função SPACE$ é a seguinte:

```
SPACE$(n)
```

n é um valor inteiro especificando o número de espaços que a cadeia terá. O valor retornado por SPACE$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento para um comando ou função que aceite valores de cadeia.

O uso mais comum da função SPACE$ é na formatação da saída, como mostra a rotina a seguir:

```
branco$ = SPACE$(15)
PRINT branco$; "Grande venda de coelhos hoje!"
```

Você pode usar SPACE$ sempre que endentar o texto coerentemente por um número de espaços definido.

A sintaxe da função STRING$ é a seguinte:

STRING$(*m, expressão-cadeia*)

m é um valor inteiro especificando o tamanho da cadeia a ser retornada e *expressão-cadeia* é o caracter a ser repetido. O valor retornado por STRING$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento para um comando ou função que aceite valores de cadeia.

Prática: Usando SPACE$ e STRING$

O uso mais comum da função STRING$ é para títulos usados na saída do programa. O programa CABEC.BAS (Figura 9-15) usa STRING$ para mostrar uma mensagem de cabeçalho no meio da tela. Observe que o uso de uma constante para especificar o número de caracteres repetidos facilita a modificação do programa mais tarde, e que as variáveis criadas por SPACE$ e STRING$ são excelentes candidatas à concatenação.

Digite o programa CABEC.BAS e execute-o.

```
' CABEC.BAS
' Este programa demonstra as funções SPACE$
'  e STRING$.

CONST TAMANHO% = 16
arquivo$ = "RECEITA.DOC"
branco$ = SPACE$(TAMANHO%)
asterisco$ = STRING$(TAMANHO%, "*")
titulo$ = branco$ + asterisco$ + " " + arquivo$ +
" " + asterisco$

CLS

PRINT titulo$
```

FIGURA 9-15.
CABEC.BAS: Um programa demonstrando as funções SPACE$ e STRING$.

Você verá o nome de arquivo *RECEITA.DOC* no meio da linha superior da tela, rodeada por números iguais de asteriscos e espaços em branco.

Busca de uma cadeia dentro de outra

Usamos as funções RIGHT$, LEFT$ e MID$ para retornar caracteres das partes direita, esquerda e do meio de uma cadeia. Essas funções são muito eficazes na extração de caracteres de um determinado local em uma cadeia, mas não servem para procurar e extrair um padrão específico de uma cadeia. O QBasic oferece esse serviço com a função INSTR, que pesquisa uma cadeia dentro de outra cadeia. A função INSTR reúne RIGHT$, LEFT$, MID$ e as funções de suporte que vimos nesta seção (UCASE$, LCASE$, LEN$, RTRIM$, LTRIM$, SPACE$ e STRING$) para entregar uma coleção completa de ferramentas para trabalho com cadeias.

A sintaxe da função INSTR é a seguinte:

INSTR([*início*,]*cadeia-base, cadeia-pesquisa*)

início é um valor inteiro opcional especificando o caracter em que a pesquisa deverá começar, *cadeia-base* é a cadeia sendo verificada, e *cadeia-pesquisa* é a cadeia sendo procurada. O valor retornado por INSTR pode ser atribuído a uma variável inteira ou fornecido como argumento parra um comando ou função que aceite valores inteiros. A tabela a seguir lista os valores que a função INSTR pode retornar:

Condição	*Valor inteiro retornado*
cadeia-pesquisa achada na *cadeia-base*	Posição na *cadeia-base* onde foi encontrada
cadeia-pesquisa não achada em *cadeia-base*	0
início maior do que o tamanho de *cadeia-base*	0
cadeia-base não possui caracteres	0
cadeia-pesquisa não possui caracteres	*início* (se houver); se não, 1

Prática: Usando a função INSTR

O programa BUSCA.BAS (Figura 9-16) usa a função INSTR para pesquisar a cadeia *ndo* na cadeia, *Andando, cantando e divertindo os ouvintes.*

Digite o programa BUSCA.BAS e execute-o.

```
' BUSCA.BAS
' Este programa demonstra a função INSTR.

CLS

pesquisa$ = "ndo"
base$ = "Andando, cantando e divertindo os ouvintes"
localCad% = INSTR(1, base$, pesquisa$)

IF (localCad% <> 0) THEN
  PRINT pesquisa$; " apareceu logo no caracter";
localCad%
ELSE
  PRINT pesquisa$; " não foi achado"
END IF
```

FIGURA 9-16.
BUSCA.BAS: Um programa demonstrando a função INSTR.

Quando você rodar o programa, verá a seguinte saída:

```
ndo apareceu logo no caracter 5
```

Embora o padrão *ndo* apareça três vezes em *base$*, a função INSTR retorna o local apenas da primeira ocorrência. Entretanto, você pode usar INSTR dentro de um laço para encontrar múltiplas ocorrências de um padrão.

É bom verificar o valor retornado por INSTR sempre que usar a função INSTR. Isso permitirá que o programa aja apropriadamente se a cadeia não for achada, ou se a cadeia de pesquisa ou cadeia-base for vazia.

Por exemplo, no programa BUSCA, se *ndo* não fosse achado, a função INSTR atribuiria o valor 0 à variável *localCad%*, e o comando IF mostraria a seguinte mensagem para indicar que *ndo* não apareceu em *base$*:

```
ndo não foi achado
```

Prática: Encontrando múltiplas ocorrências de um padrão

O programa REPETE.BAS (Figura 9-17) usa a função INSTR para encontrar múltiplas ocorrências de um padrão em uma série de linhas entradas pelo teclado.

O núcleo de REPETE encontra-se na função *Repete%*. Toda vez que INSTR encontra a cadeia de pesquisa (*pesquisa$*) na cadeia-base (*base$*), o local da cadeia de pesquisa é atribuído à variável *carAtual%*. Depois a variável *num%* é incrementada e a variável *carAtual%* é movida da sua posição para logo após a cadeia encontrada na cadeia-base (o novo local de partida para a próxima pesquisa). Este processo

continua até que o final da cadeia-base seja alcançado ou até que a cadeia de pesquisa não seja achada no restante da cadeia-base. A função *Repete%*, então, retorna o número total de combinações para a linha na variável *repeticoes%* no programa principal.

Digite o programa REPETE.BAS e execute-o. Observe que a Figura 9-17 mostra como REPETE.BAS aparece quando impresso — e não dentro do QBasic. Não se esqueça de usar o comando New SUB no menu Edit para entrar com o subprograma *PegaTexto* e o comando New FUNCTION no menu Edit para entrar com a função *Repete%*. Você poderá rever este processo no Capítulo 7.

```
' REPETE.BAS
' Este programa pede um certo número de linhas
'   e uma cadeia de pesquisa, mostrando as linhas
'   e o número de combinações achadas.

' define o número máximo de linhas que poderão ser
'   entradas e declara cadeia para conter as linhas

CONST MAXLIN% = 10
DIM linhas$(MAXLIN%)

' declara subprograma PegaTexto e função Repete%

DECLARE SUB PegaTexto (linhas$(), numLinhas%)
DECLARE FUNCTION Repete% (pesquisa$, base$)

CLS
' chama subprograma para pegar entrada do usuário;
'  ao retornar, a variável numLinhas% terá o número
'  de linhas recebidas

PegaTexto linhas$(), numLinhas%

' pega o padrão a ser procurado

PRINT
INPUT "Entre com a cadeia a procurar: ", padrao$
PRINT

' chama Repete% para saber o número de combinações
'  por linha; totalRepete% acumula o número
```

FIGURA 9-17. (*continua*)
REPETE.BAS: Um programa que lê várias linhas, procurando um padrão, e informa o número de vezes que o encontrou.

FIGURA 9-17. *continuação*

```
FOR i% = 1 TO numLinhas%
  repeticoes% = Repete%(padrão, linhas$(i%))
  totalRepete% = totalRepete% + repeticoes%
NEXT i%

' mostra as linhas entradas pelo usuário...

PRINT "Você entrou com as seguintes linhas:"
PRINT
FOR i% = 1 TO numLinhas%
  PRINT linhas$(i%)
NEXT i%

' e o número total de combinações

PRINT
PRINT "O padrão '"; padrao$; "' apareceu";
totalRepete%; "vezes."

END

SUB PegaTexto (vetorCad$(), cont%)

' O subprograma PegaTexto preenche vetorCad$ com
'  texto entrado pelo teclado. O máximo de linhas
'  aceito é determinado pela constante global
'  MAXLIN%. Tanto vetorCad$ quanto cont$ (o número
'  de linhas realmente entradas) são retornados
'  ao programa principal.

PRINT "Entre com até"; MAXLIN%; "linhas de texto;
no fim, ";
PRINT "pressione Enter em uma nova linha."
PRINT
conta% = 0

DO
  LINE INPUT "-> "; lin$  ' pega uma linha
  IF (lin% <> "") THEN    ' se não for fim, copia
   cont% = cont% + 1      '   para vetorCad$
   vetorCad$(cont%) = lin$
  END IF

' laço até cont% = MAXLIN% ou receber linha vazia
LOOP WHILE (cont% < MAXLIN%) AND (lin$ <> "")
```

(continua)

FIGURA 9-17. *continuação*

```
END SUB

FUNCTION Repete% (pesquisa$, base$)

' A função Repete% retorna o número de vezes que uma
'  cadeia de pesquisa é encontrana numa cadeia-base.

' determina tamanho das cadeias de pesquisa e base
tamPesq% = LEN(pesquisa$)
tamBase% = LEN(base$)
' deslocamento de caracter na cadeia de base
carAtual% = 1
' total acumulado de combinações achadas na base
num% = 0

' laço até toda cadeia ser processada ou
'  INSTR retornar 0

WHILE (carAtual% <= tamBase%) AND (carAtual% <> 0)
  carAtual% = INSTR(carAtual%, base$, pesquisa$)
  IF (carAtual% <> 0) THEN   ' se achou,
   num% = num% + 1           ' incrementa o contador
   ' novo início = atual mais tamanho da cadeia
   carAtual% = carAtual% + tamPesq%
  END IF
WEND

Repete% = num%      ' retorna total de combinações

END FUNCTION
```

Você verá uma saída semelhante a esta:

```
Entre com até 10 linhas de texto; no fim,
pressione Enter em uma nova linha.

-> Dez homens lutando,
-> Nove damas dançando,
-> Oito vacas pastando,
-> Sete cisnes nadando,
-> Seis cães latindo,
-> Cinco anéis brilhando,
-> Quadro andorinhas no céu,
-> Três mosqueteiros duelando,
-> Duas tartarugas caindo, e
-> Uma perdiz em uma mangueira.

Entre com a cadeia a procurar: ando
```

Você entrou com as seguintes linhas:

```
Dez homens lutando,
Nove damas dançando,
Oito vacas pastando,
Sete cisnes nadando,
Seis cães latindo,
Cinco anéis brilhando,
Quatro andorinhas no céu,
Três mosqueteiros duelando,
Duas tartarugas caindo, e
Uma perdiz em uma mangueira.

O padrão 'ando' apareceu 7 vezes.
```

COMPARAÇÃO DE CADEIAS

No Capítulo 4 você aprendeu que pode comparar uma cadeia com outra e desviar para outro local no programa com base no resultado da comparação. O comando IF a seguir compara a variável *replica$* com a cadeia literal "S" e imprime uma mensagem com base na comparação:

```
replica$ = "S"
IF (replica$ = "S") THEN
    PRINT "Os dois valores de cadeia são iguais."
ELSE
    PRINT "Os dois valores são diferentes."
END IF
```

Se você executar os comandos como aparecem aqui, receberá a seguinte saída:

```
Os dois valores de cadeia são iguais.
```

Se mudar o valor de *replica$* para *s* e depois executar o comando, verá uma resposta diferente:

```
Os dois valores são diferentes.
```

Por que isso? Afinal, um S é um s... Ou não é? Que critérios o QBasic usa para sua comparação de cadeia?

O Conjunto de Caracteres ASCII

Antes que o QBasic possa comparar um caracter com outro, ele deve converter cada caracter para um número usando uma tabela de tradução chamada *conjunto de caracteres ASCII*. O QBasic, em seguida, compara os números, chamados *códigos ASCII* e retorna o valor lógico *verdadeiro* se os códigos ASCII forem iguais ou *falso* se os códigos não forem iguais.

ASCII é uma Sigla

Assim como muitos outros termos de computador, ASCII é um acrônimo — ele significa *American Standard Code for Information Interchange* [Código Padrão Americano para Intercâmbio de Informações]. A palavra-chave aqui é *código*: como o código Morse usado em rádio e telegrafia, ASCII é um código aceito internacionalmente para representar caracteres, mas ASCII é usado em computadores e telecomunicações. O Apêndice B lista o conjunto de caracteres ASCII completo.

Cada caracter no conjunto ASCII é associado a um único número; o conjunto contém 128 caracteres (códigos de 0 a 127) ao todo:

- Caracteres de controle (de 0 a 31), incluindo caracteres que correspondem a teclas especiais no teclado, como Enter, Backspace e Escape
- Símbolos de pontuação, números e símbolos matemáticos (códigos de 32 a 64)
- Letras maiúsculas do alfabeto (códigos de 65 a 90)
- Letras minúsculas do alfabeto (códigos 97 a 122)
- Símbolos diversos (códigos de 91 a 96 e de 123 a 127)

O código ASCII para a letra *A* maiúscula é 65; o código ASCII para a letra minúscula *z* é 122. Seguindo esta lógica, é fácil ver por que o QBasic considera a letra maiúscula *S* e a letra minúscula *s* como caracteres diferentes.

O Conjunto de Caracteres Estendido da IBM

O Apêndice B contém um grupo adicional de códigos de caracteres (códigos de 128 a 255) conhecidos como *conjunto de caracteres estendido da IBM*. Este conjunto de símbolos foi desenvolvido pela IBM e para a família IBM de computadores pessoais, tendo sido mais tarde adotado pela maior parte dos fabricantes de computadores compatíveis com o PC. O conjunto de caracteres estendido da IBM (às vezes chamado conjunto de caracteres ASCII superior) contém caracteres de língua estrangeira, caracteres de desenho de linha e quadro e símbolos matemáticos. Embora você possa usar esses símbolos nos seus programas, não poderá digitá-los por meios normais — eles não aparecem no teclado. Em vez disso, mantenha presa a tecla Alt e digite o código estendido da IBM com três dígitos. (Use as teclas do teclado

numérico, e não as teclas numéricas no alto do seu teclado.) Por exemplo, para mostrar o símbolo matemático na sua tela, mantenha pressionada a tecla Alt e digite *227* no teclado numérico.

O comando PRINT a seguir demonstra um uso válido do conjunto de caracteres estendido da IBM em um programa QBasic. O símbolo ¿ foi entrado segurando-se a tecla Alt e digitando-se *168*.

```
PRINT "¿Como se llama?"
```

NOTA: Alguns monitores mais antigos não podem mostrar caracteres do conjunto estendido da IBM, e você terá problemas para imprimir a maior parte do conjunto estendido, a menos que a sua impressora tenha sido elaborada especificamente para aceitá-los. Esteja pronto para ver resultados estranhos (mas não destruidores) quando usar o conjunto de caracteres estendido.

Conversão de Códigos ASCII em Caracteres

Se souber o código ASCII de um caracter mas não estiver certo quanto ao caracter que ele representa, poderá usar a função CHR$ para retornar o símbolo na sua tela ou no seu programa.

A sintaxe para a função CHR$ é a seguinte:

```
CHR$(código)
```

código é um valor inteiro especificando um código de caracter ASCII ou estendido da IBM. O valor retornado por CHR$ pode ser atribuído a uma variável de cadeia ou fornecido como argumento a um comando ou função que aceite valores de cadeia.

Prática: Usando a função CHR$

O programa ASCII.BAS (Figura 9-18) mostra como usar a função CHR$ para mostrar os caracteres nos conjuntos de caracteres ASCII e IBM. Observe que o programa salta os 32 primeiros caracteres (controle) no conjunto ASCII e pára após cada múltiplo de 23 linhas (exceto no primeiro e último conjuntos) para que você possa ver os resultados pelo tempo que desejar.

Digite o programa ASCII.BAS e execute-o. (Devido ao tamanho da saída do programa, não iremos mostrá-la aqui.)

```
' ASCII.BAS
' Este programa mostra os conjuntos de caracteres
'  ASCII e estendido da IBM.
```

FIGURA 9-18. *(continua)*
ASCII.BAS: Um programa que usa a função CHR$ para mostrar os conjuntos de caracteres ASCII e IBM.

FIGURA 9-18. *continuação*

```
CLS

FOR i% = 33 TO 255
  PRINT "Código"; i%; "= "; CHR$(i%)
  IF (i% MOD 23 = 0) THEN INPUT "Pressione Enter para
continuar...", entra$
NEXT i%
```

Conversão de Caracteres em Códigos ASCII

Como complemento à função CHR$, o QBasic oferece a função ASC, que converte um caracter ao seu código no conjunto de caracteres ASCII ou no conjunto estendido da IBM.

A sintaxe da função ASC é a seguinte:

ASC(*expressão-cadeia*)

expressão-cadeia é uma cadeia de um caracter. O valor inteiro retornado por ASC pode ser atribuído a uma variável inteira ou fornecido como argumento para um comando ou função que aceite valores inteiros.

Prática: Usando a função ASC

O programa COMPARA.BAS (Figura 9-19) usa a função ASC para mostrar como o QBasic compara caracteres. COMPARA pede ao usuário dois caracteres e depois usa um comando SELECT CASE para mostrar uma dentre três mensagens, com base nos valores numéricos retornados por ASC.

Digite o programa COMPARA.BAS e execute-o.

```
' COMPARA.BAS
' Este programa compara dois caracteres quaisquer.

CLS

INPUT "Entre com qualquer caracter: ", primeiro$
INPUT "Entre com outro caracter: ", segundo$
PRINT

SELECT CASE ASC(primeiro$) - ASC(segundo$)
  CASE IS < 0
```

FIGURA 9-19. (*continua*)
COMPARA.BAS: Um programa que usa a função ASC para comparar dois caracteres.

FIGURA 9-19. *continuação*

```
    PRINT "'"; primeiro$; "' vem antes de '";
segundo$; "'"
    PRINT "porque"; ASC(primeiro$); "é menor que";
ASC(segundo$)
  CASE IS > 0
    PRINT "'"; primeiro$; "' vem depois de '";
segundo$; "'"
    PRINT "porque"; ASC(primeiro$); "é maior que";
ASC(segundo$)
  CASE ELSE
    PRINT "'"; primeiro$; "' é igual a '";
segundo$; "'"
    PRINT "porque"; ASC(primeiro$); "é igual a";
ASC(segundo$)
END SELECT
```

Você verá uma saída semelhante a esta:

```
Entre com qualquer caracter: d
Entre com outro caracter: J

'd' vem depois de 'J'
porque 100 é maior que 74
```

Uso de Operadores Relacionais com Cadeias

Além de testar a equivalência de caracteres, o QBasic aceita comparações de cadeia com os seguintes operadores relacionais:

Operador	*Significado*	*Operador*	*Significado*
< >	Não igual	>	Maior do que
=	Igual	< =	Menor ou igual a
<	Menor do que	> =	Maior ou igual a

Um caracter é "maior do que" outro caracter se o seu código ASCII for maior. Por exemplo, o valor ASCII da letra *B* é maior do que o valor ASCII da letra *A*, de modo que a expressão

```
"A" < "B"
```

é verdadeira, e a expressão

```
"A" > "B"
```

é falsa.

Ao comparar duas cadeias, cada uma contendo mais de um caracter, o QBasic começa comparando o primeiro caracter na primeira

cadeia com o primeiro caracter da outra cadeia e depois prossegue pelas cadeias com um caracter de cada vez, até encontrar uma diferença. Por exemplo, as cadeias *Marcos* e *Marcelo* são iguais até os quintos caracteres (*o* e *e*). Como o valor ASCII de *o* é maior do que o de *c*, a expressão

```
"Marcos" > "Marcelo"
```

é verdadeira [embora *Marcelo* seja um nome mais longo do que *Marcos*].

Se não forem achadas diferenças, as cadeias serão iguais. Se duas cadeias forem iguais por vários caracteres mas uma delas continuar e a outra terminar, a cadeia mais longa será maior do que a cadeia mais curta. Por exemplo, a expressão

```
"AAAAA" > "AAA"
```

é verdadeira.

Ordenação de Cadeias: O Programa SORTCAD.BAS

O programa SORTCAD.BAS (Figura 9-20 na página 238) usa o operador relacional <= para comparar elementos de vetor e usa o comando SWAP para trocar quaisquer elementos fora de ordem.

SORTCAD declara um vetor de cadeias chamado *linhas$* e chama o subprograma *PegaTexto*. O subprograma *PegaTexto* em SORTCAD é idêntico ao subprograma de mesmo nome em REPETE.BAS (Figura 9-17): ele lê linhas de texto em um vetor e depois retorna o vetor (*linhas$*) e o número de elementos no vetor (*numElementos%*) ao programa principal.

STRSORT em seguida chama o subprograma *ShellSort*. Observe os seguintes comandos no núcleo de *ShellSort*, que comparam os elementos do vetor e depois trocam os elementos, se estiverem fora de ordem:

```
IF vetorCad$(j%) <= vetorCad$(j% + faixa%) THEN EXIT FOR
' troca elementos fora da ordem no vetor
SWAP vetorCad$(j%), vetorCad$(j% + faixa%)
```

O operador relacional <= é usado para comparar os dois elementos do vetor.

- Se o valor da expressão de cadeia *vetorCad$(j%)* for menor ou igual ao valor da expressão de cadeia *vetorCad$(j% + faixa%)*, os elementos já estão na ordem e o comando EXIT FOR termina o laço FOR.
- Se o valor da segunda expressão de cadeia for maior do que o valor da primeira, os elementos são trocados por meio do comando SWAP. SWAP, apresentado aqui pela primeira vez, troca os valores de duas variáveis quaisquer do mesmo tipo. Neste caso,

os valores das duas variáveis de cadeia serão trocados. Quando o vetor estiver ordenado completamente, ele será passado de volta ao programa principal e impresso por completo em um laço FOR.

Prática: Usando o operador relacional <= e o comando SWAP

Digite o programa SORTCAD.BAS e execute-o. Se você digitou o programa REPETE.BAS (Figura 9-17), poderá usá-lo como base para SORTCAD.BAS deletando a função *Repete*. (Pressione F2, destaque *Repete* e selecione Delete.) Em seguida, escolha o texto no programa principal e delete-o. Depois digite o subprograma *ShellSort* e o novo programa principal. Salve o novo programa como SORTCAD.BAS.

O Shell Sort

A lógica no subprograma *ShellSort* é baseada no conhecido algoritmo Shell Sort para ordenação de um vetor de números, divulgado em 1959 por Donald Shell. O Shell Sort ordena uma lista de elementos dividindo continuamente a lista principal em sublistas com a metade do tamanho e comparando os elementos em cima e embaixo das sublistas. Se os elementos estiverem fora da ordem, serão trocados. O resultado final é um vetor de itens em ordem descendente.

```
' SORTCAD.BAS
' Este programa pede ao usuário uma lista de nomes
'  e depois ordena os nomes alfabeticamente.

' define o número máximo de linhas que podem ser
'  entradas e declara um vetor de cadeia para
'  conter as linhas

CONST MAXLINES% = 15
DIM linhas$(MAXLIN%)

' declara subprograma PegaTexto e ShellSort

DECLARE SUB PegaTexto (vetorCad$(), numElementos%)
DECLARE SUB ShellSort (vetorCad$(), numElementos%)
```

FIGURA 9-20. *(continua)*
SORTCAD.BAS: Um programa que ordena uma lista de cadeias entrada pelo usuário.

FIGURA 9-20. *continuação*

```
CLS

' chama subprograma PegaTexto para pegar entrada do
'  usuário; no retorno a variável numElementos% terá
'  o número de linhas recebidas

PegaTexto linhas$(), numElementos%

' chama subprograma ShellSort para colocar vetor
'  linhas$ em ordem alfabética

ShellSort linhas$(), numElementos%

PRINT
PRINT "Resultados da ordenação:"
PRINT

FOR i% = 1 TO numElementos% ' imprime vetor ordenado
  PRINT linhas$(i%)
NEXT i%

END

SUB PegaTexto (vetorCad$(), cont%)

' O subprograma PegaTexto preenche vetorCad$ com
'  texto entrado pelo teclado. O máximo de linhas
'  permitido é determinado pela constante global
'  MAXLIN%. Tanto vetorCad$ quanto cont$ (o número
'  de linhas realmente entradas) são retornados
'  ao programa principal.

PRINT "Entre com até"; MAXLIN%; "linhas de texto;
no fim, ";
PRINT "pressione Enter em uma nova linha."
PRINT
conta% = 0

DO
  LINE INPUT "-> "; lin$  ' pega uma linha
  IF (lin% <> "") THEN    ' se não for fim, copia
   cont% = cont% + 1      '   para vetorCad$
   vetorCad$(cont%) = lin$
```

(continua)

FIGURA 9-20. *continuação*

```
  END IF
' laço até cont% = MAXLIN% ou receber linha vazia
LOOP WHILE (cont% < MAXLIN%) AND (lin$ <> "")

END SUB

SUB ShellSort (vetorCad$(), numElementos%)

' O subprograma ShellSort ordena os elementos de
'  vetorCad$ e retorna vetorCad$ ao prog. principal.
'  O argumento numElementos% contém o número de
'  elementos em vetorCad$. O ShellSort ordena
'  elementos em ordem descendente.

faixa% = numElementos% \ 2

DO WHILE faixa% > 0
  FOR i% = faixa% TO numElementos% - 1
    j% = i% - faixa% + 1
    FOR j% = (i% - faixa% + 1) TO 1 STEP -faixa%
     IF vetorCad$(j%) <= vetorCad$(j% + faixa%) THEN
EXIT FOR
      ' troca elementos do vetor fora da ordem
      SWAP vetorCad$(j%), vetorCad$(j% + faixa%)
    NEXT j%
  NEXT i%

  faixa% = faixa% \ 2
LOOP

END SUB
```

Você verá uma saída semelhante a esta:

```
Entre com até 15 linhas de texto; no fim,
pressione Enter em uma nova linha.

-> Roberto Carlos
-> Milton Nascimento
-> Erasmo Carlos
-> Maria Betânia
-> Xuxa Meneguel
-> Tim Maia
-> Ivan Lins
-> Guilherme Arantes
->
```

```
Resultados da ordenação:

Erasmo Carlos
Guilherme Arantes
Ivan Lins
Maria Betânia
Milton Nascimento
Roberto Carlos
Tim Maia
Xuxa Meneguel
```

RESUMO

Você percorreu muito chão neste capítulo, incluindo muitas novas funções e comandos no seu repertório de ferramentas de programação. Ao todo, você aprendeu 15 novos comandos e funções:

- UCASE$, LCASE$
- LEN
- RIGHT$, LEFT$, MID$
- RTRIM$, LTRIM$
- LINE INPUT
- SPACE$, STRING$
- INSTR
- CHR$, ASC
- SWAP

Você também aprendeu a declarar, combinar, separar, comparar e ordenar cadeias. O QBasic oferece uma variedade tão grande de ferramentas para trabalhar com cadeias porque muito do que é feito com computadores pessoais gira em torno do trabalho com grandes quantidades de dados de texto. O próximo capítulo discute a respeito de meios eficientes de gerenciar todos esses dados.

PERGUNTAS E EXERCÍCIOS

1. Falso ou Verdadeiro: Por convenção, as constantes de cadeia são escritas com todas as letras maiúsculas.
2. Falso ou Verdadeiro: A declaração de uma cadeia de tamanho fixo a seguir é um comando QBasic válido:

   ```
   DIM sobreNome$ AS STRING * 20
   ```

3. Que saída produzirá o comando QBasic a seguir?

   ```
   PRINT "UM" + "DOIS" + "TRÊS"
   ```

4. Quais dos seguintes valores podem ser fornecidos como argumento para a função LEN?

   a. Uma cadeia literal
   b. Uma constante de cadeia
   c. Uma variável de cadeia
   d. O resultado de uma função de cadeia

5. O que precisa acontecer para que uma chamada à função INSTR retorne o valor 0?

6. O que apareceria na tela com o seguinte comando do QBasic?

   ```
   PRINT CHR$(ASC("h") - 32)
   ```

7. Qual é o código ASCII da letra M?

8. Escreva um programa que peça ao usuário um nome e um sobrenome, converta os nomes para maiúsculas e depois imprima-os no formato *Sobrenome, Nome*.

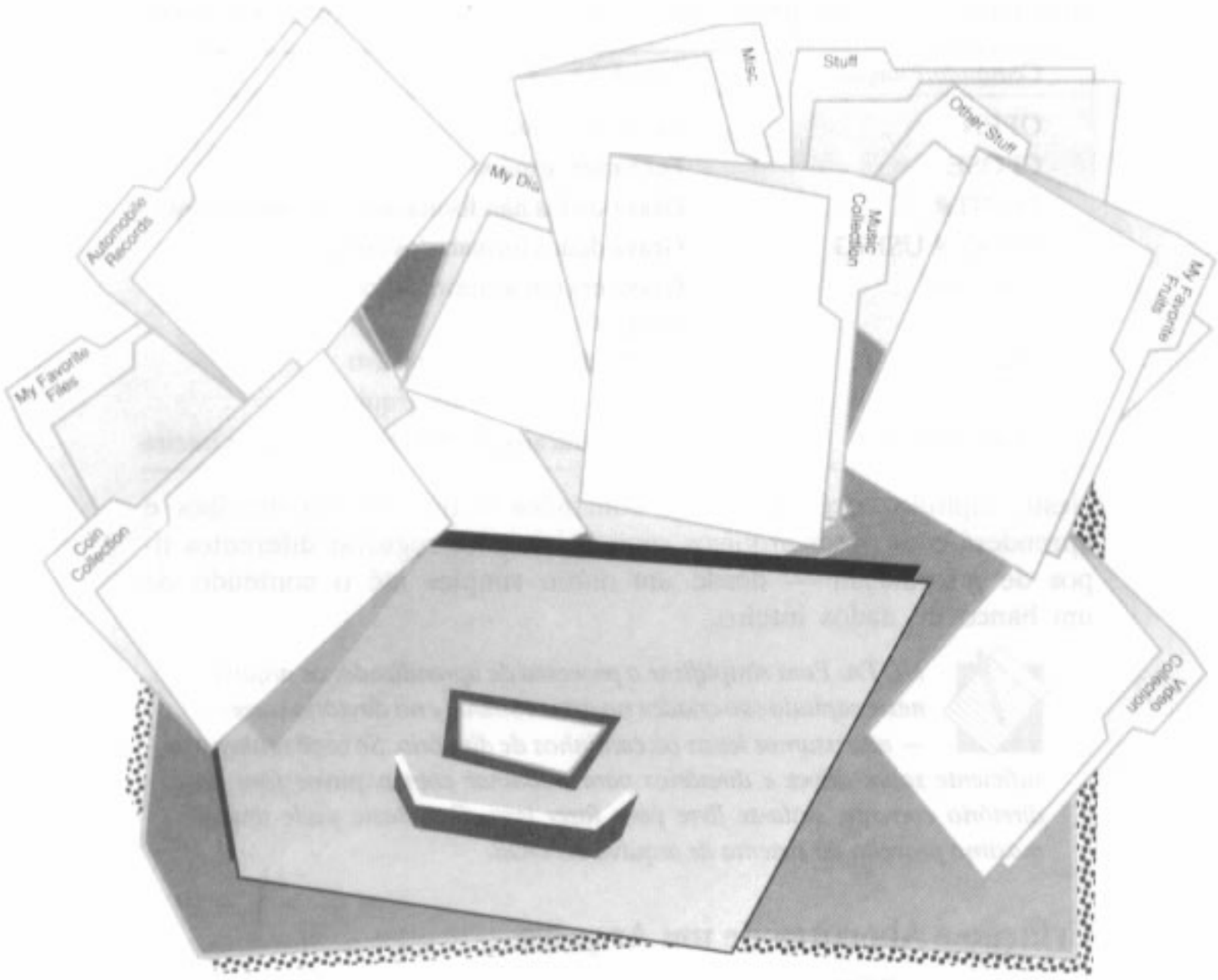
9. Escreva um programa que solicite do usuário uma cadeia de tamanho variável, inverta a ordem dos caracteres da cadeia e mostre a nova cadeia.

10. Escreva um programa que peça ao usuário um nome completo, em três partes, e o coloque numa variável de tamanho variável, copiando cada nome na cadeia para uma variável de cadeia separada. O seu programa deverá processar nomes de qualquer tamanho, desde que o usuário separe os três nomes com espaços. A saída do programa deverá ficar semelhante a esta:

    ```
    Entre com um nome: Rainha Vitória Belfield

    Primeiro nome: Rainha
    Segundo nome: Vitória
    Último nome: Belfield
    ```

CAPÍTULO 10

Uso de Arquivos



Nos Capítulos 8 e 9 você aprendeu a trabalhar com grandes quantidades de dados em um programa. Neste capítulo, você aprenderá a salvar os dados em disco e acessá-los sempre que desejar. Aprenderá também como enviar a saída do programa a uma impressora.

CRIAÇÃO E USO DE ARQUIVOS SEQÜENCIAIS

Um *arquivo* é simplesmente um grupo de dados salvos em disco. Um tipo de arquivo que você pode criar dentro de um programa QBasic é um *arquivo seqüencial*. O conteúdo de um arquivo seqüencial deve ser usado em ordem, do início ao fim. (O conteúdo do outro tipo de arquivo que você pode criar, um *arquivo de acesso randômico*, pode ser usado em qualquer ordem, mas não discutiremos os arquivos de acesso randômico neste livro.) Em seus programas QBasic, você usa estes comandos e funções para criar e trabalhar com arquivos seqüenciais:

Comando/Função	*Descrição*
OPEN	Abre um arquivo
CLOSE	Fecha um arquivo
PRINT#	Grava dados não formatados em um arquivo
PRINT# USING	Grava dados formatados em um arquivo
WRITE#	Grava em um arquivo dados organizados em campos
INPUT#	Obtém dados de um arquivo
EOF	Verifica o final de um arquivo
LINE INPUT#	Pega de um arquivo uma linha de dados inteira

Neste capítulo você verá esses comandos e funções em detalhes e aprenderá como usar arquivos seqüenciais para registrar diferentes tipos de informação — desde um diário simples até o conteúdo de um banco de dados inteiro.

NOTA: Para simplificar o processo de aprendizado, os arquivos neste capítulo são criados no drive corrente e no diretório corrente — não usamos letras ou caminhos de diretório. Se você souber o suficiente sobre drives e diretórios para trabalhar com arquivos fora do diretório corrente, sinta-se livre para fazer isso. O QBasic pode tirar o máximo proveito do sistema de arquivo do DOS.

Criação e Abertura de um Arquivo

O comando OPEN realiza uma dupla tarefa: ele cria novos arquivos e abre arquivos já existentes. A sintaxe do comando OPEN é a seguinte:

OPEN *nome-arquivo* FOR *modo* AS *#número-arquivo*

nome-arquivo é um nome-de-arquivo válido para o DOS, *modo* é uma palavra que indica como o arquivo será usado, e *número-arquivo* é um inteiro entre 1 e 255, que será associado ao arquivo que está sendo aberto.

NOTA: Como você atribui um número ao seu arquivo, poderá se referir a ele dentro do programa simplesmente chamando-o pelo seu número. Você deverá começar a numerar os arquivos dentro dos programas com 1. O número atribuído ao arquivo é válido até que você feche o arquivo.

Especificação de um modo

Você só pode abrir um arquivo em um dentre três modos possíveis de cada vez. Use o argumento *modo* para especificar como pretende usar o arquivo seqüencial. Use um dos argumentos a seguir em um *modo* do comando OPEN.

Modo	*Descrição*
OUTPUT	Cria e abre um arquivo que receberá a saída do programa. Se o arquivo já existir, seu antigo conteúdo será apagado.
APPEND	Abre um arquivo existente ao qual a saída do programa será anexada; ou seja, a saída do programa será colocada no *final* do arquivo. O conteúdo original do arquivo será preservado.
INPUT	Abre um arquivo que só poderá ser lido pelo programa. O programa não poderá alterar o arquivo, mas sim usar o seu conteúdo como entrada.

A seguir, vemos alguns comandos OPEN de exemplo que mostram o uso de cada modo.

O comando a seguir abre o arquivo seqüencial NOMES.TXT, que armazenará a saída do programa. Se NOMES.TXT não existir, o QBasic o criará. Se existir, o QBasic apapará o seu conteúdo. NOMES.TXT está associado ao número 1.

```
OPEN "NOMES.TXT" FOR OUTPUT AS #1
```

O comando a seguir abre um arquivo existente chamado NOMES.TXT, associando-o ao número 1. O programa armazena sua saída ao final do conteúdo existente em NOMES.TXT.

```
OPEN "NOMES.TXT" FOR APPEND AS #1
```

O comando a seguir abre um arquivo existente chamado NOMES.TXT, associando-o ao número 1. O programa pode usar o conteúdo de NOMES.TXT como entrada.

```
OPEN "NOMES.TXT" FOR INPUT AS #1
```

Uso de uma variável de cadeia com OPEN

Ao usar o comando OPEN, você pode até mesmo usar uma variável de cadeia no lugar de *nome-arquivo*. Por exemplo, você poderá pedir ao usuário para fornecer um nome de arquivo e depois atribuir este nome a uma variável e usá-la dentro do comando OPEN, como mostra o exemplo a seguir:

```
INPUT "Que arquivo gostaria de abrir? ", arquivo$
OPEN arquivo$ FOR OUTPUT AS #1
```

Você provavelmente desejará usar algumas habilidades de avaliação de cadeia que aprendeu no Capítulo 9 para que o programa verifique se a resposta do usuário (*arquivo$*) é um nome válido para o DOS antes de tentar abrir o arquivo.

Fechamento de um Arquivo

Quando você acabar de trabalhar com um arquivo, não se esqueça de fechá-lo com o comando CLOSE. Isso fará com que toda a informação gravada no arquivo seja realmente gravada no disco. Depois que um arquivo for fechado, o número associado a ele será liberado, podendo ser atribuído a outro arquivo.

A sintaxe do comando CLOSE é a seguinte:

```
CLOSE [#número-arquivo]
```

número-arquivo é o número associado com o arquivo que você quer fechar. Se o argumento *#número-arquivo* for omitido, *todos* os arquivos abertos serão fechados. Ao fechar um arquivo, você deverá reabri-lo antes que possa usá-lo novamente.

Vejamos alguns exemplos do comando CLOSE. O comando CLOSE a seguir fecha o arquivo 1:

```
CLOSE #1
```

O comando CLOSE a seguir fecha todos os arquivos abertos:

```
CLOSE
```

NOTA: Embora o QBasic feche automaticamente todos os arquivos abertos quando um programa termina, você deverá acostumar-se a fechar todos os arquivos assim que termine de trabalhar com eles. Caso contrário, um evento imprevisto, como uma falta de energia, poderá causar perda de dados.

Armazenagem de Dados em um Arquivo

Após um arquivo ser aberto em modo OUTPUT e APPEND, ele poderá receber dados do seu programa. Três comandos do QBasic podem enviar dados para um arquivo seqüencial aberto:

- O comando PRINT# envia dados *não-formatados* para um arquivo.
- O comando PRINT# USING envia dados *formatados* para um arquivo.
- O comando WRITE# envia para um arquivo dados organizados em *campos*.

As seções a seguir descrevem em detalhes cada um desses comandos de armazenamento de arquivo.

O comando PRINT#

O comando PRINT# é funcionalmente semelhante ao comando PRINT, mas envia dados para um arquivo, e não para a tela. A sintaxe do comando PRINT# é a seguinte:

PRINT $*número-arquivo*, [*lista-expressão*][,|;]

número-arquivo é o número do arquivo aberto, e *lista-expressão* são os dados a serem enviados ao arquivo. Se você omitir a *lista-expressão*, o arquivo receberá uma linha em branco.

Se você colocar mais de um item em *lista-expressão*, deverá separar os itens com ponto-e-vírgulas ou vírgulas. Ponto-e-vírgulas e vírgulas no comando PRINT# funcionam exatamente como no comando PRINT comum. Se o ponto-e-vírgula for o último caracter no comando PRINT#, o próximo item enviado ao arquivo aparecerá ao final da linha entrada anteriormente.

Prática: Armazenando dados não-formatados em um arquivo

O programa DADOSCAR.BAS (Figura 10-1) demonstra como usar o comando PRINT# para enviar três linhas de dados não-formatados para um arquivo chamado DADOSCAR.TXT. Os valores enviados a um arquivo podem ser valores literais, variáveis simples ou os resultados de funções ou expressões.

Digite e execute o programa DADOSCAR.BAS.

```
' DADOSCAR.BAS
' Este programa usa o comando PRINT# para enviar
'  três linhas de informação de carro em um arquivo
'  seqüencial.
```

FIGURA 10-1. (*continua*)
DADOSCAR.BAS: Um programa demonstrando o comando PRINT#.

FIGURA 10-1. *continuação*

```
' abre arquivo no drive/diretório corrente
OPEN "DADOSCAR.TXT" FOR OUTPUT AS #1

CLS

' pega dados de carros e grava no arquivo aberto

INPUT "Entre com a marca de um carro: ", marca$
INPUT "Qual é o nome do modelo? ", modelo$
INPUT "Em que ano ele foi fabricado? ", ano%

' PRINT #1, marca$, modelo$, ano%

' inclui alguns valores literais ao arquivo

PRINT #1, "Mercedes-Benz", "190 SL", 1959

' inclui novo carro ao arquivo (função RIGHT$ retorna
'  o ano corrente a partir da função DATE$)

PRINT #1, "Nissan", "Stanza", " "; RIGHT$(DATE$, 4)

CLOSE #1              ' fecha o arquivo

PRINT
PRINT "Dados foram gravados com sucesso em
DADOSCAR.TXT"
```

Você receberá uma saída semelhante à seguinte:

```
Entre com a marca de um carro: Ford
Qual é o nome do modelo? Del Rey
Em que ano ele foi fabricado? 1985

Dados foram gravados com sucesso em DADOSCAR.TXT
```

Um arquivo chamado DADOSCAR.TXT também foi criado no diretório corrente do disco no drive corrente. Para ver o arquivo, saia do QBasic e digite o seguinte comando do DOS ao aviso de comando do sistema:

```
type dadoscar.txt
```

Você verá uma saída semelhante a esta:

```
Ford          Del Rey      1985
Mercedes-Benz 190 SL       1959
Nissan        Stanza       1991
```

O comando PRINT# USING

O comando PRINT# USING segue as mesmas regras e usa os mesmos caracteres de formatação especiais do comando PRINT USING, mas envia dados formatados a um arquivo e não à tela. PRINT# USING é útil quando você precisa enviar grandes quantidades de dados tabulares a um arquivo. A sintaxe do comando PRINT# USING é a seguinte:

PRINT *#nome-arquivo*, USING *gabarito*; [*lista-expressão*][,|;]

número-arquivo é o número do arquivo aberto, *gabarito* é uma cadeia usada para formatar os valores em *lista-expressão*, e *lista-expressão* são os dados a serem formatados e enviados ao arquivo. (Você pode usar separadores de vírgula ou ponto-e-vírgula para separar os valores em *lista-expressão*.) Os caracteres de formatação em *gabarito* devem combinar um por um com os caracteres dos valores em *lista-expressão*.

Prática: Armazenando dados formatados em um arquivo

O programa FRUTAS.BAS (Figura 10-2) demonstra como usar o comando PRINT# USING para enviar três linhas de dados formatados no arquivo FRUTAS.TXT. O gabarito de formatação neste programa é a variável de cadeia *gab$*, que usa vários caracteres de formatação especiais para alinhar os dados verticalmente antes de enviá-los ao arquivo.

Digite e execute o programa FRUTAS.BAS.

```
' FRUTAS.BAS
' Este programa usa o comando PRINT# USING para
'  enviar três linhas de dados de fruta formatados
'  a um arquivo.

' abre o arquivo no drive/diretório corrente
OPEN "FRUTAS.TXT" FOR OUTPUT AS #1

CLS

' cria gabarito de formatação para PRINT# USING

gab$ = "Fruta: \        \ Caixas: ###   Preço/kg:
$$##.##"

' pega dados de frutas e grava no arquivo aberto
```

FIGURA 10-2. (*continua*)
FRUTAS.BAS: Um programa demonstrando o comando PRINT# USING.

