2ª EDIÇÃO

MANUAL APPLE

NOEL KANTARIS

INCLUINDO BASIC

Com aplicações de arquivos em disco



Editora Campus

KANTARIS
MANUAL APPLE INCLUINDO BASIC

TÍTULOS DE INTERESSE CORRELATO

APPLE

JOGOS GRÁFICOS PARA O APPLE — *P. Coletta*

APPLESOFT E DOS 3.3 — *R.U. Christmann*

APPLE — MANUAL DO PROGRAMADOR — *E.M. Oliveira*

COMO PROGRAMAR O SEU APPLE — *R. Cook e T. Hartnell*

Conheça toda a linha de Informática da Editora CAMPUS, com títulos nas áreas de: Introdução à Informática; Computação para Crianças; BASIC; COBOL; Outras Linguagens de Alto Nível; Microprocessadores e Linguagem de Máquina; Arquitetura de Computadores e Hardware; Apple; PC; TK85 e TK90X; TRS; Computação em Ambiente Empresarial; Programas e Aplicativos; Processamento de Dados; Teoria e Organização de Dados; Banco de Dados; Programação e Análise Estruturada de Sistemas; Sistemas Operacionais e Compiladores; Inteligência Artificial e Robótica; Interesse Especial; Videocassete e Videogames.

E, ainda:

DICIONÁRIO ENCICLOPÉDICO DE INFORMÁTICA — *A.H. Fragomeni*

Extenso e abrangente, reúne mais de 33.000 entradas em inglês e português pertencentes aos mais diversos campos da Informática e áreas correlatas.

Solicite nosso catálogo completo.

Procure nossas publicações nas boas livrarias ou comunique-se diretamente com:

EDITORA CAMPUS LTDA.
Livros Científicos e Técnicos
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Barão de Itapagipe 55 Rio Comprido
Telefone (021) 284-8443 Telex (00038) 021-32606
20261 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO.

MANUAL APPLE INCLUINDO BASIC



Consultor Técnico

Solon Benayon da Silva
Engenheiro Eletrônico UFRJ
Analista de Sistemas e Pesquisador
do Instituto Latino-Americano
de Pesquisas e Sistemas da IBM.

NOEL KANTARIS

MANUAL APPLE INCLUINDO BASIC

Com aplicações de arquivos em disco

2ª EDIÇÃO

Tradução

Mário del Guercio Dolacio Mendes

Editora Campus Ltda.

Rio de Janeiro

Do original:
The APPLE's BASIC Core
by Noel Kantaris
Sigma Technical Press
Copyright © 1982 by Noel Kantaris.

© 1985, Editora Campus Ltda.
2ª Edição, 1987.

Todos os direitos para a língua portuguesa reservados e protegidos pela Lei 5988 de 14/12/1973.
Nenhuma parte deste livro poderá ser reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos, mecânicos, fotográficos, gravação ou quaisquer outros.

Todo o esforço foi feito para fornecer a mais completa e adequada informação. Contudo a editora e o(s) autor(es) não assumem responsabilidade alguma pelos resultados e uso da informação fornecida.
Recomendamos aos leitores, em conseqüência, testar toda a informação antes de sua efetiva utilização.

Capa
Otavio Studart

Projeto Gráfico, Composição e Revisão
Editora Campus Ltda.
Qualidade internacional a serviço do autor e do leitor nacional.
Rua Barão de Itapagipe 55 Rio Comprido
Tel.: (021) 284 8443 Telex (00038) 021-32606
20261 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO

ISBN 85-7001-402-3
(Edição original: ISBN 0-905104-26-9 Sigma Technical Press, Cheshire, U.K.)

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

K25m
2.ed.

Kantaris, Noel
Manual APPLE incluindo BASIC / Noel Kantaris; tradução Mario del Guercio Dolacio Mendes. — 2.ed. — Rio de Janeiro: Campus, 1987.

Tradução de: The APPLE's BASIC core.
ISBN 85-7001-402-3

1. APPLE II (Computadores). 2. BASIC (Linguagem de programação para computadores). I. Título.

86-1047

CDD — 001.64
CDU — 681.3.06

Sumário

Introdução

Apresentando o microcomputador APPLE II 15
O teclado 16
Linguagens de programação 16
Usando a unidade de disco 17
Inicializando um disquete em branco 18
Atenção: O que fazer se você apertar RESET 19
Comandos para manuseio 20
Obtendo uma cópia impressa para a listagem do programa 21
Dispositivo de edição na tela 21

Capítulo 1 **Usando o BASIC**

Modo imediato 24
Modo não-imediato 24
Atribuindo valores a variáveis 25
Variáveis, constantes e expressões 26
Introduzindo um novo programa 26
Formatação da instrução PRINT 27
Problema 1.1 28
A instrução INPUT 28
Problema 1.2 30
As instruções READ e DATA 30
Problema 1.3 31
A instrução RESTORE 31
Prioridades em operações aritméticas 31
Tabela 1.1 Operadores aritméticos e suas prioridades 32

Capítulo 2 **Controle de fluxo do programa**

A instrução GOTO 34
A instrução IF 34
Fluxogramas 35
Fig. 2.1 Fluxograma do programa PERCENT 36
Operadores lógicos dentro de uma instrução IF 36
Tabela 2.1 Operadores lógicos 36
Atenção: Palavras reservadas 37
Classificação de dados 37
Problema 2.1 38
Fig. 2.2 Classificação de cartas em ordem descendente 38
Fig. 2.3 Fluxograma para classificar três números 39

A instrução GOTO computada 40
Problema 2.2 40
O loop FOR-NEXT 41
O uso de STEP (passo) 42
Ninhos de loops FOR-NEXT 42
Problema 2.3 43
O uso de loops FOR-NEXT em gráficos simples 43
Fig. 2.4 Algumas configurações de loop 44
Problema 2.4 44

Capítulo 3 **Gráficos – 1ª parte**

Gráficos de baixa resolução 46
Tabela 3.1 Cores para gráficos de baixa resolução 46
Desenhando quadros 47
Gráficos de barra e histogramas 48
Fig. 3.1 Gráfico de barra das despesas domésticas 48
Problema 3.1 49
Gráficos de alta resolução 49
Tabela 3.2 Cores para gráficos de alta resolução 50
Atenção: Uso dos comandos HGR e HGR2 50
Desenhando linhas em alta resolução 50
Plotando gráficos em alta resolução 52
Problema 3.2 52

Capítulo 4 **Strings e arrays**

Variáveis string 54
Tabela 4.1 Códigos de conversão de caracteres ASCII 55
Arrays de strings 55
Funções string 57
Problema 4.1 58
Variáveis numéricas subscritas 58
Instruções múltiplas em uma linha 60
Classificação alfabética 61
Classificação pela técnica da bolha 62
Problema 4.2 63
Concatenação de strings 63
Fig. 4.1 Fluxograma do programa de classificação pelo método da bolha . . 64
Mais algumas funções string 64
Atenção: A função VAL e valores negativos 65
Problema 4.3 66

Capítulo 5 **Subprogramas**

Funções aritméticas standard 67
Tabela 5.1 Funções standard do BASIC 67
Funções definidas pelo usuário 69
Problema 5.1 69
Sub-rotinas 69

Tabela 5.2 Funções matemáticas derivadas 70
As instruções GOSUB e RETURN 70
Fig. 5.1 Representação em diagrama de programa com sub-rotina 71
Parâmetros de sub-rotina. 71
Alinhamento de números à direita 72
Problema 5.2 73
Fig. 5.2 Fluxograma do programa dos juros. 74
Sub-rotinas aninhadas. 74
Fig. 5.3 Representação em diagrama de sub-rotinas aninhadas. 75
Problema 5.3 75
Sub-rotinas com múltiplos pontos de entrada 76
Problema 5.4 77

Capítulo 6 **Comunicação com periféricos**

Comandos DOS a partir do BASIC 78
Problema 6.1 79
Arquivos-texto. 79
Arquivos-texto seqüenciais 80
Usando a instrução INPUT nos arquivos-texto 81
Problema 6.2 82
Atualização de arquivos-texto seqüenciais 82
O uso do comando APPEND 83
Uso da instrução GET em arquivos-texto. 84
Atenção: Problemas com o uso da instrução GET 85
Problema 6.3 85
Acesso randômico a arquivos-texto. 86
Criação e recuperação de arquivos randômicos 86
Problema 6.4 88
Comprimento ótimo de registro 88

Capítulo 7 **Programas utilitários e arquivos binários**

Rotinas de manipulação de erros 90
Tabela 7.1 Códigos de erros do DOS. 90
A opção de parâmetro de volume 91
Uso de sub-rotina do monitor 92
Tabela 7.2 Sub-rotinas do monitor afetando textos 92
Edição de entrada da tecla sob BASIC. 93
Fig. 7.1 Representação visual do processo de edição 94
Problema 7.1 94
A sub-rotina GETLN 94
Formatação da tela de janelas 95
Tabela 7.3 Localizações de memória relacionadas à formatação da tela . . . 95
Atenção: Erros causados por extensão fora da faixa 95
Problema 7.2 96
Carga de um programa em linguagem de máquina 96
Como usar arquivos binários 97
Programas utilitários em linguagem de máquina. 98

Capítulo 8 **Criação de uma seqüência de programas**

O programa menu 101
Arquivos executáveis 102
Problema 8.1 103
Renumeração e intercalação de programas 103
Problema 8.2 107
Pesquisa e edição de registros 107
Problema 8.3 109
Classificação de grandes arquivos 109
Ligação de registros com ponteiros 109
Ligações para trás e para frente 110
A opção de parâmetro de byte 112
Criação de ponteiros para ligar arquivos 112
Problema 8.4 114
Desmontagem e remontagem de inteiros 114
Tabela 8.1 Representação do byte 116

Capítulo 9 **Gráficos – 2ª parte**

Tabelas de formas 118
Fig. 9.1 Representação esquemática de uma tabela de formas 119
Tabela 9.1 Movimentos do vetor 120
O comando DRAW 120
Problema 9.1 121
Escala e direção das formas 121
Tabelas de formas complexas 122
Tabela 9.2 Tabela para definição de formas 123
Salvando tabelas de formas em disco 125
Problema 9.2 125
Animação de formas 125

ANEXO A

Resumo dos comandos de acesso e comandos DOS 127
Comandos de acesso 127
Comando DOS 127
Parâmetros dos comandos DOS 128
Comandos DOS a partir do BASIC 129
Arquivos de acesso randômico 129

ANEXO B

Mensagens de erro do APPLESOFT e do DOS 131
Tabela B-1 Códigos de erro do APPLESOFT 131
Tabela B-2 Códigos de erro do DOS 131

ANEXO C

Mensagens de erro 133

ANEXO D

Técnicas para depuração de erros 136

(*a*) Erros de sintaxe 136
(*b*) Erros de lógica 137
Problema D-1 139

ANEXO E

O mapa de memória do APPLE 140
Representação de dados numéricos 140
Problema E-1 141
Organização geral de memória 141
Organização da memória RAM 141
Fig. E-1 Representação das memórias RAM e ROM do APPLE 142
Fig. E-2 Mapa da memória RAM do APPLE 142
Tabela E-1 Valores do HIMEM 143
Tabela E-2 Espaço de memória disponível 144

ANEXO F

Resumo das características de edição 145

ANEXO G

Códigos de conversão ASCII 146
Tabela G-1 Códigos de conversão decimal ASCII 146
Códigos de ESCape 147