FIGURA 10-2. *continuação*

```
INPUT "Entre com sua fruta favorita: ", fruta$
INPUT "Quantas caixas deseja comprar? ", caixas%
INPUT "Qual é o preço por quilo? $", custo!

PRINT #1, USING gab$; fruta$; caixas%; custo!

' inclui alguns valores literais ao arquivo

PRINT #1, USING gab$; "Morango"; 2; 400

PRINT #1, USING gab$; "Graviola"; 14; 290

CLOSE #1              ' fecha o arquivo

PRINT
PRINT "Dados foram gravados com sucesso em
FRUTAS.TXT"
```

Você verá uma saída semelhante à seguinte:

```
Entre com sua fruta favorita: Pêra
Quantas caixas deseja comprar? 10
Qual é o preço por quilo? 220

Dados foram gravados com sucesso em FRUTAS.TXT
```

Um arquivo chamado FRUTAS.TXT também foi criado no diretório corrente do disco no drive corrente. Para ver o arquivo, saia do QBasic e entre com o seguinte comando do DOS no aviso de comando do sistema:

```
type frutas.txt
```

Você verá uma saída semelhante a esta:

```
Fruta: Pêra          Caixas:  10     Preço/kg: $220.00
Fruta: Morango       Caixas:   2     Preço/kg: $400.00
Fruta: Graviola      Caixas:  14     Preço/kg: $290.00
```

O comando WRITE#

O comando WRITE# não formata sua saída como os comandos PRINT# e PRINT# USING — WRITE# cria um arquivo que será lido por outros programas. (Usaremos este comando mais tarde, quando criarmos e utilizarmos um arquivo de banco de dados.) A informação que WRITE# envia para um arquivo seqüencial é separado por vírgulas, sendo organizado em grupos chamados *campos*.

A sintaxe do comando WRITE# é a seguinte:

WRITE *#número-arquivo*[, *lista-expressão*]

número-arquivo é o número do arquivo aberto, e *lista-expressão* são os dados a serem enviados ao arquivo. Os elementos em *lista-expressão* são separados por vírgulas. Se você omitir *lista-expressão*, uma linha em branco será gravada no arquivo.

A Figura 10-3 mostra um comando WRITE# de exemplo, que envia cinco tipos de valores para um arquivo. Observe como as vírgulas entre os valores separam a saída em campos. Observe também o valor de cadeia rodeado por aspas: se a cadeia tivesse espaços ou vírgulas, um programa não-QBasic reconheceria os espaços ou vírgulas como parte da cadeia. Se o arquivo criado com o comando WRITE# tivesse múltiplas linhas, cada linha seria chamada de *registro*.

Comandos de exemplo:

```
OPEN "TESTE.TXT" FOR OUTPUT AS #1
```

Comando OPEN

```
a$ = "Kimberly"
b% = 25
c& = 100000
d! = 115.5
e# = .0123456789#
```

Variáveis de exemplo

```
WRITE #1, a$, b%, c&, d!, e#
CLOSE #1
```

Comando WRITE#
Comando CLOSE

Saída enviada ao arquivo:

```
"Kimberly", 25, 100000, 115.5, .0123456789
```

Campo 1 Campo 2 Campo 3 Campo 4 Campo 5

FIGURA 10.3.
O comando WRITE# armazena dados em campos e registros de um arquivo.

Prática: Armazenando dados em um arquivo com campos

O programa MOEDAS.BAS (Figura 10-4) demonstra como usar o comando WRITE# para gravar dados de coleção de moedas em um arquivo chamado MOEDAS.TXT. MOEDAS.BAS foi elaborado para permitir que o usuário entre com informações sobre quantas moedas desejar. Para terminar a entrada de dados da coleção, basta o usuário digitar *FIM* quando tiver que entrar com um país.

Digite e execute o programa MOEDAS.BAS.

```
' MOEDAS.BAS
' Este programa usa WRITE# para enviar dados
'  de coleção de moedas para um arquivo em campos.

' abre arquivo no drive/diretório corrente
OPEN "MOEDAS.TXT" FOR OUTPUT AS #1

CLS

PRINT "Este programa guarda dados de coleção de
moedas"
PRINT "em um arquivo de disco chamado MOEDAS.TXT."
PRINT "Entre com os dados e digite FIM para
terminar."
PRINT
DO WHILE (país$ <> "FIM") ' até digitar FIM...

' pega dados da coleção de moedas e grava no arquivo

  INPUT "De que país é a moeda? ", país$
  IF (país <> "FIM") THEN ' FIM, não grava
     INPUT "Qual o valor da moeda? ", valor$
     INPUT "Qual o nome da moeda? ", nome$
     INPUT "Qual o ano da moeda? ", ano%
     ' envia os campos ao arquivo
     WRITE #1, país$, valor$, nome$, ano%
  END IF
  PRINT        ' linhas em branco entre moedas

LOOP

CLOSE #1       ' fecha o arquivo
PRINT "Dados foram gravados com sucesso em
MOEDAS.TXT"
```

FIGURA 10-4.
MOEDAS.BAS: Um programa demonstrando o comando WRITE#.

Você verá uma saída semelhante a esta:

```
Este programa guarda dados de coleção de moedas
em um arquivo de disco chamado MOEDAS.TXT.
Entre com os dados e digite FIM para terminar.

De que país é a moeda? Estados Unidos
Qual o valor da moeda? 10 cents
Qual o nome da moeda? Dime
Qual o ano da moeda? 1980
```

```
De que país é a moeda? Canadá
Qual o valor da moeda? 25 cents
Qual o nome da moeda? Quarter
Qual o ano da moeda? 1960

De que país é a moeda? Hungria
Qual o valor da moeda? 2 forints
Qual o nome da moeda?
Qual o ano da moeda? 1985

De que país é a moeda? Grã-Bretanha
Qual o valor da moeda? 1 libra
Qual o nome da moeda? Libra
Qual o ano da moeda? 1981

De que país é a moeda? FIM

Dados foram gravados com sucesso em MOEDAS.TXT
```

Um arquivo chamado MOEDAS.TXT também foi criado no diretório corrente do disco no drive corrente. Para ver o arquivo, saia do QBasic e digite o seguinte comando do DOS ao aviso de comando do sistema:

```
type moedas.txt
```

Você verá uma saída semelhante a esta:

```
"Estados Unidos","10 cents","Dime",1980
"Canadá","25 cents","Quarter",1960
"Hungria","2 forints","",1985
"Grã-Bretanha","1 libra","Libra",1981
```

O arquivo MOEDAS.TXT contém quatro registros de quatro campos cada um, como mostramos na Figura 10-5:

```
Registro 1    "Estados Unidos","10 cents","Dime",1980
Registro 2    "Canadá","25 cents","Quarter",1960
Registro 3    "Hungria","2 forints","",1985
Registro 4    "Grã-Bretanha","1 libra","Libra",1981
```

FIGURA 10-5.
Cada linha no arquivo MOEDAS.TXT é um registro contendo quatro campos.

Leitura de Dados de um Arquivo

Agora que você já criou três arquivos e usou o comando TYPE do DOS para examiná-los, aprenderá como examinar e usar arquivos de dados de dentro de um programa QBasic. Você examinará dois comandos e uma função que o ajudam a trabalhar com arquivos que foram abertos para entrada:

- O comando INPUT# pega *um ou mais campos* de dados de um arquivo.
- A função EOF determina se o final do arquivo foi alcançado.
- O comando LINE INPUT# pega uma *linha inteira* de dados de um arquivo.

O comando INPUT#

O comando INPUT# é a principal ferramenta que você usa para obter dados de um arquivo seqüencial em QBasic. O comando INPUT# pega a entrada de um arquivo seqüencial de forma semelhante ao comando INPUT apanhando dados do teclado: Os dois comandos atribuem um ou mais itens de dados a variáveis de tipos correspondentes. A sintaxe do comando INPUT# é a seguinte:

INPUT *#número-arquivo, lista-variável*

número-arquivo é o número do arquivo aberto, e *lista-variável* é uma ou mais variáveis a receberem dados do arrquivo. Os itens no arquivo seqüencial podem ser campos de dados criados com o comando WRITE# ou gravados pelo comando PRINT#, pelo comando PRINT# USING ou por qualquer programa que possa criar arquivos de dados.

Prática: Lendo itens de dados de um arquivo seqüencial

O programa AMIGOS.BAS (Figura 10-6) demonstra como usar o comando INPUT# no seu programa em QBasic para obter dados de um arquivo seqüencial. O programa inicialmente abre um arquivo seqüencial no diretório corrente para saída e envia-lhe quatro cadeias com o comando WRITE#. Depois, o programa fecha o arquivo, abre-o novamente para entrada e lê as quatro cadeias de volta ao programa.

Digite e execute o programa AMIGOS.BAS.

```
' AMIGOS.BAS
' Este programa demonstra o comando INPUT#.

CLS

' abre arquivo para saída
OPEN "AMIGOS.TXT" FOR OUTPUT AS #1

PRINT "Entre com nomes de quatro amigos"
PRINT
```

FIGURA 10-6. (*continua*)
AMIGOS.BAS: Um programa demonstrando o comando INPUT#.

FIGURA 10-6. *continuação*

```
FOR i% = 1 TO 4                          ' repete 4 vezes
  INPUT "Nome do amigo: ", pal$          ' cada uma pega o nome
  WRITE #1, pal$                         ' e grava no disco
NEXT i%

CLOSE #1                                 ' fecha o arquivo

OPEN "AMIGOS.TXT" FOR INPUT AS #1  ' reabre o arquivo

PRINT
PRINT "Você entrou com os seguintes nomes:"
PRINT

FOR i% = 1 TO 4                          ' repete 4 vezes
  INPUT #1, pal$                         ' cada uma pega o nome
  PRINT pal$                             ' e mostra na tela
NEXT i%

CLOSE #1                                 ' fecha o arquivo
```

Você verá uma saída semelhante a esta:

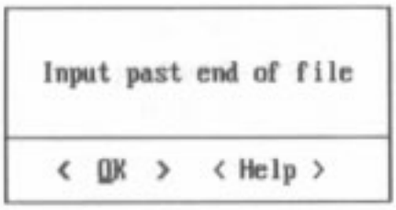
```
Entre com nomes de quatro amigos

Nome do amigo: Lauro
Nome do amigo: Wander
Nome do amigo: Gustavo
Nome do amigo: Gilberto

Você entrou com os seguintes nomes:

Lauro
Wander
Gustavo
Gilberto
```

O arquivo AMIGOS.TXT também foi criado no disco, contendo os seguintes registros de dados:

```
"Lauro"
"Wander"
"Gustavo"
"Gilberto"
```

A função EOF

Se o seu programa em QBasic não puder dizer onde o arquivo seqüencial termina, você poderá receber uma mensagem de erro de fim-de-arquivo. Isso ocorre quando o QBasic tenta ler além do final do arquivo.



Para evitar este erro, você deverá dizer explicitamente ao QBasic para acabar a leitura dos valores após encontrar o final do arquivo. Para isso, use a função EOF, que tem a seguinte sintaxe:

EOF(*número-arquivo*)

número-arquivo é o número do arquivo aberto que você deseja verificar.

EOF retorna o valor lógico *verdadeiro* se o próximo caracter a ser lido estiver além do final do arquivo, e *falso* em caso contrário.

Você pode usar a função EOF em qualquer lugar no programa, para verificar o estado de um arquivo aberto, mas ele é mais eficiente em um laço DO, como mostramos na sessão de prática a seguir.

Prática: Detectando o final de um arquivo

O programa MOEDAS2.BAS (Figura 10-7) é uma melhoria do programa MOEDAS.BAS que já incluímos anteriormente. Ele usa o comando INPUT# para mostrar dados do arquivo e a função EOF para checar o final do arquivo. Esta nova versão abre o arquivo de coleção de moedas no modo APPEND para que a informação já reunida não seja trocada pela nova informação; em vez disso, toda a nova informação é colocada ao final do arquivo.

Digite e execute o programa MOEDAS2.BAS.

```
' MOEDAS2.BAS
' Este programa usa o comando WRITE# para enviar
'  dados de coleção de moedas para um arquivo
'  seqüencial e INPUT# para apresentá-los.

' abre arquivo no modo APPEND para que o
'  conteúdo anterior não seja modificado

' abre arquivo no drive/diretório corrente
OPEN "MOEDAS.TXT" FOR APPEND AS #1

CLS
```

(*continua*)

FIGURA 10-7.
MOEDAS2.BAS: Um programa demonstrando a função EOF e o comando INPUT#.

FIGURA 10-7. *continuação*

```
PRINT "Este programa guarda dados de coleção de
moedas"
PRINT "em um arquivo de disco chamado MOEDAS.TXT."
PRINT "Entre com os dados e digite FIM para
terminar."
PRINT

DO WHILE (país$ <> "FIM") ' até digitar FIM...

' pega dados da coleção de moedas e grava no arquivo

  INPUT "De que país é a moeda? ", país$
  IF (país <> "FIM") THEN ' FIM, não grava
     INPUT "Qual o valor da moeda? ", valor$
     INPUT "Qual o nome da moeda? ", nome$
     INPUT "Qual o ano da moeda? ", ano%
     ' envia os campos ao arquivo
     WRITE #1, país$, valor$, nome$, ano%
  END IF
  PRINT          ' linhas em branco entre moedas

LOOP

CLOSE #1         ' fecha o arquivo

' espera o usuário pressionar Enter para continuar

INPUT "Pressione Enter para ver o conteúdo da sua
coleção de moedas", entra$
CLS              ' começa com uma tela limpa

' abre arquivo no modo INPUT para que o conteúdo
'  possa ser lido pelo programa

' arquivo no drive/diretório corrente
OPEN "MOEDAS.TXT" FOR INPUT AS #1

' mostra cabeçalho para informação tabular dos
' dados da coleção

PRINT "Origem da moeda    Valor da moeda    Nome da
moeda   Ano da moeda:
PRINT "- - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - -"
```

(continua)

FIGURA 10-7. *continuação*

```
PRINT

' inicializa o gabarito de formatação para uso com
'  PRINT USING

gab$ = "\            \   \          \   \          \
####"

' enquanto o fim do arquivo não for alcançado,
'  atribui itens do arquivo a variáveis e imprime

DO WHILE (NOT EOF(1))
  INPUT #1, país$, valor$, nome$, ano%
  PRINT USING gab$; país$", valor$, nome$, ano%
LOOP

CLOSE #1          ' fecha o arquivo
```

A entrada será solicitada da seguinte maneira:

```
Este programa guarda dados de coleção de moedas
em um arquivo de disco chamado MOEDAS.TXT.
Entre com os dados e digite FIM para terminar.

De que país é a moeda? Holanda
Qual o valor da moeda? 2.5 guilders
Qual o nome da moeda? Rijksdaalder
Qual o ano da moeda? 1982

De que país é a moeda? FIM
```

Após ter entrado com a nova informação, ela será mostrada junto com a informação já existente no arquivo:

Origem da moeda	Valor da moeda	Nome da moeda	Ano da moeda
Estados Unidos	10 cents	Dime	1980
Canadá	25 cents	Quarter	1960
Hungria	2 forints		1985
Grã-Bretanha	1 libra	Libra	1981
Holanda	2.5 guilders	Rijksdaalder	1982

O comando LINE INPUT#

O comando INPUT# é um meio eficaz de obter itens individuais dos arquivos, mas o que você faria se precisasse obter longas cadeias de dados textuais? O QBasic oferece o comando LINE INPUT# para esta finalidade. LINE INPUT# é semelhante a LINE INPUT; ele

simplesmente obtém a entrada de um arquivo em vez do teclado. LINE INPUT# tem a seguinte sintaxe:

LINE INPUT *#número-arquivo, nome-variável$*

número-arquivo é o número do arquivo aberto que será lido, e *nome-variável$* é a variável de cadeia que receberá a linha de entrada.

O Comando LPRINT

Quando você quiser enviar a informação à impressora ligada ao seu computador, use o comando LPRINT. LPRINT tem uma sintaxe semelhante à do comando PRINT:

LPRINT [*lista-expressão*][,|;]

lista-expressão é uma lista de expressões numéricas ou de cadeia separadas por vírgulas ou ponto-e-vírgulas. Por exemplo, para enviar a cadeia *Pense na Neve!* à impressora, use este comando LPRINT:

```
LPRINT "Pense na Neve!"
```

Para enviar um comando de *formfeed* (salto de página) à sua impressora quando terminar uma impressão, use o seguinte comando:

```
LPRINT CHR$(12)
```

Isso enviará o caracter ASCII 12 à impressora, a fim de avançar o papel.

Prática: Lendo um diário com LINE INPUT#

O programa DIARIO.BAS (Figura 10-8) demonstra como usar os comandos LINE INPUT e LINE INPUT# juntos para registrar linhas de material textual em um arquivo seqüencial. DIARIO.BAS é aberto inicialmente no modo APPEND e depois marcado com a hora e data correntes do relógio do sistema. O usuário pode, então, incluir uma ou mais linhas ao diário, que se encontra armazenado no arquivo DIARIO.TXT, no diretório corrente. Como o arquivo DIARIO.TXT é aberto no modo APPEND, cada entrada será adicionada ao final do arquivo, preservando as entradas anteriores. Se o usuário solicitar uma listagem, todas as entradas (incluindo suas marcas de hora e data) serão enviadas à impressora por meio de comandos LPRINT.

Digite e execute o programa DIARIO.BAS.

```
' DIARIO.BAS
' Este programa mantém um diário no computador em um
'  arquivo seqüencial chamado DIARIO.TXT

CLS

' abre arquivo no modo append
OPEN "DIARIO.TXT" FOR APPEND AS #1

PRINT "Entre com os pensamentos secretos para hoje;
digite FIM para terminar."
PRINT

PRINT #1, TIME$; " "; DATE$ ' grava hora e data no
arquivo
PRINT #1,                   ' grava linha em branco

DO WHILE (UCASE$(linha$) <> "FIM") ' até digitar FIM,
  LINE INPUT linha$         ' pega linha de texto
  IF (linha$ <> "FIM") THEN PRINT #1, linha$
LOOP          ' e grava no arquivo

PRINT #1,     ' grava linha em branco no arquivo
CLOSE #1      ' fecha arquivo

PRINT         ' verifica se o usuário deseja listagem
INPUT "Gostaria de imprimir o diário inteiro (S/N)?
", resp$

IF (UCASE(resp$) = "S") THEN          ' se sim,
  OPEN "DIARIO.TXT" FOR INPUT AS #1   '  abre arquivo

  LPRINT STRING$(33, "-");    ' imprime cabeçalho
  LPRINT " Meu Diário ";
  LPRINT STRING$(33, "-")
  LPRINT                      ' e uma linha em branco

  DO WHILE (NOT EOF(1))       ' até chegar ao fim
    LINE INPUT#1, linha$      '  lê linhas do arquivo
    LPRINT line$              '  e envia à impressora
  LOOP
  CLOSE #1                    ' fecha arquivo

  LPRINT CHR$(12)             ' salta a página
```

FIGURA 10-8. (*continua*)
DIARIO.BAS: Um programa simples de diário que demonstra o comando LINE INPUT#.

FIGURA 10-8. *continuação*

```
    ' informa que o conteúdo do diário foi impresso
    PRINT
    PRINT "Diário enviado à impressora"
  END IF
```

Quando rodar o programa, este lhe pedirá uma nota do diário da seguinte forma:

Entre com os pensamentos secretos para hoje; digite FIM para terminar.

Após digitar a sua entrada e a palavra *FIM*, verá a seguinte pergunta:

Gostaria de imprimir o diário inteiro (S/N)?

Se entrar com *S*, o programa enviará uma cópia do diário à impressora. Na Figura 10-9 você verá o conteúdo de uma listagem impressa.

O comando VIEW PRINT

O último comando que discutiremos nesta seção é VIEW PRINT. VIEW PRINT não é usado diretamente com arquivos, mas às vezes é usado em combinação com os comandos INPUT#, LOCATE e PRINT para mostrar o conteúdo de um arquivo em diferentes partes da tela. Até aqui, usamos a tela inteira para a saída de comandos como PRINT. Quando as 25 linhas da tela se enchem, comandos PRINT adicionais fazem com que a tela "role". Com o comando VIEW PRINT, no entanto, você pode restringir a saída impressa a apenas parte da tela — chamada *janela de texto* — enquanto o restante permanece intacto. A sintaxe do comando VIEW PRINT é a seguinte:

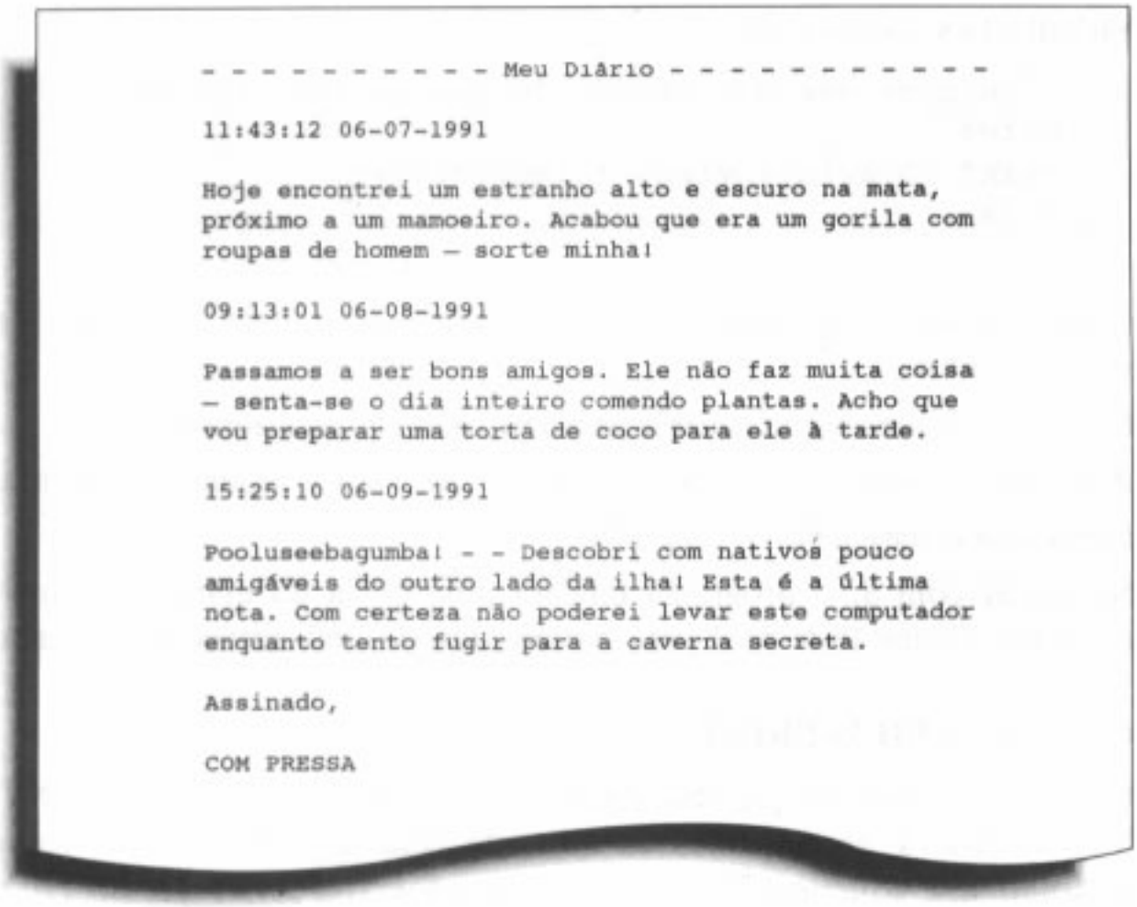
VIEW PRINT [*linha-sup* TO *linha-inf*]

linha-sup é a linha superior da janela de texto, e *linha-inf* é a linha inferior da janela de texto. Se *linha-sup* e *linha-inf* não forem incluídas, a tela inteira tornar-se-á a janela de texto (o default).

Uma vez estabelecida uma janela de texto, o rolamento ocorre apenas entre as linhas superior e inferior da janela. Isso é muito conveniente quando você deseja mostrar uma lista de instruções ou cabeçalhos de tabela e não deseja que saiam da tela. Observe que, quando uma janela estiver em efeito, você poderá usar o comando

CLS 2

para apagar apenas a parte da janela na tela e deixar o restante intacto.



```
- - - - - - - - - - Meu Diário - - - - - - - - - - -

11:43:12 06-07-1991

Hoje encontrei um estranho alto e escuro na mata,
próximo a um mamoeiro. Acabou que era um gorila com
roupas de homem — sorte minha!

09:13:01 06-08-1991

Passamos a ser bons amigos. Ele não faz muita coisa
— senta-se o dia inteiro comendo plantas. Acho que
vou preparar uma torta de coco para ele à tarde.

15:25:10 06-09-1991

Pooluseebagumba! - - Descobri com nativos pouco
amigáveis do outro lado da ilha! Esta é a última
nota. Com certeza não poderei levar este computador
enquanto tento fugir para a caverna secreta.

Assinado,

COM PRESSA
```

FIGURA 10-9.
Uma listagem produzida pelo programa DIARIO.BAS.

Prática: Criando uma janela de texto

O programa VIEW.BAS (Figura 10-10) demonstra a operação dos comandos VIEW PRINT, LOCATE, PRINT e CLS. Veremos esses comandos mais adiante, no programa de banco de dados.

```
' VIEW.BAS
' Este programa demonstra o comando VIEW PRINT.

CLS

PRINT "Bem-vindo ao programa VIEW!"   ' frase inicial
PRINT
PRINT "Vejamos algumas coisas rolando..."
PRINT STRING$(80, "-");       ' mostra linha superior

LOCATE 24, 1: PRINT STRING$(80, "-");' mostra linha
                                    '  inferior
LOCATE 25, 1: INPUT ; "Pressione Enter para iniciar
o programa...", algo$
```

FIGURA 10-10. (*continua*)
VIEW.BAS: Um programa demonstrando o comando VIEW PRINT.

FIGURA 10-10. *continuação*

```
VIEW PRINT 5 TO 23              ' ativa janela (linhas 5-23)

COLOR 2                         ' define cor verde
FOR i% = 1 TO 200
  PRINT "Esta é a linha"; i%    ' lista 200 vezes
NEXT i%
COLOR 7                         ' volta à cor branca

VIEW PRINT                      ' desativa janela
LOCATE 25,1: INPUT ; "Press. Enter para apagar a
janela...", algo$
VIEW PRINT 5 TO 23              ' ativa janela
CLS 2                           ' apaga somente a janela
```

Digite e execute o programa VIEW.BAS. Você verá uma tela com uma janela dividida, como mostra a tela seguinte.

```
Benvindo ao programa VIEW!!

Vejamos algumas coisas rolando...
--------------------------------------------------------------------------------
Esta é a linha 183
Esta é a linha 184
Esta é a linha 185
Esta é a linha 186
Esta é a linha 187
Esta é a linha 188
Esta é a linha 189
Esta é a linha 190
Esta é a linha 191
Esta é a linha 192
Esta é a linha 193
Esta é a linha 194
Esta é a linha 195
Esta é a linha 196
Esta é a linha 197
Esta é a linha 198
Esta é a linha 199
Esta é a linha 200
--------------------------------------------------------------------------------
Press. Enter para apagar a janela...
```

Equivalentes no QBasic aos Comandos do DOS

Muitos comandos do QBasic possuem correspondência no mundo do DOS. Se estiver acostumado com comandos do DOS relacionados a arquivos, poderá achar útil a Tabela 10-1, que lista os comandos do

QBasic, seus comandos equivalentes do DOS e descreve as suas finalidades.

Comando do QBasic	*Equivalente no DOS*	*Descrição*
FILES	DIR /W	Mostra uma listagem de diretório
SHELL	COMMAND	Roda um comando do DOS
NAME	RENAME	Muda o nome de um arquivo
KILL	DEL *ou* ERASE	Apaga um arquivo
MKDIR	MKDIR	Cria um diretório
CHDIR	CHDIR	Muda o diretório corrente
RMDIR	RMDIR	Remove um diretório vazio

TABELA 10-1.
Equivalentes no QBasic aos comandos do DOS.

O comando FILES

Se quiser que seu programa mostre uma lista de arquivos da qual um usuário poderá escolher, use o comando FILES. FILES tem a seguinte sintaxe:

FILES [*nome-arquivo*]

nome-arquivo é o nome opcional do arquivo que você deseja ver na listagem de diretório. Se for omitido, o QBasic mostrará uma listagem de todos os arquivos no diretório corrente. Você pode usar os curingas * e ? como parte do *nome-arquivo*, a fim de mostrar uma faixa de arquivos maior. Para mostrar uma lista para um diretório ou drive diferente do diretório corrente, inclua uma letra de drive e/ou um caminho como parte do *nome-arquivo*.

Prática: Vendo o diretório corrente

O programa MOSTRA.BAS (Figura 10-11) demonstra como usar o comando FILES para mostrar todos os arquivos no diretório corrente cujos nomes terminam com *.TXT*. Após mostrar uma listagem de diretório, MOSTRA.BAS pede ao usuário um nome de arquivo e imprime o arquivo de texto especificado.

Digite e execute o programa MOSTRA.BAS.

```
' MOSTRA.BAS
' Este programa lhe permite ver um arquivo de dados
'  no diretório corrente.

CLS

PRINT "O diretório corrente contém os seguintes";
PRINT "arquivos com extensão TXT:"
PRINT

FILES "*.TXT"
PRINT
INPUT "Que arquivo gostaria de ver? ", arquivo$

OPEN arquivo$ FOR INPUT AS #1
PRINT
PRINT "- - - - - - - - - - -"; UCASE$(arquivo$); "- -
- - - - - - - - -"
PRINT

DO WHILE (NOT EOF(1))
  LINE INPUT #1, linha$
  PRINT linha$
LOOP
```

FIGURA 10-11.
MOSTRA.BAS: Um programa demonstrando o comando FILES.

Quando rodar o programa, você verá uma saída semelhante à seguinte:

```
O diretório corrente contém os seguintes
arquivos com extensão TXT:

C:\DOS
DADOSCAR.TXT FRUTAS .TXT    MOEDAS .TXT   AMIGOS .TXT
DIARIO .TXT
 2080768 Bytes free

Que arquivo gostaria de ver? frutas.txt

---------- FRUTAS.TXT ----------

Fruta: Pêra          Caixas:  10     Preço/kg: $220.00
Fruta: Morango       Caixas:   2     Preço/kg: $400.00
Fruta: Graviola      Caixas:  14     Preço/kg: $290.00
```

O comando Shell

O comando SHELL lhe permite sair temporariamente do QBasic para que possa rodar comandos do DOS. O comando SHELL é útil quando

você deseja executar um comando não disponível no QBasic ou quando deseja dar aos usuários acesso ao DOS enquanto um programa está rodando. (Isso é oferecido por muitos programas atualmente no mercado. No Microsoft Windows, por exemplo, você pode sair temporariamente para o DOS selecionando o ícone DOS Prompt.)

A sintaxe do comando SHELL é a seguinte:

SHELL [*cadeiaComando*]

cadeiaComando é um valor de cadeia opcional ou uma variável de cadeia contendo um comando do DOS válido. Se você incluir *cadeiaComando*, o DOS executará o comando e retornará ao programa em QBasic. Se omitir *cadeiaComando*, o DOS rodará uma nova cópia temporária de si mesmo. Em quase todos os aspectos, esta cópia temporária se apresenta e funciona como a cópia principal do DOS — você pode entrar com qualquer número de comandos. Entretanto, esta cópia do DOS é apenas temporária: quando digitar *exit* e pressionar Enter, o programa do QBasic continuará executando.

Prática: Rodando um comando do DOS

O programa RODA_DOS.BAS (Figura 10-12) demonstra como usar o comando SHELL para rodar comandos do DOS de dentro de um programa em QBasic. Para ajudá-lo a identificar o que é mostrado pelo programa e pelo DOS, o programa começa definindo a cor de segundo plano em ciano. Isso assegura que os resultados dos comandos em QBasic aparecerão em ciano e que os resultados dos comandos do DOS aparecerão em branco, o default do sistema.

1. Digite e execute o programa RODA_DOS.BAS.

```
' RODA_DOS.BAS
' Este programa executa um comando do DOS.

CLS
COLOR 3                 ' define cor ciano
                        ' pega linha de comando do DOS
INPUT "Entre com linha de comando do DOS [pressione
Enter para rodar o DOS]: ", comDOS$
PRINT
SHELL comDOS$           ' sai para o DOS e executa comando

PRINT
PRINT "Comando do DOS completo"
COLOR 7                 ' define cor para branco default
```

FIGURA 10-12.
RODA_DOS.BAS: Um programa demonstrando o comando SHELL.

O QBasic mostra o seguinte pedido:

```
Entre com linha de comando do DOS [pressione Enter para rodar o DOS]:
```

2. Para testar como o comando SHELL processa um comando do DOS, digite *chkdsk* e pressione Enter. Após alguns momentos, o DOS mostrará uma informação semelhante a esta:

```
Volume MARCOS        criated 02-19-1990   4:36p
Volume Serial Number is 1810-1496

  21690368  bytes total disk space
     77824  bytes in 6 hidden files
    120832  bytes in 54 directories
  19410944  bytes in 1003 user files
   2080768  bytes available on disk

      2048  bytes in each allocation unit
     10591  total allocation units on disk
      1016  available allocation units on disk

    655360  total bytes memory
    361888  bytes free

DOS command complete
```

O comando SHELL processou o comando CHKDSK do DOS e depois retornou ao programa.

3. Agora rode novamente o programa RODA_DOS, mas desta vez basta pressionar Enter e disparar uma versão temporária do DOS. Você verá algo semelhante à saída seguinte. Observe o modo como usamos nosso tempo no DOS para copiar um arquivo.

```
Entre com linha de comando do DOS [pressione Enter para rodar o DOS]:

Microsoft(R)  MS-DOS(R) Version 5.00
              (C)Copyright Microsoft Corp 1981-1990.

C:\DOS>dir *.txt

 Volume in drive C is MARCOS
 Volume Serial Number is 1810-1496
 Directory of C:\DOS

DADOSCAR  TXT       107    06-06-91   12:40p
FRUTAS    TXT       177    06-06-91    3:56p
MOEDAS    TXT       197    06-06-91    5:34p
AMIGOS    TXT        37    06-07-91   10:19a
DIARIO    TXT       713    06-07-91    3:12p
          5 file(s)      1231 bytes
                         2080768 bytes free
C:\DOS>copy diario.txt a:
       1 File(s) copied
```

```
C:\DOSexit

DOS command complete
```

O comando NAME

O comando NAME é muito útil quando queremos renomear um arquivo ou mover um arquivo de um diretório para outro. A sintaxe do comando NAME é a seguinte:

NAME *nomeAntigo* AS *nomeNovo*

nomeAntigo é o nome de arquivo atual, e *nomeNovo* é o novo nome que o arquivo deverá ter. Você pode usar caminhos de diretório tanto em *nomeAntigo* quanto em *nomeNovo*. Se os dois caminhos forem diferentes, o QBasic moverá o arquivo para o diretório especificado por *novoNome*. Porém, você não poderá mover arquivos entre drives diferentes. Se o comando NAME não puder renomear os arquivos, o programa gerará uma mensagem de erro.

Prática: Renomeando um arquivo

O programa NOVONOME.BAS (Figura 10-13) demonstra como usar o comando NAME para mudar o nome de um arquivo no drive corrente. Observe o uso do comando FILES, que mostra os arquivos no diretório corrente.

Digite e execute o programa NOVONOME.BAS.

```
' NOVONOME.BAS
' Este programa lhe permite renomear um arquivo
'   de dados no diretório corrente.

CLS

PRINT "O diretório corrente contém os seguintes
arquivos:"
PRINT

FILES "*.*"   ' lista arquivos no diretório corrente

PRINT         ' pega nomes antigo e novo
INPUT "Que arquivo gostaria de renomear? ",
nomeAntigo$
INPUT "Qual deverá ser o novo nome? ", nomeNovo$

NAME nomeAntigo$ AS nomeNovo$      ' tenta renomear
```

FIGURA 10-13. (*continua*)
NOVONOME.BAS: Um programa demonstrando o comando NAME.

FIGURA 10-13. *continuação*

```
' se nomeAntigo$ não existir ou nomeNovo$
'  não for válido, o comando gerará um erro
'  de execução; se não, as linhas a seguir serão
'  executadas:

PRINT          ' imprime mensagem de sucesso
PRINT UCASE$(nomeAntigo$); " renomeado com sucesso"
```

Quando rodar o programa, você verá uma saída semelhante a esta:

```
O diretório corrente contém os seguintes arquivos:

C:\DOS
         .<DIR>              ..<DIR>  FRUTAS    .BAS   DADOSCAR .BAS
DADOSCAR .TXT    FRUTAS      .TXT     MOEDAS    .BAS   MOEDAS   .TXT
MOEDAS2  .BAS    AMIGOS      .BAS     AMIGOS    .TXT   DIARIO   .BAS
MOSTRA   .BAS    NOVONOME.BAS         DIARIO    .TXT   RODA-DOS .BAS

    2080013 Bytes free

Que arquivo gostaria de renomear? diario.txt
Qual deverá ser o novo nome? diariant.txt

DIARIO.TXT renomeado com sucesso
```

O comando KILL

O comando KILL é simples e permanente: ele apaga um arquivo do disco. Quando um arquivo é apagado por meio do comando KILL, ele não pode ser recuperado. Os programadores geralmente usam KILL para apagar arquivos temporários durante a execução do programa, mas você também pode usar KILL para uma limpeza geral no disco. A sintaxe do comando KILL é a seguinte:

KILL *nome-arquivo*

nome-arquivo é o nome do arquivo a ser deletado. Você também usa letras de drive, caminhos e os caracteres globais * e ? dentro de um comando KILL.

NOTA: Tenha muito cuidado com o comando KILL — sobretudo quando usar caracteres globais. Você poderá deletar acidentalmente mais arquivos do que pretende. A sessão de prática a seguir mostra algumas das medidas de segurança que você deve elaborar em um programa que use o comando KILL.

Prática: Apagando um arquivo

O programa APAGA.BAS (Figura 10-14) demonstra como usar o comando KILL para apagar um arquivo indesejado. Se o usuário incluir caracteres globais [curingas] no nome de arquivo entrado para apagar, o programa os detectará com as funções INSTR e mostrará uma mensagem de advertência em vermelho, pedindo que o usuário verifique a múltipla eliminação. Se o arquivo especificado para eliminação não existir, o comando KILL gerará uma mensagem de erro em tempo de execução.

Digite e execute o programa APAGA.BAS

```
' APAGA.BAS
' Este programa lhe permite apagar um arquivo no
'  diretório corrente.

CLS

PRINT "O diretório corrente possui os seguintes
arquivos:"
PRINT

FILES "*.*"   ' mostra arquivos no diretório corrente

PRINT         ' pega arquivo a apagar
INPUT "Que arquivo(s) gostaria de apagar? ",
arquivo$

' verifica curingas (? e *) em arquivo$
'  se houver, mostra advertência antes de apagar
'  se não houver, prossegue apagando; se arquivo não
'  existir, KILL mostrará uma mensagem de erro

IF (INSTR(arquivo$, "?")) OR (INSTR(arquivo$, "*"))
THEN
  PRINT
  COLOR 4      ' cor vermelha para o efeito
  PRINT "Perigo: Posso apagar muitos arquivos!"
  COLOR 7      ' retorna ao branco default
  PRINT        ' pede confirmação para apagar
  INPUT "Deseja prosseguir (S,N)? ", resp$
               ' se resp$=S, apaga arquivo
  IF (UCASE$(resp$) = "S") THEN
    KILL arquivo$
    PRINT
```

FIGURA 10-14. *(continua)*

APAGA.BAS: Um programa demonstrando o comando KILL.

FIGURA 10-14. *continuação*

```
      PRINT UCASE$(arquivo$); " apagado do sistema"
    END IF              ' ...se não, sai do programa
  ELSE
    KILL arquivo$       ' se não houver curinga, apaga
    PRINT               '  arquivo
    PRINT UCASE$(arquivo$); " apagado do sistema"
  END IF
```

Você verá uma saída semelhante à seguinte:

```
O diretório corrente possui os seguintes arquivos:

C:\DOS
         .<DIR>          ..<DIR>  FRUTAS   .BAS  DADOSCAR .BAS
DADOSCAR .TXT   FRUTAS   .TXT     MOEDAS   .BAS  MOEDAS   .TXT
MOEDAS2  .BAS   AMIGOS   .BAS     AMIGOS   .TXT  DIARIO   .BAS
MOSTRA   .BAS   NOVONOME.BAS      DIARIANT .TXT  APAGA    .BAS
RODA-DOS .BAS
  2078419 Bytes free

Que arquivo(s) gostaria de apagar? diariant.txt

DIARIANT.TxT apagado do sistema
```

REGISTRO DE UM BANCO DE DADOS COM ARQUIVOS SEQÜENCIAIS

Concluiremos este capítulo com um programa simples de banco de dados, MUSICABD.BAS, que usa muitos dos comandos e funções que já aprendemos para trabalhar com dados armazenados em um arquivo seqüencial. MUSICABD.BAS registra a seguinte informação para cada disco, fita e *compact disc* na sua coleção:

- Título
- Artista
- Ano de lançamento
- Estilo de música (rock, blues, jazz, clássica, etc.)
- Tipo (disco, fita, *compact disc*, etc.)

MUSICABD.BAS também oferece uma série de características em "estilo de banco de dados" para ajudá-lo a monitorar sua coleção. MUSICABD.BAS lhe permite

- Armazenar informações da coleção de música permanentemente em arquivo seqüencial.
- Examinar os registros do banco de dados da coleção um de cada vez na tela.

- Imprimir o banco de dados da coleção de música inteiro.
- Pesquisar registros por títulos ou artista.
- Carregar e examinar outros bancos de dados de coleção de música.
- Sair temporariamente para o DOS para rodar comandos do DOS.

Uma quantidade razoável do código de MUSICABD.BAS foi dedicada à interface do usuário com menus, que lhe dará uma boa idéia de como são criadas as telas típicas dos programas de aplicação para uso geral. MUSICABD.BD é generalizado ao ponto de você facilmente poder alterá-lo a fim de registrar outros tipos de informação em banco de dados. Veremos como fazer isso depois de examinarmos a operação de MUSICABD.BAS.

Registro de Informações com um Banco de Dados

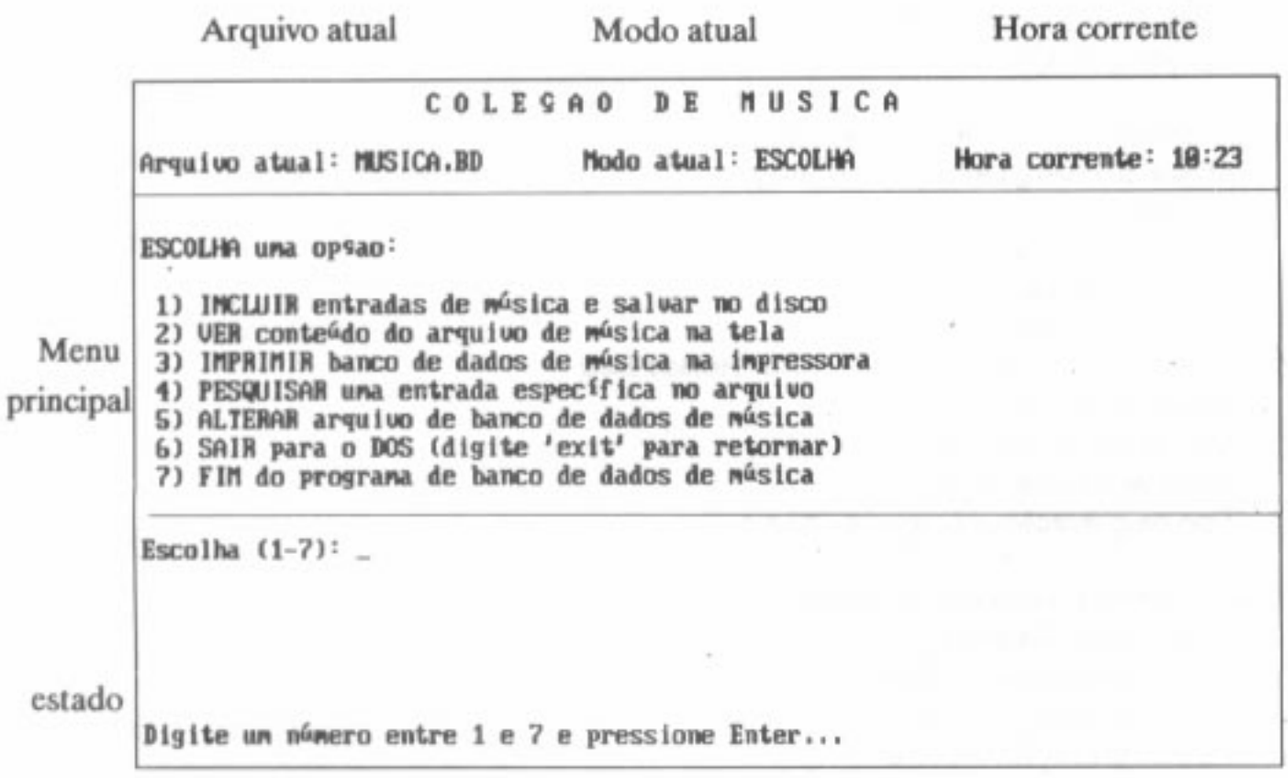
Um *banco de dados* é um conjunto de registros individuais com um formato em comum. Um catálogo telefônico, por exemplo, é um banco de dados: ele contém uma listagem alfabética de registros de nome, cada um contendo os elementos endereço e número de telefone. Um banco de dados sempre possui uma estrutura fixa; ou seja, cada registro contém o mesmo *tipo* de informação, embora não necessariamente o mesmo *conteúdo*.

Os bancos de dados são muito comuns atualmente — agitando a vida doméstica e profissional. Eis alguns exemplos do dia a dia:

- Registros de empregados
- Registros de estudantes
- Produtos disponíveis
- Uma coleção de música
- Uma coleção de moedas
- Uma coleção de filme e vídeo
- Um catálogo de peças
- Um catálogo de ficha de biblioteca
- Datas importantes
- Informação de estoque
- Estatísticas de equipe
- Movimentos financeiros

Uma Primeira Olhada em MUSICABD.BAS

Antes de digitar o programa MUSICABD.BAS, gaste algum tempo vendo o que o programa de coleção de música pode fazer. Quando você iniciar MUSICABD, verá a tela a seguir:



Você verá esses elementos na tela da coleção de música:

- *Arquivo atual* é o nome do arquivo de banco de dados de música atualmente aberto (MUSICA.BD é o default).
- *Modo atual* é a tarefa do menu que está ativa atualmente; este indicador muda à medida que novas tarefas são executadas.
- *Hora corrente* é a hora corrente, lida pelo relógio do sistema (atualizada toda vez que o menu principal é mostrado).
- *Menu principal* contém as sete diferentes tarefas que o programa pode realizar; este é o menu básico do programa, e reaparecerá após a realização de cada tarefa.
- *Linha de estado* contém instruções ou informação sobre a seleção de menu que está ativa atualmente.

Vamos examinar cada opção do menu principal e ver como poderão ser usadas.

A opção INCLUIR

A opção INCLUIR inclui itens musicais no banco de dados. Para selecionar INCLUIR, digite *1* e pressione Enter. MUSICABD mudará

o modo corrente para *INCLUIR* e lhe pedirá itens da coleção de música. O diálogo de entrada ficará mais ou menos assim:

```
Título do item: O Quebra-Nozes
   Artista: Filarmônica de Viena
   Ano de lançamento: 1982
   Estilo de música: Clássico
   Tipo de gravação: Disco

Título do item: Roll Out the Lefse
   Artista: Norwegian Polka Boys
   Ano de lançamento: 1953
   Estilo de música: Polca
   Tipo de gravação: Disco

Título do item: (T)Hanks for the Memories
   Artista: The Hanks
   Ano de lançamento: 1989
   Estilo de música: Rock
   Tipo de gravação: Compact disc

Título do item: Hurried Honeymoon
   Artista: The Magees
   Ano de lançamento: 1988
   Estilo de música: Country
   Tipo de gravação: Disco

Título do item: Der Ring des Nibelungen (Great Scenes)
   Artista: Filarmônica de Viena
   Ano de lançamento: 1985
   Estilo de música: Clássico
   Tipo de gravação: Fita

Título do item: FIM
```

Se você entrar com itens suficientes, a janela do meio rolará, deixando inalteradas as partes superior e inferior da tela (o comando VIEW PRINT em ação). Quando digitar *FIM*, o programa gravará a informação entrada em um arquivo seqüencial no disco e voltará ao menu principal.

A opção VER

A opção VER lhe permite examinar o seu banco de dados com um registro de cada vez. Para selecionar VER, digite *2* e pressione Enter. Continue a pressionar Enter até ver a mensagem *Fim do arquivo*, que indica que todos os registros foram mostrados. Observe que cada registro aparece na tela exatamente como foi entrado. Pressione Enter novamente para retornar ao menu principal.

A opção IMPRIMIR

A opção IMPRIMIR lhe permite imprimir o seu banco de dados. Para selecionar IMPRIMIR, digite *3* e pressione Enter. Se tiver uma impressora ligada ao sistema, verifique se está em linha e pronta, depois entre com *I* para enviar o conteúdo do arquivo corrente à impressora. Se não tiver uma impressora ligada ao sistema, não entre com *I*, ou o programa iniciará uma busca inútil por uma impressora que não existe. Em vez disso, entre com *R* para retornar ao menu principal.

A opção PESQUISAR

A opção PESQUISAR lhe permite procurar um título ou um artista específicos no banco de dados. Para selecionar a opção PESQUISAR, digite *4* e pressione Enter. Quando for solicitada uma categoria, você pode digitar *1* para pesquisar por título ou *2* para pesquisar por artista. A tela a seguir mostra o que acontecerá se você tiver entrado com os discos do nosso exemplo e depois solicitar uma pesquisa por artista para *Filarmônica de Viena*.

Dois álbuns são mostrados, com o item que você pesquisou — neste caso, o nome do artista — destacados em verde. Após terminar a pesquisa, pressione Enter para retornar ao menu principal.

```
                    C O L E Ç Ã O   D E   M U S I C A

Arquivo atual: MUSICA.BD        Modo atual: PESQUISAR      Hora corrente: 10:32

  1) Pesquisar por título
  2) Pesquisar por artista

Categoria (1-2) : 2

Entre com a cadeia a pesquisar: Filarmônica de Viena

Resultados da pesquisa:

Título:  O Quebra-Nozes
Artista: Filarmônica de Viena
Ano:     1982        Estilo:  Clássico  Tipo:  Disco

Título:  Der Ring des Nibelungen (Great Scenes)
Artista: Filarmônica de Viena
Ano:     1985        Estilo:  Clássico  Tipo:  Fita

Press. Enter para voltar ao menu...
```

A opção ALTERAR

A opção ALTERAR lhe permite mudar o nome do arquivo com que está trabalhando para que um outro arquivo de banco de dados de

música (no mesmo formato) possa ser examinado ou criado. Para selecionar a opção ALTERAR, digite *5* e pressione Enter. Para relembrar os arquivos que já possui, ALTERAR mostra o conteúdo do diretório atual antes de pedir um novo nome de arquivo. Você pode incluir um drive e um caminho no seu novo nome de arquivo, mas se você especificar um arquivo inválido o seu programa terminará. Se pressionar Enter sem especificar um novo nome de arquivo, o arquivo default do banco de dados, MUSICA.BD, será selecionado.

NOTA: Especifique apenas arquivos de banco de dados (novos ou existentes) com a opção ALTERAR. Não especifique arquivos de documento ou programa de outras aplicações (como aqueles terminados com .BAS, .BAT ou .DOC) — você poderá danificá-los.

A opção SAIR

A opção SAIR lhe permite sair para o DOS temporariamente, a fim de executar comandos do DOS ou rodar pequenos programas. (Você não poderá rodar grandes programas porque o QBasic e o programa MUSICABD ainda estarão na memória.) Para selecionar a opção SAIR, digite *6* e pressione Enter. Após ter digitado alguns comandos do DOS, digite *exit* e pressione Enter para retornar ao menu principal de MUSICABD.

A opção FIM

A opção FIM serve para sair do programa e retornar ao QBasic. Para selecionar a opção FIM, digite *7* e pressione Enter.

A Listagem do Programa MUSICABD.BAS

Agora que você já viu como o programa funciona, veja na listagem o que já ensinamos neste e em outros capítulos. MUSICABD.BAS (Figura 10-15) consiste em um programa principal e sete subprogramas. Embora cada subprograma cuide de uma tarefa específica, cada um foi elaborado como uma rotina de uso geral, para que você possa adaptá-la mais tarde a qualquer projeto de banco de dados com interesse especial.

Destacamos alguns itens na listagem completa para indicar onde você deverá modificar o programa se a qualquer momento quiser adaptar o banco de dados para registrar um tipo diferente de informação. Gaste algum tempo examinando a listagem e lendo os comentários explicativos. Enquanto isso, observe quais partes de cada subprograma são independentes do tipo de dado sendo usado: escolha de um item pelo menu principal; processamento da seleção de item do menu; abertura e fechamento de arquivos; e leitura, impressão e

pesquisa de itens. A alteração do programa para que se adapte a outro tipo de banco de dados — coleção de moedas, registro de empregados ou estoque — significa simplesmente remover os campos e mensagens específicas à música e incluir campos e mensagens apropriadas ao novo banco de dados. A fachada muda, mas a estrutura interna permanece inalterada.

Prática: Digitando o programa de coleção de música

Digite e execute o programa MUSICABD.BAS. Observe que a Figura 10-15 mostra como MUSICABD.BAS é listado quando impresso — e não como se parece dentro do QBasic. Para digitar este programa, você precisa usar o comando New SUB do menu Edit para entrar com cada subprograma em uma janela separada. Para maiores informações sobre este processo, leia o Capítulo 7.

```
' MUSICABD.BAS
' Este é um programa simples de banco de dados, que
'  registra uma coleção de música com um arquivo
'  seqüencial e vários subprogramas de uso geral.

DECLARE SUB MostraCab () ' declara subprogramas
DECLARE SUB PegaEscolha (escolha%)
DECLARE SUB IncluiReg ()
DECLARE SUB VeReg ()
DECLARE SUB ImprimeReg ()
DECLARE SUB Pesquisa ()
DECLARE SUB MudaArquivo ()

COMMON SHARED arquivo$, gab$ ' declara variáveis
globais
arquivo$ = "MUSICA.BD"    ' arquivo default
' assegura existência do arquivo:
OPEN arquivo$ FOR APPEND AS #1: CLOSE #1
gab$ = "Ano: ####     Estilo: \     \  Tipo: \     \"

MostraCab                 ' chama sub para preparar tela

DO
  PegaEscolha escolha% ' chama sub para escolha
```

FIGURA 10-15. *(continua)*

MUSICABD.BAS: Programa de banco de dados para uso geral usando um arquivo seqüencial para registrar coleções de música. Modifique os itens destacados para adaptar este programa a outro tipo de dados.

FIGURA 10-15. *continuação*

```
  SELECT CASE escolha% ' processa escolha
    CASE 1             ' "1" significa incluir
     LOCATE 3, 47: PRINT "INCLUIR  " ' muda modo para
                                     ' INCLUIR
     IncluiReg         ' chama sub-rotina
    CASE 2             ' "2" significa ver
     LOCATE 3, 47: PRINT "VER      " ' muda para VER
     VeReg             ' chama sub-rotina
    CASE 3             ' "3" significa imprimir
     LOCATE 3, 47: PRINT "IMPRIMIR " ' muda modo para
                                     ' IMPRIMIR
     ImprimeReg        ' chama sub-rotina
    CASE 4             ' "4" significa pesquisar
     LOCATE 3, 47: PRINT "PESQUISAR" ' muda modo para
                                     '  PESQUISAR
     Pesquisa          ' chama sub-rotina
    CASE 5             ' "5" significa alterar
     LOCATE 3, 47: PRINT "ALTERAR  " ' muda modo para
                                     ' ALTERAR
     MudaArquivo       ' chama sub-rotina
    CASE 6             ' "6" significa DOS
     CLS               ' apaga a tela
     SHELL             ' sai para shell do DOS
     MostraCab         ' prepara tela no retorno
    CASE 7             ' "7" significa terminar
     LOCATE 3, 47: PRINT "FIM      "  ' muda modo para
                                     '  FIM
  END SELECT

LOOP UNTIL (escolha% = 7)' repete até escolher FIM

END

SUB IncluiReg

' O subprograma IncluReg inclui novos itens de
' música ao banco de dados.

LOCATE 25,1            ' mensagem na linha de estado
PRINT "Entre com dados da música. Digite FIM em
título para terminar...";
VIEW PRINT 5 TO 23     ' ativa janela (linhas 5-23)
PRINT                  ' pede dados
```

(continua)

FIGURA 10-15. *continuação*

```
PRINT "Entre com nova informação de item musical
(sem vírgulas)"
PRINT

OPEN arquivo$ FOR APPEND AS #1  ' abre em modo append

' pega registros para arquivo até entrada do título
'  FIM

WHILE (UCASE$(titulo$) <> "FIM")
  INPUT "Título do item:" ", titulo$  ' pega título
  IF (UCASE$(titulo$) <> "FIM") THEN
     INPUT "    Artista: ", artista$   ' ... e outros
     INPUT "    Ano de lançamento: ", ano%
     INPUT "    Estilo de música: ", estilo$
     INPUT "    Tipo de gravação: ", tipo$
     PRINT
          ' grava registro no banco de dados
     WRITE #1, titulo$, artista$, ano%, estilo$, tipo$
  END IF
WEND

CLOSE #1       ' fecha arquivo quando terminar

END SUB

SUB MudaArquivo

' O subprograma MudaArquivo muda o nome do arquivo
'  de banco de dados atual. Se o novo arquivo não
'  existir, será criado. Se não for indicado, o
'  arquivo default MUSICA.BD será usado.
'  Nota: Este subprograma faz uma pequena
'  checagem de um nome de arquivo válido —
'  se for entrado um nome não-válido,
'  o programa terminará.

LOCATE 25, 1: PRINT "Especifique o novo arquivo de
música...";
VIEW PRINT 5 TO 23          ' mostra mensagem
                            ' na linha de estado

PRINT
```

(continua)

FIGURA 10-15. *continuação*

```
PRINT                ' pede um novo nome de arquivo
PRINT "Use esta opção para criar um novo arquivo de";
PRINT "banco de dados ou abrir um já existente."
PRINT
PRINT "O diretório atual contém os seguintes
arquivos:"
PRINT
FILES "*.*"          ' mostra todos os arquivos no
PRINT                '  diretório, para ajudar o usuário
PRINT "Que arquivo de dados de músicas gostaria de
usar?"
PRINT "(Pressione Enter para usar o arquivo default
MUSICA.BD)"
PRINT
INPUT "Arquivo: ", arquivo$ ' entra variável global

IF (arquivo$ = "") THEN   ' se nenhum arquivo entrado
  arquivo$ = "MUSICA.BD"  '  usa arquivo MUSICA.BD
ELSE                      ' ou retira espaços do nome
  arquivo$ = LTRIM$(RTRIM$(UCASE$(arquivo$)))
END IF                    '  e passa o nome para
                          '  letras maiúsculas

' abre e fecha arquivo para evitar um erro no modo
' INPUT se o arquivo não existir
OPEN arquivo$ for APPEND AS #1
CLOSE #1

END SUB

SUB MostraCab

' o subprograma MostraCab mostra dados de estado
'  nas três primeiras linhas da tela e as duas
'  linhas divisórias que separam a janela de
'  informação do programa.

CLS                  ' apaga a tela

COLOR 9              ' cor azul clara

PRINT "          C O L E Ç Ã O   D E   M Ú S I C A"
PRINT
PRINT "Arquivo atual:    "; ' mostra campos de estado
```

(continua)

FIGURA 10-15. *continuação*

```
PRINT "Modo atual:        ";
PRINT "Hora corrente:"

PRINT STRING$(80, "-")    ' mostra linhas divisórias
LOCATE 24, 1: PRINT STRING$(80, "-");' nas linhas 4
e 24

COLOR 7           ' define cor em branco default

END SUB

SUB PegaEscolha (escolha%)

' O subprograma PegaEscolha pega escolha de menu do
'  usuário e a retorna ao programa na variável
'  escolha%. O comando VIEW PRINT é usado para
'  ativar e desativar a área da janela (linhas 5-23).
'  A informação mostrada aqui não muda os dados
'  nas linhas de 1 a 4 e de 24 a 25.

escolha% = 0        ' inicializa escolha% em zero

VIEW PRINT' desativa janela para atualizar linhas
          '  3-25
LOCATE 3, 16: PRINT "          ": LOCATE 3,16: PRINT
arquivo$
LOCATE 3, 47: PRINT "ESCOLHA"   ' define modo escolha
LOCATE 3, 76: PRINT LEFT$(TIME$, 5)   ' atualiza hora
LOCATE 25, 1: PRINT "Digite um número entre 1 e 7 e
Enter...";
VIEW PRINT 5 TO 23 ' ativa janela (linhas 5-23)
CLS 2         ' apaga janela para listar escolhas

PRINT         ' pede escolha ao usuário
PRINT "ESCOLHA uma opção:"
PRINT
PRINT " 1) INCLUIR entradas de música e salvar no
disco"
PRINT " 2) VER conteúdo do arquivo de música na
tela"
PRINT " 3) IMPRIMIR banco de dados de música na
impressora"
PRINT " 4) PESQUISAR uma entrada específica no
arquivo"
```

(*continua*)

FIGURA 10-15. *continuação*

```
PRINT " 5) ALTERAR arquivo de banco de dados de
música"
PRINT " 6) SAIR para o DOS (digite 'exit' para
retornar)"
PRINT " 7) FIM do programa de banco de dados de
música"
PRINT
       ' a escolha deve ser um inteiro de 1 a 7
DO WHILE (escolha% < 1) OR (escolha% > 7)
  INPUT "Escolha (1-7): ", escolha%
LOOP

CLS 2      ' apaga janela para receber dados
VIEW PRINT ' desativa janela p/apagar linha de estado
LOCATE 25, 1: PRINT STRING$(80, " ");    ' apaga linha

END SUB

SUB ImprimeReg

' O subprograma ImprimeReg envia o conteúdo inteiro
'  do arquivo de banco de dados atual à impressora.

VIEW PRINT 5 TO 23     ' ativa janela (linhas 5-23)
PRINT             ' mostra mensagem introdutória
PRINT "Esta opção envia o conteúdo de "; arquivo$;
PRINT "à impressora."

VIEW PRINT          ' desativa janela para que a
LOCATE 25, 1        '  linha de estado seja mudada
INPUT ; "Entre com I para imprimir e R para
retornar ao menu: ", resp$
VIEW PRINT 5 TO 23     ' ativa janela (linhas 5-23)
                  ' se usuário quiser imprimir (I ou i)
IF (resp$ = "I") OR (resp$ = "i") THEN
  OPEN arquivo$ FOR INPUT AS #1 ' abre arquivo
                  ' imprime cabeçalho
  LPRINT STRING$(19, "-"); " Coleção de Música ";
STRING$(19, "-")
  LPRINT
  LPRINT "Data:     "; DATE$ ' imprime a data corrente
  LPRINT "Arquivo: "; arquivo$  ' imprime arquivo
  LPRINT
  LPRINT "Conteúdo da coleção:"
  LPRINT
```

(continua)

FIGURA 10-15. *continuação*

```
            ' até terminar o conteúdo do arquivo
  DO WHILE (NOT EOF(1))   ' lê um registro do arquivo
    INPUT #1, titulo$, artista$, ano%, estilo$, tipo$

    LPRINT "Título:  "; titulo$ ' imprime cada campo
    LPRINT "Artista: "; artista$
    LPRINT "Ano:     "; ano%
    LPRINT "Estilo:  "; estilo$
    LPRINT "Tipo:    "; tipo$
    LPRINT
  LOOP

  LPRINT CHR$(12)        ' salta página
  CLOSE #1               ' fecha arquivo
END IF

END SUB

SUB Pesquisa

' O subprograma Pesquisa procura em todo o banco
'  de dados registros combinando com uma cadeia
'  entrada pelo usuário. Atualmente aceita
'  pesquisas por título e artista, mas outros
'  podem ser incluídos com outros comandos CASE.

num% = 0      ' inicializa variável de categoria
achado% = 0   ' inicializa marcador de "reg. achado"

LOCATE 25, 1        ' atualiza linha de estado
PRINT "Entre com categoria de pesquisa e conteúdo...";

VIEW PRINT 5 TO 23     ' ativa janela (linhas 5-23)

PRINT
PRINT "Escolha uma categoria:"  ' pede um tópico de
pesquisa
PRINT
PRINT "  1) Pesquisar por título"
PRINT "  2) Pesquisar por artista"
' pedidos de categorias adicionais entram aqui...
PRINT
DO WHILE (num% < 1) OR (num% > 2)  ' pega número
  INPUT "Categoria (1-2): ", num%  ' associado ao
                                   ' tópico da pesquisa
```

(continua)

FIGURA 10-15. *continuação*

```
LOOP

PRINT                    ' pega cadeia de pesquisa
INPUT "Entre com a cadeia a pesquisar: ",
cadPesquisa$
PRINT
PRINT "Resultados da pesquisa:" ' mostra resultados
PRINT

OPEN arquivo$ FOR INPUT AS #1    ' abre arquivo

DO WHILE (NOT EOF(1))      ' lê registros do arquivo
  INPUT #1, titulo$, artista$, ano%, estilo$, tipo$

  SELECT CASE num%  ' usa num% para comparar campo...
  CASE 1    ' se num%=1, procura no campo título
     IF instr(UCASE$(titulo$), UCASE$(cadPesquisa$))
     THEN
        achado% = -1     ' se achou, muda o indicador
        COLOR 2: PRINT "Título:    "; titulo$: COLOR 7
        PRINT "Artista: "; artista$
        PRINT USING gab$; ano%; estilo$; tipo$
        PRINT ' mostra campos com título em verde
     END IF
  CASE 2    ' se num%=2, procura no campo artista
     IF INSTR(UCASE$(artista$), UCASE$(cadPesquisa$))
     THEN
        achado% = -1     ' se achou, muda o indicador
        PRINT "Título:    "; titulo$
        COLOR 2: PRINT "Artista: "; artista$: COLOR 7
        PRINT USING gab$; ano%; estilo$; tipo$
        PRINT ' mostra campos com artista em verde
     END IF
  ' inclua aqui comandos CASE p/outras categorias...
  END SELECT
LOOP

CLOSE #1                      ' fecha arquivo
IF (NOT achado%) THEN      ' se não achou, mostra
  COLOR 2: PRINT cadPesquisa$;   ' mensagem "não achou"
  COLOR 7: PRINT " não achado arquivo "; arquivo$;
END IF
VIEW PRINT' desativa janela e acerta linha de estado
LOCATE 25, 1: INPUT ; "Pressione Enter para voltar
ao menu...", algo$
```

(continua)

FIGURA 10-15. *continuação*

```
END SUB

SUB VeReg

' O subprograma VeReg mostra na tela cada registro
'  do banco de dados, um por vez.

LOCATE 25, 1          ' atualiza linha de estado
PRINT "Pressione Enter para continuar...";

VIEW PRINT 5 TO 23     ' ativa janela (linhas 5-23)
PRINT                  ' mostra mensagem de abertura
PRINT "Esta opção lhe permite ver sua coleção de
música ";
PRINT "com um registro de cada vez."
PRINT

OPEN arquivo$ FOR INPUT AS #1       ' abre arquivo

DO WHILE (NOT EOF(1))               ' pega registro
                                    '  do arquivo
  INPUT #1, titulo$, artista$, ano%, estilo$, tipo$

  PRINT "Título:   "; titulo$       ' mostra cada campo
  PRINT "Artista:  "; artista$
  PRINT USING gab$; ano%; estilo$; tipo$

  INPUT "", algo$          ' pausa após cada registro
LOOP

CLOSE #1                            ' fecha arquivo
PRINT "** Fim do arquivo **"        ' mostra mensagem de EOF
INPUT "", algo$                     ' pausa antes de retornar
                                    '  ao programa principal
END SUB
```

RESUMO

Neste capítulo, você começou a reunir suas habilidades de forma realmente útil. Você continuou a trabalhar com grandes quantidades de dados, aprendendo a ler e gravar arquivos seqüenciais. Você também aprendeu sobre o papel que o DOS e os comandos do DOS desempenham na programação em QBasic, e praticou o uso de uma série de comandos do DOS de formas úteis. Além disso, trabalhou com um programa de banco de dados de coleção de música que

combina a manipulação de cadeia e técnicas de interface com o usuário, desenvolvidas em capítulos anteriores, com as habilidades de arquivo seqüencial aprendidas neste capítulo. O resultado: um programa de banco de dados para uso geral, que você pode usar para registrar *quaisquer* informações em que esteja interessado.

PERGUNTAS E EXERCÍCIOS

1. Qual é a diferença entre abrir um arquivo para OUTPUT e abrir um arquivo para APPEND?

2. Quais dos seguintes comandos exigem itens de dados entre aspas quando os enviam para um arquivo seqüencial?

   a. INPUT#
   b. PRINT# USING
   c. PRINT#
   d. WRITE#

3. Falso ou Verdadeiro: O nome de arquivo especificado em um comando OPEN deve estar em letras maiúsculas.

4. Quando o comando LINE INPUT# é considerado mais útil do que o comando INPUT#?

5. Onde está o erro no seguinte comando em QBasic?

```
SHELL COPY TESTE.TXT TESTE2.TXT
```

6. O que mostram as seguintes linhas de programa?

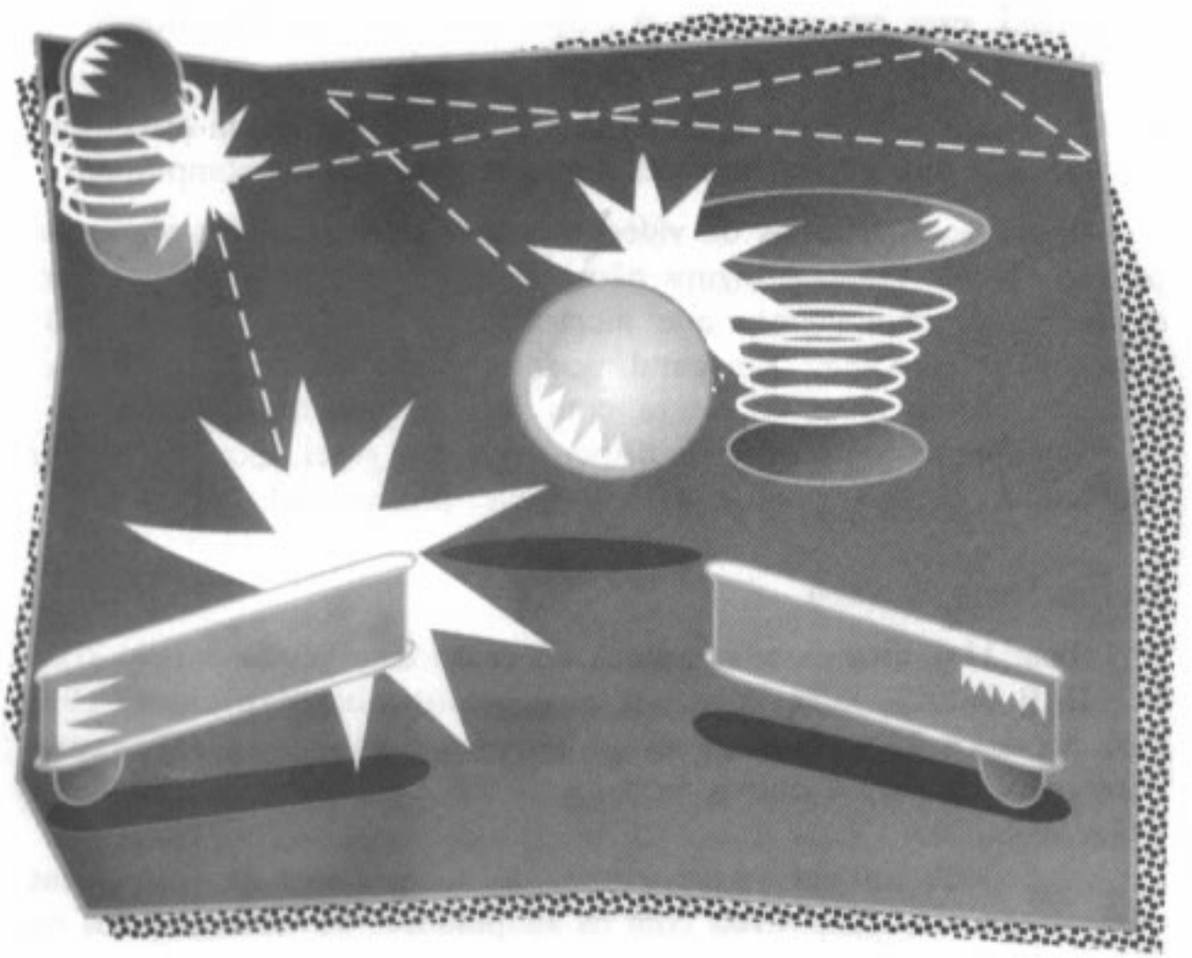
```
cadPesquisa$ = "Bea"
artista$ = "The Beatles"
IF INSTR(UCASE$(artista$), UCASE$(cadPesquisa$)) THEN
        PRINT "Cadeia achada!"
ELSE
        PRINT "Cadeia não foi achada"
END IF
```

7. Escreva um programa que peça ao usuário uma lista de cidades e armazene a informação em um arquivo seqüencial. Elabore o programa de modo que o usuário possa ver o conteúdo do arquivo após ter sido criado.

8. Escreva um programa que peça ao usuário uma lista de nomes e endereços, armazene-os em um arquivo seqüencial e depois ordene os registros alfabeticamente por nome. Dica: O meio mais fácil de resolver este problema é usar um vetor para armazenar os registros e ordená-los. Você poderá usar o Shell Sort, que foi listado no Capítulo 9.

CAPÍTULO 11

Uso de Gráficos e Som



Agora que você já aprendeu os aspectos básicos do QBasic, poderá melhorar seus programas incluindo gráficos e som. Neste capítulo você aprenderá que tipo de gráficos seu computador pode criar, e também verá como usar o som para incrementar seus programas.

INTRODUÇÃO À PROGRAMAÇÃO COM GRÁFICOS

O seu hardware determina o tipo de gráfico que você pode criar, de modo que é importante saber o tipo de equipamento que está instalado no seu computador.

O Que É o Equipamento de Vídeo?

Em um IBM PC ou compatível, o equipamento de vídeo consiste em duas partes:

- O *adaptador de vídeo*, que é uma placa eletrônica instalada dentro da sua máquina. (Nota: Em um computador IBM PS/2 ou compatível, este equipamento já existe embutido no computador — ele não vem em uma placa separada.)
- O *monitor* (às vezes chamado *tela de vídeo*), que é um dispositivo tipo TV que geralmente fica sobre o gabinete do computador.

Alguns equipamentos de vídeo mostram gráficos e cor; alguns só podem mostrar gráficos; alguns não podem mostrar nem gráficos nem cores. E do equipamento que mostra gráficos e cores, você ainda possui várias escolhas. Em geral, quanto menos dispendioso o equipamento, menos cores você terá, com uma imagem mais "granulada". O equipamento de vídeo mais dispendioso, em geral, mostra imagens com muitas cores e uma aparência suave, profissional.

Adaptadores de vídeo

A Tabela 11-1 lista os adaptadores de vídeo disponíveis para o IBM PC, IMB PC/XT, IBM PC/AT e computadores compatíveis. O IBM PS/2 Modelos 25 e 30 possuem um adaptador gráfico MCGA embutido; todos os PS/1 e outros modelos de PS/2 possuem um adaptador VGA embutido.

Você pode instalar várias marcas de adaptadores de vídeo, mas quase todos são compatíveis com os adaptadores de vídeo listados na Tabela 11-1.

Adaptador	*Tela Multicolorida*	*Gráficos*
MDA (adaptador de vídeo monocromático)	Não	Não
HCG (cartão gráfico Hercules)*	Não	Sim
CGA (adaptador gráfico/colorido)	Sim	Sim
EGA (adaptador gráfico melhorado)	Sim	Sim
MCGA (arranjo gráfico multicolorido)	Sim	Sim
VGA (arranjo gráfico de vídeo)	Sim†	Sim

* O cartão gráfico Hercules não é fabricado pela IBM. Este tipo de placa é normalmente chamado *adaptador monográfico ou adaptador gráfico monocromático*.

† Para receber saída multicolorida, você deverá ter um monitor VGA.

TABELA 11-1.
Adaptadores de vídeo para a família IBM PC de computadores e compatíveis.

Se o seu computador possui um adaptador MDA, você não poderá fazer algumas das sessões práticas deste capítulo. Se não tiver certeza quanto ao tipo de adaptador que possui, continue lendo. Logo você digitará e rodará um programa que lhe dirá se pode ou não fazer as sessões práticas deste capítulo.

Monitores

Assim como os adaptadores de vídeo, os monitores vêm numa grande variedade de tipos (lógica e conseqüentemente, com uma grande variedade de preços). Embora sejam permitidas certas misturas e combinações — ou seja, certos adaptadores de vídeo funcionam com vários tipos diferentes de monitores —, a maior parte dos vendedores normalmente combina o adaptador escolhido com o monitor apropriado.

Modo Texto versus Modo Gráfico

Você pode usar qualquer adaptador de vídeo que possa mostrar gráficos em *modo de texto* ou *modo gráfico*.

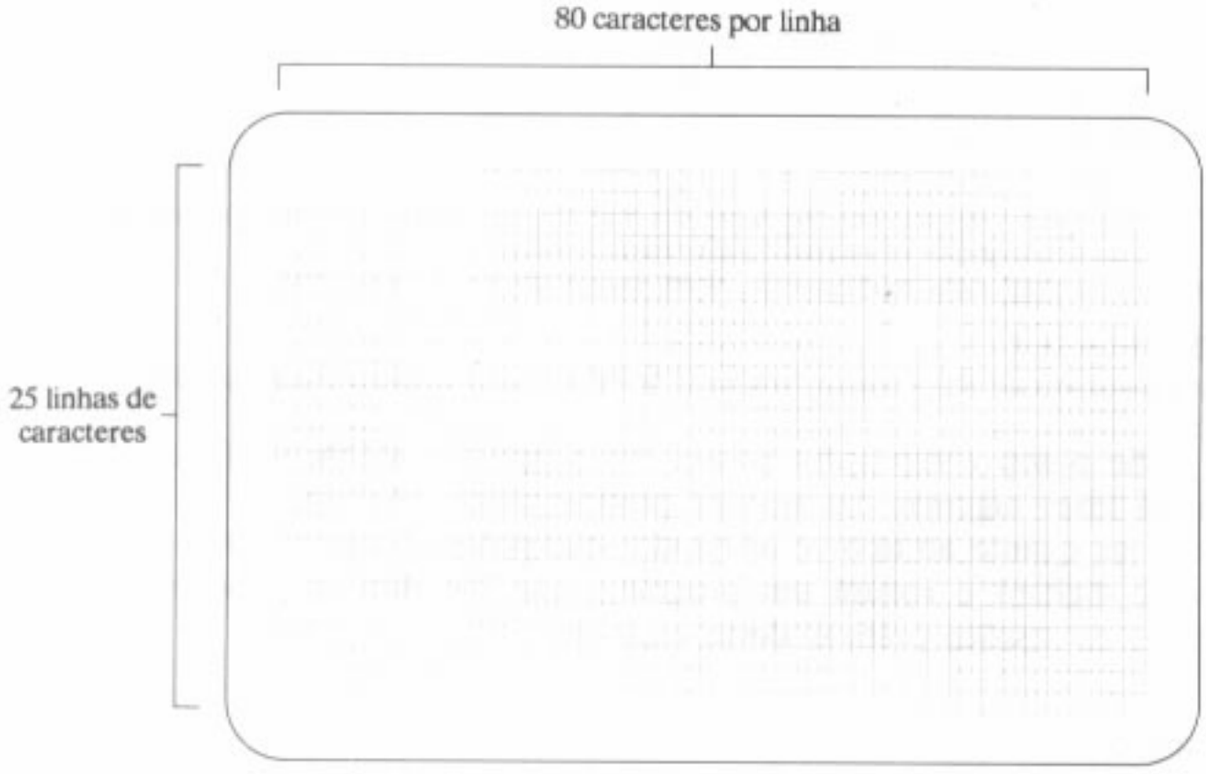
- No modo de texto, o seu adaptador de vídeo pode mostrar caracteres alfanuméricos — letras, números e sinais de pontuação — mas não pode mostrar formas (como linhas, quadros e círculos).

- No modo gráfico, o seu adaptador de vídeo ainda poderá mostrar caracteres alfanuméricos, mas também poderá mostrar formas não-alfanuméricas, como círculos e polígonos.

Vejamos esses dois modos com mais detalhes.

MODO DE TEXTO

Quando você inicia o QBasic, o seu monitor de vídeo mostra a janela View e a tela de saída no *modo de texto*: 25 linhas e 80 caracteres por linha.

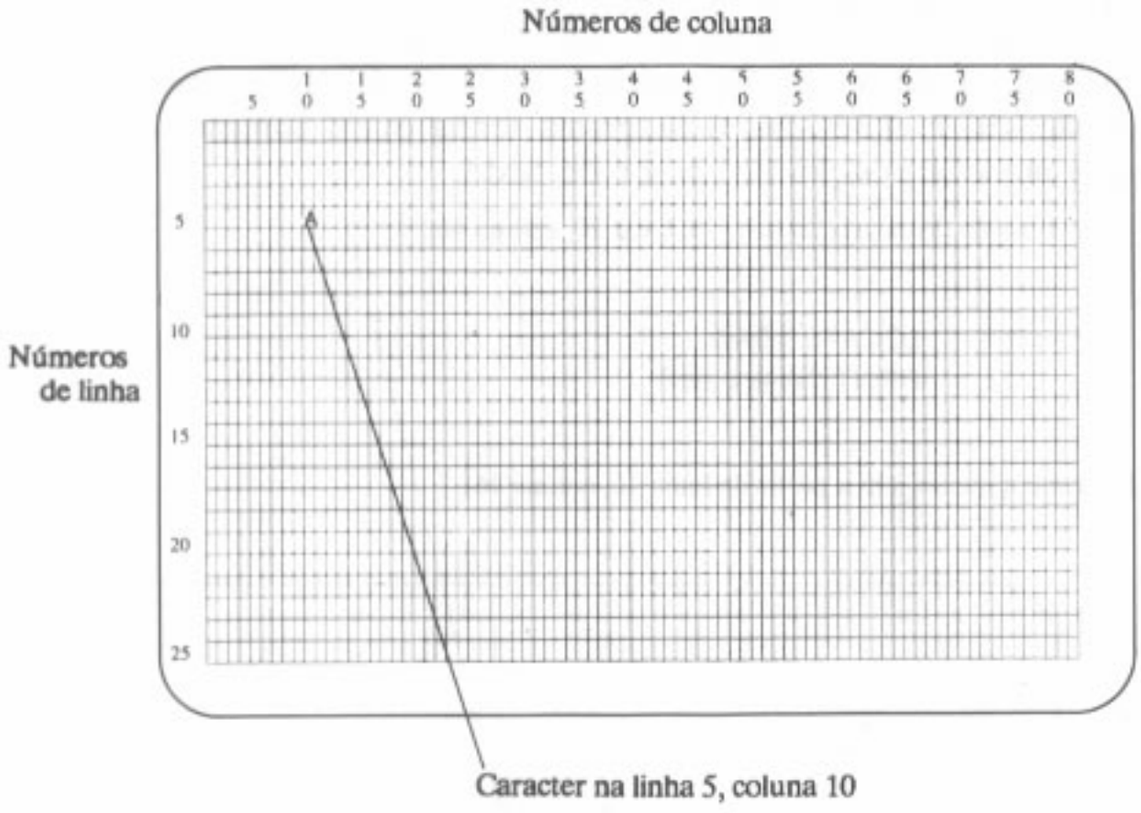


Cada linha e coluna na tela de saída é numerada, dando a cada "caixa" coordenadas únicas de linha e coluna, chamadas *coordenadas da tela*. Cada caixa só pode conter um único caracter alfanumérico.

O Cursor de Texto

Você deverá se lembrar de que, à medida que digita um programa na janela View, o cursor piscante indica onde aparecerá o próximo caracter digitado. Na tela de saída, o QBasic usa seu próprio cursor, chamado *cursor de texto*, para determinar onde os caracteres aparecerão. Diferente do cursor piscante na janela View, o cursor de texto é geralmente invisível.

O comando CLS que apaga a tela de saída serve também a outra finalidade: CLS estabelece um cursor de texto na linha 1, coluna 1 da tela de saída. Se o QBasic executar em seguida um comando PRINT, este mostrará sua mensagem a partir da linha 1, coluna 1. Supondo que o comando PRINT não termine com um ponto-e-vírgula, o QBasic moverá o cursor de texto para baixo, até a coluna 1 da linha seguinte (linha 2, coluna 1).



O Comando LOCATE

Você pode alterar o posicionamento do cursor de texto no QBasic usando o comando LOCATE. LOCATE lhe permite indicar dentro de um programa onde o cursor de texto deverá aparecer.

Veja a sintaxe do comando LOCATE:

LOCATE [*linha*][, *coluna*]

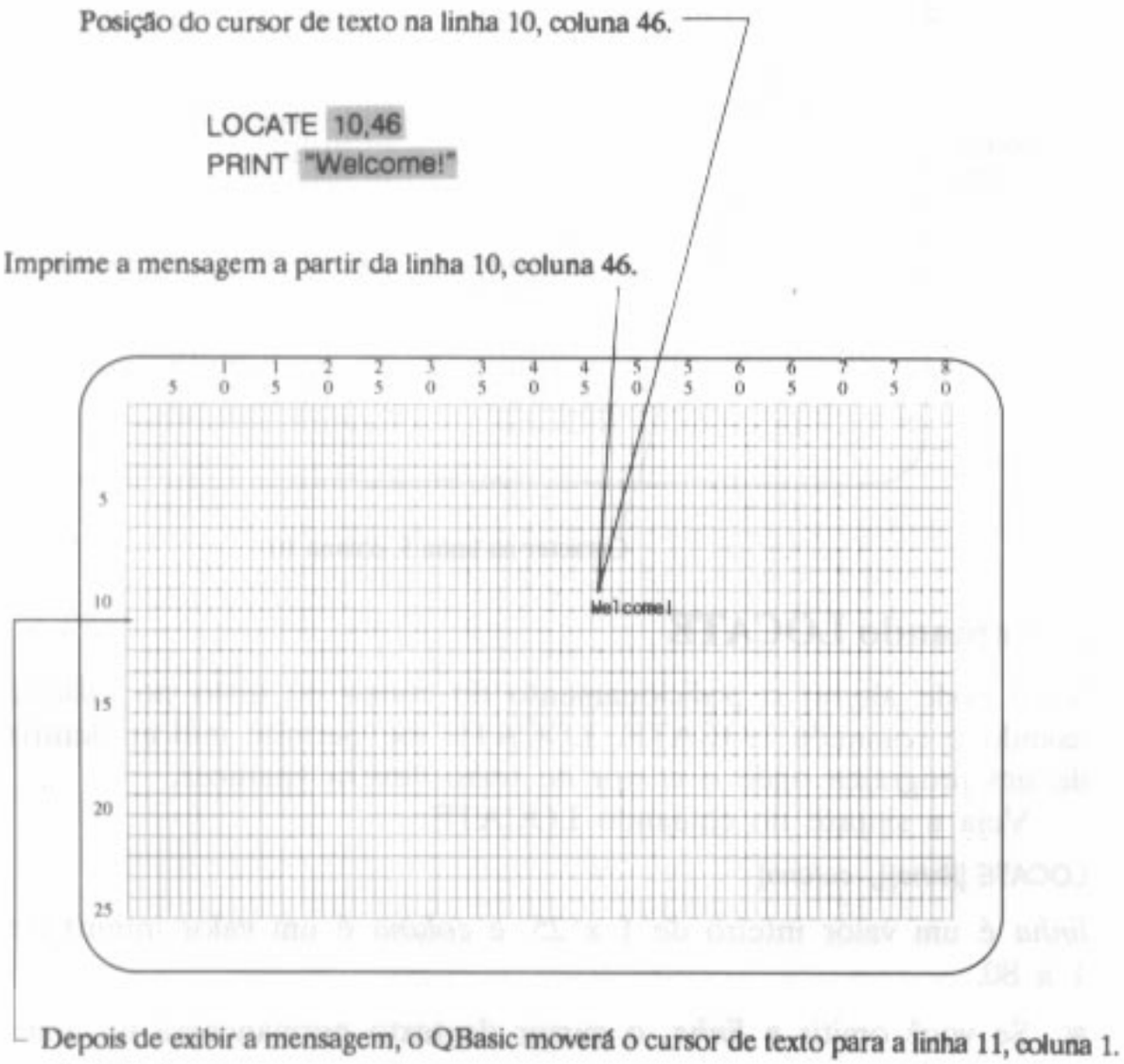
linha é um valor inteiro de 1 a 25, e *coluna* é um valor inteiro de 1 a 80.