ANEXO H

Soluções dos problemas 148
Problemas 1.1,1.2 e 1.3 148
Problema 1.4 149
Problemas 2.1 e 2.2 149
Problemas 2.3 e 2.4 150
Problemas 3.1 e 3.2 151
Problemas 4.1 (A) e (B) 152
Problema 4.2 153
Problemas 4.3 (A) e (B) 154
Problemas 5.1 e 5.2 154
Problema 5.3 155
Problema 5.4 155
Problema 6.1 156
Problema 6.2 157
Problema 6.3 158
Problema 6.4 158
Problema 7.1 159
Problema 7.2 160
Problemas 8.1,8.2,8.3 e 8.4 161
Problemas 9.1 e 9.2 168
Índice analítico 170
Índice de figuras 175
Índice de tabelas 176

Lista de programas no texto

Média 27
Uso de PRINT TAB 27
USO de HOME, HTAB e VTAB 28
Programa de percentagem 28
Percentagem geral 29
Uso da instrução READ 30
Uso de RESTORE 31
Média com READ 31
Percentagem com a instrução IF 35
Classificação de dois números 37
Classificação de três números 39
Uso da instrução ON-GOTO 40
Uso do loop FOR-NEXT 41
Acumulador 41
Soma de N números 41
Loops FOR-NEXT aninhados 42
Imprimindo letras em forma de bloco 43
Quadrados coloridos 47
Gráfico de barra de largura variável 48
Desenhando linhas em alta resolução 50
Plotando a equação Y = X ↑ 2 52
Uso de um array de strings 57
Informações sobre funcionários 56
Controle de estoque 58
Array numérico 59
Controle de estoque usando strings e arrays 60
Classificação alfabética 61
Classificação da bolha 62
Função definida pelo usuário 69
Juros 71
Sub-rotina para converter números para o formato # # # . # # 71
Sub-rotina para alinhar números à direita 72
Juros com duas sub-rotinas 73
Juros com sub-rotinas aninhadas 76
Impressão da saída na impressora 78
Fazendo uma chave de acionamento para o sistema 79
Criação de arquivo-texto seqüencial 80
Recuperação de arquivo-texto seqüencial 81
Atualização de arquivos-texto seqüencial 82

Append arquivo seqüencial 83
Criação de arquivo seqüencial com GET. 84
Recuperação de arquivo seqüencial com GET 84
Criação de arquivo randômico com INPUT. 86
Criação de arquivo randômico com GET 87
Recuperação de arquivo randômico com GET. 87
Rotina de manipulação de erros 91
Edição de entradas pelo teclado 93
Usando a sub-rotina GETLN 94
Criando uma janela 95
Colocando programas em linguagem de máquina na memória. 97
Programas de encadeamento 98
Localizador de endereço BSAVE 99
Criador de programa MENU 101
Criador de arquivos-texto 101
Criação/recuperação/atualização de arquivos randômicos 101
Editor de arquivos randômicos antigos. 107
Criador de ponteiros para concatenar arquivos alfabeticamente 112
Desmontando e remontando inteiros. 114
Desmontando e remontando números para o disco. 115
Criação de uma tabela de formas de uma linha reta 118
Carregador de formas (BLOAD) 125
Objetos voadores. 125
Gerador de seqüências binárias. 140

Prefácio

Este livro foi especialmente elaborado para ser usado com o microcomputador Apple II equipado com uma unidade de disco.

A primeira metade deste livro introduz o principiante na linguagem de programação BASIC, não pressupondo qualquer conhecimento ou familiaridade com computadores e suas linguagens. Os comandos BASIC são introduzidos e explicados com o auxílio de programas simples, e o usuário é encorajado a digitar os mesmos no computador, salvá-los em disco e aprimorá-los à medida que comandos mais complexos da linguagem e instruções forem apresentados. Isto permite ao usuário construir uma biblioteca considerável de seus próprios programas e sub-rotinas que lhe servirão de base para as técnicas de programação mais avançadas e as aplicações de arquivos em disco que serão analisadas na segunda parte do livro.

A parte final deste livro também é útil aos usuários que já possuem conhecimento gerais sobre o BASIC, e que desejem ser conduzidos através dos caminhos complexos dos programas utilitários, rotinas de erro, sub-rotinas do Monitor, manuseio da tela e rotinas de entrada por teclado sendo ilustradas e incorporadas em programas de trabalho.

Os últimos capítulos deste livro são dedicados a aplicações de arquivos em disco com ênfase em técnicas de banco de dados. Os tópicos abordados são: uso de disco, arquivos de acesso seqüencial e aleatório (randômico), programa e sub-rotinas, seqüências de programas, *menus*, uma técnica especial de banco de dados para encadear registros com ponteiros duplos, gráficos de alta resolução e métodos fáceis e rápidos de criar tabelas de formas.

Problemas de gradativa dificuldade são propostos ao longo do livro, com as suas soluções contidas no último anexo. Se você desejar obter um disquete contendo estas soluções juntamente com todos os programas utilitários mencionados neste livro, entre em contato com o autor na CAMBORNE SCHOOL OF MINES, POOL, REDRUTH, CORN WALL.

Agradecimentos: Gostaria de agradecer a Geoffrey Kantaris por sua constante ajuda e inúmeras sugestões de programação, a Lofty Humphrey por ter me apresentado as idéias de banco de dados demonstradas neste livro, e a Sandra Hoy por usar o texto como seu guia para aprender BASIC e por suas sugestões de edição.

INTRODUÇÃO

Apresentando o microcomputador APPLE II

Tire a tampa do seu computador Apple II ou Apple Plus (tendo-o desconectado da tomada), apoiando a palma de cada mão nos cantos de trás da caixa e puxando firmemente com os dedos a extremidade traseira da tampa até que os dois cantos se desencaixem. A tampa sairá ao se deslizá-la para trás.

Observe as inúmeras carreiras ordenadas de CIs (micro chips ou pastilhas) que são montadas na placa principal do computador. O mais longo, que possui o número 6502, é o chip microprocessador, que pode executar meio milhão de adições por segundo. Todos os demais chips e componentes existem simplesmente para auxiliar no funcionamento do chip 6502 e possibilitar a comunicação das instruções dadas a ele. A caixa de alumínio à esquerda contém a fonte de tensão. Processa-se de trás para frente do computador.

A coluna de 8 encaixes, na sua maioria vazios, é usada para conectar periféricos tais como uma impressora, unidade de disco, modulador de cores e a placa para música. Veja os números dos encaixes (soquetes). Convencionalmente, o encaixe 1 é usado para a impressora, o encaixe 6 para a unidade de disco, e o encaixe 7 para a placa de cores. O computador pode ser ordenado a receber ou emitir dados por qualquer um dos encaixes, um de cada vez – esta é uma das melhores qualidades do projeto desta máquina.

A próxima coluna contém o microprocessador 6502 seguido de seis chips grandes chamados ROMs que significam MEMÓRIA SOMENTE PARA LEITURA (do inglês READ ONLY MEMORY); i.e., o computador pode ler seu conteúdo mas não pode modificá-lo. O conteúdo das ROMs permanece intacto quando o computador é desligado, o que é benéfico pois contém uma variedade de programas que controlam a capacidade operacional de seu computador (conhecida como monitor do sistema), que você acessa digitando instruções no teclado. Existem vários conjuntos de ROMs disponíveis que determinam o que o computador faz quando ligado (tais como a ROM de acionamento automático) e qual linguagem de computador está disponível ao usuário (tais como a versão especial do BASIC conhecida como Applesoft BASIC).

As três colunas seguintes contêm até 25 chips, chamados RAM. Qualquer lugar vazio pode ser preenchido com chips idênticos a fim de expandir a memória de seu computador até 48 kbytes. Isto significa que o computador pode guardar até 48.000 letras ou números enquanto estiver ligado e funcionando.

Na coluna frontal de chips, embaixo do teclado e difícil de se ver, está um grande chip gerador de caracteres, que produz os sinais que exibem as suas informações na tela do televisor. Finalmente, note a existência de um alto-falante e, perto do encaixe 7, um soquete para controlador de jogos.

Coloque a tampa novamente no lugar, inicialmente deslizando as suas extremidades traseiras, até encaixar. Ligue a força e se estará pronto para investigar o teclado. Se seu computador está equipado com uma ROM de início automática, a sua unidade de disco começará a operar. Para pará-la, pressione a tecla RESET algumas vezes.

O teclado

O teclado é semelhante a uma máquina de escrever, com algumas diferenças: não existem letras minúsculas, pois só as letras maiúsculas são necessárias para se escrever programas para o computador. A tecla SHIFT permite o uso de símbolos adicionais que aparecem acima de cada tecla. A tecla SHIFT é mantida pressionada enquanto tais símbolos forem desejados. A tecla SHIFT não tem nenhum efeito quando usada com teclas que não têm nenhum símbolo na parte superior, exceto as teclas M e G.

A tecla CTRL (controle) é como uma segunda tecla SHIFT, e dá um terceiro conjunto de possíveis "significados" a cada tecla. Por exemplo: CTRL G (mantenha a tecla CTRL pressionada e aperte G) irá produzir um "bip" no alto-falante. Caracteres que são pressionados enquanto a tecla CTRL estiver pressionada não aparecerão na tela. Tente pressionar estas teclas e observe o resultado na tela.

Outra tecla pouco familiar é a REPT (repetir). Para ilustrar o seu uso, pressione a letra A e, enquanto a mantiver pressionada, aperte REPT. Você verá que a letra A aparecerá repetidamente na tela e assim permanecerá até que a tecla REPT seja solta. A "flecha esquerda" e a "flecha direita" deslocam o cursor para a esquerda ou para a direita e podem ser usadas em conjunto com a tecla REPT.

Uma tecla muito importante é a RETURN, que é pressionada quando uma instrução completa ao computador já estiver digitada e o usuário desejar uma resposta do computador. Ela também possui o efeito de retornar o cursor ao canto esquerdo da tela na linha seguinte.

Para facilitar a leitura, de agora em diante, aspas serão usadas para assinalar teclas especiais que o usuário deverá pressionar. Para 'RETURN' significa aperte a tecla RETURN, enquanto 'CTRL' C significa pressione a tecla CTRL e, mantendo-a pressionada, aperte a tecla C.

Finalmente, existe uma tecla escrita ESC (escape) que pode ser usada para editar informações escritas na tela. A tecla ESC só precisa ser pressionada momentaneamente antes de se pressionar qualquer outra tecla. Por exemplo: ESC 'SHIFT' P (aperte momentaneamente a tecla ESC e depois, enquanto pressiona a tecla SHIFT, aperte a tecla P) irá apagar tudo o que estiver na tela e mandará o cursor para o canto superior esquerdo da mesma.

Finalmente, mas não menos importante, existe uma tecla em que está escrito RESET. Se esta tecla for pressionada, ela pára o processador; quando for solta, fará o computador entrar em monitor (modelos antigos) ou (para o Apple plus) começará um ciclo que reiniciará o Applesoft. Não aperte a tecla RESET ainda, pois a discutiremos mais adiante com mais detalhes, mas se você apertá-la acidentalmente, e o símbolo * aparecer na tela, indicando entrada de monitor, digite então 3DOG (0 é um zero), do contrário não digite nada.

Linguagens de programação

Existem várias linguagens de programação disponíveis no Apple II que podem ser usadas

para escrever programas. Delas os principiantes ao Apple irão encontrar provavelmente as três seguintes:

1) Assembler
2) Integer BASIC e
3) Applesoft BASIC.

Os programas do Sistema Monitor, escritos em código de máquina com o auxílio do Assembler, estão disponíveis no instante em que o computador é ligado, pelo fato de residirem em ROM. Estes programas permitem que as informações do teclado sejam interpretadas corretamente, ou informações sejam impressas na tela da TV.