- Se você omitir a *linha*, o cursor de texto permanecerá na linha corrente.
- Se omitir a *coluna*, o cursor de texto permanecerá na coluna corrente.
- Se omitir *linha* e *coluna*, o QBasic deixará o cursor de texto em sua posição atual.
- Se especificar um valor de linha ou de coluna fora da faixa de tamanho da tela, verá uma mensagem de erro.

NOTA: Como veremos em breve, você pode instruir o seu adaptador de vídeo a mostrar apenas 40 colunas de texto na tela. Neste caso, a coluna não poderá ter um valor superior a 40.

No exemplo a seguir, o comando LOCATE diz ao QBasic para colocar o cursor na linha 10, coluna 46. Quando o QBasic executa o comando PRINT subseqüente, ele imprime a mensagem *Welcome!*

[Bem-vindo!] a partir da linha 10, coluna 46. Depois de imprimir essa mensagem, o QBasic coloca o cursor de texto na linha 11, coluna 1, exibindo a mensagem do próximo comando PRINT nesse ponto — a não ser, é claro, que o programa tenha outro comando LOCATE que mude a posição do cursor de texto.



Prática: Trabalhando com o comando LOCATE

1. Digite o programa LOCATE-1.BAS (Figura 11-1).

```
' LOCATE-1.BAS
' Este programa demonstra o comando LOCATE.

CLS

INPUT "Favor entrar com a coordenada linha (1-24):
", linha%
PRINT
```

FIGURA 11-1. (*continua*)
LOCATE-1.BAS: Um programa demonstrando o comando LOCATE.

FIGURA 11-1. *continuação*

```
INPUT "Favor entrar com a coordenada coluna (1-80):
", coluna%
PRINT
INPUT "Favor entrar com uma mensagem a exibir: ",
mensagem$

CLS

FOR i% = 0 To 70 STEP 10      ' mostra coordenadas de
  PRINT "1234567890";         '  coluna no alto da tela
NEXT i%

FOR i% = 2 TO 23             ' mostra números de linha
  LOCATE i%, 1               '  no lado esquerdo da tela
  PRINT LTRIM$(STR$(i%))     ' não mostra espaço em
NEXT i%                      '  branco antes do número

LOCATE linha%, coluna%       ' mostra mensagem nas
PRINT mensagem$              '  coordenadas do usuário
```

2. Execute o programa e entre com valores de linha e coluna maiores do que 2 (você verá por que em seguida). Depois de rodar o programa, a sua tela de saída deverá se parecer com a próxima captura de tela.

   Observe que o programa mostra números na horizontal da linha e na vertical das colunas 1 e 2. O programa faz isso para que você possa verificar com facilidade se a mensagem começa nas coordenadas cujos valores você digitou para o comando LOCATE.

A Função STR$

A função STR$ converte um valor numérico para uma cadeia de caracteres numéricos. A sintaxe da função STR$ é a seguinte:

```
STR$(valor)
```

valor é qualquer expressão numérica.

Você pode usar STR$ junto com a função LTRIM$ para livrar-se do espaço que aparece à esquerda dos números não-negativos. Isso é muito prático quando você quer posicionar um número usando o comando LOCATE. Por exemplo, os comandos a seguir mostram o número *12.34* sem o espaço inicial.

```
numero! = 12.34
PRINT LTRIM$(STR$(numero!))
```

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
2
3
4
5                    Olá!
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Press any key to continue
```

Este exemplo usa um valor de linha *5*, um valor de coluna *20*, e a mensagem *Olá!* Execute o programa algumas vezes para que possa entender bem o funcionamento do comando LOCATE.

3. Em seguida, tente executar o programa com um valor de coluna relativamente grande e uma mensagem muito longa. Por exemplo, tente usar o valor de linha *5*, o valor de coluna *70* e a mensagem, *Onde aparecerá tudo isso?* A sua tela de saída deverá se parecer com esta:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
2
3
4
5
Onde aparecerá tudo isso?
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Press any key to continue
```

Observe que o QBasic mostrou a mensagem a partir da linha 6, coluna 1, embora tivesse pedido para mostrar na linha 5. Como a mensagem excedeu o limite da coluna 80 da tela, o QBasic sabia que não poderia colocar a mensagem inteira na linha 5, de modo que começou no início da linha seguinte. Não se esqueça disso ao usar o comando LOCATE, e planeje com cuidado.

Você pode estar curioso sobre uma coisa. O programa pede para entrar com um valor de linha de 1 a 23, também numerando as linhas na tela somente até 23. Entretanto, já dissemos que o valor de linha pode ser de 1 a 25. Por que limitar o programa a 23 linhas?

4. Rode o programa novamente e use o valor de linha *24*. A sua tela de saída deverá ficar como esta:

```
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
        O que acontecerá?
Press any key to continue
```

Após imprimir a mensagem, o QBasic moveu o cursor para baixo, até a primeira coluna da linha seguinte. Ao fazer isso, o QBasic teve que deslocar o conteúdo da tela para cima uma linha, para que a linha seguinte fosse mostrada. Com isso, repare que a linha do alto foi retirada da tela.

Você pode resolver este problema incluindo um ponto-e-vírgula ao final do comando PRINT para manter o cursor na mesma linha. Os comandos a seguir, por exemplo, mostram a mensagem *Esta é a linha 24* na linha 24 sem deslocar o conteúdo da tela:

```
LOCATE 24, 1
PRINT "Esta é a linha 24";
```

Você pode usar esta mesma técnica para colocar um texto na linha 25. (Isso também foi feito no programa MUSICABD.BAS, no capítulo anterior.) Lembre-se, no entanto, de que o QBasic usa a linha 25 para mostrar sua mensagem *Pressione uma tecla para continuar*, de modo que qualquer texto mostrado nessa linha será modificado quando o programa terminar.

Uso de LOCATE para Criar Animação

O comando LOCATE também é um instrumento eficaz para criar uma animação simples. Na tela de um computador, você pode criar um certo efeito de animação apresentando ao usuário a *ilusão* do movimento. Um meio de fazer isso é imprimir um caracter repetidamente na direção em que deseja que ele se mova, e ao mesmo tempo substituir a ocorrência anterior do caracter por um espaço, como mostra a Figura 11-2. Embora o caracter não se mova realmente, o resultado gera a ilusão do movimento.

1. Imprima um caracter.

2. Imprima o mesmo caracter em um local adjacente.

3. Imprima um espaço onde estava o primeiro caracter.

FIGURA 11-2.
Animando um caracter na tela.

Prática: Trabalhando com animação baseada em texto

Digamos que você queira escrever um programa que faça com que a letra X se "mova" pela tela. Os passos a seguir lhe mostrarão como faria isso.

1. Digite o programa LOCATE-2.BAS (Figura 11-3).

```
' LOCATE-2.BAS
' Este programa demonstra o primeiro passo na
'  criação do efeito de animação.

CLS

FOR i% = 1 TO 80
  LOCATE 10, i%
  PRINT "X"
NEXT i%
```

FIGURA 11-3.
LOCATE-2.BAS: O primeiro passo na animação de um caracter.

2. Rode o programa. Quando o fizer, ele mostrará uma linha sólida de Xs pela linha 10 da sua tela. Isso por si só não oferece a ilusão do movimento, mas você deu o primeiro passo — fez com que a letra X se "movesse" pela tela. Como o QBasic executa o programa rapidamente, ele mostrará a linha de Xs quase que instantaneamente. Use as técnicas vistas no Capítulo 6 para inserir um laço no programa, diminuindo a velocidade com que o QBasic imprime os Xs. (O próximo programa inclui um laço de espera, mas tente você mesmo criar um antes prosseguir.)

   Em seguida você precisa criar a ilusão de fazer com que uma *única* letra X se mova pela tela, embora o programa realmente esteja imprimindo 80 Xs separados. Para isso, faça com que o programa imprima um X, imprima um segundo X imediatamente à direita do primeiro e depois apague o primeiro X posicionando um espaço sobre ele.

3. Digite e execute o programa LOCATE-3.BAS (Figura 11-4).

```
' LOCATE-3.BAS
' Este programa demonstra uma animação simples.

CLS
LOCATE 10, 1            ' posiciona o primeiro X
PRINT "X"
FOR i% = 2 TO 80        ' repete o restante
  LOCATE 10, i%
  PRINT "X"             ' imprime X ao lado do primeiro
  LOCATE 10, i% - 1
  PRINT " "             ' "apaga" o X anterior

  FOR j% = 1 TO 300     ' laço de espera
  NEXT j%
NEXT i%
```

FIGURA 11-4.
LOCATE-3.BAS: Um programa que anima um caracter.

Observe que o primeiro X foi impresso antes do laço iniciar. O motivo para isso é o modo como os Xs são apagados. Os valores de LOCATE para impressão de um espaço são *10, i% – 1*. Se o laço começasse com o valor 1, quando o programa tentasse imprimir um espaço pela primeira vez ele tentaria imprimi-lo na linha 10, coluna 0. Como 0 não é um valor válido, o QBasic pararia e mostraria uma mensagem de erro.

4. Rode o programa. Quando o fizer, deverá ver a letra X "movendo-se" pela tela! Se quiser, pare aqui e faça experiências com o programa. A inclusão desse tipo de animação no seu programa pode realmente chamar a atenção do usuário!

Animação vertical

Agora que você aprendeu sobre as bases da animação baseada em texto, vejamos como podemos usar o comando LOCATE para realizar uma animação ainda melhor: criar palavras caindo do céu!

Prática: Trabalhando com CAINDO.BAS

Digite CAINDO.BAS (Figura 11-5) e rode o programa.

```
' CAINDO.BAS
' Este programa demonstra a animação vertical.

CLS

FOR  i% = 1 TO 4

  READ coluna%, palavra$

  FOR linha% = 2 TO 18
     LOCATE linha%, coluna%
     PRINT palavra$
     LOCATE linha% - 1, coluna%
     PRINT SPACE$(LEN(palavra$))
     SOUND (2400 / linha%), 1
  NEXT linha%

NEXT i%

DATA 28, "O", 30, "céu", 34, "está", 39, "caindo!!"
```

FIGURA 11-5.
CAINDO.BAS: Um programa demonstrando a animação vertical.

Em vez de imprimir apenas uma única letra, como no exemplo anterior, este programa imprime palavras inteiras. E para "apagar" a

palavra anterior, este programa usa uma cadeia de espaços para cobrir a palavra inteira ao mesmo tempo.

Além disso, este programa usa o som para reforçar a ilusão de "queda" de cada palavra. Observe como o comando SOUND usa o valor atual de *linha%* como divisor do valor *2400*. Como o valor de *linha%* aumenta, o resultado da operação de divisão diminui, e a tonalidade resultante torna-se mais baixa. (Você aprenderá mais sobre o comando SOUND mais adiante, ainda neste capítulo.)

MODO GRÁFICO

Como já vimos, o modo de texto pode ser um modo interessante e relativamente fácil de se usar na apresentação de informações na tela. Você pode incluir tanto "gráficos" baseados em texto quanto animações simples aos programas sem muito esforço de programação.

A programação baseada em texto tem como ponto fraco as suas limitações. Até mesmo o menor caracter na tela, que é um ponto, ocupa uma "caixa" de caracter inteira, significando que gráficos bem definidos e bem contornados são difíceis de criar, se não impossíveis. Além disso, a animação no modo de texto normalmente parece saltitante, pois o item que você anima precisa saltar pela largura de uma "caixa" de caracter toda vez que você programa "movê-lo".

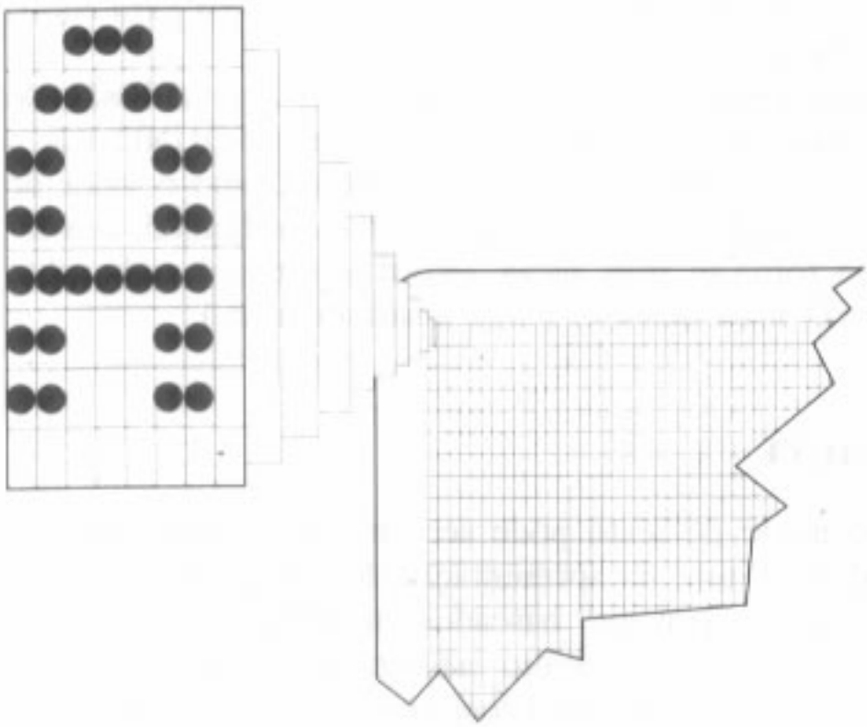
Uma saída com formas de desenho mais nítido e animação fluente é mais bem elaborada se o seu adaptador de vídeo estiver no modo gráfico e se você tirar proveito das ferramentas gráficas que o QBasic tem a oferecer.

O Que É o Modo Gráfico?

Se você olhar atentamente para um dos caracteres na sua tela, verá que, na realidade, ele é composto de pequenos pontos que, em conjunto, formam a letra. Cada uma dessas "caixas" a que nos referimos é, na verdade, uma matriz (uma grade) de 8 pontos por 8 pontos, como vemos na ilustração da página a seguir.

Na matriz do exemplo, a letra A é preenchida para mostrar como uma matriz de pontos exibe um caracter. Embora nem todos os pontos sejam usados, *todos* os pontos na matriz — preenchidos ou não — aparecem na tela quando o seu adaptador de vídeo se encontra em modo de texto. É por isso que uma "caixa" inteira é necessária para cada caracter.

No modo gráfico, as regras são outras: você pode acessar cada ponto individual.



NOTA: Alguns adaptadores de vídeo podem mostrar matrizes com tamanhos diferentes. Por exemplo, alguns podem mostrar matrizes de 8 pontos de largura por 9 de altura; outros podem mostrar matrizes de 9 pontos de largura e 14 pontos de altura, e assim por diante. Vamos considerar uma matriz de 8 por 8 neste estudo.

Resolução gráfica

Em gráficos de computador, a palavra *resolução* é definida como a nitidez de detalhes em uma imagem. De um modo geral, uma imagem em baixa resolução possui uma aparência granulada, enquanto uma imagem em alta resolução possui uma aparência suave, quase fotográfica. Quanto maior a resolução, melhor será a qualidade da imagem. Se você olhar de perto para uma fotografia de jornal, verá que, na realidade, a imagem é composta de milhares de pequeninos pontos. Segure a figura de longe e poderá ver que, coletivamente, esses pontos aparentemente aleatórios formam uma imagem reconhecível.

Uma tela de computador (ou um aparelho de televisão comum, da mesma forma) também mostra pequenos pontos que, em conjunto, formam uma imagem. Quanto menores os pontos — ou seja, quanto maior a resolução —, melhor será a qualidade da imagem.

Na tela de um computador, cada um desses pontos é chamado *pixel*, uma abreviação de *picture element* [elemento de imagem]. Diferente dos pontos em uma fotografia de jornal (que possuem diferentes tamanhos, para criar efeitos de áreas mais claras e mais escuras), todos os pixels na tela do computador possuem o mesmo tamanho.

NOTA: Lembre-se de que o adaptador de vídeo, e não o monitor, determina a resolução do vídeo. O adaptador também é responsável por mostrar os caracteres reais e figuras na tela. Portanto, quando falarmos sobre o hardware e as capacidades gráficas do seu computador, estaremos falando do adaptador de vídeo.

Antes de começar a tirar proveito do modo gráfico, você deverá primeiro passar o seu adaptador de vídeo para o modo gráfico. Em QBasic, isso é feito usando o comando SCREEN.

O Comando SCREEN

O comando SCREEN faz com que o QBasic coloque o seu adaptador de vídeo no modo gráfico quando mostrar sua saída. A forma mais simples do comando SCREEN é a seguinte:

SCREEN *modo*

modo é um número de 0 a 13 (exceto 5 e 6), que diz ao QBasic para que modo gráfico você deseja que ele passe o seu adaptador de vídeo. Embora existam 12 modos gráficos disponíveis, o seu adaptador de vídeo só poderá usar alguns deles. E o seu adaptador de vídeo poderá mostrar apenas um modo de cada vez.

Cada modo é único em termos de resolução e número de cores que aceita: alguns modos mostram 25 linhas e 80 colunas de texto; outros mostram 25 linhas e 40 colunas. E alguns modos mostram mais do que as 25 linhas de texto normais.

A Tabela 11-2 resume os 14 modos de vídeo. As resoluções mostradas representam as resoluções máximas possíveis para os modos em particular. Os valores são dados como *horizontal* X *vertical*, de modo que 320 X 200 significa que a tela possui 320 pixels de largura por 200 de altura.

O número de cores mostrado para cada modo representa o número máximo de cores disponíveis nesse modo. Observe que alguns adaptadores poderiam ser incapazes de mostrar o número máximo de cores para um modo em particular, mesmo que o adaptador seja capaz de apresentar esse modo.

A resolução de texto mostrada para cada modo indica o tamanho em que o texto será mostrado se você usar comandos PRINT no seu programa. Observe que a maior parte dos modos usa 25 linhas; neste ponto, o mais importante é o número de colunas de texto que um modo em particular mostrará.

Modo de vídeo	*Resolução*	*Número de cores*	*Resolução do texto*
0	(Apenas texto)	16	80 x 25
1	320 x 200	4	40 x 25
2	640 x 200	2	80 x 25
3*	720 x 348	1	80 x 25
4†	640 x 400	1	80 x 25
5	Não aceito		
6	Não aceito		
7	320 x 200	16	40 x 25
8	640 x 200	16	80 x 25
9	640 x 350	16	80 x 25
10	640 x 350	4‡	80 x 25
11	640 x 480	2	80 x 30
12	640 x 480	16	80 x 30
13	320 x 200	256	40 x 25

* Somente placas Hercules e compatíveis.

† Somente computadores pessoais Olivetti e AT&T 6300.

‡ O modo 10 mostra diferentes *atributos*, e não diferentes cores. Em outras palavras, as imagens são mostradas em forma normal, intensa ou piscante, e não em diferentes cores.

TABELA 11-2.
Os modos de vídeo disponíveis e suas características.

Prática: Identificando o adaptador no seu computador

Digite e execute o programa TESTEVID.BAS (Figura 11-6). Este programa oferece uma lista de valores de modo que você pode usar com o comando SCREEN. Você poderá marcar ao lado da Tabela 11-2 cada modo aceito pelo seu adaptador de vídeo em particular.

```
' TESTEVID.BAS
' Este programa determina quais os modos de vídeo que
'  o seu adaptador de vídeo aceita.

DIM modosOk%(18)    ' vetor contendo modos em teste

CLS

' diz ao QBasic o que fazer se houver um erro
ON ERROR GOTO NãoDefineModo
```

FIGURA 11-6. (*continua*)
TESTEVID.BAS: Um programa que testa o seu adaptador de vídeo.

FIGURA 11-6. *continuação*

```
FOR i% = 0 TO 13          ' tenta definir o adaptador
  SCREEN i%               '  em todos os 14 modos; se
  IF codErro% = 0 THEN '  houver erro (modo não aceito)
   modosOk%(i%) = 1       '  salta para o fim do programa
   melhorModo% = i%
  END IF
  codErro% = 0
NEXT i%

SCREEN melhorModo% ' passa a tela para o melhor modo

PRINT melhorModo%
FOR i% = 0 TO 13
  IF modosOk%(i%) = 1 THEN
     PRINT "Você pode usar o modo"; i%; "com o seu
computador."
  END IF
NEXT i%

END

NãoDefineModo:   ' rotina "fictícia" que serve apenas
  codErro% = 1   '  (para que o QBasic não pare e
  RESUME NEXT    '  mostre uma mensagem de erro)
```

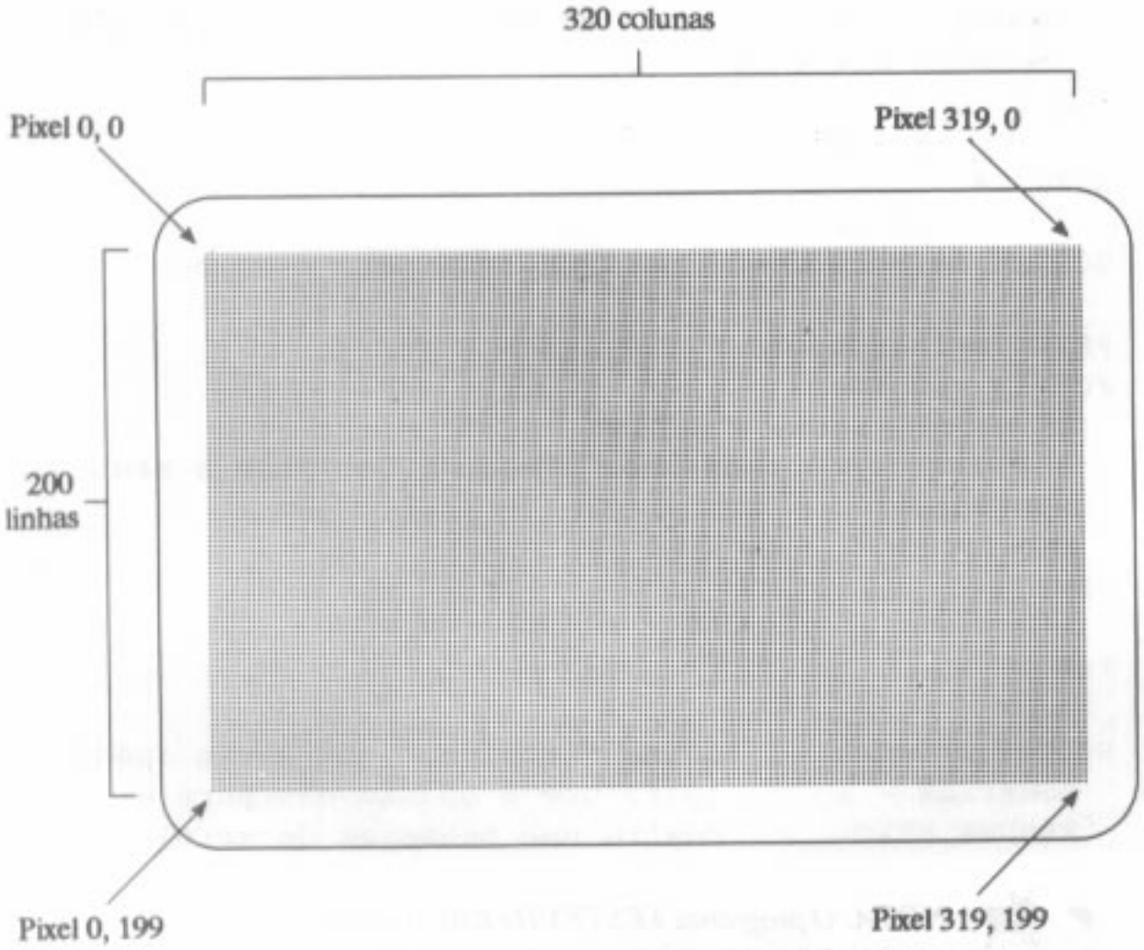
NOTA: O programa TESTEVID.BAS usa dois comandos novos – ON ERROR GOTO e RESUME NEXT – para dizer ao QBasic o que fazer se encontrar um modo de vídeo que o seu adaptador não aceite. Além deste uso simples, ON ERROR GOTO e RESUME NEXT estão além do escopo deste livro. Use a ajuda em linha para descobrir mais sobre esses comandos.

As Coordenadas no Modo Gráfico

Com o comando LOCATE você aprendeu que, especificando coordenadas de linha e coluna, poderia impor o local onde o texto apareceria na tela de saída. No modo gráfico, a tela de saída na realidade possui dois tipos de coordenadas, dependendo do tipo de informação a ser apresentada:

- Coordenadas de texto, que funciona no modo gráfico da mesma forma como no modo de texto
- Coordenadas gráficas, que você usa para colocar informações não-texto (como linhas e círculos) na tela.

Vamos examinar o sistema de coordenadas para o modo de vídeo 1. Como você viu na Tabela 11-2, o modo de vídeo 1 possui uma resolução gráfica de 320 X 200 e uma resolução de texto de 40 X 25, como vemos na tela da ilustração seguinte.



Observe uma diferença muito importante entre o sistema de coordenadas gráficas e o sistema de coordenadas de texto: o pixel superior esquerdo na tela gráfica está nas coordenadas (*0, 0*), enquanto o caracter superior esquerdo na tela de texto está nas coordenadas (*1, 1*).

Uma outra diferença importante está em como você especifica as coordenadas: no modo de texto, você informa as coordenadas como *linha, coluna*; no modo gráfico, você especifica as coordenadas como *coluna, linha*.

Observe também que no modo gráfico, assim como no modo de texto, cada local da tela possui coordenadas próprias.

Uso de Pixels Individuais

Após ter usado o comando SCREEN para colocar o seu adaptador de vídeo no modo gráfico, você poderá usar dois comandos do QBasic para colocar pontos individuais na tela em locais específicos: PSET e PRESET.

O comando PSET

Em sua forma mais simples, a sintaxe de um comando PSET é a seguinte:

PSET (*coordenadax, coordenaday*)[, *cor*]

coordenadax é o número da coluna em que você deseja que um pixel apareça. A faixa de valores que você pode usar depende do modo de vídeo gráfico definido com o comando SCREEN. Por exemplo, se escolher o modo de vídeo 1, o valor de *coordenadax* poderia variar de 0 até 319, pois o número máximo de colunas no modo de vídeo 1 é 320.

coordenaday é o número da linha em que você deseja colocar o pixel. Novamente, a faixa de valores que você pode usar depende do modo de vídeo definido com o comando SCREEN. Por exemplo, se você definiu o modo de vídeo 1, o valor de *coordenaday* variaria de 0 a 199, pois o número máximo de linhas no modo de vídeo 1 é 200.

NOTA: Os parênteses em torno dos valores de coordenadax *e* coordenaday *não são opcionais. Você terá que usar parênteses em torno desses valores no comando PSET.*

cor é um valor representando a cor do pixel. Novamente, a faixa de valores de cor que você pode escolher depende do modo de vídeo gráfico definido com o comando SCREEN. Se for omitida, o QBasic mostrará o pixel na cor atual do primeiro plano.

Um comando PSET só define um pixel de cada vez, mas você pode colocar um comando PSET dentro de um laço para desenhar vários pixels em uma linha.

Para apagar um pixel desenhado com o comando PSET, coloque as coordenadas do pixel que deseja apagar em outro comando PSET e indique a cor de segundo plano com o valor de *cor*.

Prática: Usando o comando PSET

1. Digite o programa PSET.BAS (Figura 11-7).

```
' PSET.BAS
' Este programa demonstra o comando PSET.

CLS

INPUT "Favor entrar com o modo de vídeo (0-13): ",
numModo%
SCREEN numModo%
```

(*continua*)

FIGURA 11-7.
PSET.BAS: Uma demonstração do comando PSET.

FIGURA 11-7. *continuação*

```
LOCATE 22, 23
PRINT "Col: Linha:"

FOR i% = 0 TO 200 STEP 20    ' valores de coordenadax
  FOR j% = 0 TO 135 STEP 15  ' valores de coordenaday
    PSET (i%, j%)   ' usa cor primeiro plano default
    LOCATE 23, 1    ' posiciona cursor de texto
    PRINT "Coordenadas atuais: ("; i%; ","; j%; ")"

    FOR k% = 1 TO 2000    ' laço de espera
    NEXT k%

    PSET (i%, j%), 0  ' apaga com cor segundo plano
    LOCATE 23, 22
    PRINT SPACE$(13)  ' apaga coordenadas impressas
  NEXT j%
NEXT i%
```

2. Rode o programa e entre com um valor de modo da lista de valores que você pode usar no seu adaptador de vídeo em particular (a lista obtida na sessão de prática anterior). O programa, então, começará a "mover" um pixel em saltos regulares pela tela. Depois que o programa colocar cada pixel na tela, ele atualizará a mensagem na base da tela para mostrar suas coordenadas.

   Se você puder usar mais de um modo de vídeo com o seu adaptador, rode o programa novamente e entre com um número de modo diferente. Observe a diferença na resolução — o programa quase cobrirá com pixels uma tela de baixa resolução (como 320 X 200), mas apenas cobrirá o canto superior esquerdo de uma tela de alta resolução (como 640 X 480).

Alteração das cores de primeiro e segundo planos

O Capítulo 6 fez uma pequena introdução ao comando COLOR. Neste ponto, devemos fazer uma pequena revisão.

O comando COLOR tem a seguinte sintaxe:

COLOR [*primeiro-plano*][, *segundo-plano*]

primeiro-plano e *segundo-plano* são valores para as cores de primeiro e segundo planos, respectivamente. A faixa de valores que você pode usar depende inteiramente do modo gráfico em que o seu adaptador se encontra. (Veja a Tabela 11-2.) Se você omitir um valor, o valor corrente será usado. Se omitir *primeiro-plano*, deverá preceder o *segundo-plano* com uma vírgula.

Chamadas de Função Ilegais

Quase sempre você tentará rodar um programa incluindo comandos gráficos ou de som, descobrindo que o QBasic o fará voltar à janela View com o seguinte quadro de diálogo:

Além disso, o QBasic destacará o comando que teve problemas para executar. Por exemplo, se você tivesse tentado rodar o programa anterior com um modo de vídeo 0, o QBasic teria parado o programa, retornado à janela View e mostrado o quadro de diálogo, destacando o primeiro comando PSET. Por quê?

Se você examinar a Tabela 11-2, notará que o modo de vídeo 0 trabalha apenas com texto, o que significa que não podem ser usados comandos gráficos (como PSET) enquanto o adaptador de vídeo estiver nesse modo.

Outras condições podem causar um erro semelhante:

- Tentativa de usar um comando COLOR com valor não aceito pelo modo de vídeo.
- Tentativa de usar um comando LOCATE com valor 0 ou superior ao número disponível de linhas/colunas.
- Tentativa de usar um comando SCREEN com um valor não aceito pelo seu adaptador de vídeo.

Se você encontrar esse quadro de diálogo, reveja a Tabela 11-2, examine a lista de modos de vídeo válidos (obtida pelo programa TESTEVID.BAS) e depois corrija o programa.

O valor de primeiro-plano determina a cor do texto e das formas gráficas. Você pode mudar a cor de primeiro plano a qualquer ponto no programa sem afetar o texto ou formas já exibidas na tela.

O valor de segundo plano especifica a cor do fundo. (Você só pode mudar a cor do fundo nos modos 0, 7, 8, 9 e 10.) Observe que alterar a cor do fundo não terá um efeito imediato. Para mudar a cor do fundo inteiro para a nova cor, você deverá incluir, no seu programa, uma instrução que atualize a cor do fundo. O comando CLS é um candidato excelente, embora deva ser feito cedo no programa,

já que um comando CLS apagará qualquer texto ou gráfico já apresentado na tela.

O comando PRESET

O comando PRESET funciona como o comando PSET, com uma inversão importante: se você omitir o argumento *cor*, o QBasic mostrará o pixel na cor do segundo plano. Isso lhe permite apagar os pixels simplesmente omitindo *cor*. O comando PRESET tem a seguinte sintaxe:

PRESET (*coordenadax, coordenaday*)[, *cor*]

Observe que o comando PRESET apaga qualquer coisa localizada em (*coordenadax, coordenaday*). Por exemplo, se você usar o comando PRINT para imprimir uma mensagem na tela e depois usar um comando PRESET cujas coordenadas estejam na mesma área da mensagem, o comando PRESET pode "pegar um pedaço" de uma letra na mensagem.

Prática: Trabalhando com o comando PRESET

1. Digite o programa PRESET.BAS (Figura 11-8).

```
' PRESET.BAS
' Este programa demonstra o comando PRESET.

CLS

INPUT "Favor entrar um modo de vídeo (0-13): ",
modo%
INPUT "Pressione Enter para começar a chuva...",
algo$

SCREEN modo%
PRINT "Pressione uma tecla para terminar"

DO
  RANDOMIZE TIMER
  ' número coluna aleatório para "chuva"; supõe uma
  ' resolução horizontal de 320;
  ' mudar para outros modos com resolução maior
  randCol%=INT(RND(1)*320)

  FOR i% = 1 TO 199     ' resolução vertical = 200
                        ' mudar para outros modos
    PSET (randCol%, i%)
```

FIGURA 11-8. (*continua*)
PRESET.BAS: Uma demonstração do comando PRESET.

FIGURA 11-8. *continuação*

```
      PRESET (randCol%, i% - 1)

      FOR j% = 1 TO 35   ' laço espera
      NEXT j%

   NEXT i%
LOOP UNTIL INKEY$ <> ""
```

2. Rode o programa. Se o seu adaptador de vídeo permitir, entre com o valor *1*, *7* ou *13* para obter um efeito máximo. Os comentários no programa lhe mostram o que deve ser mudado se quiser que o programa tire proveito dos modos de vídeo com resolução maior.

Quando o programa é rodado, você pode notar um dos efeitos colaterais interessantes de PRESET. Logo antes da "chuva" aparecer, o programa imprime a mensagem *Pressione uma tecla para terminar.* Enquanto o programa continua, você verá os comandos PRESET demolindo esta mensagem à medida que a "chuva" cai sobre ela.

Observe o uso da função INKEY$, introduzido aqui pela primeira vez. INKEY$ verifica o teclado para ver se uma tecla foi pressionada. Se tiver sido, INKEY$ retorna o código ASCII associado com a tecla; se não, retorna uma cadeia vazia. Diferente dos comandos INPUT e LINE INPUT, a função INKEY$ não espera que o usuário pressione Enter antes de continuar.

Posicionamento de Pixels com Coordenadas

Você pode usar dois sistemas de coordenadas para posicionar pixels na tela: *coordenadas absolutas* e *coordenadas relativas.*

Coordenadas absolutas

O sistema de coordenadas que você usou até agora é o sistema de coordenadas absolutas. Nele, todos os valores de coordenada são baseados na coordenada do canto superior esquerdo, *(0, 0)*. Para posicionar um pixel na tela, você conta (ou, mais provavelmente, adivinha) o número de colunas à direita e o número de linhas para baixo, a fim de gerar os valores de coordenada da posição do pixel.

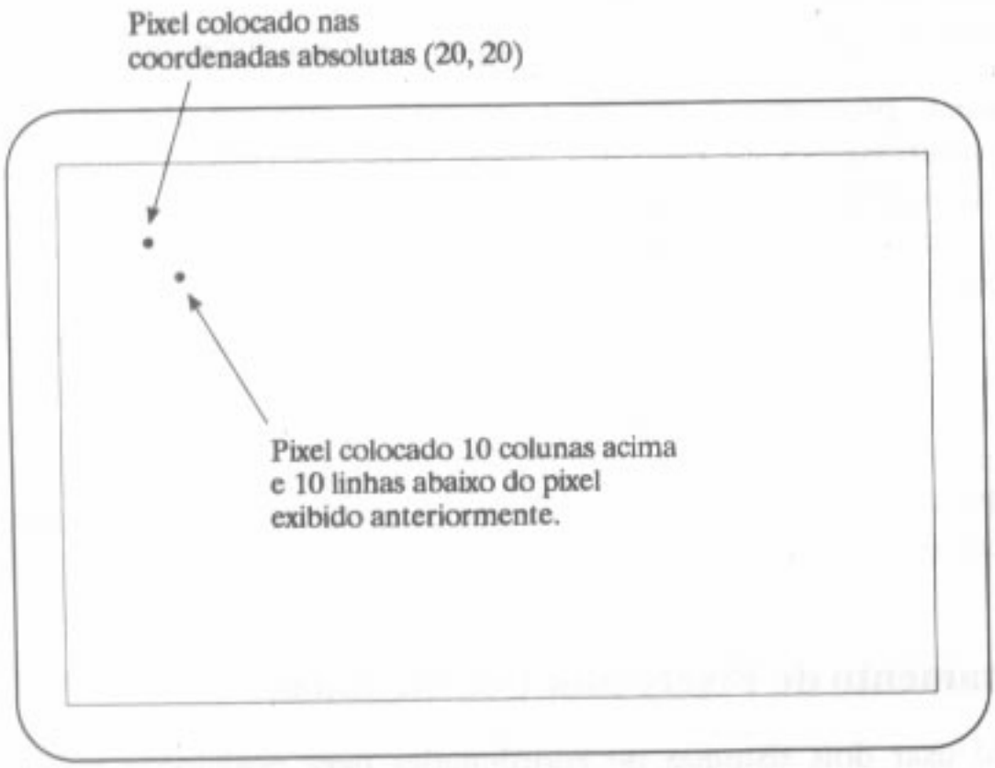
Como vimos, o sistema de coordenadas absolutas funciona bem, mas se você quiser usar pixels individuais para criar uma forma em particular, terá que usar muita adivinhação e contagem. O sistema de coordenadas relativas torna muito mais fácil o trabalho de determinar os valores de coordenada.

Coordenadas relativas

Com as coordenadas relativas, as coordenadas que você especifica para um pixel específico são *relativas* ao último pixel posicionado na tela.

A ilustração a seguir demonstra isso. Para colocar o primeiro pixel na tela, você deve usar as coordenadas absolutas, pois não existe um local de pixel relativo. Para o segundo pixel em diante, no entanto, você poderá usar coordenadas relativas.

Para usar o sistema de coordenadas relativas, use a palavra-chave STEP dentro de um comando PSET ou PRESET.



A palavra-chave STEP

A sintaxe de um comando PSET que contém uma palavra-chave STEP é a seguinte:

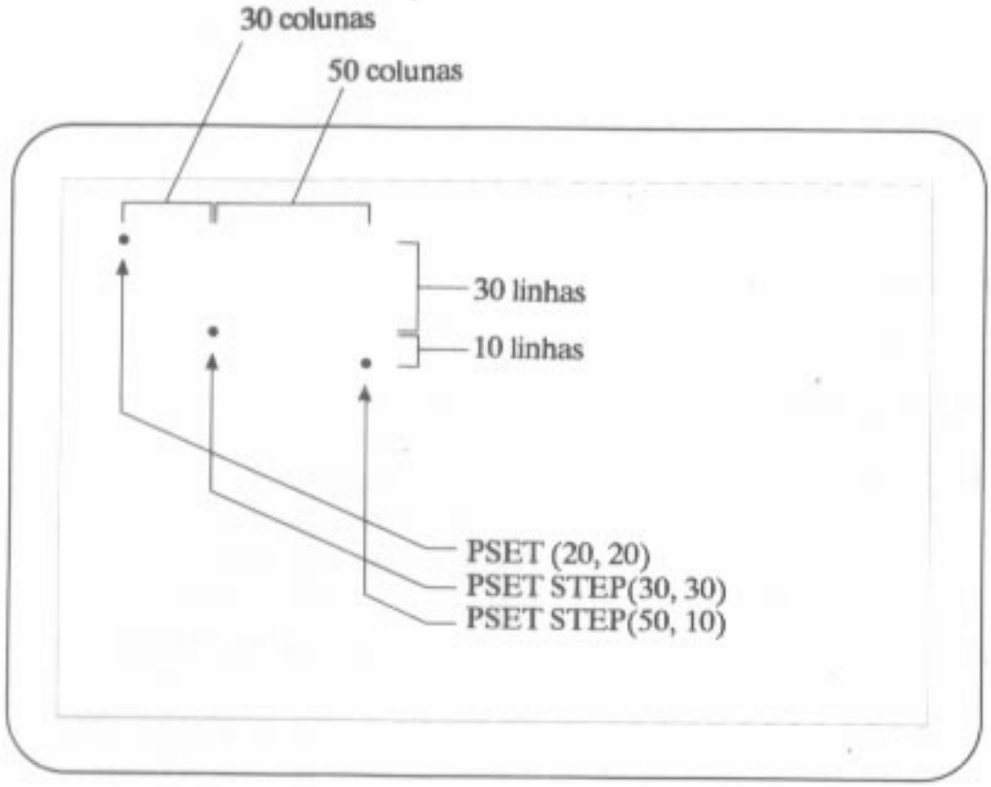
PSET STEP(*coordenadax, coordenaday*)[, *cor*]

(A sintaxe é idêntica para um comando PRESET que contém a palavra-chave STEP; basta substituir o comando PRESET pelo comando PSET.)

Os valores de *coordenadax* e *coordenaday* são números positivos e negativos que dizem ao QBasic onde posicionar o pixel em relação ao último pixel posicionado com um comando PSET e PRESET. Se não existir um pixel anterior, o QBasic usará as coordenadas absolutas (*0, 0*).

A ilustração de tela a seguir demonstra este efeito em "cadeia": a localização de cada pixel além do primeiro depende do local do

pixel anterior. Nunca se esqueça de que a palavra-chave STEP faz com que o QBasic posicione um pixel com relação ao local do pixel posicionado *mais recentemente*.



Prática: Trabalhando com a palavra-chave STEP

1. Digite o programa STEP-1.BAS (Figura 11-9).
2. Rode o programa e entre com um modo de vídeo apropriado. O programa pressupõe uma resolução de 320 X 200; portanto, use um valor *1*, *7* ou *13*, se for possível. (Verifique o que pode ser aceito pelo seu adaptador de vídeo.) Pressione qualquer tecla para interromper o programa.

Este programa oferece a ilusão de atravessar o espaço simulando "estrelas" em sua direção. Dentro do laço DO, o programa define um pixel "invisível" no centro da tela. Este é o ponto de referência no qual os comandos PSET posteriores baseiam suas coordenadas.

Em seguida, o número randômico *quad%* determina em que quadrante a estrela aparecerá. Observe os dois comandos IF — um para os quadrantes 0 e 1 e outro para os quadrantes 0 e 3. (O quadrante 2 usa valores positivos, de modo que nenhuma modificação é necessária.) Os valores modificados nos comandos IF afetam as coordenadas de PSET dentro do laço FOR — alguns usam coordenadas positivas, outros usam coordenadas negativas, e ainda outros usam ambas.

```
' STEP-1.BAS
' Este programa demonstra o uso da ilusão de sair
'  pelo espaço para demonstrar a palavra-chave STEP.

CLS

CONST ESP% = 200    ' controla velocidade do disparo

INPUT "Favor entrar com o modo de vídeo (0-13): ",
modo%
SCREEN modo%

DO
  PSET (160, 100), 0        ' define centro (supondo
                            '  resolução 320 por 200)
  quad% = INT(RND(1) * 4)   ' número randômico para o
                            '  quadrante de onde a
                            '  "estrela" disparará
  randX% = INT(RND(1)*10)   ' números randômicos para
  randY% = INT(RND(1)*10)   '  mover coluna e linha

  ' movimento normal para baixo; inverte y para subir
  IF quad% = 0 OR quad% = 1 THEN
    randY% = -randY%
  END IF
  ' movimento normal à direta; inverte x para esquerda
  IF quad% = 0 OR quad% = 3 THEN
    randX% = -randX%
  END IF

  FOR i% = 1 TO 20
    ' desenha a "estrela"
    PSET SETP(-randX% * i%, randY% * i%)

    FOR j% = 1 TO ESP%    ' laço de espera
    NEXT j%

    PRESET STEP(0, 0)     ' apaga a "estrela"
  NEXT i%

LOOP UNTIL INKEY$ <> ""
```

FIGURA 11-9.
STEP-1.BAS: Saindo pelo espaço com a palavra-chave STEP.

No comando PSET, um número aleatório é multiplicado pelo valor corrente da variável do laço *i%*. A cada laço o valor resultante aumenta, fazendo com que a "estrela" se mova para mais longe do

centro da tela. A "estrela" começa lenta no centro e acelera à medida que chega perto da borda da tela.

Finalmente, observe como o comando PRESET é usado. A palavra-chave STEP faz com que o comando PRESET use coordenadas relativas — coordenadas determinadas pelo comando PSET. Como as coordenadas fornecidas são *(0, 0)*, o comando PRESET faz com que o QBasic coloque o pixel PRESET no mesmo local do pixel PSET anterior — logo, apagando-o.

Só por passatempo, pense em escrever este programa com coordenadas absolutas. Você poderá fazê-lo (poderá até mesmo gostar do desafio), mas logo descobrirá que as coordenadas relativas são muito mais adequadas a este truque de programação em particular.

CRIAÇÃO DE FORMAS COMPLEXAS

Plotar pixels individuais garante um controle exato ao criar gráficos na tela de saída. Entretanto, para formas complexas como linhas, quadros, círculos e polígonos, você teria que fazer muitos cálculos de coordenadas. Para uma linha, você poderia usar um laço para plotar rapidamente uma série de pontos individuais, mas ainda teria que fazer muitos cálculos. Felizmente, o QBasic oferece uma "caixa de ferramentas" para facilitar o trabalho de criar formas complexas num instante.

O Comando LINE

O comando LINE oferece uma forma de criar linhas com um único comando. Se usar o comando LINE, o QBasic fará quase todo o trabalho por você.

Desenho de uma linha simples

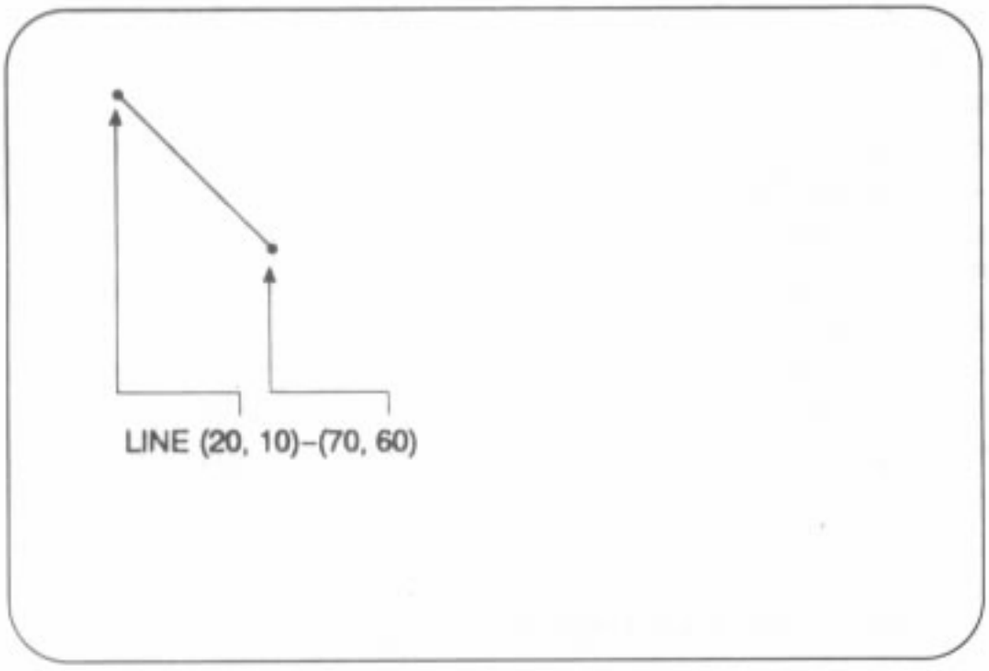
Em sua forma mais simples, o comando LINE usa a seguinte sintaxe:

LINE *(x1, y1)–(x2, y2)*[, *cor*]

Observe que, diferente dos comandos PSET e PRESET, que usam apenas um único conjunto de coordenadas, um comando LINE usa *dois* conjuntos de coordenadas:

- *x1* e *y1* são as coordenadas de linha e coluna do ponto inicial da linha.
- *x2* e *y2* são as coordenadas de linha e coluna do ponto final da linha.

Veja um exemplo de como funciona o comando LINE:



O ponto inicial não precisa estar à esquerda do ponto final. Na verdade, você pode iniciar uma linha no canto inferior direito da tela e prosseguir para cima e à esquerda, se desejar. Basta especificar as coordenadas, e o QBasic "ligará os pontos".

Assim como em qualquer outro comando gráfico, a faixa de valores válidos para *x1, y1, x2* e *y2* é baseada no modo do seu adaptador de vídeo. Você deverá delimitar as coordenadas com parênteses e separar os conjuntos com um hífen.

Prática: Trabalhando com o comando LINE

1. Digite o programa LINE-1.BAS (Figura 11-10).

```
' LINE-1.BAS
' Este programa demonstra o comando LINE.

CLS

INPUT "Favor entrar com um modo de vídeo (0-13): ",
modo%
INPUT "Quantas cores? ", numCores%
SCREEN modo%

CONST ESP% = 100        ' controla a espera após
                        '  cada linha desenhada
DO
  cor% = INT(RND(1) * numCores%) + 1  ' cor da linha
```

(*continua*)

FIGURA 11-10.
LINE-1.BAS: Desenhando linhas randômicas com o comando LINE.

FIGURA 11-10. *continuação*

```
    posx1% = INT(RND(1) * 320)     ' coordenada inicial
    posy1% = INT(RND(1) * 200)
    posx2% = INT(RND(1) * 320)     ' coordenada final
    posy2% = INT(RND(1) * 200)
    LINE (posx1%, posy1%)-(posx2%, posy2%), cor%

    FOR i% = 1 TO ESP%             ' laço de espera
    NEXT i%

  LOOP WHILE INKEY$ = ""
```

2. Rode o programa. Ele lhe pedirá um modo de vídeo e um número máximo de cores, exibindo em seguida linhas aleatórias na tela, com várias cores. Pressione qualquer tecla para interromper o programa.

Desenho de um quadro

Desenhar um quadro, vazio ou cheio, é tão fácil quanto desenhar uma única linha. Você poderia usar quatro comandos LINE para realizar o trabalho, mas os inventores do Basic acharam que os quadros eram usados com freqüência suficiente para garantir um comando próprio na linguagem — economizando muito tempo e cálculo.

O interessante é que você também usa um comando LINE para criar um quadro:

```
LINE (x1, y1)-(x2, y2)[, [cor][, [B[F]]]]
```

Este é o mesmo comando LINE que você acabou de aprender, mas com uma diferença importante. Se você incluir a opção B ao final do comando, o QBasic desenha um quadro [um retângulo] usando as posições inicial e final como cantos opostos do quadro. Veja, na figura da página seguinte, como isso funciona.

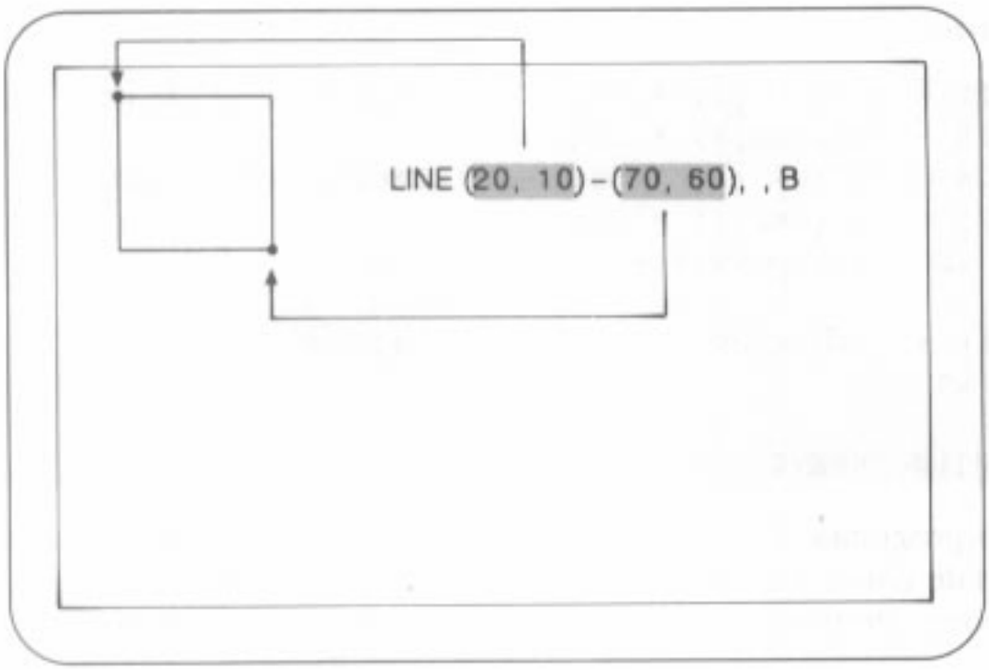
Observe que o QBasic *não* desenha uma linha diretamente entre os pontos inicial e final, como faria em um comando LINE simples.

Se você escolher incluir a opção F imediatamente após a opção B, o QBasic preencherá o quadro com a cor de primeiro plano atual. Você só poderá usar a opção F se usar também a opção B.

Prática: Trabalhando com quadros vazios e cheios

1. Digite o programa LINE-2.BAS (Figura 11-11).

2. Rode o programa. Com base nas respostas, o programa cria um padrão aleatório de quadros vazios ou de quadros cheios. Como vemos neste programa, os quadros podem ter qualquer tamanho ou forma, e você poderá colocá-los em qualquer lugar da tela.




```
' LINE-2.BAS
' Este programa demonstra as capacidades de desenho
'  de quadro do comando LINE.

CLS
INPUT "Entre com um modo de vídeo (0-13): ", modo%
INPUT "Quantas cores? ", numCores%
INPUT "Quadros vazios ou cheios (V ou C)? ", quadro$
SCREEN modo%

CONST ESP% = 800

DO
  cor% = INT(RND(1) * numCores%) + 1  ' cor da linha
  posx1% = INT(RND(1) * 320)     ' coordenada inicial
  posy1% = INT(RND(1) * 200)
  posx2% = INT(RND(1) * 320)     ' coordenada final
  posy2% = INT(RND(1) * 200)

  IF UCASE%(quadro$) = "V" THEN
    LINE (posx1%, posy1%)-(posx2%, posy2%), cor%, B
  ELSEIF UCASE$(quadro) = "C" THEN
    LINE (posx1%, posy1%)-(posx2%, posy2%), cor%, BF
  END IF

  FOR i% = 1 TO ESP%             ' laço de espera
  NEXT i%
LOOP UNTIL INKEY$ <> ""
```

FIGURA 11-11.
LINE-2.BAS: Desenhando quadros randômicos com o comando LINE.

Desenho de formas complexas

As linhas, quadros vazios e quadros cheios com que trabalhamos até aqui usavam coordenadas absolutas. Assim como em PSET e PRESET, você pode usar a palavra-chave STEP com o comando LINE para especificar as coordenadas relativas para uma linha ou um quadro. Pode ainda omitir o ponto inicial da linha ou do quadro — forçando o QBasic a usar o ponto final do último objeto desejado (pixel ou linha) como ponto inicial para a nova linha.

Veja a sintaxe de um comando LINE usando coordenadas relativas:

LINE [[STEP](*x1, y1*)]-[STEP]-(*x2, y2*)[, [cor][, [B[F]]]]

O uso de coordenadas relativas permite a criação de desenhos de linha complexos, que levariam um certo tempo para calcular se fossem usadas coordenadas absolutas. A Figura 11-12 mostra alguns comandos LINE de exemplo que usam coordenadas absolutas e relativas. Você poderá ver o resultado de cada comando. Em cada caso, suponha que o último ponto tenha sido marcado em (*20, 20*).

Comando LINE	*Resultado*
LINE -(30, 50)	Desenha de (20, 20) a (30, 50)
LINE -STEP(30, 50)	Desenha de (20, 20) a (50, 70)
LINE (40, 40) -STEP(60, 60)	Desenha de (40, 40) a (100, 100)
LINE STEP(30, 50)-(70, 70)	Desenha de (50, 70) a (70, 70)
LINE STEP(30, 50)-STEP(70, 70)	Desenha de (50, 70) a (120, 140)

FIGURA 11-12.
Os resultados de comandos LINE quando usados com coordenadas absolutas e relativas.

Eis um resumo de algumas conclusões que podemos tirar da Figura 11-12:

- Se você omitir as coordenadas iniciais, o QBasic usará as coordenadas do ponto mais recente para as coordenadas iniciais.
- Se você omitir as coordenadas iniciais e usar a palavra-chave STEP com as coordenadas finais, o QBasic posicionará o ponto final *com relação* às coordenadas do ponto mais recente.
- Se você incluir as coordenadas iniciais e usar a palavra-chave STEP com as coordenadas finais, o QBasic posicionará o ponto final *com relação* às coordenadas especificadas.
- Se você usar a palavra-chave STEP com as coordendas iniciais e não usar a palavra-chave STEP com as coordenadas finais, o QBasic posicionará as coordenadas iniciais *com relação* às coordenadas

do ponto mais recente e posicionará o ponto final nas coordenadas *absolutas* especificadas.

- Se você usar a palavra-chave STEP com as coordenadas iniciais e finais, o QBasic posicionará as coordenadas iniciais *com relação* às coordenadas do ponto mais recente e posicionará o ponto final *com relação* às coordenadas do ponto inicial recém-calculado.

Usar coordenadas relativas permite que você "encadeie" linhas e quadros, simplificando a tarefa de desenhar formas complexas.

Prática: Trabalhando com LINE e coordenadas relativas

1. Digite o programa STEP-2.BAS (Figura 11-13).

```
' STEP-2.BAS
' Este programa demonstra o uso da palavra-chave
'  STEP no comando LINE.

CLS

INPUT "Entre com um modo de vídeo (0-13): ", modo%
SCREEN modo%

espaço$ = SPACE$(39)

PRINT "Entre com 0 e 0 para terminar"
PSET (150, 50)      ' estabelece ponto inicial

DO
  LOCATE 22, 1
  INPUT "Movimento horizontal (+ ou -): ", horiz%
  INPUT "Movimento vertical (+ ou -):   ", vert%
  LINE -STEP(horiz%, vert%)
  LOCATE 22,1
  PRINT espaço%
  PRINT espaço%
LOOP UNTIL horiz% = 0 AND vert% = 0
```

FIGURA 11-13.
STEP-2.BAS: Um programa simples de desenho de linha.

2. Rode o programa. Na realidade, este é um programa simples para desenho. O programa começa plotando um pixel em (*150, 50*), o ponto inicial do seu desenho. O núcleo do trabalho é feito no laço DO. O programa lhe pedirá os valores horizontal e vertical, que podem ser positivos ou negativos. Após entrar com esses valores, o programa usa o comando LINE junto com a palavra-chave STEP para desenhar a linha.

O Comando CIRCLE

O QBasic também permite o desenho de círculos. Usando o comando CIRCLE, você poderá criar círculos de vários tamanhos que, em conjunto com as outras ferramentas gráficas à sua disposição, permitem a criação de gráficos interessantes e úteis com o seu programa.

Em sua forma mais simples, a sintaxe de um comando CIRCLE é a seguinte:

CIRCLE (*coordenadax, coordenaday*), *raio*[, *cor*]

coordenadax e *coordenaday* são as mesmas coordenadas de tela horizontal e vertical com que você já trabalhou. Esses valores dizem ao QBasic onde posicionar o centro do círculo. *raio* é o valor que indica, em pixels, o raio do círculo. (O raio de um círculo é a metade do seu diâmetro.) *cor* é um inteiro que diz ao QBasic em que cor deverá desenhar o círculo. Se omitir a *cor*, o QBasic usará a cor atual para o primeiro plano.

Assim como em qualquer outro comando gráfico, as coordenadas que você usa dependem inteiramente do modo gráfico definido com o comando SCREEN.

Prática: Trabalhando com o comando CIRCLE

1. Digite o programa CIRCLE-1.BAS (Figura 11-14).

```
' CIRCLE-1.BAS
' Este programa demonstra o comando CIRCLE.

CLS

INPUT "Entre com um modo de vídeo (0-13): ", modo%
SCREEN modo%

DO
  LOCATE 21, 1
  INPUT "Favor entrar com o número da coluna: ", col%
  INPUT " Favor entrar com o número da linha: ", lin%
  INPUT " Favor entrar com o raio do círculo: ", raio%
  CIRCLE (col%, lin%), raio%
  INPUT "Pressione Enter para continuar ou F para
sair: ", algo$
  CLS
LOOP UNTIL UCASE$(algo$) = "F"
```

FIGURA 11-14.
CIRCLE-1.BAS: Um programa simples de desenho de círculo.

2. Rode o programa e entre com um modo de vídeo apropriado. O programa lhe pedirá para entrar com um valor de coluna, um valor de linha e um valor de raio. O comando CIRCLE no meio do programa desenha um círculo com base nos números entrados. Depois, quando pressionar Enter para continuar, o comando CLS "apagará a lousa", e o processo será repetido até que você entre com *f* ou *F* de fim.

Tente entrar com valores maiores do que as coordenadas da tela, ou entre com um valor de raio relativamente grande. Você poderá usar esta técnica para criar um efeito em particular, como um arco próximo a uma borda da tela.

Desenho de arcos

O comando CIRCLE também lhe permite desenhar arcos — segmentos de círculos. Use um comando CIRCLE com valores iniciais e finais para desenhar um arco:

CIRCLE (*coordenadax, coordenaday*), *raio*[, [*cor*][, [*início*][, *fim*]]]

Esta é a mesma sintaxe do comando CIRCLE simples, mas você inclui os valores de *início* e *fim*. Os valores de *início* e *fim* são as medidas, em radianos, do ponto inicial e do ponto final do arco. Embora você talvez esteja mais acostumado a medir os arcos em termos de *graus*, o comando CIRCLE exige que você forneça medidas iniciais e finais em *radianos*. A Figura 11-15 ilustra a relação entre o grau e o sistema de medida por radianos. Ao desenhar um arco, simplesmente veja a Figura 11-15 para determinar os pontos inicial e final do seu arco. Incluímos também a informação de conversão, útil se você souber as medidas do seu arco em graus.

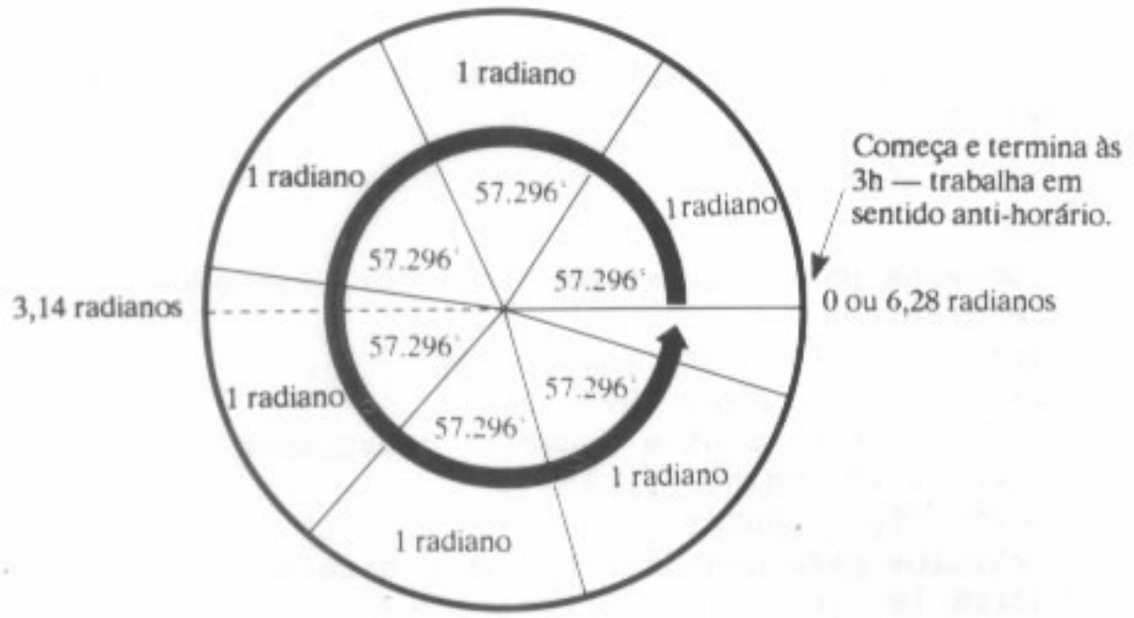
As medidas começam em 0 (3 horas) e continuam em sentido anti-horário até um máximo de 6,28 radianos. O ponto inicial 0 e o ponto final 6,28 estão no mesmo local.

Os valores fornecidos para *início* e *fim* são números de 0 a 6,28. (Nota: O valor 6,28 é tecnicamente 2 X π; estamos usando números arredondados para simplificar.)

Se você colocar um sinal de menos no valor inicial ou final, o QBasic desenhará uma linha (o raio) desse ponto no arco até o centro do "círculo". O exemplo de arco a seguir mostra uma linha do ponto final até o centro do "círculo", devido ao sinal de menos na frente do valor *final* no comando CIRCLE.

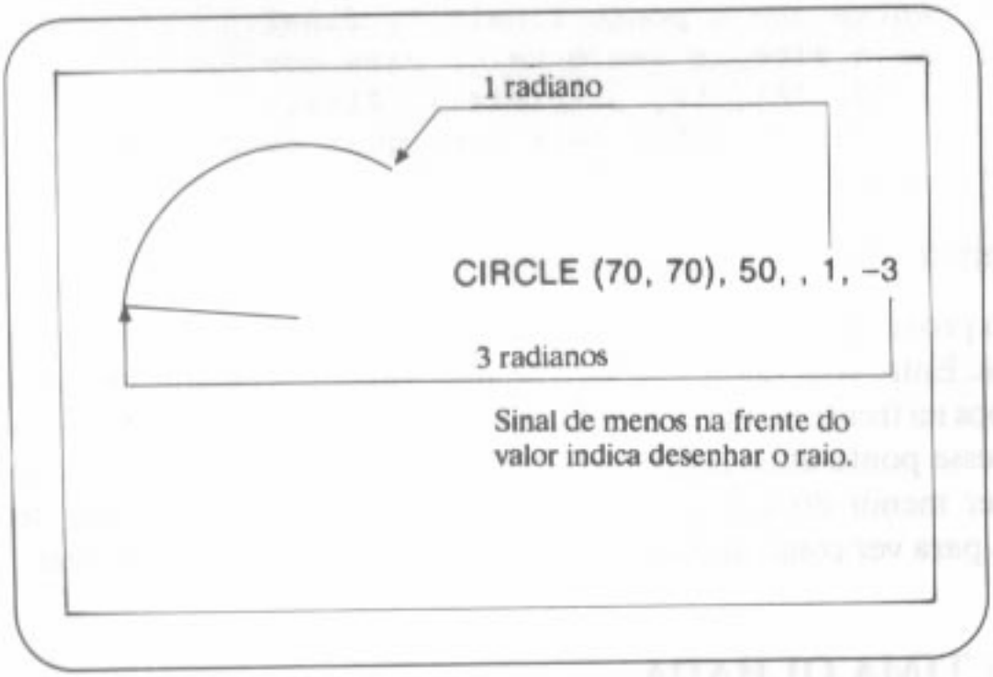
Prática: Trabalhando com arcos

1. Digite o programa CIRCLE-2.BAS (Figura 11-16).



Radianos = (3,14159 X (ângulo em graus)) ÷ 180
1 grau = 0,0174532 radiano

FIGURA 11-15.
Medindo um arco em radianos.




```
' CIRCLE-2.BAS
' Este programa demonstra como criar arcos
'  com o comando CIRCLE.

CLS
```

(*continua*)

FIGURA 11-16.
CIRCLE-2.BAS: Um programa que cria arcos e fatias.

FIGURA 11-16. *continuação*

```
INPUT "Entre com um modo de vídeo (0-13): ", modo%
SCREEN modo%

DO
  CLS
  ' desenha pontos superior, inferior, esquerdo
  ' e direito
  PSET (150, 30): PSET (150, 125)
  PSET (95, 78): PSET (205, 78)
  ' rótulos para pontos superior e esquerdo
  LOCATE 2,18: PRINT "1,57"
  LOCATE 10, 7: PRINT "3,14"
  ' rótulos para pontos inferior e direito
  LOCATE 18, 18: PRINT "4,71"
  LOCATE 10, 28: PRINT "0 ou 6,28"

  LOCATE 20, 1
  PRINT "Entre com valores entre 0 e 6,28"
  PRINT "(Um valor negativo desenha um raio.)"
  INPUT "Entre com o ponto inicial: ", inicial!
  INPUT "Entre com o ponto final: ", final!
  ' desenha o círculo sem o valor para cor
  CIRCLE (150, 78), 50, , inicial!, final!
  INPUT "Pressione Enter para continuar ou F para
sair: ", algo$

LOOP UNTIL UCASE$(algo$) = "F"
```

2. Rode o programa. Este lhe dará uma chance para jogar com a criação de arcos. Entre com valores inicial e final de 0 a 6,28. (Coloque um sinal de menos na frente de um número se quiser que o QBasic desenhe uma linha desse ponto até o centro do círculo.) Observe que o valor inicial pode ser menor do que o valor final. Tente várias combinações de valores para ver como afetam os arcos que o comando CIRCLE cria.

UMA ÚLTIMA OLHADA NOS GRÁFICOS DO QBASIC

Já vimos muita coisa nova até aqui. O QBasic oferece algumas ferramentas de desenho úteis e poderosas. Lamentavelmente não podemos ver todas elas aqui. Mas as ferramentas que você já viu neste capítulo serão suficientes para mantê-lo ocupado por muito tempo permitindo a criação de alguns gráficos "espertos" em seus próprios programas. Agora você poderá incluir, em seus programas, as melhorias listadas a seguir.

- Gráficos de barra, que ilustram dados graficamente.
- Gráficos de setores [torta], que mostram a distribuição dos dados em diferentes categorias.
- Ilustrações, que tornam os seus programas mais interessantes.
- Guardadores de tela, que protegem o seu monitor.
- Animação, que torna seus jogos e simulações mais realísticos.

NOTA: Você sempre poderá checar o sistema de ajuda em linha para obter mais informações sobre gráficos. Experimente — é o melhor meio de aprender!

INTRODUÇÃO À PROGRAMAÇÃO COM SOM

Você acabou de usar gráficos para dar mais vida aos seus programas. Agora irá aprender a incrementá-los ainda mais com as capacidades de som do seu computador e do QBasic.

O Comando SOUND Revisto

No Capítulo 6, quando você aprendeu sobre laços, teve uma rápida introdução ao comando SOUND. Chegou o momento apropriado para uma rápida revisão.

O comando SOUND tem a seguinte sintaxe:

SOUND *freqüência, duração*

freqüência é um valor inteiro de 37 a 32.767, que especifica a freqüência, em hertz (ciclos por segundo), do som que você deseja. Freqüências baixas criam sons baixos; freqüências altas criam sons altos. Observe que a faixa de audição humana é de aproximadamente 20 a 20.000 hertz e que os sons com freqüências acima de 16.000 hertz são mais audíveis para cães do que para a maioria das pessoas. A Figura 11-17 lista os valores padronizados de *freqüência* e suas notas associadas, conforme estabelecido pela American Standards Association [Associação de Padrões Americanos] em 1936.

duração é um valor inteiro que diz ao QBasic quanto tempo um som durará. O valor de *duração* especifica por quantos "pulsos de relógio" a nota soará — existem 18,2 pulsos de relógio por segundo, de modo que o valor de *duração* 9,1 faria com que o QBasic tocasse a nota por meio segundo.

Você deverá especificar tanto a *freqüência* quanto a *duração* em um comando SOUND.

Nota	*Freqüência*	*Nota*	*Freqüência*	*Nota*	*Freqüência*
D#1	38,89	G3	196,00	A#5	932,33
E1	41,20	G#3	207,65	B5	987,77
F1	43,65	A3	220,00	C6	1046,50
F#1	46,25	A#3	233,08	C#6	1108,73
G1	49,00	B3	246,94	D6	1174,66
G#1	51,91	C4	261,63	D#6	1244,51
A1	55,00	C#4	277,18	E6	1328,51
A#1	58,27	D4	293,66	F6	1396,91
B1	61,74	D#4	311,13	F#6	1479,98
C2	65,41	E4	329,63	G6	1567,98
C#2	69,30	F4	349,23	G#6	1661,22
D2	73,42	F#4	369,99	A6	1760,00
D#2	77,78	G4	392,00	A#6	1864,66
E2	82,41	G#4	415,30	B6	1975,53
F2	87,31	A4	440,00	C7	2093,00
F#2	92,50	A#4	466,16	C#7	2217,46
G2	98,00	B4	493,88	D7	2349,32
G#2	103,83	C5	523,25	D#7	2489,02
A2	110,00	C#5	554,37	E7	2637,02
A#2	116,54	D5	587,33	F7	2793,83
B2	123,47	D#5	622,25	F#7	2959,96
C3	130,81	E5	659,26	G7	3135,96
C#3	138,59	F5	698,46	G#7	3322,44
D3	146,83	F#5	739,99	A7	3520,00
D#3	155,56	G5	783,99	A#7	3729,31
E3	164,81	G#5	830,61	B7	3951,07
F3	174,61	A5	880,00	C8	4186,01
F#3	185,00				

FIGURA 11-17.
Freqüências de notas musicais variando em cerca de sete oitavas.

Os efeitos sonoros nos programas de exemplo que vimos até aqui foram criados sem um som particular em mente, mas você poderá escolher valores de *freqüência* combinando com as notas musicais. Como o menor valor de *freqüência* que você poderá usar com o comando SOUND é 37, a nota mais baixa que poderá ser reproduzida é D#1. (Na realidade, a tabela de freqüências da Associação de Padrões Americanos começa com C_0, que tem uma freqüência de 16,35.)

O número subscritado ao final de cada nota indica a oitava dessa nota. Se você estudar a Figura 11-17, notará que para elevar uma nota em uma oitava, bastará duplicar sua freqüência — aproximadamente. Por exemplo, C_2 tem uma freqüência de 65,41 hertz, e C_3 tem uma freqüência de 130,81 — quase o dobro da freqüência de

C_2. Este fato poderá ajudá-lo quando precisar incluir música em seus programas — tudo que você precisa fazer é duplicar uma freqüência em particular para elevar uma nota em uma oitava.

Prática: Trabalhando com o comando SOUND

1. Digite o programa SOUND-1.BAS (Figura 11-18).

```
' SOUND-1.BAS
' Este programa demonstra o comando SOUND.

CLS

INPUT "Entre com um valor de oitava (1-7): ",
oitava%

' toca uma oitava

FOR i% = 1 TO 8
  READ nota%                ' lê freqüência da nota
  ' eleva a nota à oitava desejada
  nota% = nota% * (2 ^ oitava%)
  SOUND nota%, 12           ' toca a nota
NEXT i%

' freqüências das principais notas na segunda oitava

DATA 65, 73, 82, 87, 98, 110, 123, 131
```

FIGURA 11-18.
SOUND-1.BAS: Um programa que toca uma escala musical.

2. Rode o programa e depois entre com um valor de oitava. Observe que a tabela de freqüências no comando DATA na realidade começa em C_2, de modo que se você entrar com um valor de oitava 1, o programa tocará uma escala maior de C_2 a C_3. E como o comando SOUND só aceita valores inteiros como freqüências, o comando DATA terá valores que foram arredondados para o inteiro mais próximo.

Notação Musical e o QBasic

Não estamos acostumados a ver notas musicais em termos da freqüência de cada nota, mas o QBasic exige que você diga que nota deverá tocar usando suas freqüências.

Se pretende incluir música em seus programas — seja uma canção que você mesmo escreveu ou alguma música pautada que deseja

copiar —, terá que converter as notas em freqüências. Você poderá fazer uma "folha de conversão" para ajudá-lo a converter as notas mais rapidamente.

Trabalhando com música no papel

Embora não seja objetivo deste livro explicar a notação musical com detalhes, você poderá aproveitar as dicas a seguir para incluir música em seus programas. Veja a Figura 11-19, que mostra os nomes das notas e suas posições na pauta musical.

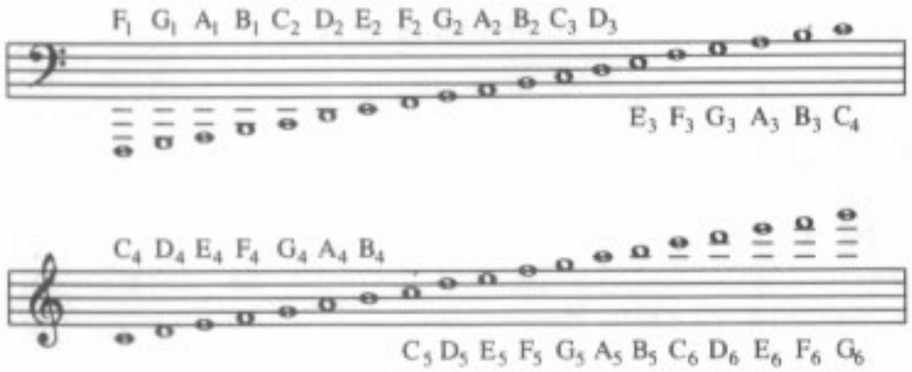


FIGURA 11-19.
Os nomes das notas na pauta musical.

Se estiver trabalhando a partir de uma folha pautada, combine as posições de suas notas com as notas da Figura 11-19, procurando os valores de freqüência correspondentes na Figura 11-17. Depois você poderá escrever ao lado de cada nota da sua folha de música o valor de freqüência que precisa usar com o comando SOUND.

Para chegar aos valores de duração nos comandos SOUND — ou seja, por quanto tempo a nota será tocada —, basta decidir quanto tempo deseja que uma nota permaneça tocando e depois dividir o valor por 2 para notas de meio tempo [colcheias], por 4 para notas de um quarto de tempo [semicolcheias] e assim por diante. Depois de estabelecer estes valores, poderá voltar à folha pautada novamente e anotar o valor de duração ao lado de cada nota.

Depois que determinar o valor de freqüência e a duração de cada nota, estará pronto para incluir uma melodia ao seu programa.

Prática: Tocando uma música com o QBasic

A Figura 11-20 mostra a música de "My Bonnie Lies Over the Ocean". Calculamos o valor de freqüência e a duração de cada nota e escrevemos um pequeno programa para tocar a canção.

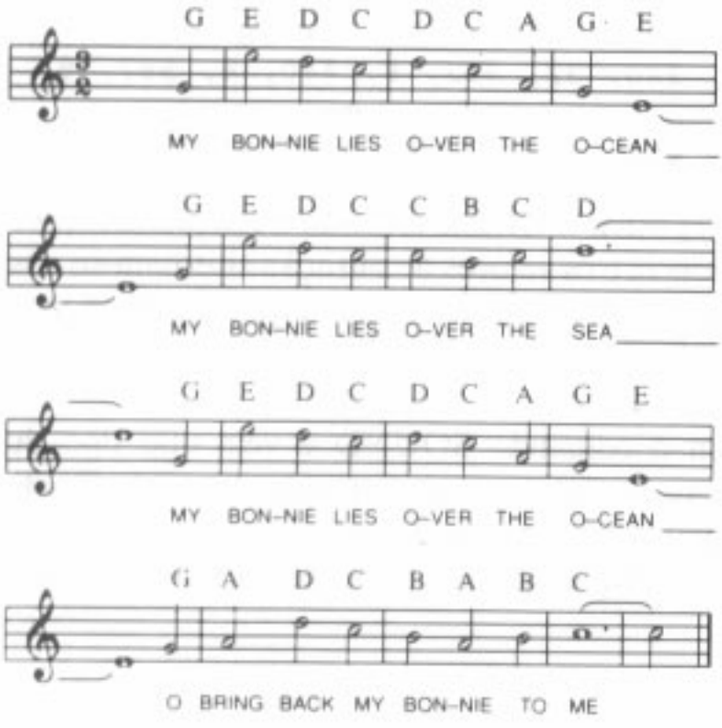


FIGURA 11-20.
Trecho da música de "My Bonnie Lies Over the Ocean".

1. Digite o programa SOUND-2.BAS (Figura 11-21).

```
' SOUND-2.BAS
' Este programa demonstra como tocar uma música no
'  QBasic.

CLS

INPUT "Pressione Enter para começar...", algo$

FOR i% = 1 TO 34
  READ nota%, duração%
  SOUND nota%, duração%
NEXT i%

'    My       bon-     nie      lies     o-       ver
DATA 392, 8, 659, 8, 587, 8, 523, 8, 587, 8, 523,
'       the      o-       cean,    My       bon-
DATA 8, 440, 8, 392, 8, 330, 32, 392, 8, 659,
'      nie      lies     o-       ver      the      sea;
DATA  587, 8, 523, 8, 523, 8, 494, 8, 523, 8, 587, 40
'    My       bon-     nie      lies     o-       ver
DATA 392, 8, 659, 8, 587, 8, 523, 8, 587, 8, 523, 8
```

FIGURA 11-21. (*continua*)
SOUND-2.BAS: Um programa que toca uma música.

FIGURA 11-21. *continuação*

```
'    the      o-       cean-     o        bring    back
DATA 440, 8, 392, 8, 330, 32, 392, 8, 440, 8, 587, 8
'    my       bon-     nie       to       me!
DATA 523, 8, 494, 8, 440, 8, 494, 8, 523, 32
```

2. Rode o programa e poderá ouvir o QBasic tocando a canção. Observe como os valores de freqüência e os valores de duração alternam-se nos comandos DATA. Dentro do laço FOR, o comando READ lê dois valores — um para a freqüência e outro para a duração — cada vez que este é executado.

   Faça experiências com o comando SOUND e veja que tipo de controle você possui sobre os valores nos comandos DATA. Tente multiplicar *nota%* por 2 para elevar todas as notas em uma oitava, por exemplo, e tente dividir a *duração%* por 2 para ver o efeito do andamento da música.

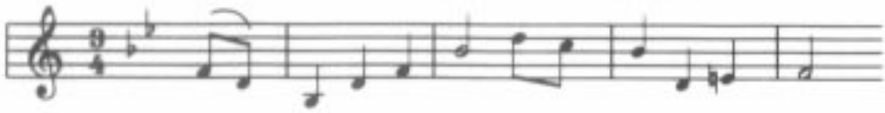
RESUMO

O QBasic possui uma grande faixa de ferramentas que você pode usar para criar gráficos e sons — e só falamos sobre o básico. Após ter completado os exercícios deste livro, gaste algum tempo modificando os programas que vimos neste capítulo. Ou então use a facilidade de ajuda em linha do QBasic para obter mais informações. Gráficos e sons não são os efeitos de programação mais fáceis de dominar, mas certamente são os dois aspectos mais compensadores da programação em QBasic.

PERGUNTAS E EXERCÍCIOS

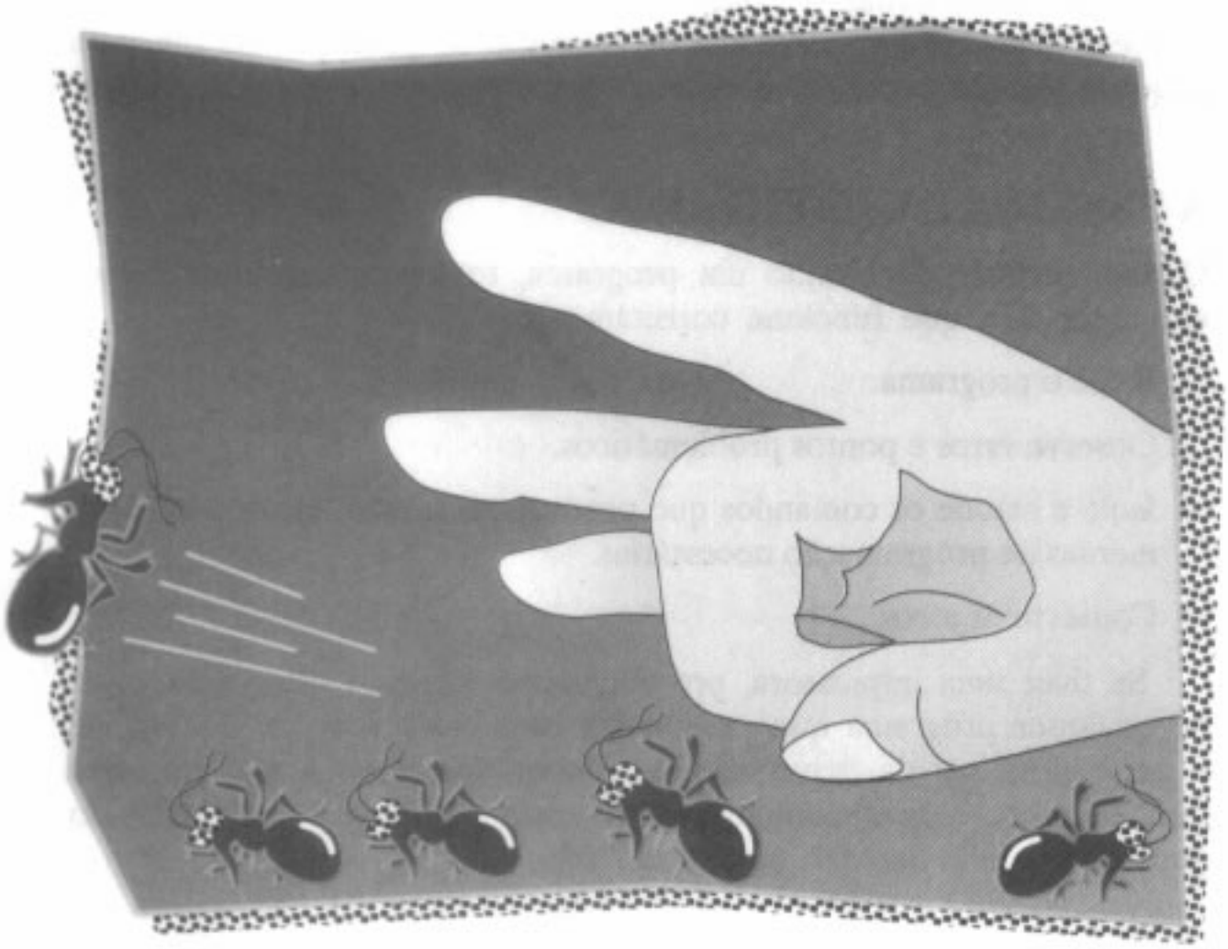
1. Quais são os dois componentes do equipamento de vídeo do seu computador? Qual é a função de cada um deles?
2. Qual é a diferença entre o modo de texto e o modo gráfico?
3. O que o comando LOCATE faz?
4. Escreva um pequeno programa que peça ao usuário para entrar com um nome e depois mostre esse nome no canto esquerdo da tela, "movendo-o" até o canto direito.
5. O que faz o comando SCREEN?
6. O que fazem os comandos PSET e PRESET, e qual a diferença entre eles?
7. Qual a diferença entre coordenadas [illegible]

8. Falso ou Verdadeiro: Você pode usar um comando LINE para criar um quadro preenchido com uma cor.
9. Falso ou Verdadeiro: Você pode usar um comando CIRCLE para criar um círculo preenchido.
10. Escreva um programa que desenhe um círculo próximo ao canto esquerdo da tela e depois "mova-o" para o canto direito. (Dica: Não se esqueça de apagar o círculo anterior.)
11. Converta a melodia a seguir em um programa QBasic:



CAPÍTULO 12

Depuração de Programas em QBasic



Lembra-se do seu primeiro programa em QBasic? Você já fez muita coisa desde que a sua tela de saída mostrou *Ward, I'm worried about the Beaver*. Certamente você merece um parabéns! Mas embora estando a caminho de ser um mestre em programação QBasic, chegará o dia em que o programa recém-digitado recusar-se-á totalmente a rodar. Nessas horas, você precisa ter confiança na sua capacidade de diagnosticar e solucionar o problema de programação. Aí é que entram as habilidades de depuração.