Integer BASIC e Applesoft BASIC são versões do BASIC que podem tanto ser carregadas de fita cassete ou disquete, ou existir na ROM. Qualquer que seja a linguagem que tenha em ROM, você terá de carregar a outra de cassete ou disquete para poder usá-la. Os modelos mais recentes do computador irão automaticamente carregar a linguagem necessária, se esta estiver no mesmo disquete do programa desde que seu computador esteja equipado com um cartão de linguagem, o qual é conectado ao encaixe 0, do contrário ele responderá escrevendo na tela a mensagem

```
LANGUAGE NOT AVAILABLE.
(LINGUAGEM NÃO-DISPONÍVEL.)
```

Existem algumas diferenças entre Integer BASIC e Applesoft BASIC, mas o resto deste livro dedicar-se-á somente à versão Applesoft do BASIC.

Usando a unidade de disco

Estamos admitindo que o leitor está usando uma unidade de disco (disk "drive"), cujo cartão de controle está conectado ao encaixe 6 (veja seu Manual de DOS). A unidade de disco é muito mais rápida e mais fácil de usar do que um gravador cassete. Além disso, o usuário pode salvar em disquete ou carregar de disquete os seus programas com o mínimo de dificuldade.

Para poder usar a unidade de disco é importante adicionar os comandos do Sistema Operacional de Disco, conhecido pela sigla DOS (do inglês Disk Operating Sistem), ao BASIC em seu Apple. Esta operação "DOS" é conhecida como boot* (a qual denominaremos unidade DOS). Isto é obtido facilmente seguindo as instruções dadas a seguir:

Insira o Master disk (disquete mestre) na unidade de disco, feche a porta e ligue o computador. De acordo com a versão de Apple você verá que o computador estará ou em Monitor ou em Applesoft BASIC. Para chamar a unidade de disco, digite um dos seguintes comandos de acesso:

1) Para o Monitor (cujo caractere de aviso é *) digite

```
6'CTRL'P                (seguido por 'RETURN')
```

2) Para Applesoft (cujo caractere de aviso é) digite

```
IN #6                   (seguido por 'RETURN')
```

onde IN # significa 'entrada pelo encaixe' (do inglês INPUT). Digitar PR # 6 tem o mesmo efeito, e é usado alternadamente com IN # 6.

* N. do T. Não há tradução consagrada para a palavra "booting", por isso a mantivemos em inglês.

Para o Apple II Plus (com acionamento automático), a chamada do DOS acontece automaticamente quando o computador é ligado, se o disquete estiver dentro da unidade. Se não estiver, a unidade começará a funcionar até que RESET seja pressionado. Em seguida insira o disquete na unidade e digite IN # 6.

Em qualquer um dos casos, o computador deverá agora responder com o caractere], que indica que o sistema foi chamado e o Applesoft está disponível. Se, em vez disso, o computador responder com o caractere > (que indica que o sistema foi chamado e o Integer BASIC está disponível), digite

```
FP                    (seguido por 'RETURN')
```

FP significa 'Ponto Flutuante' (do inglês Floating Point). A unidade irá funcionar e o caractere [aparecerá na tela.

```
CATALOG               (seguido por 'RETURN')
```

Uma lista de nomes deverá aparecer na tela. Estes são os 'nomes de arquivo' dos programas que estão atualmente armazenados no disco. Se, após ter digitado CATALOG, você tiver a resposta

```
SYNTAX ERROR
```

então você não deve ter digitado o comando corretamente; redigite-o e observe a resposta do computador.

Nota: Um resumo dos comandos de acesso e DOS estão no Anexo A. Listas de mensagens de erro tanto do Applesoft como do DOS são dadas nas Tabelas B-1 e B-2 do Anexo B.

Inicializando um disquete em branco

O Master disk que contém programas muito importantes está protegido (não pode ser regravado) e deve ser mantido em lugar seguro (Veja o Manual DOS). A fim de se criar um disquete de trabalho que possa ser reconhecido pelo computador como sendo seu, para que programas nele possam ser gravados, devem ser seguidos os seguintes passos.

Remova o Master disk da unidade e substitua-o por um novo disquete em branco. A fim de que o Apple reconheça tal disco como sendo um dos seus, o computador deve inicializar o novo disquete com um programa BASIC especial, chamado programa de "apresentação", que devemos digitar no computador. No entanto, antes de digitar um novo programa, a memória do computador deve ser limpa, apagando todos os programas existentes, digitando-se

```
NEW
```

Digite o seguinte programa de apresentação, lembrando de apertar a tecla RETURN no final de cada linha. Se você cometer erros, simplesmente aperte RETURN e redigite a linha.

```
10 PRINT "ALO"
20 END
```

Agora digite os comandos do DOS

```
INIT ALO              (seguido por 'RETURN')
```

A unidade de disco irá operar por uns 20 segundos e quando parar o disco estará inicializado. Agora digite

```
CATALOG
```

e não esqueça o RETURN. De fato, esta será a última vez que se apertará a tecla RETURN. A unidade de disco irá operar e o único nome de arquivo exibido na tela do televisor será o nome do programa de apresentação (neste caso ALO).

O novo disquete inicializado pode ser usado para iniciar o sistema desde que o Applesoft esteja na ROM. Se, no entanto, o Applesoft precisar ser carregado na RAM, troque então o disquete no drive pelo disquete do sistema mestre e tecle os seguintes comandos DOS

```
LOAD APPLESOFT
```

O disk drive * irá operar até que todo o programa esteja carregado na memória do computador. Quando a luz "em uso" do drive apagar, troque novamente o disquete mestre do sistema pelo o que acabou de inicializar e digite os comandos DOS

```
SAVE APPLESOFT
```

Este comando fará o drive operar e o programa Applesoft, que está na memória, será transferido para o novo disquete. Quando o aviso retornar à tela isto poderá ser verificado digitando-se CATALOG. Dois nomes de arquivo estarão exibidos na tela (o programa de apresentação ALO e Applesoft). Agora o novo disquete pode ser usado para inicializar o sistema. Tente fazê-lo. Desligue o computador, depois ligue-o novamente. Se você possuir a versão com acionamento automático do Apple II, o boot do disquete ocorrerá automaticamente. Caso contrário, você precisa digitar o comando de acesso 6'CTRL'P.

Agora já é possível escrever e salvar programas no novo disquete. Para demonstrá-lo, digite o seguinte programa:

```
10 PRINT "PROGRAMA TESTE"
20 END
```

e SALVE-o no disquete digitando o comando

```
SAVE TESTE
```

onde TESTE será o nome de arquivo do programa salvado.

ATENÇÃO: O que fazer se você apertar RESET

Se em qualquer momento a tecla RESET for pressionada (tente) ver-se-á que em todas as versões do Apple II (exceto a Apple Plus) o computador vai entrar no monitor (indicado por *). Com o Apple II Plus, um ciclo de "RESET" é iniciado, retomando o sistema para Applesoft. Para reentrar em Applesoft, de monitor, sem perder o programa então armazenado na memória, digite o comando de monitor 3D0G (0 é um zero). Alternadamente, se seu sistema possuir o BASIC na ROM, poderá usar o comando 'CTRL'C.

Se, no entanto, você tivesse de dar "boot" novamente no sistema, o programa na memória seria perdido. Tente todos estes passos e, após cada um, verifique se o programa-teste ainda está na memória, digitando o comando

```
LIST
```

Desde que o programa esteja na memória, o BASIC irá listá-lo. Se eventualmente o programa for apagado da memória, poderá ser recarregado do disquete digitando-se

```
LOAD TESTE
```

* Vide nota do tradutor.

Comandos para manuseio

Um programa em BASIC pode ser removido do disquete, digitando-se o comando

```
DELETE TESTE
```

Isto pode ser verificado digitando-se CATALOG depois que o drive parar de operar. No entanto, o programa ainda está na memória do computador e pode ser regravado (save), se necessário. Salve o programa com o nome de arquivo-teste e use o comando

```
RENAME TESTE, PROG TESTE
```

Agora, no catálogo do disquete, ver-sé-á que o nome do arquivo-teste foi substituído pelo nome de arquivo PROG TESTE.

Deve-se tomar cuidado para não renomear um arquivo com um nome já existente no disquete, pois isto resultará em dois programas diferentes com um mesmo nome de arquivo, gerando conseqüencias indesejáveis.

Agora digite o comando

```
LOCK TEST PROG
```

Se agora você digitar CATALOG, verá na lista de nome de arquivo que o programa PROG TESTE possui um asterisco seguido pela letra A e um número antecedendo o respectivo nome de arquivo. O asterisco significa que o arquivo está travado e não pode ser apagado acidentalmente, como na tentativa de se salvar um outro programa com o mesmo nome de arquivo. Para demonstrar isto, digite

```
DELETE TEST PROG
```

O DOS responderá escrevendo:

```
FILE LOCKED (TRAVADO)
```

Para deletar um arquivo travado, deve-se antes destravá-lo. Digite

```
UNLOCK 'nome de arquivo'
```

em seguida

```
DELETE 'nome de arquivo'
```

A letra A que segue o asterisco no catálogo do disquete representa a linguagem em que o programa foi escrito, neste caso Applesoft. Um programa escrito em Integer BASIC é indicado pela letra I, enquanto que um programa escrito em linguagem de máquina (sistema binário) é indicado pela letra B. Arquivos de texto, que discutiremos mais tarde, são indicados pela letra T.

Todos os comandos apresentados até agora são válidos para todos os diferentes tipos de arquivos mencionados acima, exceto em três casos:

(i) para carregar (load) programas binários (em linguagem de máquina) use o comando

```
BLOAD 'nome de arquivo'
```

(ii) para salvar um programa binário você precisa saber o seu endereço de início e seu comprimento (explicaremos isto mais tarde)

(iii) os comandos LOAD e SAVE não podem ser usados com os arquivos de textos (falaremos mais sobre isto mais adiante).

Finalmente, o número que antecede cada arquivo no catálogo do disquete é o comprimen-

to do programa nos setores dos disquetes. Cada setor do disquete pode armazenar até 256 bytes de informações. Existem 35 trilhas concêntricas em cada disquete, e cada trilha é subdividida em 16 setores (para DOS 3.3). Isto significa que cada disquete pode conter cerca de 140.000 bytes.

Obtendo uma cópia impressa para listagem do programa

À medida que o programa se torna mais longo que as 24 linhas disponíveis na tela de sua TV, torna-se imperativa a possibilidade de se obter uma cópia impressa (uma cópia no papel) da listagem do programa. Acabamos de ver que para acessarmos * a unidade de disco (disk drive) tínhamos de digitar IN # 6, desde que a placa de controle de disco estivesse no soquete 6. O comando geral para enviar uma saída para um soquete é:

PR #*s*

onde PR # s significa 'Imprima (do inglês Print) no soquete *s*'. Logo, dirigindo-se a qualquer número de soquete, podemos acessar o equipamento que a ele estiver conectado. Normalmente a placa de interface para a impressora é conectada ao soquete 1.

A fim de se obter uma cópia impressa da listagem de um programa, o processo dado a seguir deve ser observado.

(i) carregue (LOAD) o programa na memória digitando

LOAD 'nome de arquivo'

(ii) supondo que a impressora esteja conectada ao soquete 1, digite

PR # 1

(iii) digite

LIST

Ao apertar-se a tecla RETURN, o programa será listado no papel. Se, no entanto, a impressora estiver desligada, o sistema ficará 'preso', isto é, o computador continuará tentando se comunicar com a impressora e só poderá ser interrompido pressionando-se RESET. Para retornar à linguagem que você estava usando antes de pressionar RESET, sem perder o programa que está atualmente armazenado na memória, digite o comando de monitor 3D0G, ou para um computador com BASIC em ROM digite o comando 'CTRL'C. Para obter uma cópia impressa de seu programa ligue a impressora e continue a partir do segundo passo (ii) acima.

(iv) Quando a impressão de seu programa estiver concluída, digite

PR # 0

que irá causar o aparecimento do sinal na tela significando que o Apple está desconectado do soquete da impressora.

Tente este procedimento usando o seu programa-teste.

Dispositivo de edição na tela

Até agora, toda vez que um erro era cometido ao se introduzir uma linha ou comando no computador, simplesmente redigitávamos toda a linha ou comando. O seu computador,

* Vide nota do tradutor.

no entanto, foi suprido com alguns dispositivos de edição de tela que torna mais fácil a correção de vários tipos de erros. Estes podem ser agrupados em duas amplas categorias, dependendo do fato de ter sido apertada ou não a tecla RETURN antes da constatação do erro.

(*a*) Correção de erros antes de pressionar RETURN

Para corrigir um erro percebido antes de pressionar a tecla RETURN use a tecla com a flecha esquerda para colocar o cursor em cima do caractere incorreto. Redigite o caractere correto e use a tecla com a flecha direita para retornar o cursor do final da linha digitada, a fim de continuar digitando-a, ou pressione a tecla RETURN. Usando a barra de espacejamento para mover o cursor para a direita, apagam-se os caracteres pelos quais ele passa. Apertando-se RETURN, quando o cursor estiver num ponto qualquer da linha entrará na memória a parte da linha que estiver à esquerda do cursor.

Tanto a flecha esquerda quanto a direita podem ser usadas em conjunto com a tecla REPT para movimentos rápidos de cursor. Para usar este recurso, pressione a flecha esquerda ou direita e, mantendo-a pressionada, aperte a tecla REPT. Tente digitar a primeira linha de seu programa-teste adicionando alguns erros e em seguida use os recursos de edição citados para corrigi-la.

Uma linha indesejada em um programa pode ser deletada simplesmente digitando-se o número da linha, seguido por 'RETURN'. Tente fazê-lo deletando a linha 20 de seu programa-teste.

Entretanto, no caso de você mudar de idéia após ter digitado o número da linha, ou estiver no meio da edição de uma linha, é possível "abortar", isto é, abandonar tudo já feito digitando-se o seguinte comando:

'CTRL'X

que irá colocar uma marca no ponto onde o cursor estava, mas deixará a linha original inalterada.

(*b*) Correção de erros após ter pressionado RETURN

Quando se percebe um erro em uma linha após se ter pressionado a tecla RETURN, pode-se tanto redigitar toda a linha, se esta for curta, ou, se for muito longa, corrigi-la seguindo o processo dado a seguir.

Primeiro, liste a linha digitando LIST seguido do número da linha requerida. Em seguida, pressione a tecla ESC, solte-a e pressione a tecla B. Isto levará o cursor de volta ao extremo esquerdo da tela. Agora pressione (e solte) a tecla ESC, seguida pela tecla D que irá deslocar o cursor uma linha para cima na tela. Repita este último passo (i.e., aperte e solte ESC e em seguida aperte e solte D) até que o cursor esteja no primeiro dígito do número da linha desejada. Agora use a flecha direita para pressionar o cursor em cima do caractere incorreto. Digite-o corretamente e copie o resto da linha antes de pressionar RETURN, o que introduzirá a linha correta na memória do computador.

Existem cinco teclas que são usadas só para movimentos do cursor. Estas são as ESC, A, B, C e D. 'A' movimenta o cursor para a direita, 'B' para a esquerda, 'C' para baixo e 'D' para cima. Pratique o uso destas teclas, digitando novamente a linha 10 de seu programa-teste, adicionando um ou dois erros. Lembre-se de que o cursor só se movimentará com o uso destas teclas se antes de cada uso você tiver pressionado e soltado a tecla ESC.

O Apple II Plus possui um recurso adicional para o movimento de cursor. As cinco teclas nele utilizadas são: ESC, I, J, K e M. Observe o posicionamento desta tecla no te-

clado. A sua posição física relativa fornece a direção de deslocamento do cursor; 'I' desloca o cursor para cima, 'J' para a esquerda, 'K' para a direita e 'M' para baixo. Além da disposição facilitada das teclas, você só precisa pressionar a tecla ESC uma única vez antes de efetuar movimentos de cursor utilizando as quatro teclas alfabéticas. Novamente, digite a linha 10 de seu programa-fonte e use estes recursos para editá-la.

Edição de longas linhas de texto

O uso dos códigos ESC em conjunto com flecha direita para editar longas linhas de texto que possam estar contidas em uma introdução PRINT, como mostrado a seguir, pode causar um ligeiro problema. Para ilustrar, digite a seguinte linha:

```
10 PRINT      "COMO CORRIGIR LONGAS LINHAS DE TEXTO"
20 END
```

Agora LISTE o programa. Você deverá ver na sua tela:

```
10 PRINT      "COMO CORRIGIR LONGAS LINHAS DE TEXTO"
20 END
```

Observe que o uso do comando LIST formata longas linhas em sua tela até um comprimento de 33 caracteres por linha, com mais um espaço adicional no meio da palavra "de" para facilitar a leitura. No entanto, se você usasse a flecha direita para copiar uma destas linhas longas, você iria introduzir estes espaços extras em seu texto. Para superar este problema siga um dos dois processos sugeridos a seguir.

(*a*) Digite LIST 10
Use o código ESC para colocar o cursor em cima do primeiro dígito do número da linha.
Use a flecha direita para copiar até a letra *D* (inclusive) da palavra "de".
Use o código ESC para deslocar o cursor para a direita até que ele esteja na letra *E* (da palavra "de").
Use a flecha direita para copiar o restante do texto.

(*b*) Digite POKE 33,33 (que formata a tela para a largura de 33 caracteres)
Digite LIST 10 (a linha está agora listada sem o acréscimo de qualquer espaço extra)
Use o código ESC para colocar o cursor no primeiro dígito do número da linha.
Use a flecha direita com a tecla REPT para copiar toda a linha
Digite POKE 33,40 (que dá à tela a sua formatação normal, de 40 caracteres de largura)

Estes recursos de edição dados podem parecer complicados; apesar disto seria muito útil você dominá-los, pelo fato de tornarem o processo de programação menos tedioso.

Capítulo 1
Usando o BASIC

Modo imediato

Computadores podem ser usados em modo imediato (i.e., sem programa) para executar cálculos aritméticos, desde que o sistema esteja chamado e o BASIC disponível. Assumindo que este seja o caso de seu computador, digite

```
PRINT 15 + 5                    (seguido por 'RETURN')
```

O BASIC irá responder escrevendo

```
20
```

Se, ao contrário, a mensagem SYNTAX ERROR (erro de sintaxe) aparecer na tela, você deve ter cometido um erro quando digitou a instrução. Tente outra, digitando

```
PRINT 25-14
```

O BASIC irá responder, escrevendo

```
11
```

Agora digite

```
PRINT 25 * 3
```

O BASIC irá escrever

```
75
```

Observe que o asterisco (*) é um caractere especial para multiplicação e nenhum outro caractere pode ser usado para indicar essa operação aritmética. Agora digite

```
PRINT 15/3
```

O BASIC irá escrever

```
5
```

Novamente observe que a barra (/) é um caractere especial para divisão e nenhum outro caractere pode ser usado como alternativa para esta operação. Agora digite

```
PRINT 2 ↑ 3
```

o BASIC irá escrever

```
8
```

Observe que o símbolo (↑) significa 'elevado à potência' e a instrução acima é interpretada como '2 elevado à potência 3' (i.e., 2x2x2). Novamente, o caractere (↑) é o único que pode ser usado para indicar este tipo de operação matemática.

Modo não-imediato

Na última seção, instruções tais como PRINT eram executadas imediatamente após se

pressionar a tecla RETURN. Existe, no entanto, uma outra forma em que as instruções BASIC podem ser escritas de forma que suas execuções sejam retardadas seguido por 'RETURN'. De fato, todos os comandos devem ser seguidos de 'RETURN'.

Instruções não-imediatas devem ser precedidas de um número, referido como o número de linhas, que deve ser um inteiro e estar entre 0 e 63999, com o limite superior dependendo do sistema específico. Para permitir a inclusão de novas linhas, é aconselhável numerá-las em intervalos de dez. Digite as seguintes instruções, lembrando-se de pressionar a tecla RETURN ao final de cada linha.

```
10 PRINT 5 * 2 – 2
20 PRINT 30/2 + 5
30 END
```

Isto é um programa. A instrução da linha 30 diz ao BASIC que ela é o fim do programa, e a execução irá terminar naquele ponto. Agora digite o comando

```
RUN
```

O BASIC irá escrever

```
 8
 20
```

Agora, suponha que queiramos saber não somente os resultados individuais das linhas 10 e 20, mas também a sua soma. Poderíamos conseguir isto incluindo no programa a instrução

```
25 PRINT 5 * 2 – 2 + 30/2 + 5
```

Após ter digitado esta linha, digite o comando

```
LIST
```

O BASIC irá responder listando o programa na memória, com os números de linha em ordem ascendente, incorporando a linha 25 no local correto. Agora digite

```
RUN
```

O BASIC irá responder escrevendo

```
 8
 20
 28
```

que indica que as instruções do programa, na realidade, também são executadas na ordem ascendente dos números de linha.

Atribuindo valores a variáveis

A linha 25 do último programa é, de uma certa forma, a repetição das linhas 10 e 20. Um método de se escrever este programa seria o de atribuir às expressões das linhas 10 e 20 duas variáveis de nomes diferentes. Por exemplo:

```
5  A = 5 * 2 – 2
10 PRINT A
15 B = 30/2 + 5
20 PRINT B
25 PRINT A + B
```

Se você listar o programa verá que as linhas antigas, as quais possuíam os mesmos núme-

ros que as que acabou de digitar, foram substituídas pelas novas, mas a linha 30 ainda está lá. Se agora você rodar (RUN) o programa, o BASIC irá responder escrevendo a mesma resposta anterior. Agora digite

25 (seguido por 'RETURN')

Quando listar o programa, você verá que a linha 25 foi removida do programa e só poderá ser reincluída digitando-a. Por este processo, linhas indesejáveis podem ser removidas de um programa BASIC.

Variáveis, constantes e expressões

Variáveis

Uma variável é um símbolo que representa um número. Em Applesoft BASIC a variável é formada por uma letra, uma letra seguida de outra letra ou um número de um único dígito. São exemplos de variáveis permitidas

A, AA, AZ, A1, A9

Os nomes de variáveis podem ser mais longos, mas só os dois primeiros caracteres são significativos. Os nomes de variáveis nas instruções não podem usar palavras reservadas do BASIC (veja a seção de Palavras Reservadas no Capítulo 2). Em outras versões de BASIC, particularmente aquelas disponíveis em microcomputadores, os nomes de variáveis estão restritos a uma letra ou uma letra seguida por um número de um único dígito.

Constantes

O BASIC aceita constantes expressadas tanto na forma inteira, decimal ou exponencial.

255	é um número inteiro
26.75	é um número decimal (ou de ponto flutuante)
2.368E9	é um número exponencial. A letra E, neste caso, é lida como 'vezes dez elevado à potência de'.

Os resultados de cálculos são impressos na forma inteira ou decimal quando o número couber em nove caracteres. Além deste limite os resultados de cálculos que forem muito grandes ou pequenos são impressos na forma exponencial. Por exemplo: um número como 2369000000 será impresso como 2.369E9 e 0.00000005126 como 5.126E-8.