Depuração é o processo de acompanhar e consertar erros em um programa. A capacidade de depurar um programa problemático é uma das mais importantes que você poderá desenvolver como programador iniciante. A depuração exige que você pense em modularidade — para analisar um programa como um computador faz, monitorando a sua execução a cada passo do caminho — a fim de descobrir a falha que não deixa o programa realizar sua tarefa.

Felizmente você não está sozinho. O QBasic oferece vários e poderosos comandos de depuração para ajudá-lo a examinar seus programas. Este capítulo apresenta esses comandos e descreve como programar para evitar a criação de erros em primeiro lugar. Você também depurará um programa de exemplo do princípio ao fim para adquirir alguma experiência com os comandos de depuração.

A PRÁTICA DA DEPURAÇÃO

Quando estiver depurando um programa, repita os seguintes passos até fazer com que funcione corretamente:

1. Rode o programa.
2. Observe erros e pontos problemáticos.
3. Isole e estude os comandos que produziram o erro, usando as ferramentas de programação necessárias.
4. Conserte os erros.

Se tiver uma impressora, provavelmente desejará listar uma cópia de qualquer programa que precise ser depurado. Estudar o código é a parte principal da depuração; e se você for como a maioria, verá que a leitura e acompanhamento de uma listagem é mais fácil do que acompanhar um programa pela tela.

COMANDOS DE MENU DO QBASIC

O QBasic oferece comandos de menu que o ajudam a acompanhar os erros superficiais que aparecem quando você roda o seu programa. Nas seções a seguir, discutimos os comandos relacionados à depuração

Três Tipos de Erros

Podem ocorrer três tipos de erro em um programa QBasic: *erro de sintaxe*, *erro em tempo de execução*, e *erro de lógica*.

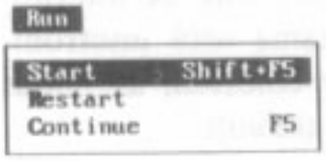
- Um erro de sintaxe é um engano na programação que interfere com as regras do QBasic (como um separador faltando ou uma palavra errada.) O QBasic aponta os erros de sintaxe no seu programa à medida que você o digita, e não permitirá a execução do programa até que cada um tenha sido consertado.
- Um erro em tempo de execução [*run-time*] é um erro que fez com que o programa parasse inesperadamente durante a execução. Os erros em tempo de execução ocorrem quando um evento externo, ou um erro de sintaxe, oculto até então, força um programa a parar enquanto estiver sendo executado. Ultrapassar os limites de um vetor ou tentar abrir um arquivo com um nome errado são exemplos de erros em tempo de execução.
- Um erro de lógica é um erro humano — um erro de programação que faz com que o programa produza resultados errados. A maior parte dos esforços de programação estão voltados para a descoberta de erros lógicos introduzidos pelo programador.

Lembre-se de usar seus recursos de ajuda em linha quando encontrar mensagens de erro produzidas por erros de sintaxe ou erros em tempo de execução.

nos menus Run, Debug, View e Options. Para obter informações mais detalhadas sobre esses comandos, consulte a ajuda em linha.

O Menu Run

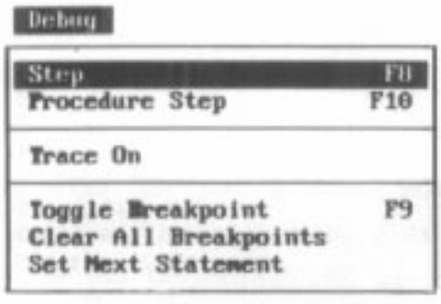
O menu Run contém três comandos relacionados à depuração:



- Start (Shift-F5). Roda o programa atualmente carregado desde o início em plena velocidade. Pressione Ctrl-Break para terminar um programa que está rodando.
- Restart. Cancela todas as variáveis do programa e destaca o primeiro comando executável. Use Restart para mover até o topo do programa antes de executar os comandos paulatinamente com os comandos Step e Procedure Step do menu Debug.
- Continue (F5). Roda o programa atualmente carregado a partir da linha corrente. Use Continue para continuar a execução do seu programa em plena velocidade.

O Menu Debug

O menu Debug contém seis comandos, todos relacionados à depuração:

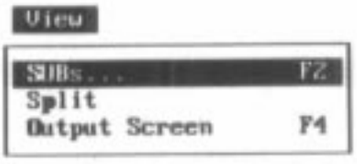


- Step (F8). Executa um comando do programa e destaca o próximo comando executável. Use Step para executar o seu programa com uma instrução de cada vez.
- Procedure Step (F10). Executa um comando do programa e destaca o próximo comando executável. Diferente de Step, Procedure Step executa um subprograma inteiro ou uma função definida pelo usuário em um único passo.
- Trace On. Um comando de chave que prepara o seu programa para a execução em "câmera lenta". Quando Trace On estiver ativo, um programa rodará lentamente e cada comando será destacado à medida que for executado. Quando Trace On estiver inativo (o default), o programa rodará em sua velocidade normal.
- Toggle Breakpoint (F9). Marca ou desmarca um ponto de interupção (uma linha em que a execução do programa será interrompida) no seu programa. Se tiver um monitor colorido, as linhas com *breakpoint* aparecerão em vermelho por default.
- Clear All Breakpoints. Desmarca todos os pontos de interrupção marcados pelo comando Toggle Breakpoint. Use Clear All Breakpoints para desmarcar múltiplos breakpoints ao mesmo tempo.

- Set Next Statement. Marca a linha onde o cursor se encontra como o próximo comando a ser executado. Use Set Next Statement para reexecutar os comandos ou para saltar sobre comandos do programa que não deseja executar.

O Menu View

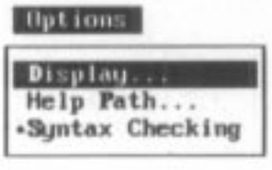
O menu View contém um comando relacionado à depuração:



- Output Screen (F4). Mostra o conteúdo da tela de saída. Use este comando enquanto estiver acompanhando o seu programa, para ver o que o programa mostra na tela de saída. Para retornar ao ambiente de edição, pressione qualquer tecla.

O Menu Options

O menu Options contém um comando relacionado à depuração:



- Syntax Checking. Um comando de chave que desativa o analisador de sintaxe automático do QBasic, que se encontra ativo por default.

Além desses comandos relacionados à depuração, você pode usar a tecla F7 para executar a linha corrente e as linhas entre a linha corrente e o cursor em velocidade total. Pressionar esta tecla é um atalho para a definição de um breakpoint com a tecla F9 e a execução a partir da linha corrente até o breakpoint com a tecla F5.

ACOMPANHANDO AS VARIÁVEIS COM PRINT

Outra ferramenta de depuração muito útil é um comando com o qual você já está acostumado: o comando PRINT. Colocando comandos PRINT dentro de um programa, ou usando-os na janela Immediate, você poderá examinar o conteúdo das variáveis à medida que um programa é executado. (Você provavelmente desejará mostrar os comandos PRINT em uma cor diferente para diferenciá-los da saída do

programa.) Depois de usar comandos PRINT para a depuração, lembre-se de removê-los do seu programa.

A sessão de prática a seguir usa PRINT para encontrar um erro lógico em um programa.

Prática: Usando PRINT para acompanhar uma alteração de variável.

O programa ERROLOG.BAS (Figura 12-1) é um simples jogo de adivinhação que pede ao usuário para adivinhar o tamanho em litros do maior milkshake que já foi feito (um recorde impressionante estabelecido em Ohio, EUA, em 1988). Embora o programa rode sem erros de sintaxe, ele não opera corretamente: não importa o número que o usuário entre, o programa responde que o valor é muito baixo. Você consegue descobrir o erro lógico que fez com que o programa falhasse?

Digite e execute o programa ERROLOG.BAS.

```
' ERROLOG.BAS
' Este programa contém um erro lógico.
' Tente descobri-lo.

CONST MAIORSHAKE% = 658   ' número de litros correto

CLS

PRINT "Quantos litros havia no maior milkshake já
feito?"
PRINT

DO WHILE (chute% <> MAIORSHAKE%)   ' faz comparação
  INPUT ; "Adivinhe: ", chute      ' pede um valor

  COLOR 2              ' muda a cor para verde
  ' determina a proximidade do chute e mostra dica
  IF (chute% < MAIORSHAKE%) THEN
     PRINT , "muito baixo"
  ELSEIF (chute% > MAIORSHAKE%) THEN
     PRINT , "muito alto"
  ELSE
     PRINT
  END IF
  COLOR 7                          ' volta à cor default
LOOP
```

FIGURA 12-1. *(continua)*

ERROLOG.BAS: Um jogo de adivinhação contendo um erro de lógica.

FIGURA 12-1. *continuação*

```
PRINT
PRINT "Correto! Foi feito um milkshake de morango
com";
PRINT MAIORSHAKE%; "litros em Ohio, EUA, em 1988."
```

Você verá uma saída semelhante a esta:

```
Quantos litros havia no maior milkshake já feito?

Adivinhe: 100     muito baixo
Adivinhe: 200     muito baixo
Adivinhe: 10000   muito baixo
Adivinhe: 658     muito baixo
Adivinhe: Ctrl-C
```

Não importa o valor entrado (incluindo o valor correto, 658), a mesma mensagem aparecerá: *muito baixo*. O programa fica preso em um laço infinito. O único meio de terminá-lo é pressionando Ctrl-C ou Ctrl-Break.

Vamos tentar localizar o erro usando um comando PRINT para acompanhar o valor da variável *chute%*, que muda toda vez que o usuário entra com um novo valor no laço DO.

1. Coloque os seguintes comandos (que mostram o valor da variável *chute%* em vermelho) no final do laço DO, logo acima do comando LOOP:

   ```
   COLOR 4 : PRINT "DEBUG chute% ="; chute%: COLOR 7
   ```

   Quando o fizer, sua tela ficará assim:

O comando PRINT é rodeado por dois comandos COLOR: o primeiro muda a cor do primeiro plano para vermelho, e o segundo a muda de volta para branco. Você poderá sempre usar essa segunda cor para distinguir a saída do programa da saída dos comandos de depuração. Se o seu monitor não mostrar cores, omita os comandos COLOR.

2. Rode o programa novamente. Você verá uma saída semelhante à seguinte, completa com a informação de depuração:

```
Quantos litros havia no maior milkshake já feito?

Adivinhe: 10        muito baixo
DEBUG: chute% = 0
Adivinhe: 30000  muito baixo
DEBUG: chute% = 0
Adivinhe: 658       muito baixo
DEBUG: chute% = 0
Adivinhe: Ctrl-C
```

O resultado do comando PRINT é interessante — ele mostra que, por algum motivo, o chute do valor não foi atribuído à variável *chute%*. Existem duas explicações para isso: algo saiu errado com o comando INPUT ou a variável *chute%* foi modificada em algum ponto na linha. Você já pode descobrir o erro?

Se examinar o comando INPUT no programa, verá que o problema está na variável *chute%*: no comando INPUT, a variável *chute%* não possui o caracter de declaração de tipo %:

```
INPUT ; "Adivinhe: ", chute
```

Como resultado, o QBasic considera *chute* e *chute%* como duas variáveis separadas.

3. Conserte o comando INPUT no programa ERROLOG.BAS incluindo um símbolo % após *chute*:

```
INPUT ; "Adivinhe: ", chute%
```

4. Rode o programa novamente para ter certeza de que está rodando corretamente. Se estiver, remova os comandos de depuração incluídos no Passo 1.

Erros Comuns de Programação

Os erros lógicos a seguir aparecem de vez em quando como resultado de erros de digitação ou erros de projeto. Verifique-os com freqüência!

- **Atribuições de tipo incorretas.** Asssegure-se de que os valores dos dados não foram atribuídos a variáveis do tipo errado. Atribuições de tipo incorretas podem passar pelo QBasic e causar problemas mais tarde no seu programa. Verifique principalmente as atribuições em cada comando INPUT, INPUT# e READ do seu programa.
- **Confusão em nome de variável.** Não erre na digitação das variáveis e nem omita os caracteres de declaração de tipo. Também verifique uma fonte de confusão comum: variáveis locais e globais com o mesmo nome. Lembre-se: você estará sozinho ao usar nomes de variáveis — o QBasic não verifica se estão corretos.
- **Comparações lógicas incorretas.** Verifique as comparações lógicas feitas por operadores relacionais em comandos IF, SELECT, CASE e em laços. Use programas de exemplo ou a janela Immediate para verificar se as comparações estão corretas.
- **Laços sem fim.** Cuidado com laços que não conseguem terminar. Use a janela Immediate para checar as condições de término dos seus laços.
- **Saída confusa.** O uso incorreto de comandos PRINT, PRINT USING, LOCATE e SCREEN pode produzir uma tela de saída confusa ou misturada. Assegure-se de que os seus programas podem cuidar de qualquer dado que o usuário possa entrar.
- **Problemas com vetor.** Não confunda um número de índice de vetor com o valor armazenado no vetor. E para evitar a mensagem *Subscript out of range* [subscrito fora de faixa], não leia ou escreva após o final de um vetor.

DEPURANDO UM PROGRAMA PASSO A PASSO

Para ter mais experiência na procura e conserto de erros, vamos depurar o programa DOCES.BAS, que usa um vetor para registrar suas sobremesas favoritas e o número de calorias que elas contêm.

Um DOCES.BAS Sem Erros

DOCES.BAS pede ao usuário seu nome e o número de doces que ele deseja entrar. O número de entradas do usuário é usado para dimensionar um vetor unidimensional de valores de cadeia em que os nomes de doces serão armazenados. Um subprograma, em seguida, inclui dados ao vetor e totaliza o número de calorias que o usuário entrou. O programa imprime o conteúdo do vetor e mostra o total de calorias em verde piscante.

Quando DOCES.BAS for rodado corretamente, produzirá a saída da seguinte forma:

```
Bem-vindo ao Programa de Doces!

Favor entrar com o seu nome: Marcos

Quantos doces gostaria de entrar, Marcos? 2

  Nome do doce: Morangos com creme
  Calorias no doce: 125

  Nome do doce: Torta de chocolate
  Calorias no doce: 310

A seguir está uma lista dos doces favoritos de Marcos:

  Morangos com creme
  Torta de chocolate

A lista contém um total de 435 calorias!
```

Prática: Rodando o programa DOCES.BAS

Todavia, DOCES.BAS (Figura 12-2) contém dois erros lógicos. Digite o programa DOCES.BAS exatamente como mostramos e execute-o para descobrir o que há de errado.

```
' DOCES.BAS
' Este programa possui dois erros lógicos. Tente
'  descobri-los.

              ' declara subprograma PegaDados
DECLARE SUB PegaDados (vetor$(), num%,
totalCalorias%)

CLS
```

FIGURA 12-2. (*continua*)
DOCES.BAS: Um programa contendo dois erros lógicos.

FIGURA 12-2. *continuação*

```
                ' mostra mensagem de bem-vindo
PRINT "Bem-vindo ao Programa de Doces!"
PRINT             ' pede nome do usuário
INPUT "Favor entrar com o seu nome: ", nomeUsu$
PRINT             ' pede número de sobremesas
PRINT "Quantos doces gostaria de entrar,"; nomeUsu$;
INPUT num%
PRINT

DIM doces$(num%)    ' dimensiona vetor de cadeia

' chama o subprograma
PegaDados doces$(), num%, totalCalorias%

PRINT "A seguir está uma lista dos doces favoritos
de "; nomeUsu$
PRINT

FOR i% = 1 TO num%
  PRINT "  "; doces$(i%)  ' mostra conteúdo do vetor
NEXT i%

PRINT             ' mostra total de calorias
                  '  com o número em vermelho
PRINT "A lista contém um total de";
COLOR 20: PRINT totalCalorias%; : COLOR 7
PRINT "calorias!"

END

SUB PegaDados (vetor$(), num%, totalCalorias%)

' O subprograma PegaDados pega dados de
'  doces do usuário.

' laço por num% vezes (passado pelo prog. principal)
FOR i% = 1 TO num%
  ' pede nome e calorias
  INPUT "  Nome do doce: ", vetor%(num%)
  INPUT "  Calorias no doce: ", calorias%
  PRINT
  totalCalorias% = calorias% ' total acumulado
NEXT i%

' retorna vetor$ e totalCalorias% ao prog. principal
END SUB
```

Você verá uma saída semelhante a esta:

```
Bem-vindo ao Programa de Doces!

Favor entrar com o seu nome: Marcos

Quantos doces gostaria de entrar, Marcos? 2

  Nome do doce: Morangos com creme
  Calorias no doce: 125

  Nome do doce: Torta de chocolate
  Calorias no doce: 310

A seguir está uma lista dos doces favoritos de Marcos:


  Torta de chocolate

A lista contém um total de 310 calorias!
```

Observe os dois problemas óbvios: o programa só lista um dos valores armazenados no vetor *doces$*, e o programa não mostra o total de calorias correto. Lembrando dessas duas observações, você pode começar a depurar o programa passo a passo.

Depuração do Programa DOCES.BAS

A parte introdutória do programa parece estar correta — ela mostra informações úteis na tela e obtém dados do usuário. Vamos passar por estes comandos definindo um ponto de interrupção próximo ao meio do programa e rodando a primeira seção do programa a toda velocidade.

Definição de um ponto de interrupção

Para definir um ponto de interrupção e depois rodar DOCES.BAS até esse breakpoint:

1. Selecione o comando Restart pelo menu Run para preparar o programa para execução desde o início. O comando CLS, o primeiro comando executável no programa, será destacado.

2. Mova o cursor para baixo, até a 17ª linha do programa (a linha contendo o comando DIM) e selecione o comando Toggle Breakpoint pelo menu Debug (F9).

3. Selecione o comando Continue pelo menu Run (F5) para rodar o programa até o breakpoint. Entre com o seu nome ao primeiro pedido

e com o número *2* ao segundo pedido de entrada. O programa parará quando atingir o *breakpoint*.

4. Escolha o comando Toggle Breakpoint (F9) para apagar o ponto de interrupção — ele não é mais necessário.

Análise do subprograma *PegaDados*

Agora que estamos na região do erro, vamos percorrer o código num ritmo mais lento. Fique de olhos atentos — algo que fez sentido para quem escreveu o subprograma *PegaDados* não está funcionando.

1. Escolha o comando Step pelo menu Debug (F8) e execute o comando DIM, que dimensiona o vetor *doces$*. *doces$* deverá ser grande o suficiente para conter todos os nomes de doces que o usuário digitar — um número que já sabemos e armazenamos na variável *num%*. Observe que *num%* aparece no comando DIM e que o comando DIM é executado sem erro — parece que tudo está bem até aqui.

2. Escolha o comando Step (F8) três vezes para entrar no subprograma *PegaDados* e executar os dois primeiros comandos nele. Entre com *Torta de cereja* ao pedido do nome do doce.

3. Dê uma boa olhada no conteúdo do subprograma *PegaDados* para ver se algo parece fora de ordem — às vezes um erro poderá se tornar visível imediatamente. Um meio excelente de examinar o processo de entrada até este ponto é entrar com um comando PRINT de depuração na janela Immediate. Vamos fazer isso agora para ver se o primeiro nome de sobremesa foi armazenado corretamente no primeiro elemento de *vetor$*. Pressione F6 para passar à janela Immediate e depois entre com a seguinte linha:

```
COLOR 4: PRINT "DEBUG: vetor$(1) ="; vetor$(1):
COLOR 7
```

(O sinal de dois pontos lhe permite colocar mais de um comando em uma única linha.) Depois disso, a informação de depuração será mostrada na sua tela em vermelho:

```
DEBUG: vetor$(1) =
```

vetor$(1), o primeiro elemento do vetor chamado *vetor$*, deveria conter a cadeia *Torta de cereja* — acabamos de digitar! Mas não; logo, algo deve estar errado com o comando INPUT. Especificamente, algo deve estar errado com o elemento de índice da atribuição do vetor: o elemento deve ser avaliado como 1 na primeira passagem pelo laço.

4. Pressione qualquer tecla para retornar à janela Immediate.

Isolamento do primeiro erro lógico

Descobrimos o sintoma do nosso primeiro erro lógico. Vamos isolar sua causa dando uma olhada mais de perto no comando INPUT e no seu papel no laço FOR. Uma regra prática fundamental para índices de vetores e laços é que, se o laço tiver uma variável contadora e cada elemento do vetor tiver que ser acessado, a variável contadora (ou alguma derivada da mesma) deverá ser usada como índice do vetor. O subprograma *PegaDados* tem tudo certo, menos o índice do vetor: o comando INPUT deveria estar usando a variável contadora *i%* como índice do vetor, e não a variável *num%* [que, no caso, é mais constante do que variável].

Conserte o erro, pressionando F6 para retornar à janela View e depois mudando o comando INPUT para o seguinte:

```
INPUT "  Nome do doce: ", vetor$(i%)
```

Teste do reparo do erro

Você corrigiu o erro lógico; agora vamos testar o programa novamente:

1. Deixe o cursor na linha que contém a linha de entrada do doce e selecione o comando Set Next Statement pelo menu Debug. O comando Set Next Statement lhe permite reexecutar o comando INPUT para testar nosso reparo do erro (economizando o tempo que levaria para rodar o programa todo de novo).

2. Pressione F8 para executar o novo comando INPUT e depois entre novamente com *Torta de cereja*. Para verificar se *Cherry pie* está agora armazenado no primeiro elemento de *vetor$*, pressione F6 para passar à janela Immediate, mova o cursor para cima até o comando de depuração PRINT e pressione Enter para executá-lo. Desta vez o valor de cadeia correto, *Torta de cereja*, deverá aparecer na tela. Pressione qualquer tecla para retornar à janela Immediate e depois pressione F6 para retornar à janela View.

3. Continue o seu teste do laço FOR movendo o cursor para baixo, até a linha que contém o comando END SUB — a última linha no subprograma. Selecione o comando Toggle Breakpoint (F9) para definir um ponto de interrupção nessa linha, e depois escolha o comando Trace On do menu Debug. Esses dois passos nos permitem ver as instruções restantes no laço FOR em "câmera lenta". Especificamos dois doces no início do programa, de modo que o laço será executado duas vezes. Selecione o comando Continue (F5) para começar a rodar o programa lentamente. Ao executar o laço, você deverá entrar com os dados do doce.

4. Parece que o nosso conserto resolveu o problema — o programa saltou e parou no ponto de interrupção sem um erro. Selecione o comando Toggle Breakpoint (F9) para remover o ponto de interrupção, e depois selecione o comando Trace On novamente para retornar a velocidade de execução ao normal.

Procura do segundo erro

Mais um para terminar. Vamos devagar para procurar o último erro do programa.

1. Pressione F8 mais uma vez para retornar ao programa principal e mais sete vezes para mostrar alguns comandos PRINT e o conteúdo do vetor *doces$*, que foi passado de volta ao programa principal pelo subprograma *PegaDados*.
2. Pressione F4 para ver a tela de saída atual. Todos os elementos corretos do vetor estão lá! Observe que a tela de saída começa a ficar um pouco confusa com as mensagens de depuração e reinícios que fizemos. Não se preocupe muito com isso — eles desaparecerão na versão final do programa. Pressione qualquer tecla para retornar à janela View.
3. Pressione F8 mais sete vezes para executar os comandos que mostram a contagem final de calorias na tela e terminam o programa. Pressione F4 para ver os resultados. A contagem de calorias em vermelho piscante impressiona, mas está completamente errada. Vamos usar um comando PRINT para estudar a variável *totalCalorias%*. Pressione qualquer tecla para retornar à janela View.
4. Passe para a janela Immediate, mude o comando de depuração PRINT para

   ```
   COLOR 4: PRINT "DEBUG: totalCalorias% ="; totalCalorias%:
   COLOR 7
   ```

   e pressione enter. O comando PRINT mostra o último valor de calorias que você entrou! O que aconteceu aqui? O subprograma *PegaDados* não mantinha um total acumulado do número de calorias?

Isolamento do segundo erro lógico

Vamos retornar ao subprograma *PegaDados* para ver se podemos achar a fonte do erro. Um método inicial fácil é verificar a grafia — talvez um nome de variável foi digitado de forma errada no subprograma. Retorne à janela View e depois use o comando Find pelo menu Search para pesquisar a variável *totalCalorias%*. (O comando Find é muito útil quando você está trabalhando com grandes programas.) O comando Find localiza a variável *totalCalorias$* na lista de

parâmetros do subprograma *PegaDados*. Selecione o comando Repeat Last Find pelo menu Search (F3) várias vezes para encontrar todas as ocorrências de *totalCalorias%* no programa. Ele apareceu seis vezes — três vezes em *PegaDados* e três vezes no programa principal.

Nosso problema não é erro de sintaxe; esses nomes de variável estão todos corretos. Vamos verificar o comando no subprograma PegaDados que mantém um total acumulado dos valores de caloria que o usuário entra. Pressione F2 e selecione *PegaDados* para editar. Examine o comando de atribuição no final do laço FOR:

```
totalCalorias% = calorias%
```

Será que o comando atualiza corretamente a variável *totalCalorias%* para manter o total acumulado?

Conserto do segundo erro lógico

A resposta é *Não!* Como vimos em todo este livro, para atualizar um total acumulado você precisa incluir o total atual *e* o próximo valor somado ou lido e depois atribuir o resultado ao total acumulado. Para consertar este comando de atribuição, mude-o para:

```
totalCalorias% = totalCalorias% + calorias%
```

Depois de fazer esta mudança, o programa estará pronto para rodar. Teste o programa executando-o do princípio ao fim, até ter certeza de que está completamente sem erros.

Como Evitar Erros

Agora que você já aprendeu a procurar e consertar erros, aqui estão algumas dicas que o ajudarão a escrever programas sem erros no futuro.

Planeje seu programa cuidadosamente

Antes de começar a digitar, assegure-se de que entende o que deseja que o programa faça e como pretende escrever o programa em QBasic. Pense nos vários algoritmos que o programa usará, como a entrada e saída do programa serão organizadas e como os dados serão armazenados e manipulados. Comece de forma simples, aumentando aos poucos a complexidade.

Trabalhe com um passo de cada vez

Não tente escrever seu programa inteiro ao mesmo tempo. Crie e teste o programa um trecho de cada vez. Isole as diferentes tarefas sempre que puder, criando subprogramas e funções.

Rode o seu programa com freqüência

Quando tiver feito uma mudança no programa, execute-o por inteiro para ter certeza de que tudo funciona bem em conjunto. Seguindo esta regra, você poderá descobrir erros de programação simples logo no início — antes que se acumulem e se tornem grandes problemas de programação.

Teste novas idéias na janela Immediate

Use a janela Immediate para testar pequenas partes do código do programa *antes* de colocá-las no programa.

Teste o seu programa exaustivamente

Teste bem o seu programa a cada passo do caminho, e saiba como o seu programa responderá a qualquer tipo de entrada. Considere as seguintes perguntas:

- Qual é o maior número ou cadeia com que este programa pode trabalhar?
- Qual é o menor?
- O que fará com que este programa "falhe"?
- Como posso evitar que um usuário cause uma falha neste programa?
- O usuário entenderá o que este programa faz?

RESUMO

Neste capítulo você encontrou uma série de ferramentas, dicas e técnicas de depuração. Essas ferramentas têm sido muito aperfeiçoadas nos últimos anos, e temos muita sorte em possuir uma coleção tão poderosa bem aqui no QBasic. Porém, como programador aprendendo a depurar seus próprios programas (e, pior que isso, os programas dos outros), você precisa de mais do que ferramentas de depuração para ajudá-lo. As soluções vêm ao raciocinarmos lógica e criativamente sobre nossos programas, examinando a execução do programa passo a passo. Quanto mais você souber a respeito do QBasic e suas regras de sintaxe, maiores serão suas chances de detectar erros no programa e encontrar soluções para eles.

PERGUNTAS E EXERCÍCIOS

1. Que passos você deve seguir para isolar um erro no seu programa?
2. Qual é a diferença entre um erro de sintaxe e um erro em tempo de execução?
3. Se você estivesse isolado em uma ilha deserta com apenas dois comandos do menu Debug, quais deles você desejaria ter?
4. O que faz a tecla de função F7?
5. Qual é a finalidade do comando de menu Set Next Statement?
6. Que tipo de erro lógico você acha mais difícil de descobrir e consertar?
7. O programa a seguir (ARTISTAS.BAS) contém dois erros de sintaxe. Digite o programa, encontre os erros e conserte-os.

```
' ARTISTAS.BAS
' Este programa possui dois erros de sintaxe.
' Você consegue descobri-los?

CLS

DIMM artistas$(5)              ' dimensiona vetor
                               ' de cadeia

PRINT "Entre com os nomes dos seus cinco artistas
favoritos."
PRINT

FOR i% = 1 TO 5                ' pega 5 cadeias
  INPUT "Artista: ", artistas$(i%)
NET i%

PRINT
PRINT "Você entrou com os seguintes nomes:"
PRINT

FOR i% = 1 TO 5                ' mostra as 5 cadeias
  PRINT artistas$(i%)
NEXT i%
```

8. O programa a seguir (NOMES.BAS) possui dois erros lógicos. Digite o programa e use as ferramentas de depuração discutidas neste capítulo para descobrir e consertar os erros.

```
' NOMES.BAS
' Este programa separa o primeiro e último nomes e
' os imprime. Você pode achar os dois erros lógicos?

CLS

PRINT "Entre com o primeiro e último nomes assim: ";
PRINT "Último, Primeiro"
PRINT

INPUT "Nome: ", nomeTodo$

localVirgula% = INSTR(1, nomeTodo$, ",")

IF (localVirgula% < 0) THEN
  ultimo$ = LEFT$(nomeTodo$, localVirgula% - 1)
  primeiro$ = RIGHT$(nomeTodo$, LEN(nomeTodo$) -
localVirgula% - 1)

  PRINT
  PRINT "Que belo nome! Prazer em conhecê-lo, ";
  PRINT primeiro$; " "; ultimo$; "!"
ELSE
  PRINT
  PRINT "O nome não está no formato Último, Primeiro."
END IF
```

CAPÍTULO 13

Aprenda Mais sobre o Basic

Parabéns! Você chegou ao final deste curso. Embora sendo um ponto final nesta caminhada em particular, esperamos que seja o ponto de partida para a sua própria exploração da linguagem Basic. Neste capítulo, revisaremos rapidamente os tópicos abordados neste livro e apresentaremos alguns outros livros e software que poderão ajudá-lo a expandir e melhorar as habilidades de programação Basic adquiridas aqui. Em particular, mostraremos as características do Microsoft QuickBasic Compiler, uma versão avançada do QBasic que você poderá obter quando estiver pronto para o próximo nível de programação.

O QUE VOCÊ APRENDEU

Você já percorreu um longo caminho em pouco tempo. Pare um pouco e reflita a respeito do seu progresso: volte aos capítulos introdutórios e rode alguns dos primeiros programas para ver o quanto aprendeu sobre a linguagem QBasic e como o computador realiza o seu trabalho. Entre outras coisas, você aprendeu a respeito da estrutura de um programa em QBasic e como criar um programa usando comandos do menu. Você aprendeu a criar um programa usando comandos de menu. Aprendeu a armazenar informações em um lugar e usá-las em outro, e também como pegar a entrada do usuário e usá-la no seu programa. Você aprendeu a programar de forma eficiente — usando laços para tarefas repetitivas e declarando constantes para valores que nunca mudam. Você aprendeu a projetar seus programas e consertá-los quando não rodarem corretamente. Aprendeu até mesmo a usar gráficos e som para tornar seus programas divertidos *e* funcionais. O principal é que tudo isso foi aprendido na prática, escrevendo programas reais — algo a se ter em mente à medida que continuar o seu aprendizado no QBasic.

À medida que você trabalhar nos seus próprios projetos de programação, não reinvente a roda a todo o momento. Reutilize as partes de uso geral dos seus programas sempre que puder. Crie uma pasta com uma listagem de cada programa que você escrever e mantenha-a próxima ao computador — você nunca saberá quando os antigos programas poderão ser úteis.

O QUE VOCÊ AINDA TEM A APRENDER

Os métodos de trabalho do QBasic com dados, entrada e saída e operações repetitivas — os métodos que você aprendeu neste livro — são os blocos de montagem de cada programa que você escreve. Mas existe muito mais a aprender. Você provavelmente descobrirá, ao continuar trabalhando com o QBasic, que existem situações que conduzem a perguntas mais específicas: Como o QBasic trabalha com o

mouse ligado ao meu sistema? Como posso acessar as capacidades especiais da minha impressora? Como funcionam os arquivos de acesso randômico? Como posso usar o QBasic junto com outras linguagens de programação? Os livros que descrevemos a seguir poderão ajudá-lo a encontrar as respostas a essas perguntas.

Microsoft QuickBASIC Programmer's Toolbox, de John Clark Craig (Microsoft Press, 1988) é um conjunto de 250 subprogramas e funções que lhe permitem estender a linguagem QBasic em seus programas. Recomendamos este livro para programadores Basic intermediários e avançados, envolvidos em projetos de programação maiores. O livro inclui informações sobre a programação com linguagens misturadas e o uso do mouse.

Microsoft QuickBASIC, 3ª ed., de Douglas Hergert (Microsoft Press, 1989), é um texto intermediário sobre a escrita de grandes programas, orientados por tarefa, em QuickBasic. Recomendamos este livro para programadores com experiência anterior em BASICA, GW-BASIC ou QBasic. O livro inclui informações sobre arquivos de acesso randômico e até mesmo *trapping*.

MS-DOS QBasic, de Kris Jamsa, da Microsoft Quick Reference Series (Microsoft Press, 1991), é um guia de referência prático para cada comando e função do QBasic.

Microsoft WordBasic Macro Primer, de Russell Borland (Microsoft Press, 1991), introduz os usuários na poderosa linguagem de macro tipo Basic do Microsoft Word for Windows e Microsoft Word for OS/2. Se estiver interessado em aproveitar ao máximo o seu processamento de texto com o Basic, este livro é para você.

Programmers at Work (Microsoft Press, 1989). Entrevistas de Susan Lammers com programadores de computadores pessoais que montaram a indústria, inclui uma conversa com Bill Gates, que escreveu a primeira versão do Basic para um microcomputador, em 1975.

The Waite Group's MS-DOS QBasic Programmer's Reference (Microsoft Press, 1991) é uma referência abrangente para cada palavra-chave e elemento da linguagem QBasic. Recomendamos este livro para programadores QBasic de iniciantes a profissionais.

The Waite Group's Microsoft QuickBASIC Primer Plus (Microsoft Press, 1990) é uma introdução completa à linguagem e compilador Microsoft QuickBasic. Este livro inclui informações sobre a criação de bibliotecas e arquivos executáveis do Quick, uso de portas seriais e impressoras e desenvolvimento de programas para o MS-DOS.

Usando MS-DOS, 5ª ed., versão 5.0 de Van Wolverton (Microsoft Press, publicado pela Editora Campus, 1991), é uma introdução geral aos microcomputadores e ao sistema operacional MS-DOS. Recomendamos este livro para o aprendizado dos comandos do DOS e a operação da família de computadores pessoais da IBM.

OUTRAS VERSÕES DO BASIC

O MS-DOS QBasic não é o único pacote de software para a programação Basic disponível pela Microsoft Corporation. Em termos de poder e características, o QBasic realmente encontra-se no final da família de produtos de programação Basic da Microsoft. À medida que as suas habilidades em programação crescerem, você desejará investigar outros pacotes de linguagem Basic da Microsoft:

- *The Microsoft QuickBasic Compiler*
  Esta versão completa do MS-DOS QBasic contém mais de duas dezenas de opções de menu adicionais, aceitando a depuração em alto nível, gerenciamento de arquivo e procedure e programação modular. A principal vantagem do QuickBasic Compiler é que ele pode criar bibliotecas e arquivos executáveis isolados, levando-o ao nível de outros produtos de linguagem de programação completos. O QuickBasic Compiler é o próximo passo lógico se você quiser continuar a programar em Basic.

- *Microsoft QuickBasic para o Apple Macintosh*
  Este compilador QuickBasic roda na família de microcomputadores Apple Macintosh. A versão do QuickBasic para o Macintosh é semelhante à versão do PC, mas foi adaptada para aceitar o ambiente operacional do Macintosh, controlado por menus e janelas.

- *The Microsoft Basic Professional Development System*
  Este compilador Basic avançado é voltado para as necessidades do programador Basic profissional. Ele inclui suporte para o sistema operacional OS/2 e grandes projetos de programação. Ele vem com uma cópia do Microsoft QuickBasic Compiler para lhe dar tudo de que poderá precisar para programar qualquer coisa em Basic.

O MICROSOFT QUICKBASIC COMPILER

O MS-DOS QBasic é um produto sólido e útil para se aprender a programar em Basic e para desenvolver suas próprias aplicações. Se quiser mais características, poderá considerar a aquisição do Microsoft QuickBasic Compiler. O Microsoft QuickBasic Compiler possui uma série de vantagens sobre o QBasic:

- Tem a capacidade de criar arquivos executáveis (.EXE) isolados.
- Tem a capacidade de criar bibliotecas de procedures muito usadas.
- Inclui suporte melhorado para a depuração.

- Aceita programas com múltiplos módulos.
- Oferece um maior controle sobre o ambiente de programação.
- Contém uma série de novas opções de menus.

Melhor que tudo isso, cada uma dessas características é integrada ao ambiente de programação familiar do QBasic, de modo que as habilidades de programação já adquiridas serão aplicáveis de imediato. Além disso, a linguagem de programação Basic é idêntica nos dois produtos, de modo que cada programa já escrito em QBasic rodará no compilador sem quaisquer modificações.

RESUMO

Neste capítulo procuramos informar o que você já aprendeu. Porém, mais importante do que isso, esperamos que este capítulo — e o livro inteiro — o tenha motivado a explorar ainda mais, a aprender o que ainda não foi ensinado. Talvez você tenha pego este livro pensando em escrever o maior programa do mundo, algum dia. Ou talvez quisesse apenas saber o que significa programar um computador. Não importa o motivo, as habilidades aprendidas aqui lhe servirão muito bem. Orgulhe-se disso: Você deu os primeiros passos exploratórios para um novo mundo. Apegue-se a eles! E programe, programe, programe!

APÊNDICE A

Uso de Menus e Opções do QBasic

Este apêndice é uma referência instantânea para o ambiente de programação controlado por menus no QBasic. Ele lhe será útil para

- Selecionar menus e comandos
- Usar quadros de diálogo
- Usar comandos de edição do teclado
- Selecionar texto
- Obter ajuda em linha
- Imprimir seus programas ou partes do sistema de ajuda
- Mudar as cores da tela
- Usar as opções de inicialização

Para obter uma ajuda mais completa, veja as opções *Survival Guide* e *Contents* do sistema de ajuda em linha do QBasic.

SELEÇÃO DE MENUS E COMANDOS

Os oito menus drop-down do QBasic contêm comandos para gerenciamento de arquivo, obtenção de ajuda e edição, execução e depuração de programas. Você pode selecionar menus e comandos usando o mouse ou o teclado.

Com o Mouse...

1. Mova o apontador do mouse diretamente sobre o nome do menu que deseja selecionar e clique com o botão esquerdo do mouse para arriar o menu que deseja.
2. Mova o apontador do mouse até o comando que deseja e clique no botão esquerdo novamente.

Com o Teclado...

1. Pressione a tecla Alt para ativar a barra de menu.
2. Pressione a tecla representando a primeira letra do nome do menu que deseja selecionar.
3. Use as teclas de direção para destacar o comando que deseja e depois pressione a tecla Enter. (Ou simplesmente pressione a letra que está destacada no nome do comando.)

USO DE QUADROS DE DIÁLOGO

Quando o QBasic precisar de informação adicional antes de executar um comando, mostrará um *quadro de diálogo*. A Figura A-1 mostra o quadro de diálogo que aparece quando você seleciona o comando Open do menu File. (O conteúdo do seu quadro de diálogo poderá ser diferente.)

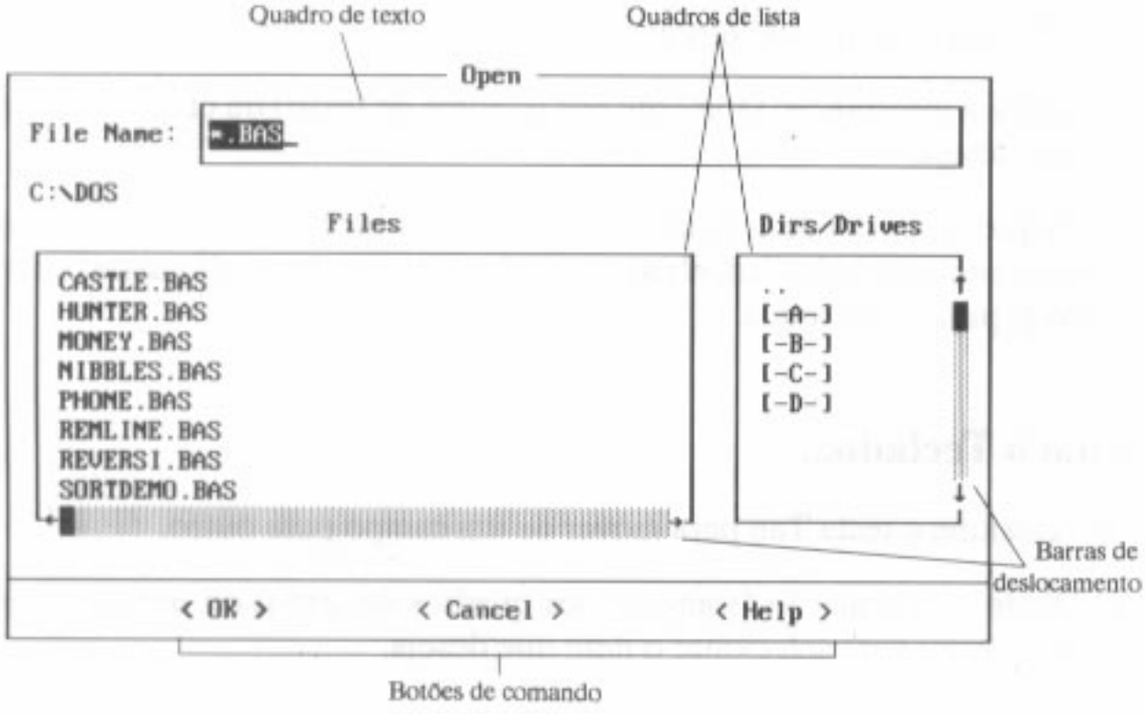


FIGURA A-1.
Os elementos do quadro de diálogo Open.

Um quadro de diálogo típico contém vários campos:

- **Quadros de texto** recebem informações que você digita pelo teclado, como nomes de arquivo ou padrões de pesquisa.
- **Quadros de lista** lhe permitem selecionar um item (como um nome de arquivo, um nome de diretório ou uma cor) a partir de uma série de escolhas. *Barras de deslocamento* lhe darão acesso a mais informações do que podem ser vistas dentro de um quadro de lista.
- **Quadros de tique** lhe permitem indicar se um elemento está em ação pela presença ou ausência de um "X" entre colchetes.
- **Botões de opção** são grupos de botões dos quais você só pode fazer uma seleção.

- **Botões de comando** controlam a operação de um quadro de diálogo. Você usa botões de comando para executar um comando, cancelar um comando ou obter ajuda.

Você poderá preencher campos do quadro de diálogo usando o mouse, o teclado ou uma combinação dos dois.

Com o Mouse...

1. Clique no campo que deseja modificar.
2. Digite a informação (somente nos quadros de texto) ou clique no item que deseja.
3. Clique no botão de comando *OK* para executar o comando, clique no botão de comando *Cancel* para cancelar o comando ou clique no botão *Help* para obter ajuda.

Com o Teclado...

1. Pressione a tecla Tab para mover de um campo para outro.
2. Digite a informação (somente nos quadros de texto) ou use as teclas de direção para selecionar o item que deseja.
3. Pressione Enter para executar o comando, ou pressione Esc para cancelar o comando.