Expressões

Uma expressão, quando referida neste texto, indica uma constante, uma variável ou uma combinação de uma delas ou ambas, separadas por operadores aritméticos.

Introduzindo um novo programa

Para introduzir um novo programa no computador, digite o comando

NEW

e para ver o que acontece, digite LIST. Com o comando NEW, o programa que estava armazenado na memória é apagado e perdido completamente. O computador está pronto para receber um novo programa. Tente usar estes comandos.

O que se segue é um exemplo de programa que atribui valores a três variáveis e em seguida calcula e imprime a sua média.

```
10 REM MÉDIA
20 A = 26.9
30 B = 28.2
40 C = 27.3
50 PRINT A, B, C
60 D = A + B + C
70 X = D/3
80 PRINT "MÉDIA = " ; X
90 END
```

Quando rodar (RUN) o programa, o BASIC irá responder escrevendo

```
26.9              28.2              27.3
MÉDIA = 27.466667
```

Existem algumas instruções desconhecidas neste programa que precisam ser explicadas.

Na linha 1Ø a instrução REM (do inglês Remark *) permite que o usuário inclua notas e comentários em um programa. Observações ao longo do corpo de um programa longo ajudam o programador a lembrar-se da função de cada trecho de um programa. A instrução REM não afeta a execução do programa, mas ocupa espaço na memória.

Na linha 50 existem três variáveis em uma instrução PRINT, separadas por vírgula. Isto causa impressão dos valores destas variáveis na mesma linha nos pontos de tabulação 1, 16 e 32, respectivamente. As duas primeiras variáveis são impressas em um campo de 16 caracteres cada, enquanto que a terceira é impressa em um campo de 8 caracteres, dando um total de 40 caracteres impressos por linha, assumido para a largura da tela de seu monitor. Se as três variáveis A, B e C fossem separadas por ponto-e-vírgula, o BASIC iria imprimir os seus valores sem espaços entre eles. Tente fazê-lo, redigitando a linha 50 e rodando (RUN) o programa.

A linha 80 tem a palavra MÉDIA = entre aspas. Isto faz com que o computador imprima tudo que estiver entre aspas exatamente como foi escrito. É uma maneira de se colocar cabeçalhos nas saídas (output) de seu computador.

Formatação da instrução PRINT

A apresentação de resultados pode ser feita facilmente utilizando-se a expressão PRINT TAB, que permite a tabulação da mesma forma que em máquinas de escrever. Existem 40 posições de tabulação em cada uma das 24 linhas disponíveis em sua tela. O programa abaixo ilustra o uso desta instrução.

```
10 REM USO DE PRINT TAB
20 A = 15
30 B = 25
40 C = 10
50 D = 20
60 PRINT TAB (5); "A"; TAB (10); "B"; TAB (15); "C"; TAB (20); "D"
```

* N. do T. – Remark = observação.

```
70 PRINT TAB (5); A; TAB (10); B; TAB (15); C; TAB (20); D
80 END
```

Digite e rode (RUN) este programa. Observe que se o valor numérico entre parênteses da instrução PRINT TAB for menor que 1 ou maior que 40, você receberá a mensagem

```
?ILLEGAL QUANTITY ERROR
(QUANTIDADE ILEGAL)
```

na execução.

A instrução PRINT TAB não pode ser usada para mover para a esquerda da posição corrente de impressão em uma dada linha. Somente movimentos progressivos para a direita são obedecidos.

Problema 1.1

Modifique o programa 'média' incorporando a instrução PRINT TAB para que a saída (output) apareça na forma tabular, abaixo dos respectivos cabeçalhos como mostrado abaixo

VALORES: A B C MÉDIA

O Applesoft BASIC possui mais duas instruções de tabulação adicionais, HTAB e VTAB, que devem ser usadas junto com a instrução PRINT. Isto permite o posicionamento do cursor em qualquer ponto da tela, independente da sua posição corrente. No entanto, para ilustrarmos o seu uso, faz-se necessário limpar toda a tela. Isto pode ser feito digitando-se ESC 'SHIFT' P, como vimos anteriormente, ou usando-se o comando HOME.

O comando HOME pode ser usado tanto em modo imediato como não-imediato. Em ambos os casos ele limpa a tela e posiciona o cursor no canto superior esquerdo da tela. Este comando, no modo não-imediato, está incorporado no programa abaixo que imprime a letra X no centro da tela

```
10 REM USO DE HOME,
20 REM HTAB E VTAB
30 HOME
40 HTAB 20
50 VTAB 12
60 PRINT "X"
70 END
```

Observe que o valor de HTAB deve estar entre 1 e 40, assumindo que 40 caracteres seja a largura de seu monitor, enquanto que para VTAB deve estar entre 1 e 24. Valores fora destes limites causam a emissão de uma mensagem de erro.

A instrução INPUT

A instrução INPUT é usada para entrada de dados do teclado. Isto será ilustrado escrevendo-se um programa que calcula 15% de qualquer número. Este número é introduzido através da instrução INPUT, como está mostrado a seguir.

```
10 REM PROGRAMA DE PERCENTAGEM
20 INPUT A
30 B = 15
```

```
40  X = A * B/100
50  PRINT X
60  END
```

A instrução INPUT na linha 20 leva o BASIC a pausar a execução, exibir um ponto de interrogação (?) e esperar que o usuário digite um valor numérico. Tente-o, rodando o programa. Quando o computador exibir o ponto de interrogação, digite 300 (sem esquecer 'RETURN'). O BASIC irá executar agora as instruções remanescentes e imprimirá *

```
45
```

O programa pode ser generalizado usando-se a instrução INPUT na linha 3Ø. Desta forma, qualquer percentagem pode ser calculada. Para evitar que se cometam erros ao responder as interrogações, é aconselhável incorporar uma mensagem na instrução INPUT no programa. Por exemplo: troque as linhas 2Ø e 3Ø pelas seguintes:

```
20  INPUT "ENTRE VALOR"; A
30  INPUT "ENTRE %"; B
```

Quando rodar (RUN) este programa, o computador irá escrever

```
ENTRE VALOR
```

com o cursor, após a última letra da mensagem, esperando por uma entrada numérica. Se digitarmos (por exemplo) 4ØØ, o computador irá então executar a linha 3Ø e escreverá

```
ENTRE %
```

e esperará por uma entrada numérica. Digite 1Ø. O computador executará o resto do programa e escreverá

```
40
```

Observe que quando uma palavra delimitada (chamada de string) é incorporada a uma instrução de INPUT, o ponto de interrogação, que foi previamente impresso pelo computador, desaparece.

Uma versão ainda melhor do último programa será:

```
10  REM PERCENTAGEM GERAL
20  INPUT "ENTRE VALOR "; A
30  INPUT "ENTRE % "; B
40  X = A * B/100
50  PRINT B; "% OF "; A;" = "; X
60  END
```

Quando rodar este programa e entrar com os mesmos valores para A e B usados anteriormente, o computador escreverá:

```
10 % OF 400 = 40
```

que, visivelmente, é uma resposta mais clara para o usuário. Observe que cada string nas instruções INPUT deste programa possui um espaço antes de fechar as aspas. Isto foi introduzido deliberadamente a fim de se obter um espaço entre o fim do string e o início do valor numérico proveniente do teclado.

SALVE (SAVE) este programa, para uso posterior, sob o nome de arquivo G. PERCENT.

* Vide nota: imprimir = exibir.

Problema 1.2

Escreva um programa, usando a instrução INPUT, que converta graus Fahrenheit (F) em graus Celsius (C). Use a fórmula

C = (F – 32) * 5/9

As instruções READ e DATA

Em exemplos anteriores, valores constantes eram atribuídos a variáveis no próprio programa ou através da utilização da instrução INPUT; se mais variáveis fossem necessárias, mais atribuições como estas seriam feitas. No entanto, em programas que requerem muitas variáveis e constantes, as instruções READ e DATA devem ser usadas.

A instrução DATA introduz uma constante numérica, ou uma série de constantes, em um programa. A instrução READ associa nome de variáveis seqüencialmente com os valores das constantes fornecidos pela instrução DATA. As instruções READ e DATA devem se acompanhar em um programa, embora não precisem aparecer aos pares. Se cinco variáveis aparecem em uma ou mais instruções READ, são necessárias pelo menos cinco constantes em uma ou mais instruções DATA.

No exemplo seguinte, todos os dados são introduzidos em uma única instrução DATA. Ela é usada em dois pontos do programa por duas instruções READ.

```
10 REM USO DA INSTRUÇÃO READ
20 READ A, B
30 X = A + B
40 PRINT A, B, X
50 READ C, D
60 Y = C + D
70 PRINT C, D, Y
80 DATA 1, 5, 2, 6
90 END
```

Quando rodar este programa, o BASIC irá responder escrevendo

```
1             5             6
2             6             8
```

Na execução deste programa, o BASIC ignora a instrução DATA até encontrar uma instrução READ. Quando isto ocorrer, ele retorna à linha de menor número do programa para procurar uma instrução DATA. No exemplo ele encontra uma na linha 80, a penúltima linha do programa. Tomando valores constantes seqüencialmente, ele os associa às variáveis na instrução READ, também dispostas seqüencialmente: O valor 1 é atribuído à variável A e o valor 5 à variável B. Deixando uma marca no elemento seguinte da instrução DATA, 2, o computador retorna à linha 30, que é a próxima instrução a ser executada. Ao chegar à linha 50, onde outra instrução READ é encontrada, o BASIC não procura pela instrução DATA, mas retorna à marca por ele deixada para obter o elemento seguinte da instrução DATA que ainda não foi usada. Os valores 2 e 6 são atribuídos às variáveis C e D, respectivamente.

Se o número total de constantes em todas as instruções DATA for inferior ao número total de variáveis em instruções READ, o BASIC irá emitir uma mensagem de erro indicando falta de dados. O excesso de dados é ignorado.

Problema 1.3

Escreva um programa, usando as instruções READ e DATA, que permita a atribuição de três números às variáveis D, H, e M. A primeira variável representa o número de dias, a segunda o número de horas e a terceira o número de minutos. Em seguida calcule e imprima o número total de minutos representado pelos números da instrução DATA.

A instrução RESTORE (rearmazenar)

A instrução RESTORE não possui parâmetros ou opções. Ela simplesmente possibilita a reciclagem da instrução DATA, começando pela linha de menor número que contenha esta instrução. O exemplo seguinte ilustra o seu uso.

```
10 REM USO DE RESTORE
20 READ A, B
30 PRINT A, B
40 RESTORE
50 READ C, D
60 PRINT C, D
70 DATA 2, 7
80 END
```

Ao rodar o programa o BASIC irá escrever

```
2             7
2             7
```

A instrução RESTORE, na linha 40, permite que a instrução READ da linha 50 obtenha valores da instrução DATA, apesar dos mesmos valores já terem sido usados anteriormente na instrução READ da linha 20. Sem a instrução RESTORE, a emissão de uma mensagem de erro indicando a falta de dados para a segunda instrução READ teria ocorrido. Esta instrução simplesmente desloca o marcador da lista de dados de volta ao início da mesma. Por isso, ela não terá efeito em programas que não possuam as instruções READ e DATA.

Prioridades em operações aritméticas

Vamos agora examinar o programa 'Média', usado no início deste capítulo, a fim de analisarmos detalhadamente como foram executadas as operações aritméticas. Abaixo, reescreveremos o programa usando as instruções READ e DATA para ler os números cuja média será calculada.

```
10 REM MÉDIA COM READ
20 READ A, B, C
30 D = A + B + C
40 X = D/3
50 PRINT X
60 DATA 26.9, 28.2, 27.3
70 END
```

Os cálculos, neste programa, são efetuados nas instruções 30 e 40. Combinando-as em uma linha, escreveríamos

X = (A + B + C)/3

É importante o numerador desta expressão estar entre parênteses. Se não estivesse, o BASIC iria calcular inicialmente C/3 e depois somá-lo com A + B, o que daria um resultado incorreto. Isto é devido a um sistema de prioridades embutido no BASIC (veja Tab. 1.1). Ao calcular expressões aritméticas, o BASIC as executa em ordem de prioridade indicada na Tab. 1.1. As porções das expressões que estiverem entre parênteses são calculadas primeiro. Nos "ninhos" de parênteses os cálculos são efetuados a partir do agrupamento mais interno e para fora.

O BASIC não admite dois operadores consecutivos. Por exemplo: A * – N não é aceito. Esta expressão deve ser escrita como A * (– N). Através do uso de parênteses, a ordem de prioridade de execução pode ser alterada, alterando também o valor final da expressão. Se uma linha possuir uma expressão que contenha vários operadores de igual prioridade, o BASIC os calculará da esquerda para a direita.

Examinemos agora como uma expressão complicada, tal como

Y = (A + B * X) ↑ 2/C – D * X ↑ 3

é calculada

TABELA 1.1 Operadores aritméticos e suas prioridades

Símbolo BASIC	*Exemplo*	*Prioridade*	*Função*
↑	A ↑ N	1	Exponenciação
*	A * N	2	Multiplicação
/	A/N	2	Divisão
+	A + N	3	Adição
–	A – N	3	Subtração

Nós assumimos que os valores de A, B, C, D e X são conhecidos. Primeiro, a porção entre parênteses da expressão será calculada. Dentro dos parênteses, a multiplicação tem um grau de prioridade maior; por esta razão ela será calculada primeiro. O número A será então somado ao produto anterior que resultará em um valor numérico que chamaremos de Z. Agora a expressão está reduzida ao seguinte:

Y = Z ↑ 2/C – D * X ↑ 3

A expressão acima possui duas expressões exponenciais, sendo calculada primeiro a que estiver mais à esquerda. Chamando de Z1 o resultado de Z ↑ 2 e X1 para o de X ↑ 3, a expressão é agora reduzida a

Y = Z1/C – D * X1

Analogamente, como a divisão e multiplicação possuem o mesmo grau de prioridade, a expressão que estiver mais à esquerda será calculada primeiro. Finalmente o resultado da multiplicação é subtraído do resultado da divisão e atribuído a Y.

Todo este processo, é claro, é feito automaticamente pelo BASIC, mas aqueles que pretendem usar expressões matemáticas complicadas devem estar familiarizados com ele.

Observe que o que aparece acima como uma equação é, de fato, uma instrução de atribuição e não uma identidade algébrica. Desde que sejam conhecidos os valores das variáveis à direita do sinal de igualdade, o resultado calculado será atribuído à variável à esquerda do sinal de igualdade. Como exemplo, considere as seguintes linhas de programa.

```
10 K = 0
20 K = K + 1
30 PRINT K
40 GOTO 20
```

A linha 20, se fosse uma expressão algébrica, não teria significado. Em termos de computação a instrução diz: "Tome o valor corrente em K, adicione um a ele e armazene o resultado em K'. A primeira vez que a linha 20 é executada, o valor de K (determinado na linha 10) é zero, e adicionando um a ele resulta em um novo valor para K igual a um. Ao se reexecutar a linha 20 pela segunda vez, o valor corrente de K é agora um e adicionando-se um a ele resulta em dois, que é armazenado em K. Em outras palavras, K é incrementado de um toda vez que a linha 20 for executada.

Problema 1.4

Para cada um dos seguintes exemplos, as variáveis usadas têm os seguintes valores:

Variável:	A	B	C	D	E	F	G
Valor:	5	3	8	4	7	2	6

Use seu computador em modo imediato para calcular o resultado correto das expressões dadas a seguir. Para chegar à resposta final, calcule todos os passos intermediários na ordem regida pelo processo de prioridades.

```
X1 = A * B ↑ E + F
X2 = A * B ↑ (E + F)
X3 = A * B/C * D
X4 = A * B/(C * D)
X5 = A + B * G + C * G ↑ 2 + D * G ↑ 3
X6 = (A + B) * G + C * G ↑ 2 + D * G ↑ 3
X7 = (A ↑ F + (B – 1/C) ↑ F) ↑ 0.5
X8 = (A ↑ F + B – 1/C ↑ F) ↑ 0.5
X9 = A/B ↑ 2 – C * D/((E + F) + G ↑ 3)
```

Para saber a resposta correta destas expressões a fim de conferir as suas respostas, escreva um programa em BASIC que atribua valores às variáveis A, B, C etc. e em seguida calcule as expressões X1, X2, X3 etc.

CAPÍTULO 2

Controle de fluxo do programa

A instrução GOTO

Esta instrução provoca um salto imediato a um número de linha indicado. A execução do programa prossegue seqüencialmente a partir do ponto indicado. Para ilustrar esta instrução, carregue (LOAD) o programa G. PERCENT, LISTe-o e digite

```
55 GOTO 20
```

Ao rodar este programa, você verá que o fluxo é automaticamente transferido à linha 20, que faz com que o programa pergunte continuamente por um novo conjunto de entrada. A única maneira de se parar este programa é digitando

```
'CTRL'C
```

Este é o único comando capaz de parar a listagem ou a impressão de programa.

A instrução IF

A instrução IF possibilita a ramificação de programas. Digite

```
25 IF A=0 GOTO 60
```

e re-rode (RUN) o programa. Se agora você desejar parar a execução, simplesmente digite 0 (zero) quando 'ENTRE VALOR' aparecer na tela. Quando o BASIC encontrar esta instrução IF, ele compara o valor da variável A com a constante que aparece após o operador lógico (neste caso, o sinal de igualdade). Se a condição teste for satisfeita, o restante da instrução IF é executado (neste caso transferindo o controle da linha 60). Se, no entanto, a condição teste não for satisfeita, a linha seguinte é executada (neste caso a linha 30).

Existe uma segunda variação da instrução IF que permite a inclusão de outras instruções BASIC em seu complemento. Um exemplo disto seria

```
25 IF A=0 THEN END
```

Observe que o uso de GOTO como complemento obrigava-o a ser precedido de um "número de linha", enquanto que o uso de THEN permite a inserção de uma outra instrução BASIC tal como END. A última modificação da linha 25 torna a linha 60 do programa supérflua e por isso pode ser deletada. Mais adiante, para possibilitar o aparecimento de linhas em branco entre execuções sucessivas do programa, digite

```
52 PRINT
```

Ao encontrar esta linha, o BASIC simplesmente imprimirá um espaço e pulará para a próxima linha (se houver impressora o papel subirá uma linha). Faça tudo isto e SALVE

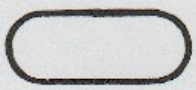
(SAVE) o programa no disquete sob o nome de arquivo PERCENT, recarregue-o (LOAD) e liste-o. Você deve ter agora na sua tela o seguinte programa

```
10  REM PERCENTAGEM
20  INPUT "ENTRE VALOR "; A
25  IF A = 0 THEN END
30  INPUT "ENTRE % "; B
40  X = A * B/100
50  PRINT B;" % OF "; A;" = "; X
52  PRINT
55  GOTO 20
```

Fluxogramas

Uma ferramenta importante da programação é o fluxograma, ou diagrama de bloco, que permite ao programador planejar a seqüência de operações antes de escrevê-las. O fluxograma é composto de um conjunto de "blocos", cujos formatos indicam a natureza da operação neles descrita, interligados por linhas e flechas para indicar o fluxo de controle entre as diversas operações.

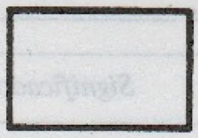
Os seguintes formatos são os mais usuais:



O oval indica o ponto de início ou término do programa



Um pararelogramo indica uma operação de entrada ou saída de dados



Um retângulo indica qualquer operação de processamento, exceto uma decisão



Um losango indica uma decisão



Um pequeno círculo indica uma ligação entre dois pontos



Uma flecha indica a direção do fluxo

Como exemplo, o fluxograma da última versão do programa PERCENT é apresentado a seguir.

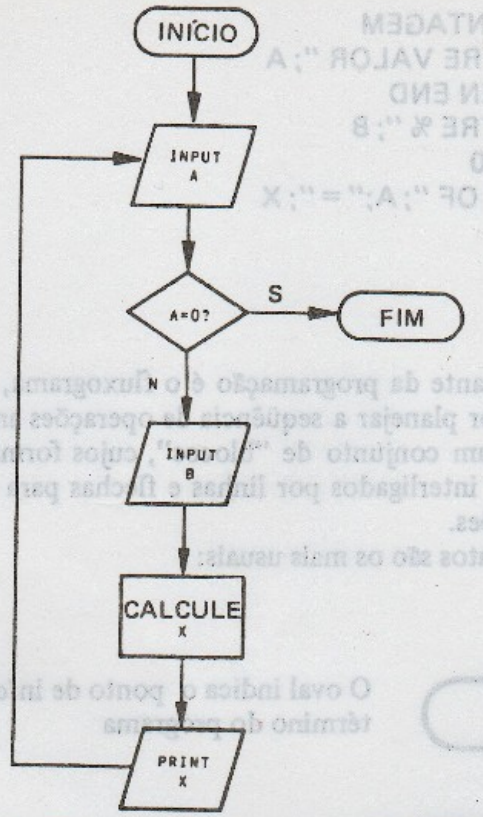


Fig. 2.1 Fluxograma do programa PERCENT.

Operadores lógicos dentro de uma instrução IF

A Tab. 2.1 mostra todos os operadores lógicos permitidos em uma instrução IF.

TABELA 2.1 Operadores Lógicos

BASIC símbolo	*Exemplo*	*Significado*
=	A = B	A igual a B
<	A < B	A menor que B
<=	A <= B	A menor ou igual a B
>	A > B	A maior que B
>=	A >= B	A maior ou igual a B
<>	A <> B	A não igual a B

Espaçamento das instruções BASIC

Já deve, sem dúvida, ser óbvio que você deve deixar espaços entre palavras ao digitar uma instrução no computador. O BASIC insere os seus próprios espaços a fim de tornar as instruções legíveis. Para testar isto, digite

```
251FA=0THENEND
```

e depois liste-o digitando 'LIST 25'. O BASIC irá inserir espaços para facilitar a nossa leitura.

ATENÇÃO: Palavras reservadas

Existe um tipo de instrução IF que, mesmo tendo sido bem digitada, não será corretamente interpretada pelo Applesoft. Para ver isto, adicione a seguinte linha ao programa PERCENT

```
35 IF B<A THEN PRINT "CORRETO"
```

Ao rodar o programa, o BASIC irá acusar um erro na linha 35. Se agora você listar a linha 35 ('LIST 35'), verá que o Applesoft modificou a sua instrução para

```
35 IF B<AT HEN PRINT "CORRETO"
```

Ele uniu a variável A e a letra T (da palavra THEN) e conseqüentemente não reconheceu a palavra HEN. Isto se dá porque a palavra AT é uma "palavra reservada". A lista a seguir dá todas as palavras reservadas de suas letras em Applesoft BASIC.

```
AT FN GR IF ON OR TO
```

Para palavras-chave com mais de duas letras consulte o seu Manual de Applesoft.

Para retificar o erro causado pela linha 35, redigite-a da seguinte forma:

```
35 IF A>B THEN PRINT "CORRETO"
```

Esta linha possui a mesma operação lógica contida na última linha 35, mas evita que A e THEN apareçam próximas uma da outra.