Você pode usar as seguintes teclas dentro de um quadro de diálogo:

Tecla(s)	*Finalidade(s)*
Tab	Mover para o campo seguinte
Shift-Tab	Mover para o campo anterior
Teclas de direção	Selecionar item no quadro de lista; selecionar botão de opção; trocar quadro de tique
PgUp/PgDn	Rolar quadro de lista vários itens de cada vez
Barra de espaço	Trocar quadro de tique; executar botão de comando
F1	Obter instruções para preencher quadro de diálogo
Enter	Executar comando
Esc	Cancelar comando

Barras de Deslocamento

Barras de deslocamento lhe permitem ver mais texto do que aparece atualmente em um quadro de diálogo. Use as barras de deslocamento para fazer com que um texto escondido apareça.

Uso de barras de deslocamento com o mouse

Para usar as barras de deslocamento com o mouse, clique nas setas da barra para ver linhas de informação adicionais e clique na área entre as setas para mover rapidamente pela lista. Você também pode arrastar o quadro de deslocamento para mover até um local específico na lista.

Uso de barras de deslocamento com o teclado

Para usar as barras de deslocamento com o teclado, pressione as teclas de direção para ver linhas de informação adicionais e pressione PgUp ou PgDn para rolar muitas linhas de cada vez.

Você também pode mover diretamente até itens começando com uma letra em particular, pressionando a letra correspondente. Continue a pressionar essa letra para percorrer todas as entradas subseqüentes da lista começando com essa letra.

EQUIVALENTES NO TECLADO DOS COMANDOS DE MENU

O QBasic oferece teclas de atalho para a maior parte dos comandos de menu e para uma série de comandos que não possuem equivalentes no menu. A tabela a seguir descreve as teclas de atalho e suas funções.

Tecla(s)	*Finalidade*	*Menu/Comando*
F1	Mostra ajuda específica	Help/Topic
Shift-F1	Mostra ajuda geral	Help/Help
F2	Mostra lista de procedures	View/SUBs
Shift-F2	Mostra próxima procedure	Nenhum
Ctrl-F2	Mostra procedure anterior	Nenhum
F3	Acha próxima ocorrência da cadeia de pesquisa	Search/Repeat Last Find
F4	Mostra tela de saída	View/Output Screen
F5	Continua execução do programa	Run/Continue

Tecla(s)	*Finalidade*	*Menu/Comando*
Shift-F5	Executa programa do início	Run/Start
F6	Ativa janela seguinte	Nenhum
Shift-F6	Ativa janela anterior	Nenhum
F7	Executa programa até o cursor	Nenhum
F8	Executa próximo comando (passo)	Debug/Step
F9	Troca ponto de interrupção	Debug/Toggle Breakpoint
F10	Executa comando seguinte (salta procedure)	Debug/Procedure Step
Ctrl-F10	Troca entre tela completa e múltiplas janelas	Nenhum

COMANDOS DO TECLADO PARA EDIÇÃO

Juntamente com os comandos de edição no menu Edit, o QBasic aceita uma série de comandos de edição pelo teclado, sendo que a maior parte possui equivalentes no WordStar. A tabela a seguir descreve os comandos de edição mais significativos.

Tecla(s)	*Teclas do WordStar*	*Finalidade*
←	Ctrl-S	Move um caracter à esquerda
→	Ctrl-D	Move um caracter à direita
Ctrl-←	Ctrl-A	Move uma palavra à esquerda
Ctrl-→	Ctrl-F	Move uma palavra à direita
↑	Ctrl-E	Move uma linha para cima
↓	Ctrl-X	Move uma linha para baixo
Home	Ctrl-QS	Move para o primeiro caracter na linha
End	Ctrl-QD	Move para o último caracter na linha
Ctrl-Enter	Ctrl-J	Move para o início da linha seguinte
Ctrl-Home	Ctrl-QR	Move para o início da procedure
Ctrl-End	Ctrl-QC	Move para o final da procedure
Ins	Ctrl-V	Troca o modo de inserção
Shift-Ins	Nenhuma	Insere conteúdo do Clipboard no cursor
Shift-Del	Nenhuma	Retira texto selecionado (coloca no Clipboard)
Nenhuma	Ctrl-Y	Retira linha selecionada (coloca no Clipboard)

Tecla(s)	*Teclas do WordStar*	*Finalidade*
Nenhuma	Ctrl-QY	Retira do cursor até o fim da linha (coloca no Clipboard)
Del	Ctrl-G	Apaga o texto ou caracter selecionado no cursor
Nenhuma	Ctrl-T	Apaga a palavra no cursor
Shift-Tab	Nenhuma	Apaga um nível de endentação para as linhas selecionadas
Ctrl-Ins	Nenhuma	Copia texto selecionado para o Clipboard
PgUp	Ctrl-R	Rola uma página para cima
PgDn	Ctrl-C	Rola uma página para baixo
Ctrl-P	Nenhuma	Entra com um caracter de controle (digite Ctrl-P e depois o caracter de controle que deseja)

SELEÇÃO DE TEXTO

Selecionar o texto é o processo de destacar um bloco de texto com o teclado ou com o mouse. O texto selecionado pode ser usado em vários comandos do QBasic, inclusive nos comandos a seguir.

Menu/Comando	*Tecla(s)*	*Finalidade*
Edit/Cut	Shift-Del	Apaga o texto selecionado (coloca no Clipboard)
Edit/Copy	Ctrl-Ins	Copia o texto selecionado no Clipboard
Edit/Paste	Shift-Ins	Insere o conteúdo do Clipboard na posição do cursor.
File/Print	Nenhuma	Imprime o texto selecionado
Search/Find	Nenhuma	Procura um texto selecionado
Search/Change	Nenhuma	Procura e muda um texto selecionado
Edit/New SUB	Nenhuma	Usa o texto selecionado como nome para um novo subprograma
Edit/New Function	Nenhuma	Usa o texto selecionado como nome para uma nova função

NOTA: Tenha cuidado com o texto selecionado. Se digitar qualquer caracter (até mesmo um espaço) enquanto uma linha de texto está selecionada, tudo que for digitado substituirá todo o texto selecionado (que é perdido permanentemente). Para cancelar uma seleção (para "desselecioná-la"), clique o botão esquerdo do mouse ou pressione qualquer tecla de direção.

Você pode selecionar o texto usando o mouse ou o teclado.

Com o Mouse...

1. Coloque o apontador do mouse no primeiro caracter que deseja selecionar. (O "primeiro caracter" depende da direção em que você move o cursor — veja a tabela seguinte.)
2. Segure o botão esquerdo do mouse.
3. Mova o apontador do mouse para o último caracter que deseja selecionar.
4. Solte o botão do mouse.

Os movimentos do mouse têm os seguintes efeitos quando você seleciona o texto:

Movimento	*Efeito*
Duplo clique	Seleciona palavra no apontador do mouse
À direita uma coluna	Seleciona caracter no apontador do mouse
À esquerda uma coluna	Seleciona caracter à esquerda do apontador do mouse
Acima uma linha	Seleciona linha no apontador do mouse e a linha acima*
Abaixo uma linha	Seleciona linha no apontador do mouse e a linha abaixo†

* Se o apontador do mouse começar na primeira coluna, somente a linha acima do ponto de partida será selecionada.

† Se o apontador do mouse começar na primeira coluna, somente a linha contendo o ponto inicial será selecionada.

Com o Teclado...

1. Use as teclas de direção para mover o cursor até o primeiro caracter que deseja selecionar.
2. Segure a tecla Shift.
3. Use as teclas de direção para mover o cursor até o último caracter que deseja selecionar.
4. Solte a tecla Shift.

As teclas de direção têm os seguintes efeitos quando você seleciona o texto:

Tecla(s) de direção	*Efeito*
→	Seleciona caracter no cursor
←	Seleciona caracter à esquerda do cursor
↑	Seleciona linha no cursor e acima*
↓	Seleciona linha no cursor e abaixo†
PgUp	Seleciona linha no cursor e uma tela acima
PgDn	Seleciona linha no cursor e uma tela abaixo
Home	Seleciona a partir do caracter à esquerda do cursor até o início da linha
End	Seleciona a partir do cursor até o final da linha
Ctrl-Home	Seleciona a partir da linha do cursor até o início da procedure
Ctrl-End	Seleciona a partir da linha do cursor até o final da procedure

* Se o apontador do mouse começar na primeira coluna, somente a linha acima do ponto de partida será selecionada.

† Se o apontador do mouse começar na primeira coluna, somente a linha contendo o ponto inicial será selecionada.

AJUDA EM LINHA

O QBasic possui uma documentação em linha útil para palavras-chave e elementos da linguagem QBasic, comandos de menu, mensagens de erro e tópicos gerais de programação. Você poderá obter ajuda em linha de três maneiras:

- Para obter ajuda com o seu programa, coloque o cursor na palavra-chave ou elemento da linguagem sobre o qual deseja ver a documentação e pressione a tecla F1. Se estiver usando um mouse, clique o botão direito do mouse na palavra-chave ou elemento da linguagem.
- Para obter ajuda com um comando de menu ou mensagem de erro, pressione F1 quando o comando de menu estiver destacado ou quando a mensagem de erro for mostrada.
- Para obter ajuda com um tópico de programação geral ou com o próprio sistema de ajuda, pressione Alt-H para escolher o menu Help e depois selecione um dos seguintes comandos:
  - *Index* para ver uma lista alfabética das palavras-chave e assuntos do QBasic

- *Contents* para ver uma lista dos tópicos gerais de programação, incluindo o uso do QBasic, elementos da linguagem QBasic e informações técnicas sobre o QBasic
- *Topic* para ver a documentação em linha para a palavra-chave do QBasic onde o cursor se encontra
- *Using Help* para ver informações sobre o sistema de ajuda

O sistema de ajuda do QBasic é composto de elementos interconectados: Cada quadro de diálogo de ajuda contém *hyperlinks* [conexões associadas], que o levarão a outras partes do sistema de ajuda. Para selecionar um desses hyperlinks com o mouse, clique duas vezes sobre ele. Para selecionar um deles com o teclado, pressione Tab até que o cursor esteja sobre o hyperlink que deseja e depois pressione Enter.

Se o texto na janela Help for maior do que uma tela, use PgUp e PgDn ou as barras de deslocamento para ver as partes que estão fora da tela. Para passar do sistema de ajuda para a janela View, e daí para a janela Immediate, pressione F6.

Quando acabar de trabalhar com o sistema de ajuda, pressione Esc para sair.

NOTA: Se o seu computador apitar quando você pressiona F1, é porque não existe ajuda em linha para a palavra onde o cursor se encontra.

Cópia de Programas pelo Sistema de Ajuda

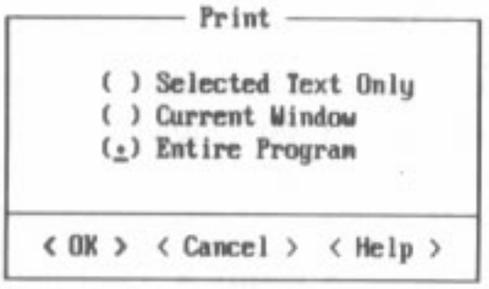
Para copiar um código de exemplo do sistema de ajuda para o seu programa:

1. Localize o código de exemplo que deseja copiar e selecione-o.
2. Escolha Copy pelo menu Edit.
3. Pressione Esc para fechar a janela Help.
4. Mova o cursor até o ponto onde deseja que o código de programa apareça.
5. Escolha Paste pelo menu Edit.

IMPRESSÃO

Se houver uma impressora ligada ao sistema, você poderá imprimir todo ou parte do programa com o comando Print no menu File.

1. Selecione o comando Print. Aparecerá o seguinte quadro de diálogo:



2. Selecione a opção apropriada:

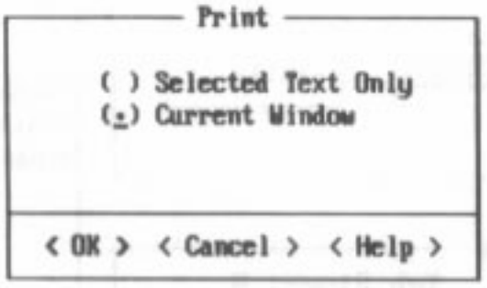
   - *Selected Text Only* imprime apenas o texto selecionado. Antes de usar esta opção, você deverá selecionar, no seu programa, o texto que deseja imprimir.
   - *Current Window* imprime o conteúdo somente da janela ativa — útil para imprimir o programa principal, uma procedure ou o conteúdo da janela Immediate.
   - *Entire Program* imprime o programa inteiro, incluindo todos os subprogramas e funções.

3. Pressione Enter ou clique em *OK* para executar o comando Print.

Impressão de Parte do Sistema de Ajuda

Você também pode usar o comando PRINT para imprimir partes selecionadas do sistema de ajuda.

1. Entre no sistema de ajuda e localize a entrada que deseja imprimir.
2. Selecione o comando Print. Aparecerá na tela o seguinte quadro de diálogo:



3. Escolha a opção apropriada.

   - *Selected Text Only* imprime apenas o texto selecionado. Antes de usar esta opção, você precisa selecionar a parte da documentação que deseja imprimir.
   - *Current Window* imprime a entrada de ajuda inteira.

4. Pressione Enter ou clique em *OK* para executar o comando Print.

ALTERAÇÃO DAS CORES DA TELA

O QBasic lhe permite alterar as cores de primeiro e segundo planos na tela, além de modificar outras características do ambiente, adequando-as às suas preferências pessoais. O seu adaptador de vídeo e monitor indicarão as cores que podem ser usadas.

1. Escolha o comando Display no menu Options. A Figura A-2 mostra o quadro de diálogo Display, que lhe permite mudar as cores de primeiro e segundo planos para três elementos na tela:

   - *Normal text* para alterar as cores do texto normal da tela.
   - *Current statement* para alterar as cores do próximo comando a ser executado pelo QBasic.
   - *Breakpoint lines* para alterar as cores das linhas designadas como pontos de interrupção pelo comando Toggle Breakpoint do Debug.

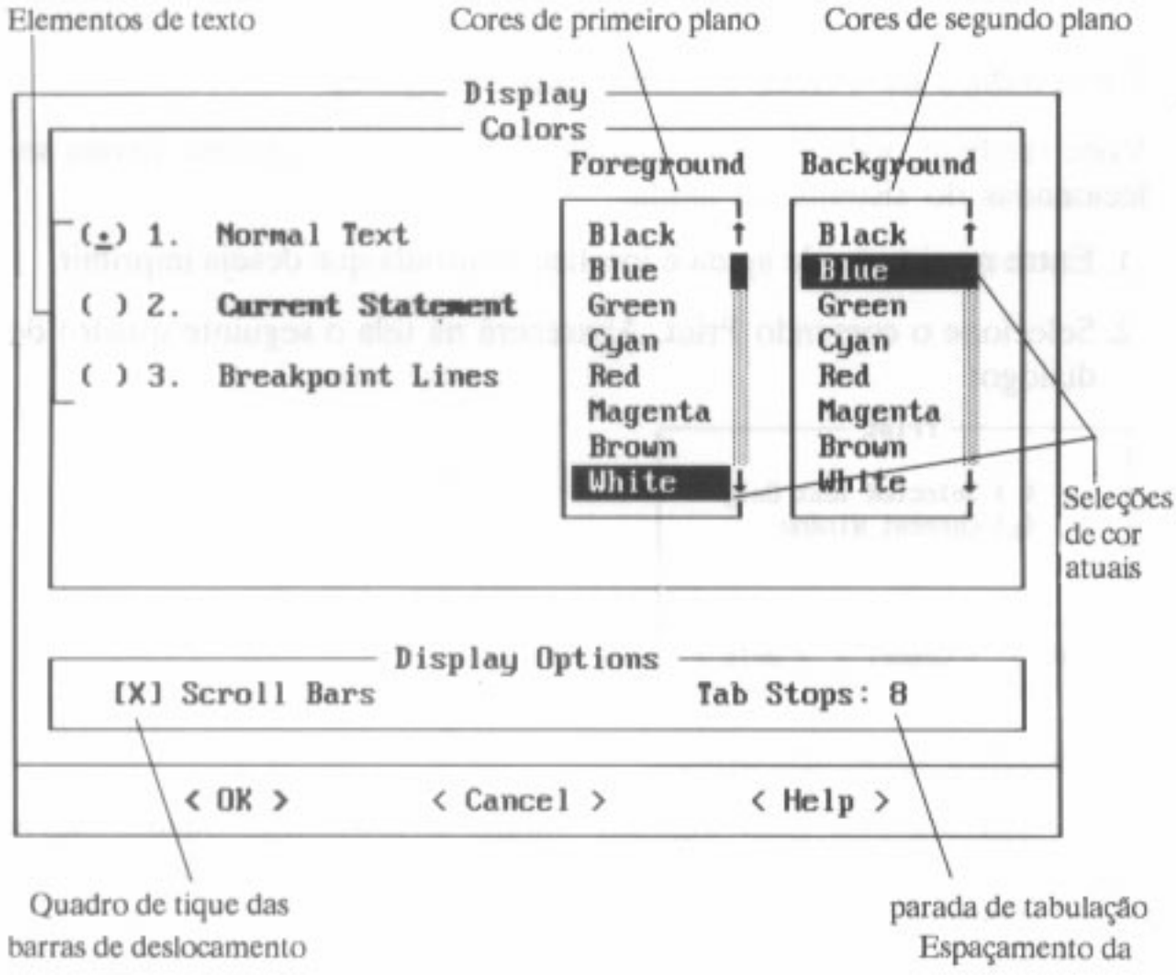


FIGURA A-2.
Os elementos do quadro de diálogo Display.

2. Escolha o elemento de texto para o qual deseja alterar as cores.

3. Escolha as cores que deseja para o primeiro e segundo planos.

4. Pressione Enter ou clique em *OK* para executar o comando Display.

NOTA: Se você escolher cores diferentes dos valores default fornecidos pelo QBasic, resista à tentação de combinar cores brilhantes ou contrastantes. Com o tempo, seus olhos lhe agradecerão pela limitação!

A parte Display Options do quadro de diálogo lhe dá as opções de remover as barras de deslocamento da tela e ajustar o espaçamento das suas paradas de tabulação. Removendo as barras de deslocamento, você poderá ver mais do programa de cada vez; alterando *Tab Stops* para um valor menor do que 8, poderá colocar mais código na tela e nas suas listagens (supondo que use a tecla Tab para endentar as linhas do seu programa).

OPÇÕES DE PARTIDA

Quando iniciar o QBasic, você poderá fazer com que execute várias tarefas especiais simplesmente incluindo algumas informações extras, conhecidas como *opções da linha de comando*, ao comando padrão do QBasic. A tabela a seguir mostra uma lista parcial dessas opções. Para obter uma lista completa, veja a parte *QBasic Command Line* [Linha de Comando do QBasic] do sistema de ajuda.

Opção	*Descrição*	*Exemplo*
programa	Faz com que o QBasic carregue o programa indicado	qbasic reversi
/run *programa*	Faz com que o QBasic carregue e execute o programa indicado antes de mostrar o código	qbasic /run reversi
/b	Roda o QBasic em modo preto-e-branco	qbasic /b
/ed	Roda o QBasic em modo de documento (idêntico ao programa EDIT do MS-DOS)	qbasic /ed
/h	Roda o QBasic com um número máximo de linhas por tela que o seu hardware pode mostrar (em geral, 25, 43 ou 50)	qbasic /h

APÊNDICE B

Conjunto de Caracteres ASCII e Estendido da IBM

As Tabelas B-1 e B-2 mostram os 256 caracteres disponíveis à maioria dos computadores pessoais que rodam o MS-DOS. A Tabela B-1 mostra o conjunto de caracteres ASCII, e a Tabela B-2 mostra o conjunto de caracteres estendido da IBM. Você poderá mostrar a maior parte desses caracteres usando PRINT e outros comandos do QBasic, podendo compará-los usando os operadores relacionais do QBasic. O Capítulo 9 descreve com maiores detalhes algumas técnicas de exibição e comparação.

Tabela B-1: O conjunto de caracteres ASCII (códigos 0-127)

Caracter	*ASCII*	*Caracter*	*ASCII*	*Caracter*	*ASCII*	*Caracter*	*ASCII*
	0	<space>	32	@	64	`	96
☺	1	!	33	A	65	a	97
●	2	"	34	B	66	b	98
♥	3	#	35	C	67	c	99
♦	4	$	36	D	68	d	100
♣	5	%	37	E	69	e	101
♠	6	&	38	F	70	f	102
•	7	'	39	G	71	g	103
◘	8	(	40	H	72	h	104
○	9	)	41	I	73	i	105
◙	10	*	42	J	74	j	106
♂	11	+	43	K	75	k	107
♀	12	,	44	L	76	l	108
♪	13	-	45	M	77	m	109
♫	14	.	46	N	78	n	110
☼	15	/	47	O	79	o	111
►	16	0	48	P	80	p	112
◄	17	1	49	Q	81	q	113
↕	18	2	50	R	82	r	114
‼	19	3	51	S	83	s	115
¶	20	4	52	T	84	t	116
§	21	5	53	U	85	u	117
▬	22	6	54	V	86	v	118
↨	23	7	55	W	87	w	119
↑	24	8	56	X	88	x	120
↓	25	9	57	Y	89	y	121
→	26	:	58	Z	90	z	122
←	27	;	59	[	91	{	123
∟	28	<	60	\	92	¦	124
↔	29	=	61	]	93	}	125
▲	30	>	62	^	94	~	126
▼	31	?	63	_	95	⌂	127

Tabela B-2: O conjunto de caracteres estendido da IBM (códigos 128-255)

Caracter	*ASCII*	*Caracter*	*ASCII*	*Caracter*	*ASCII*	*Caracter*	*ASCII*
Ç	128	á	160	@	192	a	224
ü	129	í	161	A	193	b	225
é	130	ó	162	B	194	G	226
â	131	ú	163	C	195	p	227
ä	132	ñ	164	D	196	(	228
à	133	Ñ	165	E	197	s	229
å	134	ª	166	F	198	m	230
ç	135	º	167	G	199	t	231
ê	136	¿	168	H	200	F	232
ë	137	⌐	169	I	201	U	233
è	138	¬	170	J	202	V	234
ï	139	½	171	K	203	d	235
î	140	¼	172	L	204	`	236
ì	141	¡	173	M	205	f	237
Ä	142	«	174	N	206	e	238
Å	143	»	175	O	207	∩	239
É	144	4	176	P	208	[	240
æ	145	7	177	Q	209	6	241
Æ	146	9	178	R	210	≥	242
ô	147	3	179	S	211	≤	243
ö	148	4	180	T	212	⌠	244
ò	149	5	181	U	213	⌡	245
û	150	6	182	V	214	4	246
ù	151	7	183	W	215	'	247
ÿ	152	8	184	X	216	°	248
Ö	153	9	185	Y	217	•	249
Ü	154	:	186	Z	218	·	250
¢	155	;	187	[	219	√	251
£	156	<	188	\	220	h	252
¥	157	=	189	]	221	²	253
₧	158	>	190	^	222	▪	254
ƒ	159	?	191	_	223		255

Alguns dos caracteres ASCII (de 0 a 32, além do 127) são chamados *caracteres de controle*. Você os utiliza para executar tarefas especiais, como soar a "campainha" do computador ou avançar o papel na impressora. A Tabela B-3 lista os caracteres de controle mais usados.

Tabela B-3: Caracteres de controle ASCII mais usados

Código ASCII	*Caracter de controle*	*Descrição*
7	Campainha	Emite um som pelo alto-falante
8	Retrocesso	Recua um caracter na tela
9	Tabulação	Move uma parada de tabulação à direita na tela
10	Mudança de linha	Avança uma linha na impressora
12	Mudança de página	Avança uma página na impressora
13	Retorno de carro	Avança o cursor até a próxima linha na tela

APÊNDICE C

Comandos e Funções do QBasic

Neste apêndice você encontrará o nome de cada comando e função disponível no QBasic. Para ver a documentação completa para cada um desses itens, incluindo os programas de exemplo, inicie o QBasic, digite o nome do item e pressione a tecla F1.

COMANDOS

BEEP
BLOAD
BSAVE
CALL
CALL ABSOLUTE
CHAIN
CHDIR
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COM
COMMON
CONST
DATA
DATE$
DECLARE
DEF
DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR
DIM
DO...LOOP
DRAW
END
ENVIRON
ERASE
ERROR
EXIT
FIELD
FILES
FOR...NEXT
FUNCTION
GET
GOSUB
GOTO
IF...THEN...ELSE
INPUT
INPUT#
IOCTL
KEY
KILL
LET
LINE
LINE INPUT
LINE INPUT#
LOCATE
LOCK...UNLOCK
LPRINT
LPRINT USING
LSET
MID$
MKDIR
NAME
ON ERROR
ON
OPEN
OPTION BASE
OUT
PAINT
PALETTE
PCOPY
PEN
PLAY
PMAP
POKE
PRESET
PRINT
PRINT USING
PRINT# USING
PSET
PUT
RANDOMIZE
READ
REDIM
REM
RESET
RESTORE
RESUME
RETURN
RMDIR
RSET
RUN
SCREEN
SEEK
SELECT CASE
SHARED
SHELL
SLEEP
SOUND
STATIC
STOP
STRIG

SUB
SWAP
SYSTEM
TIME$
TIMER
TROFF
TRON
TYPE
UNLOCK
VIEW
WAIT
WHILE...WEND
WIDTH
WINDOW
WRITE

FUNÇÕES

ABS
ASC
ATN
CDBL
CHR$
CINT
CLNG
COS
CSNG
CSRLIN
CVD
CVDMBF
CVI
CVL
CVS
CVSMBF
DATE$
ENVIRON$
EOF
ERDEV
ERDEV$
ERL
ERR
EXP
FILEATTR
FIX
FRE
FREEFILE
HEX$
INKEY$
INP
INPUT$
INSTR
INT
IOCTL$
LBOUND
LCASE$
LEFT$
LEN
LOC
LOF
LOG
LPOS
LTRIM$
MID$
MKDMBF$
MKD$
MKI$
MKL$
MKSMBF$
MKS$
OCT$
PEEK
PEN
POINT
POS
RIGHT$
RND
RTRIM$
SCREEN
SEEK
SGN
SIN
SPACE$
SPC
SQR
STICK
STR$
STRIG
STRING$
TAB
TAN
TIME$
UBOUND
UCASE$
VAL
VARPTR
VARPTR$
VARSEG

APÊNDICE D

Conversão de Programas do GW-BASIC para o QBasic

Neste apêndice, descreveremos as diferenças entre a linguagem de programação GW-BASIC fornecida com as versões anteriores do MS-DOS e a linguagem de programação QBasic, fornecida com a versão 5 do MS-DOS, dando dicas para a conversão de programas existentes em GW-BASIC para que rodem sob o MS-DOS QBasic. Você precisa desta informação se tiver programas do GW-BASIC que queira rodar ou melhorar sob o QBasic. A boa notícia é que o QBasic é muito compatível com o GW-BASIC, de modo que a maior parte dos programas do GW-BASIC rodará no QBasic sem modificação. Todavia, às vezes você terá que modificar os seus programas do GW-BASIC para que rodem corretamente. Neste apêndice, você terá a base de que necessita para iniciar o processo.

VERSÕES ANTERIORES DO BASIC

Uma "velha" tradição da indústria de computadores pessoais é incluir uma versão do Basic com o MS-DOS. Os distribuidores do MS-DOS reconheceram desde o início da vida do microcomputador que haveria ocasiões em que nenhum programa disponível comercialmente atenderia as necessidades do usuário do computador, e que de vez em quando as pessoas precisariam escrever seus próprios programas. Desde que o IBM PC foi lançado em agosto de 1981, a Microsoft, a IBM e outros OEMs (fabricantes de equipamento original) entregam uma versão ou outra do Basic com o MS-DOS.

No início, os limites do hardware do PC significavam que as primeiras versões do Basic entregues com o DOS — BASIC, GW-BASIC, BASICA e outros Basics com nomes dos seus fabricantes — tinham que ser edições compactas da linguagem. Com o passar dos anos, o Basic evoluiu até se tornar um ambiente de programação que se compara às características avançadas das linguagens de programação profissionais e às capacidades de edição de poderosos processadores de texto. O QBasic contém quase todas as características do poderoso Microsoft QuickBasic Compiler — o produto de linguagem mais vendido da Microsoft —, sendo ainda compatível com as versões anteriores do Basic.

NOTA: Muitas implementações da linguagem Basic, ligeiramente diferentes, foram distribuídas com o hardware e software de microcomputador com o passar dos anos. Por questão de simplicidade, chamaremos de GW-BASIC *todas as versões do Basic distribuídas com o MS-DOS antes da versão 5. Consulte a sua documentação original do Basic para ver como a sua versão anterior difere do QBasic.*

Qual É a Diferença?

O MS-DOS QBasic difere do GW-BASIC de várias maneiras importantes. As características avançadas do QBasic, e mais a compatibilidade do QBasic com versões anteriores do Basic, tornam-no um sucessor extraordinário do GW-BASIC.

- **O QBasic é uma linguagem de programação mais rica.** O QBasic contém mais comandos, funções e outros elementos de linguagem do que o GW-BASIC, dando-lhe um controle maior sobre o seu computador e permitindo a criação de programas mais eficientes.
- **O QBasic possui um ambiente de edição melhor.** O ambiente de programação controlado por menus do QBasic possui muitas das características de processamento de texto encontradas em editores de programa e processadores de texto avançados. Além disso, o QBasic aceita comandos de edição por teclado e mouse, contendo uma seleção de poderosas ferramentas de depuração.
- **O QBasic possui documentação em linha.** O sistema de ajuda em linha do QBasic oferece uma documentação completa da linguagem e normas para uso do ambiente de edição do QBasic.
- **O QBasic é mais rápido do que o GW-BASIC.** O interpretador *threaded p-code* avançado do QBasic permite que ele rode programas em Basic muitas vezes mais rápido do que o GW-BASIC. E as ferramentas de programação fornecidas com o QBasic tornam o próprio processo de desenvolvimento (escrita, edição e depuração de programas) muito mais rápido do que no GW-BASIC.

EXECUÇÃO DE PROGRAMAS DO GW-BASIC SOB O QBASIC

Para rodar um programa do GW-BASIC sob o QBasic, você salva o programa em formato ASCII, abre o programa dentro do QBasic e executa-o. A maior parte dos programas do GW-BASIC rodará corretamente sob o QBasic sem qualquer modificação — a única diferença que você notará é que o programa do GW-BASIC rodará um pouco mais rápido sob o QBasic.

Se o QBasic não puder rodar o programa do GW-BASIC, ele mostrará uma mensagem *sintax error* em um quadro de diálogo próximo ao primeiro problema que achar. Se isso acontecer, você terá que modificar o código do GW-BASIC para que se adapte às regras da linguagem QBasic.

Como Salvar o Programa em GW-BASIC no Formato ASCII

Todos os programas em QBasic devem estar em formato ASCII (texto puro). Por default, o editor de programas do GW-BASIC armazena os programas em um formato comprimido especial que o QBasic não pode ler, de modo que deverá usar a opção A do comando SAVE do GW-BASIC para salvar o programa antes de carregá-lo no QBasic. Para salvar um programa do GW-BASIC em formato ASCII, siga estes passos:

1. Localize uma cópia do GW-BASIC (ou seu equivalente) e execute-a. (O MS-DOS 5 não vem com uma cópia do GW-BASIC — procure-o em uma versão anterior do MS-DOS.)
2. Use o comando LOAD para carregar o programa GW-BASIC que desejar rodar.
3. Use o comando LIST para verificar o conteúdo do programa.
4. Use o comando SAVE e a opção A para salvar o programa. Por exemplo, para salvar um programa chamado INTEREST.BAS em formato ASCII, digite o seguinte comando no GW-BASIC:

   ```
   save "interest.bas", a
   ```

5. Saia do GW-BASIC digitando o seguinte comando no aviso de comando do GW-BASIC:

   ```
   system
   ```

A tela superior da página seguinte mostra o processo de carga, listagem, salvamento e saída em ação com os programas BASICA.EXE e INTEREST.BAS que acompanham o COMPAQ DOS 3.31.

Execução do Programa do GW-BASIC no QBasic

Após ter salvo o programa do GW-BASIC em formato ASCII, você poderá carregá-lo e executá-lo no QBasic. Para carregar o programa INTEREST.BAS, por exemplo, digite o seguinte no aviso de comando do MS-DOS:

```
C:\>qbasic interest.bas
```

Você verá uma tela semelhante à tela inferior da página seguinte.

```
The COMPAQ Personal Computer BASIC
Version 3.31

(C) Copyright Compaq Computer Corp. 1982, 1987
(C) Copyright Microsoft Corp. 1983, 1987
60133 Bytes free
Ok
load "interest.bas"
Ok
list
10 INPUT "PRINCIPAL?                $",PRINCIPAL
20 INPUT "ANNUAL INTEREST RATE(%)? ",ANNUALRATE
30 INPUT "NUMBER OF MONTHS?         ",MONTHS
40 MONTHRATE = ANNUALRATE/12
50 FACTOR = (MONTHRATE/100)+1
60 PRINT "THE COMPOUNDING FACTOR IS "FACTOR
70 FOR N = 1 TO MONTHS
80 PRINCIPAL = PRINCIPAL * FACTOR
90 PRINT "AFTER MONTH "N"THE NEW PRINCIPAL IS $"PRINCIPAL
100 NEXT N
Ok
save "interest.bas",a
Ok
system
1LIST  2RUN+  3LOAD"  4SAVE"  5CONT+  6,"LPT1 7TRON+  8TROFF+ 9KEY    0SCREEN
```

```
File  Edit  View  Search  Run  Debug  Options                    Help
                        INTEREST.BAS
10 INPUT "PRINCIPAL?                $", PRINCIPAL
20 INPUT "ANNUAL INTEREST RATE(%)? ", ANNUALRATE
30 INPUT "NUMBER OF MONTHS?         ", MONTHS
40 MONTHRATE = ANNUALRATE / 12
50 FACTOR = (MONTHRATE / 100) + 1
60 PRINT "THE COMPOUNDING FACTOR IS "; FACTOR
70 FOR N = 1 TO MONTHS
80 PRINCIPAL = PRINCIPAL * FACTOR
90 PRINT "AFTER MONTH "; N; "THE NEW PRINCIPAL IS $"; PRINCIPAL
100 NEXT N

                          Immediate
<Shift+F1=Help> <F6=Window> <F2=Subs> <F5=Run> <F8=Step>      00001:001
```

Se a listagem ficar confusa e tiver vários caracteres de controle e caracteres do conjunto estendido da IBM, o programa provavelmente não estará no formato ASCII. Saia do QBasic e salve o programa em formato ASCII, como explicamos na página 382.

Para rodar o programa INTEREST.BAS dentro do QBasic, arrie o menu Run e selecione Start. Se o programa for totalmente compatível com o QBasic, você verá a saída do programa.

E Se Não Rodar?

Se o programa do GW-BASIC tiver um erro de sintaxe, a execução parará imediatamente e uma mensagem de erro mostrará a natureza e o local do problema. Consertar um erro de sintaxe pode ser fácil, mas normalmente exige um bom conhecimento da linguagem QBasic e boas habilidades em investigação. O melhor local para iniciar a coleta de informações é o sistema de ajuda em linha do QBasic, que contém informações detalhadas sobre cada tipo de erro de sintaxe e a documentação completa para cada elemento da linguagem QBasic. Para obter informações detalhadas sobre o erro de sintaxe encontrado, pressione F1 enquanto o quadro de diálogo do erro ainda aparece na tela. Depois de entender a natureza do problema, coloque o cursor no comando ou função associado ao erro e pressione F1 para ver a documentação em linha para ele. Ao ver esta informação, você deverá saber como corrigir o erro e fazer com que o programa em GW-BASIC continue rodando.

Conserto de Erros Típicos

Embora o processo de conversão de programas em GW-BASIC para rodar sob o QBasic possa parecer mais artístico do que científico, alguns problemas costumam surgir amiúde. Nas seções seguintes, veremos os problemas mais comuns e como consertá-los.

Erros de sintaxe em geral

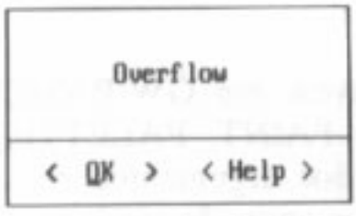
Às vezes o erro de sintaxe que interrompe um programa é uma simples incompatibilidade na pontuação entre elementos de comando (uso incompatível de vírgula, dois pontos, ponto-e-vírgula ou a tecla Enter). Às vezes o erro é uma indicação de que a sintaxe da linguagem GW-BASIC é ligeiramente diferente da sintaxe da linguagem QBasic. Para consertar esse erro de compatibilidade, verifique a documentação em linha para o elemento da linguagem em questão e compare-o de perto com a sintaxe do elemento da linguagem GW-BASIC contendo o erro. Em geral, a diferença na sintaxe será aparente, e você verá como modificar o comando do GW-BASIC para que siga as regras do QBasic.

Antes de Experimentar

Se você não escreveu o programa GW-BASIC em questão (e portanto não sabe exatamente o que o programa faz no interior), pode ter alguma dificuldade para consertar um problema de incompatibilidade. Não há problema em experimentar, mas, antes de fazer isso, não deixe de fazer uma cópia de reserva do programa original como medida de segurança, caso o problema se torne ainda pior. Também é bom gastar algum tempo estudando uma listagem do programa, para determinar exatamente o que o código faz, antes de tentar uma conversão detalhada.

Tipos de variáveis

Observe que os tamanhos e tipos de variáveis, sobretudo inteiras, variam entre as versões do Basic. Se o QBasic tiver um problema com o tratamento das variáveis em um programa, você verá uma mensagem de erro. A mensagem a seguir, por exemplo, aparece se uma variável inteira normal for muito pequena para conter o número a ela atribuído:



Se isso acontecer, mude o tipo da variável inteira para *long* a fim de conter a informação extra. O comando a seguir, válido em um programa BASIC da Hewlett-Packard (HP), gera um erro de estouro em QBasic:

```
NUM% = 620 * 400
```

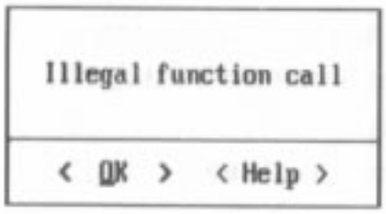
Quando você converter o programa parao QBasic, terá que mudar o tipo da variável para *long* para permitir o valor mais alto:

```
NUM& = 620 * 400
```

Quando mudar uma variável, lembre-se de modificar o tipo em todos os lugares do programa — o QBasic mantém separadas as variáveis com o mesmo nome mas com tipos diferentes.

Modo de vídeo correto

Muitos programas em GW-BASIC, geralmente os jogos de vídeo mais antigos, foram projetados para aceitar modos de vídeo especiais não aceitos pelo QBasic ou pelo adaptador de vídeo em um computador comum. Se você rodar um programa que aceite modos de vídeo não aceitos pelo QBasic ou pelo adaptador de vídeo do seu computador, verá a seguinte mensagem de erro:



Para consertar este problema, você precisa fazer com que o programa aceite um modo de vídeo também aceito pelo QBasic e compatível com o adaptador de vídeo do seu computador. (Veja no Capítulo 11, "Uso de Gráficos e Som", maiores informações sobre os adaptadores de vídeo e o comando SCREEN.) Observe que se o seu programa em GW-BASIC utilizar muitos comandos gráficos, você poderá ter que converter algumas coordenadas de tela no programa para que combinem com o novo modo de vídeo que o programa passará a aceitar.

Cores corretas

As cores de primeiro e segundo planos disponíveis, em GW-BASIC e em QBasic, são controladas pelos comandos PAINT, PALETTE, SCREEN e COLOR. As faixas e significados dos argumentos para esses comandos foram alterados com o passar dos anos, de modo que alguns programas em GW-BASIC poderão mostrar cores diferentes quando você os rodar sob o QBasic. Consulte a documentação em linha do QBasic para PAINT, PALETTE, SCREEN e COLOR, a fim de atualizar o programa com os valores de cor desejados. Observe que, à medida que o hardware de vídeo do computador continua a evoluir, valores adicionais de cores e modos de vídeo provavelmente serão adicionados a esses comandos, dando-lhe um maior controle sobre a aparência do seu programa.

Ele Roda Mesmo Tão Rápido?

Um problema comum com programas mais antigos em GW-BASIC, sobretudo com jogos de vídeo, é que sob o QBasic os programas podem rodar *muito mais rápido*. A velocidade torna-se um problema quando uma aplicação foi projetada para rodar em uma determinada velocidade em um certo computador e um computador ou interpre-

tador de programa mais rápido aparece e muda as condições de execução. Isso certamente ocorre em alguns programas em GW-BASIC quando rodam em computadores mais rápidos e modernos sob o QBasic.

Se você tiver que diminuir a velocidade de um programa que foi escrito especificamente para um computador mais antigo e para o interpretador GW-BASIC mais lento, inclua um ou mais *laços de tempo* no programa. Um laço de tempo é um laço FOR que se repete muitas vezes em um programa sem fazer nada além de passar o tempo — o laço de tempo simplesmente atrasa a ação para dar ao usuário um tempo para absorver o que aconteceu na tela. Um laço de tempo comum possui o valor inicial *1* e um valor final grande:

```
FOR I% = 1 TO 2000
NEXT I%
```

Durante o laço, nada acontece, mas quando ele termina, já se passou um tempo suficiente para que a velocidade pareça ter diminuído.

Se você escolher usar um laço de tempo, coloque-o no seu programa logo após o código que mostra a saída na tela (a saída que o usuário deverá ter tempo para absorver). Pode ser preciso testar um pouco para chegar ao valor correto do laço, dependendo da velocidade do seu computador e do efeito que deseja dar ao laço.

TIRANDO PROVEITO DO QBASIC

Fazer com que os seus programas do GW-BASIC rodem sem um erro no QBasic é apenas o primeiro passo para tirar o máximo proveito do QBasic. Depois que o programa estiver rodando sem problemas, gaste algum tempo aprendendo a usar as novas e poderosas características da moderna e estruturada linguagem QBasic. Essas características não somente tornam a programação em QBasic mais fácil e mais rápida de aprender, mas também tornam os seus programas mais fáceis de entender e revisar. As seguintes características do QBasic (não disponíveis no GW-BASIC) já foram discutidas neste livro. Use-as para melhorar os seus programas sempre que precisar.

- **Tipos estendidos de variável e constante.** O QBasic aceita nomes de variáveis com até 40 caracteres em qualquer combinação de maiúsculas e minúsculas. O QBasic também aceita variáveis *inteiras longas*, variáveis de *cadeia de tamanho fixo* e *constantes* numéricas e de cadeia. (Para maiores informações sobre esses tipos de variável e constantes, veja o Capítulo 4, "Variáveis e Operadores do QBasic".)

- **Novas estruturas de decisão.** Comandos IF multilinha são novos no QBasic e podem conter um número ilimitado de cláusulas ELSEIF para ajudar a controlar o caminho tomado pelo seu programa. O comando SELECT CASE oferece um novo meio de

realizar o desvio condicional com base no valor de uma variável. (Para maiores informações sobre esses comandos, veja o Capítulo 5, "Controle de Fluxo do Programa".)

- **Laços melhorados.** Além do laço FOR, o QBasic aceita o laço DO para a execução de um bloco de código repetidamente até que uma certa condição seja atendida. (Para maiores informações sobre o laço DO, veja o Capítulo 6, "Uso de Laços do QBasic".)
- **Subprogramas e funções definidos pelo usuário.** O QBasic aceita blocos de código em múltiplas linhas (com variáveis locais, independentes) chamados *subprogramas* e *funções*, que podem ser executados várias vezes sem qualquer redigitação. Os subprogramas e funções definidas pelo usuário substituem as rotinas GOSUB e as funções DEF FN de única linha normalmente encontrados em programas GW-BASIC. (Para maiores informações, veja o Capítulo 7, "Criação de Subprogramas e Funções".)
- **Melhorias de estilo e de ambiente.** O QBasic inclui um verificador de sintaxe embutido que avisa sobre erros de sintaxe assim que forem cometidos, ajudando-o a organizar e tornar seus programas mais fáceis de ler, passando palavras-chave do QBasic para maiúsculas, endentando os laços automaticamente e forçando a coerência nos nomes de variáveis. O QBasic não precisa de números de linha (o GW-BASIC sim), de modo que você pode construir programas fáceis de ler com base em estruturas de decisão, laços e procedures cuidadosamente planejados — sem o código "espaguete" de GOTO e GOSUB típido de programas em GW-BASIC. (Para maiores informações, veja o Capítulo 2, "Primeiros Passos com o QBasic".)