Classificação de dados

O programa a seguir nos permite entrar com dois números, testar qual deles possui maior valor e imprimi-los em ordem descendente.

```
10 REM CLASSIFICAÇÃO DE DOIS NÚMEROS
20 INPUT A, B
30 IF A<0 THEN END
40 IF A>=B THEN PRINT A, B
50 IF A>=B GOTO 20
60 PRINT B, A
70 GOTO 20
```

O programa pode ser interrompido introduzindo-se um valor negativo para A. Do contrário, A é comparado com B e a instrução PRINT apropriada será executada.

O problema de classificação se torna mais complicado, contudo, se em vez de dois números nós inserirmos um terceiro. De fato, sem a ajuda de um fluxograma seria quase impossível seguir a lógica exigida para solucionar corretamente este problema. Para a classificação de dois números tínhamos duas possíveis instruções PRINT (1 * 2). Para a classi-

ficação de três números, no entanto, o número total de instruções PRINT passa a ser seis (1 * 2 * 3), sendo as combinações (A, B, C), (A, C, B), (C, A, B), (C, B, A), (B, C, A) e (B, A, C).

Uma pequena variação na instrução IF pode ajudar a reduzir o número total de instruções exigidas neste problema. Podemos, por exemplo, perguntar se A é maior que B e B maior que C em uma instrução, logo combinando duas instruções em uma. O formato desta instrução IF é:

```
IF A > B e B > C THEN PRINT A, B, C
```

O complemento desta instrução IF só será executado se, e somente se, A for maior que B e B maior que C.

Existe outra combinação da instrução IF que é útil:

```
IF A > B OU B > C THEN PRINT A, B, C
```

Problema 2.1

Desenhe um fluxograma e depois escreva um programa para classificar três números em ordem descendente, incorporando uma instrução IF composta. Rode (RUN) o programa e teste-o para diversas combinações de números para as variáveis A, B e C.

Finalmente, eis uma forma como, com apenas duas instruções IF e uma instrução PRINT, a mesma solução para o problema de classificação de três números pode ser obtida. Ela usa uma lógica diferente e é explicada aqui com o auxílio da imagem de três cartas de baralho. Imagine que você esteja segurando três cartas em sua mão e que deseja ordená-las em ordem descendente. Olhe para as duas primeiras (Fig. 2.2a) e arrume-as de forma que a de maior valor fique na frente. Agora olhe para as duas últimas (Fig. 2.2b) e arrume-as de forma que a maior destas duas fique na frente. Obviamente, se a maior carta mostra a Fig. 2.2c), e uma repetição de todo o processo seria necessária para assegurar que a carta de maior valor esteja na frente (Fig. 2.2d).

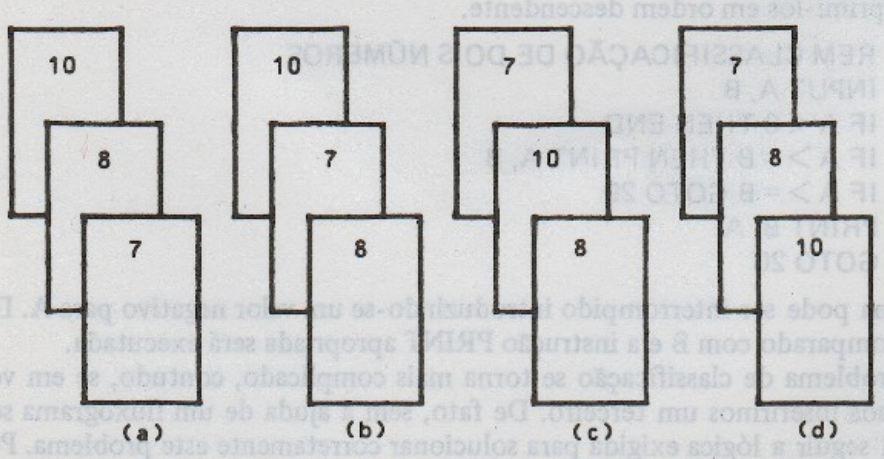


Fig. 2.2 Classificação das cartas em ordem descendente.

O fluxograma para esta lógica está na Fig. 2.3. Observe as duas trocas ou rotações dos números que ocorrem.

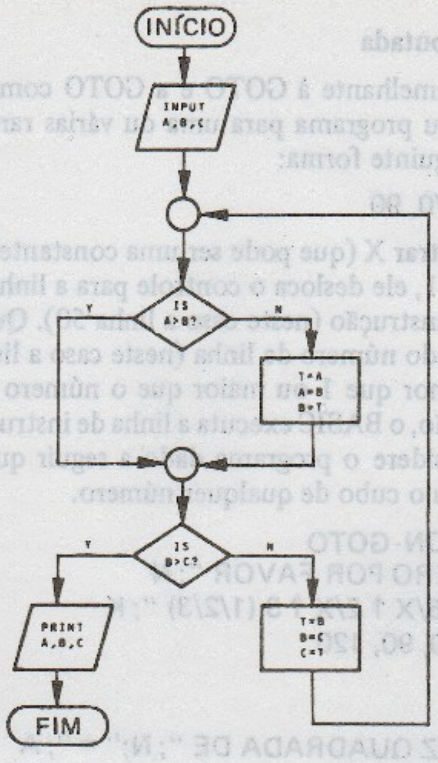


Fig. 2.3. Fluxograma para classificar três números

Se o valor em A for menor do que em B, troque-os de forma que o valor em A esteja agora armazenado em B e o valor de B armazenado em A. Note, entretanto, que ao colocar o valor de B em A, perderemos o número armazenado em A (por superposição). Por isso, transferimos o conteúdo de A para uma variável temporária T, em seguida o conteúdo de B para A e finalmente o conteúdo de T para B. A segunda rodada, necessária quando B é menor que C, é obtida de uma maneira semelhante. Agora podemos escrever o programa para obtermos esta solução.

```
10   REM CLASSIFICAÇÃO DE TRÊS NÚMEROS
20   INPUT A, B, C
30   IF A > B GOTO 70
40   T = A
50   A = B
60   B = T
70   IF B > C GOTO 120
80   T = B
90   B = C
100  C = T
110  GOTO 30
120  PRINT A, B, C
130  END
```

Digite este programa e salve-o (SAVE) sob o nome de arquivo CLASSIF TRES NUM. É claro que existem diversas soluções possíveis para este problema de classificação, mas dos três métodos até agora sugeridos, este último é o mais curto e eficiente.

A instrução GOTO computada

Uma instrução muito semelhante à GOTO é a GOTO computada, que permite que você desvie a execução do seu programa para uma ou várias ramificações em seu programa. A instrução é escrita da seguinte forma:

```
ON X GOTO 50, 70, 90
```

Quando o BASIC encontrar X (que pode ser uma constante ou uma expressão) ele calcula o seu valor e, se este for 1, ele desloca o controle para a linha cujo número está em primeiro no complemento da instrução (neste caso a linha 50). Quando X é igual a 2, o controle é deslocado para o segundo número de linha (neste caso a linha 70), e assim por diante. Se X assumir um valor menor que 1 ou maior que o número total de números de linha no complemento da instrução, o BASIC executa a linha de instrução seguinte à do ON-GOTO.

Como exemplo, considere o programa dado a seguir que pode executar tanto a raiz quadrada, o quadrado ou o cubo de qualquer número.

```
10  REM USO DE ON-GOTO
20  INPUT "NUMERO POR FAVOR "; N
30  INPUT "X ↑ 0.5/X ↑ 2/X ↑ 3 (1/2/3) "; K
40  ON K GOTO 60, 90, 120
50  END
60  A = N ↑ 0.5
70  PRINT "A RAIZ QUADRADA DE "; N;" = "; A
80  GOTO 20
90  B = N ↑ 2
100 PRINT "A RAIZ QUADRADA DE "; N;" = "; B
110 GOTO 20
120 C = N ↑ 3
130 PRINT "O CUBO DE "; N;" = "; C
140 GOTO 20
```

Para parar este programa digite 0 (zero) ou um número maior que 3 como resposta à segunda instrução INPUT.

Problema 2.2

Escreva um programa que possa efetuar qualquer uma das seguintes conversões:

(a) galões para litros (1 galão = 4,54609 litros)
(b) pés para metros (1 pé = 0,3048 metros)
(c) libras para quilogramas (1LB = 0,453592 kg)

Use a instrução READ para entrar as constantes de conversão no computador, e a instrução INPUT para a entrada do número a ser convertido e o tipo de conversão desejada.

O loop FOR-NEXT

As instruções FOR e NEXT são usadas para marcar o início e término de loops (laços) do programa. Qualquer instrução entre FOR e seu NEXT correspondente será executada repetidamente de acordo com as condições impostas pela 'variável de controle' contida na instrução FOR. Um exemplo é dado a seguir.

```
10 REM USO DO LOOP FOR-NEXT
20 FOR K = 1 TO 5 STEP 1
30 PRINT K
40 NEXT K
50 END
```

Na linha 20, a variável de controle K assume os valores 1, 2, 3, 4 e 5. Como ela não pode possuir estes valores simultaneamente, um loop (laço) é formado, iniciando com o FOR na linha 20 e terminando com o NEXT na linha 40. As instruções dentro do loop são reexecutadas cinco vezes, cada vez com um novo valor para K. O NEXT causa saltos repetidos à linha 20 até que K exceda o seu valor máximo de 5. Quando isto acontece, o controle passa para a linha seguinte a instrução *next* (neste caso a linha 50).

O loop FOR-NEXT substitui a necessidade de inicialização, teste e incrementação do contador K, assim como a necessidade da instrução GOTO como a do programa seguinte.

```
10 REM ACUMULADOR
20 K = 1
30 IF K > 5 GOTO 70
40 PRINT K
50 K = K + 1
60 GOTO 30
70 END
```

As linhas 20 e 30 deste programa realizam o mesmo que a linha 20 do programa anterior, enquanto que as linhas 50 e 60 possuem a mesma função da linha 40 do programa anterior. Observe o formato da instrução da linha 50 do último programa. Na execução, ela adiciona um ao antigo valor de K e armazena o resultado na variável K. Logo, ela pode ser pensada como um acumulador.

O programa seguinte utiliza o loop FOR-NEXT assim como um acumulador para calcular a soma de uma lista de números.

```
10  REM SOMA DE N NUMEROS
20  READ N
30  X = 0
40  FOR I = 1 TO N STEP 1
50  READ A
60  X = X + A
70  NEXT I
80  PRINT "SOMA DE "; N;" NUMEROS = "; X
90  DATA 5, 20.5, 21.3, 20.8, 20.6, 21.1
100 END
```

Ao rodar este programa, N assume o valor 5 que é o número total de entradas a serem somadas. O acumulador X é então zerado na linha 30, e um loop FOR-NEXT é montado

entre as linhas 40 e 70. Observe que os limites da variável de controle na instrução FOR pode ser escrito em função de outras variáveis. Neste caso, o valor máximo é representado por N (número total de dados). Dentro do loop, cada número é lido em A na linha 50 e acumulado em X na linha 60. Uma vez completado o loop, X possui a soma de todos os números.

O uso de STEP (passo)

No último exemplo, o modificador STEP era igual a + 1. Para este caso, o modificador STEP pode ser omitido e a linha 40 pode ser escrita como

```
40 FOR I = 1 TO N
```

Neste caso o passo é assumido como sendo igual a + 1.