APÊNDICE E

Respostas às Perguntas e Exercícios

Este apêndice contém as respostas às perguntas e exercícios ao final dos Capítulos de 2 a 12.

CAPÍTULO 2

1. Digite *qbasic* no aviso de comando do sistema.
2. Falso.
3. A janela View contém o seu programa e é a principal janela no QBasic. A janela Immediate lhe permite testar as linhas antes de usá-las no seu programa.
4. O comando New apaga a janela View e prepara o QBasic para um novo programa. O comando Open o levará ao quadro de diálogo Open para que possa carregar um programa existente no disco.
5. A janela View mostra o programa em que está trabalhando. A tela de saída mostra a saída do programa quando você roda o programa.
6. O comando Cut apaga um bloco de texto selecionado da janela View e o coloca no Clipboard. Copy coloca uma cópia do bloco de texto selecionado no Clipboard sem remover o texto original da janela View.
7. Você pode apagar um texto de um programa das seguintes maneiras:
   - □ Com o comando Cut do menu Edit (texto selecionado)
   - □ Com a tecla Del (único caracter ou texto selecionado)
   - □ Com a tecla Backspace (único caracter)
   - □ Com a combinação de tecla Ctrl-Y (linha)
8. O cursor de inserção inclui texto em uma linha sem escrever sobre os caracteres existentes. O cursor de superposição substitui os caracteres existentes pelos novos caracteres digitados. A diferença visual entre os dois é que o cursor de inserção aparece como um sublinhado piscante e o cursor de superposição aparece como um retângulo piscante. Alterne entre os dois cursores pressionando a tecla Ins.
9. Oito.
10. Os seguintes nomes de arquivo são válidos:

Nome inválido	*Motivo*
IMPRINOME.BAS	Muito grande (IMPRINOME tem 9 caracteres)
BOM+MAU.BAS	Caracter + não permitido
BOM MAU.BAS	Espaço não permitido
HOM-E-MUL.BAS	Muito grande (HOM-E-MUL tem 9 caracteres)
PROG[1].BAS	Caracteres *[* e *]* não permitidos
PROG*1.BAS	Curinga * não permitido

11. Selecione Exit pelo menu File para sair do QBasic. Se o seu arquivo tiver mudanças não salvas, o QBasic mostrará um quadro de diálogo contendo as seguintes opções: *Yes* (salvar o programa), *No* (desconsiderar as mudanças não salvas), *Cancel* (retornar ao QBasic) e *Help* (mostrar ajuda).

CAPÍTULO 3

1. Um comando opera de forma direta, em geral produzindo resultados óbvios ou visíveis (um grupo de caracteres na tela, por exemplo). Uma função geralmente faz seu trabalho no fundo e retorna um valor ao programa, que poderá ser usado como argumento para um comando.
2. BEEP — comando
   CLS — comando
   DATE$ — função
   PRINT — comando
   TIME$ — função
3. Um item entre colchetes é opcional. O caracter | significa que você pode escolher apenas um dos valores entre colchetes.
4. Um argumento é uma informação adicional dada a um comando ou a uma função.
5. Uma cadeia é um grupo de caracteres (que poderá incluir letras, números e símbolos) rodeados por aspas. Uma expressão numérica é um número ou variável numérica (ou, como vimos no Capítulo 4, qualquer expressão que produza um resultado numérico).
6. A saída dos dois comandos PRINT é igual.
7. Tanto um ponto-e-vírgula quanto uma vírgula ao final de um comando PRINT faz com que a saída do próximo comando PRINT seja mostrada na mesma linha. Um ponto-e-vírgula faz com que a saída do próximo comando PRINT apareça imediatamente após a saída do comando PRINT corrente. Uma vírgula faz com que o próximo comando PRINT apareça na próxima zona de impressão.

CAPÍTULO 4

1. Inteira normal, inteira longa, ponto flutuante com precisão simples e ponto flutuante com precisão dupla. Elas diferem no tipo e no tamanho do número que podem conter e nos seus caracteres de declaração de tipo.
2. Reservar um espaço para um sinal de menos no caso do número ser ou poder se tornar um valor negativo.

3. Uma vírgula inválida provavelmente apareceu dentro de um número do QBasic.

4. Um número fora da faixa do tipo de dado numérico de uma variável foi atribuído à variável.

5. a. Inteira normal
   b. Ponto flutuante com precisão simples
   c. Ponto flutuante com precisão simples
   d. Ponto flutuante com precisão dupla
   e. Inteira normal ou ponto flutuante com precisão simples
   f. Ponto flutuante com precisão simples
   g. Inteira longa
   h. Ponto flutuante com precisão dupla
   i. Ponto flutuante com precisão simples

6. A divisão normal retorna um resultado fracionário; a divisão inteira retorna apenas um inteiro, descartando qualquer resto; a divisão de resto retorna apenas o resto.

7. Exponenciação (^); multiplicação e divisão (*, /, \, MOD); adição e subtração (+, –).

8. 300.

9. Uma solução possível para este problema é o programa CALC.BAS:

```
' CALC.BAS
' Este programa calcula e imprime quatro fórmulas.

CLS

PRINT "ABS(-10) + 5 ="; ABS(-10) + 5
PRINT "SQR(36) ="; SQR(36)
PRINT "SQR(4) ^ 2 ="; SQR(4) ^ 2
PRINT "COS(3.14) ="; COS(3.141592654#)
```

10. Uma solução possível para este problema é o programa CIRCULO .BAS:

```
' CIRCULO.BAS
' Este programa mostra a circunferência de um
'  círculo quando o raio é fornecido pelo usuário.

CONST PI# = 3.141592654#     ' declara constante

CLS                          ' apaga a tela

PRINT "Este programa calcula a circunferência de ";
```

(*continua*)

continuação

```
PRINT "um círculo a partir do seu raio."
PRINT                          ' entrada do usuário
INPUT "Entre com o raio do círculo: ", raio!

circum! = 2 * PI# * raio!' calcula circunferência

PRINT                          ' mostra resultado
PRINT "A circunferência do círculo é"; circum!
```

CAPÍTULO 5

1. Uma expressão numérica usa operadores e gera um resultado numérico. Uma expressão condicional usa operadores condicionais e gera um resultado verdadeiro ou falso.
2. b, c, f, i, j, k.
3. Verdadeiro.
4. AND exige que as duas condições sejam verdadeiras para que uma ação seja tomada; OR exige que pelo menos uma delas seja verdadeira para que uma ação seja tomada.
5. ELSE lhe permite executar um bloco de comandos quando uma expressão condicional em um comando IF é avaliada como falsa. ELSE é o oposto de THEN.
6. ELSEIF lhe permite avaliar outra condição após um comando IF ou ELSEIF anterior ter sido avaliado como falso. A palavra-chave THEN deverá aparecer na mesma linha de ELSEIF.
7. Uma solução possível para este problema é o programa PERGUNTA.BAS:

```
' PERGUNTA.BAS
' Este programa faz uma pergunta sobre programaçãc
'  em Basic.

CLS

INPUT "Está gostando de programar em Basic até aqui
(S/N)? ", resp$
PRINT

IF (resp$ = "S") OR (resp$ = "s") THEN
```

(continua)

continuação

```
  PRINT "Muito bem! Há muita diversão a caminho!"
ELSEIF (resp$ = "N") OR (resp$ = "n") THEN
  PRINT "Sinto muito. Não se preocupe - - vai
melhorar!"
ELSSE
  PRINT "Favor rodar o programa novamente."
  PRINT "Ao pedido, entre com 'S' para Sim ou 'N'
para não."
END IF
```

8. O QBasic avalia a expressão que aparece após o SELECT CASE e depois verifica nas cláusulas CASE seguintes um valor combinando. O QBasic executa os comandos após a cláusula CASE cujos valores combinam com a expressão no comando SELECT CASE.

9. CASE ELSE lhe permite especificar um bloco de comandos que será executado se todas as condições CASE em um comando SELECT CASE forem avaliadas como falsas.

10. IS lhe permite usar uma expressão condicional em um comando CASE. TO lhe permite especificar uma faixa de valores numéricos em um comando CASE.

11. Uma solução possível para este problema é o programa ESTADOS.BAS:

```
' ESTADOS.BAS
' Este programa mostra informações sobre um
'  dentre três estados.

CLS

PRINT "Sobre qual dos seguintes estados gostaria de
estudar?"
PRINT
PRINT "  1) Rio de Janeiro"
PRINT "  2) São Paulo"
PRINT "  3) Bahia"
PRINT
INPUT "Estado (1-3): ", resp%
PRINT

SELECT CASE resp%
  CASE 1
    PRINT "** Rio de Janeiro **"
```

(continua)

continuação

```
      PRINT "   Capital: Rio de Janeiro"
      PRINT "   Região: Sudeste"
      PRINT "   Principais produtos: laranja, cana"
   CASE 2
      PRINT "** São Paulo **"
      PRINT "   Capital: São Paulo"
      PRINT "   Região: Sudeste"
      PRINT "   Principais produtos: café, soja"
   CASE 3
      PRINT "** Bahia **"
      PRINT "   Capital: Salvador"
      PRINT "   Região: Nordeste"
      PRINT "   Principais produtos: cacau, dendê"
   CASE ELSE
      PRINT "Favor rodar o programa novamente e ";
      PRINT "escolher um número de 1 a 3."
END SELECT
```

CAPÍTULO 6

1. A variável contadora identifica o *valor* do laço — um inteiro entre os limites *inicial* e *final* do laço.

2. *início* e *fim* podem ser constantes numéricas, variáveis numéricas ou expressões numéricas. Os valores podem ser positivos, negativos ou zero.

3. 4.

4. O comando SOUND faz com que o alto-falante do computador emita um som com a freqüência e duração especificadas.

5. Um laço endentado é um laço FOR, WHILE ou DO dentro de outro laço FOR, WHILE ou DO.

6. Colocar a condição no topo de um laço DO assegura que a condição deva ser atendida antes que o laço seja executado. Colocar a condição no final de um laço DO assegura que os comandos no laço serão executados pelo menos uma vez.

7. Use um laço FOR quando quiser executar um bloco de comandos por um número específico de vezes. Use um laço WHILE ou DO para executar um bloco de comandos com base no valor de uma condição.

8. Uma solução possível para este problema é o programa GASTO-GAS.BAS:

```
' GASTOGAS.BAS
' Este programa usa um laço FOR para acompanhar os
'  gastos semanais com gasolina.

CLS

PRINT "Para cada um dos sete dias da semana, ";
PRINT "entre com o valor gasto com gasolina."
PRINT

FOR dia% = 1 TO 7
  PRINT "  Valor gasto no dia"; dia%;
  INPUT "- - > $", totalDia!
  totalSemana! = totalSemana! + totalDia!
NEXT dia%

PRINT
PRINT "Puxa! $"; totalSemana!; "de gasolina em uma
semana!"
```

9. Uma solução possível para este problema é o programa SONS.BAS:

```
' SONS.BAS
' Este programa toca uma nota com base em valores de
'  freqüência e duração entrados pelo usuário.

CLS

PRINT "Entre com valores de freqüência e duração
para o"
PRINT "som que deseja ouvir. Termine entrando com
-999."
PRINT

DO
  INPUT "Freqüência (37-32767): ", freq%
  IF (freq% <> -999) THEN
     INPUT "Duração (0-65535): ", duração&
     SOUND freq%, duração&
     PRINT
  END IF
LOOP UNTIL (freq% = -999)
```

10. Uma solução possível para este problema é o programa ROLADADO.BAS:

```
' ROLADADO.BAS
' Este programa rola um dado simulado 10 vezes e
'  mostra "Boa jogada!" se o dado mostrar 6.

CLS

INPUT "Pressione Enter para rolar o dado 10 vezes.
Pense em seis...", algo$
PRINT

RANDOMIZE TIMER

FOR i% = 1 TO 10
  rola% = INT(RND * 6) + 1
  PRINT "Número: "; rola%
  IF (rola% = 6) THEN
     COLOR 2
     PRINT "Boa jogada!"
     COLOR 7
  ELSE
     PRINT
  END IF
NEXT i%
```

CAPÍTULO 7

1. Todas são vantagens.
2. O subprograma *EntraNome$* possui um caracter de declaração de cadeia ($). Isso não é correto — somente as funções são marcadas pelo tipo de dados que retornarão. O comando SUB correto seria o seguinte:

   ```
   SUB EntraNome (nome$, sobrenome$)
   ```

3. F2.
4. O comando Procedure Step no menu Debug (F10).
5. COMMON SHARED total%
6. Um subprograma lhe permite executar uma série de tarefas ou enviar uma série de valores ao programa principal. Uma função lhe permite executar uma tarefa menor e mais específica, retornando um valor ao programa principal.
7. O subprograma *PegaDadosCarro* pode ser escrito da seguinte forma:

   ```
   SUB PegaDadosCarro (marca$, modelo$, ano%, cor$)

   PRINT "Favor entrar com dados sobre o carro"
   PRINT
   ```

```
INPUT " Marca do carro: ", marca$
INPUT " Modelo do carro: ", modelo$
INPUT " Ano do carro: ", ano%
INPUT " Cor do carro: ", cor$

END SUB
```

Um comando que chama *PegaDadosCarro* poderia ser escrito da seguinte forma:

```
PegaDadosCarro marcaCarro$, modeloCarro$, anoCarro%, corCarro$
```

8. A função *AchaMaior%* pode ser escrita da seguinte forma:

```
FUNCTION AchaMaior% (int1%, int2%)

IF (int1% >= int2%) THEN
    AchaMaior% = int1%
ELSE
    AchaMaior% = int2%
END IF

END FUNCTION
```

Um comando que chama *AchaMaior%* poderia ser escrito da seguinte forma:

```
PRINT "O maior inteiro é"; AchaMaior%(primeiro%, segundo%)
```

9. Uma solução possível para este problema é o programa FORMAS.BAS:

```
' FORMAS.BAS
' Este programa mostra uma forma preenchida com o
' seu caracter favorito.

' declara constantes usadas como argumentos no
' comando COLOR

CONST CIANO% = 3
CONST BRANCO% = 7

' declara subprogramas antes de usar; os nomes e os
'  parâmetros devem combinar com os dos subprogramas

DECLARE SUB PegaForma (simbolo$, escolha%)
DECLARE SUB MostraLinha (car$)
DECLARE SUB MostraRetang (car$)
DECLARE SUB MostraTriang (car$)
```

(*continua*)

continuação

```
CLS

' chama subprograma PegaForma para obter
'  caracter e forma

PegaForma caracter$, forma%    'passa dois argumentos

PRINT
PRINT
COLOR CIANO%

' usa CASE para chamar subprograma solicitado

SELECT CASE forma%
  CASE 1          ' forma% = 1, mostra um triângulo
    MostraTriang caracter$
  CASE 2          ' forma% = 2, mostra um retângulo
    MostraRetang caracter$
  CASE 3          ' forma% = 3, mostra uma linha
    MostraLinha caracter$
END SELECT

COLOR BRANCO%

END

SUB PegaForma (simbolo$, escolha%)

' O subprograma PegaForma pede um símbolo e uma
'  forma ao usuário e retorna ao programa principal
'  nas variáveis simbolo$ e escolha%.

PRINT "Este programa mostra um grupo de caracteres ";
PRINT "na forma especificada."
PRINT
INPUT "Que caracter gostaria de usar? ", simbolo$
PRINT
PRINT "Que forma gostaria de ver na tela?"
PRINT
PRINT "    1) Triângulo"
PRINT "    2) Retângulo"
PRINT "    3) Linha"
```

(*continua*)

continuação

```
PRINT

DO   ' pede até que a escolha esteja na faixa
  INPUT "Forma (1, 2 ou 3): ", escolha%
LOOP WHILE (escolha% < 1) OR (escolha% > 3)

END SUB ' fim do subprograma - - volta ao principal

SUB MostraLinha (car$)

' O subprograma MostraLinha recebe um argumento do
'  programa principal e usa-o para mostrar uma linha
'  com 30 caracteres de extensão.

CONST TAMANHO% = 30     ' define o tamanho da linha

FOR i% = 1 TO TAMANHO% ' mostra o caracter 30 vezes
  PRINT car$;          ' usa o ponto e vírgula para
NEXT i%                '  imprimir um após o outro

PRINT

END SUB

SUB MostraRetang (car$)

' O subprograma MostraRetang recebe um argumento do
'  programa principal e o utiliza para mostrar um
'  retângulo de 50 caracteres por 7 de altura.

CONST TAMANHO% = 50       ' define a extensão em 50
CONST ALTURA% = 7         ' define a altura em 7

FOR i% = 1 TO ALTURA%     ' para cada uma das 7 linhas
  FOR j% = 1 TO TAMANHO% '  mostra 50 caracteres
    PRINT car$;
  NEXT j%
  PRINT                   ' retorna após cada linha
NEXT i%

END SUB

SUB MostraTriang (car$)
```

(continua)

continuação

```
' O subprograma MostraTriang recebe um argumento do
'  programa principal, usando-o para mostrar um
'  triângulo equilátero. A função Tab move o cursor
'  para o local de coluna correto.

CONST LINHAS% = 10      ' define número de linhas em 10
esq% = LINHAS%          ' usa esq% para o lado esquerdo
dir% = LINHAS% + 1      ' usa dir% para o lado direito

FOR contaLin% = 1 TO LINHAS%     ' para cada linha
  FOR i% = esq% TO LINHAS%
    PRINT TAB(i%); car$;         ' mostra lado esquerdo
  NEXT i%

  FOR i% = LINHAS+1 TO dir%-1    ' mostra lado direito
    PRINT TAB(i%); car$
  NEXT i%

  PRINT        ' retorna ao final da linha
  ' primeiro caracter na linha seguinte começará um
  '  espaço mais à esquerda e terminará um espaço a
  '  mais em direção à margem direita
  esq% = esq% - 1
  dir% = dir% + 1
NEXT contaLin%

END SUB
```

CAPÍTULO 8

1. Comandos READ aparecem primeiro, mas por convenção.
2. DATA, READ e RESTORE funcionam melhor com dados das seguintes categorias:
   - □ Dados que você já conhece em avanço (antes do programa ser rodado)
   - □ Dados que sempre aparecem na mesma ordem
   - □ Dados que podem ser repetidos várias vezes, como os dias da semana
3. Uma solução possível para este problema é o programa NOMES.BAS:

```
' NOMES.BAS
' Este programa lê os nomes de Beto e seus amigos
'  em um comando DATA.

CLS

FOR i% = 1 TO 7
  READ pal$
  PRINT pal$
NEXT i%

DATA Beto, Walter, Lúcia, Wander, Gustavo, Edu, Laura
```

4. Falso.

5. DIM números!(99)

6. Um marcador de fim-de-dados é um número ou cadeia indicando que não existem mais itens de dados em uma lista qualquer.

7. Uma solução possível para este problema é o programa DALLAS.BAS:

```
' DALLAS.BAS
' Este programa usa um vetor unidimensional de
'  cadeia para guardar nomes e personagens do
'  filme "Dallas".

OPTION BASE 1       ' base dos vetores em 1

CLS

PRINT "** Este programa reúne nomes de personagens
do ";
PRINT "filme Dallas **"
PRINT
INPUT "Quantos nomes gostaria de entrar? ", nomes$

DIM personagens$(nomes%)     ' dimensiona vetor

PRINT                        ' preenche vetor dinâmico
FOR i% = 1 TO nomes%
  INPUT "  Nome: ", personagens$(i%)
NEXT i%

PRINT
PRINT "Você entrou com os seguintes nomes:"
```

(*continua*)

continuação

```
PRINT

FOR i% = 1 TO nome%     ' mostra conteúdo do vetor
  PRINT personagens$(i%)
NEXT i%
```

8. Um erro do tipo "fora da faixa" é uma tentativa do programa referenciar um elemento que não existe em um vetor.

9. Uma solução possível para este problema é o programa TENIS.BAS:

```
' TENIS.BAS
' Este programa mantém o marcador de um jogo de
'  tênis com um vetor bidimensional chamado escore%.

OPTION BASE 1          ' define primeiro elemento em 1
DIM escore%(2, 5)      ' dimensiona vetor de 2x5
                       ' para o escore

' pega nomes e nacionalidades dos jogadores

visita$ = "John Davis": nacionalVis$ = "Estados
Unidos"
casa$ = "Pedro da Silva": nacionalCasa$ = "Brasil"

CLS

PRINT "Entre com os escores de cada jogador em um
jogo de tênis de cinco sets."
PRINT

FOR set% = 1 TO 5   ' pega pontos de cada set
  PRINT "Set"; set%; "- ->  "; visita$;
  INPUT ; ": ", escore%(1, set%)
  PRINT "    "; casa$;
  INPUT ": ", escore%(2, set%)
        ' ... acumula total para cada jogador
  escoreVis% = escoreVis% + escore%(1, set%)
  escoreCasa% = escoreCasa% + escore%(2, set%)
NEXT set%

' determina o ganhador do jogo e mostra resultados

PRINT
COLOR 2' define cor verde para "Notícias do Dia"
```

(continua)

continuação

```
IF (escoreVis% > escoreCasa%) THEN
  PRINT "Notícias do Dia: "; nacionalVis$; " vence ";
nacionalCasa$;
  PRINT escoreVis%; "sets a"; escoreCasa%
ELSEIF (escoreCasa% > escoreVis%) THEN
  PRINT "Notícias do Dia: "; nacionalCasa$; " vence
"; nacionalVis$;
  PRINT escoreCasa%; "sets a"; escoreVis%
ELSE
  PRINT "Notícias do Dia: "; nacionalVis$; " empata
com "; nacionalCasa$;
  PRINT escoreVis%; "a"; escoreCasa%
END IF

COLOR 7        ' volta à cor branca

' mostra o placar final

PRINT
PRINT "Set        1   2   3   4   5"
PRINT "- - - - - - - - - - - - - - -"

FOR jogador% = 1 TO 2 ' para cada jogador
  IF (jogador% = 1) THEN PRINT visita$, ELSE PRINT
casa$,
  FOR set% = 1 TO 5    ' ... e para cada set no jogo
    PRINT escore%(jogador%, set%); " ";
  NEXT set%            ' mostra o número de pontos
  PRINT
NEXT jogador%
```

CAPÍTULO 9

1. Verdadeiro.
2. Falso. A declaração correta não possui o caracter de declaração de tipo *$*:

   DIM sobrenome AS STRING * 20

3. UMDOISTRÊS
4. a, b, c, d.
5. O valor 0 de INSTR significa um dos seguintes casos:
   - *cadeiapesquisa* não foi encontrada em *cadeiabase*
   - *início* é maior do que o tamanho da *cadeiabase*

- *cadeiabase* não possui caracteres

6. H

7. 77.

8. Uma solução possível para este problema é o programa PEGANOME.BAS:

```
' PEGANOME.BAS
' Este programa pega o primeiro e último nomes do
'  usuário e os mostra em maiúsculas.

CLS

INPUT "Primeiro nome: ", nome$
INPUT "Sobrenome:     ", sobrenome$
PRINT
PRINT UCASE(sobrenome$); ", "; UCASE$(nome$)
```

9. Uma solução possível para este problema é o programa INVERTE.BAS:

```
' INVERTE.BAS
' Este programa inverte a ordem dos caracteres
'  de uma cadeia.

CLS
                              ' pega cadeia do usuário
INPUT "Entre com uma cadeia a ser convertida: ",
cadeia$
tamanho% = LEN(cadeia$)       ' acha tamanho da cadeia

' retrocede na cadeia e extrai uma letra de cada vez
'  a fim de montar a nova cadeia
FOR i% = tamanho% TO 1 STEP -1
  carTemp$ = MID$(cadeia$, i%, 1)
  inverso$ = inverso$ + carTemp$
NEXT i%

PRINT                         ' mostra a nova cadeia
PRINT "Os caracteres em ordem invertida são ";
inverso$
```

10. Uma solução possível para este problema é o programa DIVIDE.BAS:

```
' DIVIDE.BAS
' Este programa divide uma cadeia em três partes.

CONST ESPACO$ = " "     ' declara constante de cadeia
car$ =  ""              ' inicializa variáveis
contaCar% = 1
contaNome% = 0

CLS                     ' apaga a tela

PRINT "Entre com o nome neste formato: Primeiro
Meio Último"
INPUT "Nome: ", nomeTodo$     ' pega nome em 3 partes
tamanho% = LEN(nomeTodo$)     ' acha tamánho do nome

' laço até que a cadeia inteira tenha sido analisada

DO WHILE (contaCar% <> tamanho% + 1)

' lê caracteres um de cada vez até achar um espaço
'  ou o final da cadeia; atribui caracteres a nome$

  DO WHILE (car$ <> ESPACO$) AND (contaCar% <>
(tamanho% + 1))
    car$ = MID$(nomeTodo$, contaCar%, 1)
    nome$ = nome$ + car$
    ' acompanha número de caracteres lidos
    contaCar% = contaCar% + 1
  LOOP

  car$ = ""                    ' resseta car$
  contaNome% = contaNome%+1  ' incrementa contaNome$

  ' atribui cadeia às variáveis de nome com base
  '  no valor de contaNome%
  SELECT CASE contaNome%
    CASE 1
       primNome$ = nome$
    CASE 2
       nomeMeio$ = nome$
    CASE 3
       ultNome$ = nome$
  END SELECT
  nome$ = ""                     resseta nome$
LOOP
```

(*continua*)

continuação

```
PRINT
PRINT "Resultados do processo de separação:"
PRINT
PRINT "Primeiro nome: "; primNome$     ' mostra
PRINT "Segundo nome: "; nomeMeio$      ' resultados
PRINT "Último nome: "; ultNome$
```

CAPÍTULO 10

1. OUTPUT apaga o conteúdo do arquivo existente; APPEND adiciona dados ao final do arquivo existente.
2. d.
3. Falso.
4. LINE INPUT# é mais útil do que INPUT# quando você precisa ler longas linhas de dados de cadeia ou linhas contendo vírgulas em um arquivo.
5. Um argumento literal para o comando SHELL deve ser colocado entre aspas. O comando correto seria o seguinte:

   SHELL "copy teste.txt teste2.txt"

6. Cadeia achada!
7. Uma solução possível para este problema é o programa CIDADE.BAS:

```
' CIDADE.BAS
' Este programa armazena uma lista de cidades em um
'  arquivo seqüencial.

' abre arquivo no drive/diretório corrente
OPEN "CIDADE.TXT" FOR OUTPUT AS #1

CLS

PRINT "Este programa armazena nomes de cidade no
arquivo CIDADE.TXT"
PRINT "Entre com suas cidades favoritas e digite
FIM para terminar."
PRINT

' pega nomes até o usuário entrar com FIM, gravando
'  os nomes no arquivo
DO WHILE (cidade$ <> "FIM")
```

(*continua*)

continuação

```
  INPUT " Nome da cidade: "; cidade$
  IF (cidade$ <> "FIM") THEN PRINT#1, cidade$
LOOP

CLOSE #1              ' fecha o arquivo

PRINT
INPUT "Pressione Enter para ver as cidades. ", algo$
PRINT

' abre o arquivo para entrada dos nomes
OPEN "CIDADE.TXT" FOR INPUT AS #1

' lê e imprime os nomes até chegar ao fim do arquivo
DO WHILE (NOT EOF(1))
  INPUT #1, cidade$
  PRINT cidade$
LOOP

CLOSE #1              ' fecha o arquivo
```

8. Uma solução possível para este problema é o programa ORDLISTA.BAS:

```
' ORDLISTA.BAS
' Este programa pega nomes e endereços do usuário,
'  ordena-os alfabeticamente por sobrenome e
'  armazena-os no arquivo seqüencial NOMES.TXT.

DECLARE SUB IncluiNomes ()       ' declara procedures
DECLARE FUNCTION NumeroDeNomes% ()
DECLARE SUB ArquivoParaVetor (nomes$(),
enderecos$(), numItens%)
DECLARE SUB ShellSort (nomes$(), enderecos$(),
numElementos%)
DECLARE SUB VetorParaArquivo (nomes$(),
enderecos$(), numItens%)
DECLARE SUB MostraNovoArq ()

OPTION BASE 1         ' define base dos vetores em 1

CLS                   ' apaga a tela

IncluiNomes           ' chama rotina para entrada
```

(continua)

continuação

```
numNomes% = NumeroDeNomes%    ' chama função para
                              '  contar nomes

' dimensiona vetores de nomes e endereços
DIM nomes$(numNomes%)
DIM endereços$(numNomes%)

' copia nomes e endereços no arquivo para vetores,
'  preparando para a ordenação
ArquivoParaVetor nomes$(), enderecos$(), numNomes%

' ordena vetores nomes$ e enderecos$ por sobrenome
ShellSort nomes$(), enderecos$(), numNomes%

' copia vetores ordenados de volta ao arquivo
VetorParaArquivo nomes$(), enderecos$(), numNomes%

' mostra o novo arquivo na tela
MostraNovoArq

END

SUB IncluiNomes

' abre como APPEND para não perder nomes existentes
OPEN "NOMES.TXT" FOR APPEND AS #1

PRINT "Este programa inclui nomes e endereços ao
arquivo"
PRINT "NOMES.TXT e ordena o arquivo alfabeticamente.
PRINT
PRINT "Entre com nomes no formato Sobrenome, Nome.
Digite FIM para terminar."
PRINT

' pega nomes até o usuário entrar com FIM
DO WHILE (nomeTodo$ <> "FIM")
  LINE INPUT " Nome (Último, Primeiro): "; nomeTodo$
  IF (nomeTodo$ <> "FIM") THEN
     PRINT #1, nomeTodo$      ' grava dados
     ' usa LINE INPUT para permitir vírgulas
     LINE INPUT " Endereço: "; endereco$
     PRINT #1, endereco$
END IF
```

(continua)

continuação

```
  PRINT
LOOP

CLOSE #1

END SUB

SUB VetorParaArquivo (nomes$(), enderecos$(),
numItens%)

' abre como OUTPUT para gravar sobre arquivo
'  fora de ordem
OPEN "NOMES.TXT" FOR OUTPUT AS #1

FOR i% = 1 TO numItens%
  PRINT #1, nomes$(i%) ' grava vetor no arquivo
  PRINT #1, enderecos$(i%)
NEXT i%

CLOSE #1

END SUB

SUB ArquivoParaVetor (nomes$(), enderecos$(),
numItens%)

' abre como INPUT para apanhar as entradas
OPEN "NOMES.TXT" FOR INPUT AS #1

FOR i% = 1 TO numItens%
  LINE INPUT #1, nomes$(i%)  ' lê conteúdo do arquivo
  LINE INPUT #1, enderecos$(i%)
NEXT i%

CLOSE #1

END SUB

SUB MostraNovoArq

INPUT "Pressione Enter para ver NOMES.TXT. ", algo$
PRINT
```

(*continua*)

continuação

```
' abre como INPUT para apanhar as entradas
OPEN "NOMES.TXT" FOR INPUT AS #1

' lê o arquivo de dados até o seu final e mostra
'  os dados na tela
DO WHILE (NOT EOF(1))
  LINE INPUT #1, nomeTodo$
  LINE INPUT #1, endereco$
  PRINT nomeTodo$; " - - "; endereco$
LOOP

CLOSE #1

END SUB

FUNCTION NumeroDeNomes%

' abre como INPUT para apanhar as entradas
OPEN "NOMES.TXT" FOR INPUT AS #1

conta% =          ' variável para contar os itens

' lê nomes e endereços até o final do arquivo
DO WHILE (NOT EOF(1))
  LINE INPUT #1, nomeTodo$
  LINE INPUT #1, endereco$
  conta% = conta% + 1      ' incrementa o contador
LOOP

CLOSE #1

' retorna contador de itens ao programa principal
NumeroDeNomes% = conta%

END FUNCTION

SUB ShellSort (nomes$(), enderecos$(),
numElementos%)
' Há uma discussão sobre o Shell Sort no Capítulo 9
' Observe que esta versão ordena dois vetores com
'  base no conteúdo de nomes$().

faixa% = numElementos% \ 2
```

(*continua*)

continuação

```
DO WHILE (faixa% > 0)
  FOR i% = faixa% TO numElementos% - 1
   j% = i% - faixa% + 1
   FOR j% = (i% - faixa% + 1) TO 1 STEP -faixa%
     IF nomes$(j%) <= nomes$(j%+faixa%) THEN EXIT FOR
     ' troca elementos fora de ordem
     SWAP nomes$(j%), nomes$(j% + faixa%)
     SWAP enderecos$(j%), enderecos$(j%+faixa%)
   NEXT j%
  NEXT i%

  faixa% = faixa% \ 2
LOOP

END SUB
```

CAPÍTULO 11

1. Os dois componentes são o adaptador de vídeo e o monitor. O adaptador de vídeo determina os modos de vídeo, gera o texto e os gráficos a serem mostrados e gera as cores. O monitor simplesmente mostra o que o adaptador de vídeo lhe enviar.
2. No modo de texto, o adaptador de vídeo só pode mostrar caracteres alfanuméricos. No modo gráfico, o adaptador de vídeo pode mostrar texto e formas gráficas (pixels, linhas, quadros, círculos, etc.)
3. LOCATE lhe permite posicionar o cursor de texto na tela de saída.
4. Uma solução possível para este problema é o programa MOVENOME.BAS:

```
' MOVENOME.BAS
' Este programa "move" um nome pela tela.

CONST ESP% = 400

CLS

INPUT "Favor entrar com o seu nome: ", nome$
LOCATE 10,1
PRINT nome$

FOR i% = 2 TO 70
  LOCATE 10, i% - 1
```

(continua)

continuação

```
    PRINT SPACE$(10)
    LOCATE 10, i%
    PRINT nome$

    FOR j% = 1 TO ESP%      ' laço de espera
    NEXT j%

NEXT i%
```

5. SCREEN muda o modo do seu adaptador de vídeo e determina a resolução de tela e (se for o caso) o número de cores que o seu adaptador de vídeo pode mostrar.

6. Os comandos PSET e PRESET definem pixels individuais na tela em locais específicos. O comando PSET marca um pixel na cor atual do primeiro plano; o comando PRESET marca um pixel na cor atual do segundo plano.

7. As coordenadas absolutas são calculadas usando um ponto inicial *(0, 0)*, que é o canto superior esquerdo da tela. As coordenadas relativas usam o último ponto plotado como ponto de partida para o cálculo.

8. Verdadeiro, desde que você use as opções B e F.

9. Falso

10. Uma solução possível para este problema é o programa MOVECIRC.BAS:

```
' MOVECIRC.BAS
' Este programa "move" um círculo pela tela.

CONST ESP% = 50

SCREEN 1   ' mude este valor conforme o seu adaptador

CIRCLE (20, 100), 20

FOR i% = 21 TO 299 ' supondo resolução de 320 x 200
  CIRCLE (i% - 1, 100), 20, 0 ' apaga círculo anterior
  CIRCLE (i%, 100), 20        ' desenha novo círculo

  FOR j% = 1 TO ESP%          ' laço de espera
  NEXT j%

NEXT i%
```

11. Uma solução possível para este problema é o programa HINO.BAS:

```
' HINO.BAS
' Este programa toca a parte inicial de
'  "The Star-Spangled Banner."

CLS

INPUT "Pressione Enter para começar...", algo$

FOR i% = 1 TO 12
  READ nota%, duracao%
  SOUND nota%, duracao%
NEXT i%

DATA 349, 4, 294, 4, 233, 8, 294, 8, 349, 8, 466, 16
DATA 587, 4, 523, 4, 466, 8, 294, 8, 330, 8, 349, 16
```

CAPÍTULO 12

1. a. Rode o seu programa.
   b. Observe erros e pontos problemáticos na execução do programa.
   c. Usando listagens, ferramentas de programação e o seu conhecimento da sintaxe do QBasic, estude os comandos que produziram o erro.
   d. Conserte o erro e teste o programa.

2. Um erro de sintaxe é uma falha na programação que infringe as regras elementares do QBasic. Um erro em tempo de execução é aquele que faz com que um programa pare inesperadamente durante a sua execução.

3. Não há uma resposta correta aqui. Nossos favoritos são os comandos Step e Toggle Breakpoint.

4. A tecla de função F7 executa, em plena velocidade, todas as linhas entre a linha corrente e o cursor.

5. Set Next Statement define a linha onde o cursor se encontra como o próximo comando a ser executado.

6. Novamente, não há uma resposta correta aqui (sobretudo com tantas para escolher!). Os piores que podemos lembrar são erros lógicos resultantes de uma aritmética incorreta ou de expressões condicionais falhas.

7. DIM e NEXT estão escritos incorretamente. Veja a versão correta de ARTISTAS.BAS:

```
' ARTISTAS.BAS
' Versão correta de ARTISTAS.BAS

CLS

DIM artistas$(5)        ' dimensiona vetor de cadeia

PRINT "Entre com os nomes dos seus cinco artistas
favoritos."
PRINT

FOR i% = 1 TO 5         ' pega 5 cadeias
  INPUT "Artista: ", artistas$(i%)
NEXT i%

PRINT
PRINT "Você entrou com os seguintes nomes:"
PRINT

FOR i% = 1 TO 5         ' mostra as 5 cadeias
  PRINT artistas$(i%)
NEXT i%
```

8. O comando INPUT deveria estar em um comando LINE INPUT para aceitar a vírgula na entrada do usuário, e a expressão condicional no comando IF deveria usar um operador de maior que (>) em vez do operador menor que (<). Veja a versão correta de NOMES.BAS:

```
' NOMES.BAS
' Este programa separa o primeiro e último nomes
'  e os imprime.

CLS

PRINT "Entre com o primeiro e último nomes assim: ";
PRINT "Último, Primeiro"
PRINT

LINE INPUT "Nome: ", nomeTodo$

localVírgula% = INSTR(1, nomeTodo$, ",")

IF (localVírgula% > 0) THEN
  ultimo$ = LEFT$(nomeTodo$, localVirgula% - 1)
```

(*continua*)

continuação

```
  primeiro$ = RIGHT$(nomeTodo$, LEN(nomeTodo$) -
localVirgula% - 1)

  PRINT
  PRINT "Que belo nome! Prazer em conhecê-lo, ";
  PRINT primeiro$; " "; ultimo$; "!"
ELSE
  PRINT
  PRINT "O nome não está no formato Último, Primeiro."
END IF
```

Índice Remissivo

Referências a figuras e a ilustrações aparecem em itálico.

Caracteres Especiais

A

B

C

D

E

F

G

H

I

J

Q

R

S

T

U

V

W

Microsoft e Campus
garantia de qualidade

MICROSOFT WINDOWS 3.1 — TÉCNICAS DE PROGRAMAÇÃO
Microsoft Press
Para o usuário aproveitar ao máximo todo o potencial de programação do sistema operacional Windows, versão 3.1. Explica como e quando utilizar as mais variadas ferramentas para desenvolvimento de aplicações para Windows.
cód.785 * 280 pp.

EXPLORANDO DOS 6 AO MÁXIMO
Gookin, D.
Acompanha disquete gratuito
Para melhorar o desempenho, proteger dados e criar a última palavra em sistema MS-DOS, com mais de 40 arquivos de lotes, utilitários e programas. O disquete possui dois utilitários para ajudá-lo a configurar adequadamente e otimizar o sistema e todos os programas, arquivos de lote e utilitários especiais apresentados no livro.
cod. 833 * 426 pp.

PROGRAMAÇÃO EFICAZ COM MICROSOFT ASSEMBLER
Duncan, R.
Um guia em forma de exemplos, com soluções para os problemas mais comuns na programação em Assembler. Acompanha um disquete com mais de 20 exemplos de programas que permitem experimentar as rotinas e mostrar como integrá-las aos programas.
cod.794 * 408 pp.

EXPLORANDO WINDOWS 3.1
The Cob Group/Lorenz/ O'Mara/Borland
O mais abrangente livro de Windows, escrito pelos próprios criadores do programa. Explicações passo a passo, centenas de ilustrações, tradução que mantém a terminologia da Microsoft, com correspondência inglês/português.
cod.769 * 602 pp.

GUIA PARA A PROGRAMAÇÃO DO MS DOS 5
Aitken, P.
Introdução prática, em forma de exemplos para programadores em C e Basic. O ponto de partida perfeito para explorar e compreender a programação do MS DOS. Mesmo que os conhecimentos da Basic e C sejam apenas iniciais, o usuário poderá acrescentar velocidade, eficiência e flexibilidade aos programas.
cod.795 * 456 pp.

DICIONÁRIO DE INFORMÁTICA
Microsoft Press
Inglês/Português e Português/Inglês
Indispensável em qualquer biblioteca de informática. Mais de 5 mil verbetes em inglês com a respectiva tradução em português.
cod.748 * 500 pp.

MICROSOFT MOUSE — GUIA DE REFERÊNCIA PARA PROGRAMADORES
2ª edição americana
Microsoft Press
Escrito por especialistas da divisão de Hardware da Microsoft, contém todas as informações necessárias para incorporar uma sofisticada interface de mouse aos programas.
cod.741 * 312 pp.

Editora Campus • A Qualidade da Informática

Editora Campus

Vamos nos conhecer melhor

☐ Sim, estou interessado em me cadastrar junto à Editora Campus para receber mais informações sobre livros de

☐ Informática ☐ Negócios ☐ Marketing

NOME

ESCOLARIDADE

ENDEREÇO

FONE

CIDADE/ESTADO

CEP

FAX

EMPRESA

CARGO

1. Onde costuma comprar livros ?

☐ livrarias ☐ feiras ☐ por telefone ☐ por correio

☐ outros (especificar): _____

2. Quantos livros de computação você compra por ano ?

☐ 1 - 2 ☐ 3 - 5 ☐ 6 - 8 ☐ 9 - 12 ☐ mais de 13

3. Possui computador em casa ?

☐ Sim ☐ Não De que tipo ? _____

4. Onde você mais usa o computador ?

☐ em casa ☐ no trabalho ☐ nos dois

5. Qual o tipo de software que você mais utiliza ?

☐ Processador de textos ☐ Banco de dados ☐ Planilhas

☐ Outros (especificar): _____

6. Qual a linguagem que você mais utiliza ?

☐ Assembler ☐ C ☐ C ++ ☐ Clipper ☐ Cobol ☐ DBase

☐ Outras (especificar): _____

7. O que mais influencia a sua decisão ao adquirir um livro de informática? (marque no máximo até três opções, em ordem decrescente de importância)

☐ prestígio do autor ☐ recomendação de terceiros ☐ preço

☐ folhear o livro ☐ nome da editora ☐ boa apresentação

☐ primeiro a chegar ao mercado sobre o assunto

☐ crítica ou matéria na mídia

8. Qual o seu nível de formação em informática ?

☐ iniciante ☐ intermediário ☐ avançado

NOSSOS LIVROS ENCONTRAM-SE
EM TODAS AS BOAS LIVRARIAS

DOBRE AQUI

SELO

Editora Campus

LIVROS CIENTÍFICOS E TÉCNICOS
CAIXA POSTAL 13007 CEP 20159-900
RIO DE JANEIRO RJ BRASIL
Endereço Telegráfico: CAMPUSRIO

COLE

USANDO MS-DOS® QBasic™

MS-DOS QBasic é a linguagem de programação interessante, completa e moderna que vem com a versão 5 do MS-DOS. A interface objetiva MS-DOS QBasic, além das suas incríveis ferramentas de ajuda, edição e depuração, tornam o aprendizado da programação fácil e divertido. Se o seu computador tiver o DOS 5 ou se você estiver pensando em passar para o DOS 5, este livro será o caminho mais rápido para usar a linguagem de programação embutida no próprio DOS 5.

Mesmo não tendo muita ou nenhuma experiência com programação **Usando MS-DOS QBasic** é o caminho melhor e mais fácil de compreender os conceitos de programação e a escrita dos seus primeiros programas em Basic. Este curso completo inclui:

Uma Introdução Completa e um Guia Prático de Uso. Informações claras sobre conceitos de projeto de programa, instruções passo a passo para a escrita de programas em Basic, conselhos de depuração e um guia prático para converter programas do GW-Basic para o MS-DOS QBasic.

Programas de Exemplo. Dezenas de programas ilustram as boas técnicas de programação. Eles são úteis por ensinar e divertir!

Perguntas e Exercícios. Perguntas, respostas e exercícios elaborados especialmente para este livro reforçam a sua proficiência na linguagem MS-DOS QBasic.

ISBN 85-7001-699-9

9 788570 016997