Se o valor desejado para o STEP (passo) não for igual a + 1, o modificador STEP deve ser incluído. Por exemplo:

```
10 FOR I = 5 TO 20 STEP 5
20 C = 2.54 * I
30 PRINT I, C
40 NEXT I
50 END
```

irá converter 5, 10, 15 e 20 polegadas em centímetros.

Um modificador STEP negativo é permitido em BASIC. Por exemplo:

```
10 FOR J = 5 TO 1 STEP – 1
20 PRINT J
30 NEXT J
40 END
```

irá imprimir os valores 5, 4, 3, 2 e 1.

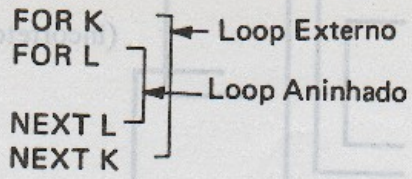
Para passos (STEP) de valores positivos, o loop é executado enquanto a variável de controle for menor ou igual ao seu valor final. Para passos negativos, o loop continua enquanto a variável de controle for maior ou igual ao seu valor final. Se o valor inicial for maior que o valor final e um passo (STEP) positivo for indicado, ou se o valor inicial for menor que o valor final e um passo (STEP) negativo for indicado, o corpo do loop não é executado.

Ninhos de loops FOR-NEXT

As instruções FOR-NEXT podem ser aninhadas para permitir a programação de loops dentro de loops como mostra o exemplo a seguir.

```
10 REM LOOP FOR-NEXT ANINHADOS
20 FOR K = 1 TO 9
30 FOR L = K TO 9
40 PRINT L;
50 NEXT L
60 PRINT
70 NEXT K
80 END
```

Ao rodar este programa, dois loops são montados da seguinte forma:



O loop externo é inicializado com K = 1 e, imediatamente, o loop interno é executado 9 vezes. Em seguida a variável de controle K é incrementada de 1, de modo que agora K = 2 e o loop aninhado é executado 8 vezes. Isto é repetido até que K seja igual a 9, quando o loop aninhado é executado uma só vez. A saída deste programa é a seguinte:

```
123456789
23456789
3456789
456789
56789
6789
789
89
9
```

O ponto-e-vírgula após a variável L na linha 40 permite que as saídas (output) sejam impressas juntas na mesma linha. Entretanto, cada linha de impressão precisa ser concluída com um salto para a linha seguinte. Isto é obtido com a instrução PRINT na linha 60, do contrário todos os números que estão dispostos na tela serão impressos na mesma linha.

Problema 2.3

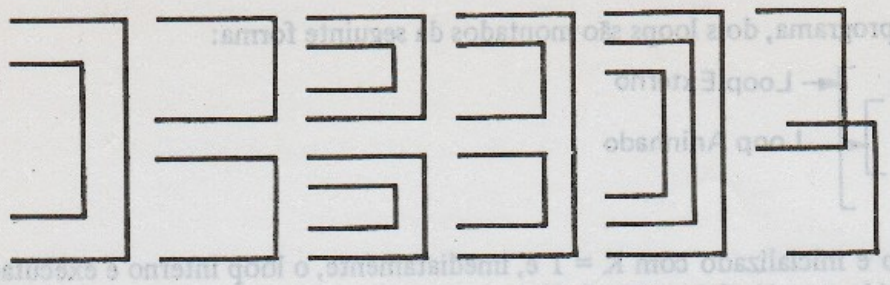
Modifique o programa acima para que a saída (OUTPUT) seja um quadrado de 15 x 15 caracteres, usando a letra X como caractere de saída.

Níveis adicionais de aninhamento são possíveis. Contudo, aninhamentos profundos ocupam muito espaço de memória. A Fig. 2.4 mostra algumas configurações de loop. Os primeiros cinco exemplos são de loops permitidos, enquanto que o sexto não é. As linhas de união entre as instruções FOR-NEXT não devem se cruzar.

O uso de loops FOR-NEXT em gráficos simples

Uma forma limitada de gráficos pode ser obtida com o uso da instrução PRINT. Por exemplo: considere o seguinte programa que irá imprimir a letra L em forma de bloco.

```
10  PRINT "LL"
20  PRINT "LL"
30  PRINT "LL"
40  PRINT "LL"
50  PRINT "LL"
60  PRINT "LL"
70  PRINT "LL"
```



(incorreto)

Fig. 2.4. Algumas configurações de loop

```
80  PRINT "LLLLLLLLL"
90  PRINT "LLLLLLLLL"
100 END
```

Evidentemente, o uso da instrução PRINT desta forma tende a se tornar um tanto grosseiro. No entanto, é bom notar que o programa é composto de apenas dois tipos diferentes de instrução PRINT que são repetidas várias vezes. A instrução da linha 10 é repetida sete vezes, enquanto que a da linha oitenta é repetida duas vezes. É possível então evitar esta repetição introduzindo-se o loop FOR-NEXT como é mostrado no programa a seguir:

```
10 FOR I = 1 TO 7
20 PRINT "LL"
30 NEXT I
40 FOR I = 1 TO 2
50 PRINT "LLLLLLLLL"
60 NEXT I
70 END
```

Problema 2.4

Escreva um programa semelhante, usando os loops FOR-NEXT e instruções PRINT delimitadas, para imprimir a letra E em forma de bloco. Use dois caracteres para a largura de cada linha da letra, nove caracteres para sua largura total e dez caracteres para sua altura.

Seria bom notar que é fácil representar letras ou outros caracteres em forma de bloco, desde que sejam compostos somente de linhas verticais e horizontais. Faz-se necessário, no entanto, o uso da instrução PRINT TAB para formas tais como a letra N que possui uma linha na diagonal.

O programa a seguir irá imprimir uma letra N na forma de bloco.

```
10 PRINT "NN"; TAB (10); "NN"
20 PRINT "NNN"; TAB (10); "NN"
30 FOR I = 1 TO 6
40 PRINT "NN";
50 PRINT TAB (I + 2); "NN"; TAB (10); "NN"
60 NEXT I
```

```
70 PRINT "NN"; TAB (9); "NNN"
80 PRINT "NN"; TAB (10); "NN"
90 END
```

Digite este programa em seu computador, salve-o (SAVE) e em seguida rode-o (RUN). Agora tente incorporar as instruções HOME, HTAB e VTAB para ele imprimir no centro de sua tela. Note que HTAB funciona de forma semelhante a TAB e precisa ser repetida a fim de endentar linhas sucessivas de impressão.

Obs.: A essa altura, seus programas estão ficando bem complicados e correm o risco de terem erro na lógica. Uma vez que isto ocorra, torna-se difícil depurar tal programa, pois nenhuma mensagem de erro é emitida pelo computador para lógica errada. Por isso, precisamos de determinados comandos BASIC que nos ajudem a traçar o fluxo da lógica de um programa em execução, porem o programa em pontos predeterminados, examinem o conteúdo das variáveis e continuem a execução sem perder o valor armazenado nas variáveis do programa.

Todos os comandos e instruções BASIC que podem nos ajudar a depurar um programa estão no Anexo C. Sugerimos que você estude estas técnicas agora e pratique o seu uso.

Capítulo 3
Gráficos — 1.ª parte

Gráficos de baixa resolução

No modo de gráficos de baixa resolução, o Apple usa as mesmas localizações de memória que usa no modo de texto, porém em diferente formato. Para entrar o modo gráfico digite

```
GR
```

Isto irá apagar as vinte linhas de texto superiores na tela do televisor e deixará apenas quatro sinais na parte inferior para que um texto possa ser incluído. Digitando-se

```
TEXT
```

retorna o computador para o modo de texto.

Quando em modo gráfico, a área superior de vinte linhas de texto é dividida em 40 por 40 quadradinhos, cada um dos quais pode ser acessado e receber uma cor (ou para televisores preto-e-branco, diferentes tonalidades de cinza). Existem no total 16 cores disponíveis que possuem valores numéricos de 0 a 15, como é mostrado na Tab. 3.1.

Tab. 3.1 Cores para gráficos de baixa resolução

0	preto	4	verde-escuro	8	marrom	12	verde-claro
1	magenta	5	cinza 1	9	laranja	13	amarelo
2	azul escuro	6	azul-médio	10	cinza 2	14	água marinha
3	roxo	7	azul-claro	11	rosa	15	branco

As localizações* na tela em baixa resolução são as seguintes:

```
0,0   1,0   2,0   3,0   ...  ...  ...  ...  39,0
0,1   1,1   2,1   3,1   ...  ...  ...  ...  39,1
0,2   1,2   2,2   3,2   ...  ...  ...  ...  39,2
...   ...   ...   ...   ...  ...  ...  ...  ...
...   ...   ...   ...   ...  ...  ...  ...  ...
...   ...   ...   ...   ...  ...  ...  ...  ...
0,39  1,39  2,39  3,39  ...  ...  ...  ...  39,39
```

O comando GR pode ser usado em um programa BASIC em modo não-imediato, como mostramos a seguir. O programa irá colorir o quadrado 2,5 (2ª coluna, 5ª linha) de amare-

* N. do T. – Localizações = posições

lo, se seu Apple estiver equipado com uma placa para cores; caso contrário você verá somente um diferente tom de cinza. Digite o programa e rode-o

```
10 GR
20 COLOR = 13
30 PLOT 2,5
40 END
```

Tendo rodado o programa acima, retorne o Apple para o modo de texto e escreva um programa que irá colorir os quatro cantos da tela com cores de sua escolha.

Suponhamos agora que desejamos colorir de laranja todos os 40 quadrados da 5ª linha. Poderíamos escrever um programa que contenha a instrução PLOT 40 vezes. ou então usar um loop FOR-NEXT com apenas uma instrução PLOT, como se segue.

```
10 GR
20 COLOR = 9
30 FOR I = 0 TO 39
40 PLOT I, 5
50 NEXT I
60 END
```

Digite este programa e rode-o. Em seguida, escreva um programa que desenhe uma linha ao redor do bordo da área gráfica.

O Apple possui duas instruções específicas que permitem desenhar linhas verticais e horizontais de forma ainda mais simples que o loop FOR-NEXT. Estas são HLIN e VLIN. Para colorir a primeira linha de roxo e a última coluna de rosa, digite

```
10 GR
20 COLOR = 3
30 HLIN 0,39 AT 0
40 COLOR = 11
50 VLIN 0,39 AT 39
60 END
```

A instrução na linha 30 faz com que uma linha horizontal seja desenhada da coluna 0 à coluna 39 na linha 0, enquanto que a instrução na linha 50 faz com que uma linha vertical seja desenhada da linha 0 à linha 39 na coluna 39.

Desenhando quadros

O programa seguinte usa um loop FOR-NEXT para desenhar uma sucessão de linhas horizontais e verticais, de fora para dentro, a partir do bordo da região gráfica, sendo cada linha sucessiva encurtada de dois quadrados em relação à anterior. Isto é repetido 16 vezes, cada vez desenhando um quadrado menor e de cor diferente.

```
10  REM QUADRADOS COLORIDOS
20  GR
30  FOR K = 0 TO 15
40  COLOR = K
50  L = 39 - K
60  HLIN K, L AT M
70  HLIN K, L AT L
80  VLIN K, L AT M
```

```
 90  VLIN K, L AT L
100  NEXT K
110  END
```

Gráficos de barra e histogramas

A representação de dados estatísticos é uma área em que gráficos de baixa resolução são muito úteis. Usaremos como exemplo o custo de manutenção de uma casa e assumiremos que o custo total semanal é de 16 unidades gastas em gêneros alimentícios, 8 com carne, 6 com legumes, 4 com leite e 2 com pão. Uma maneira de representar esta informação é com o uso de gráficos de barra (Fig. 3.1).

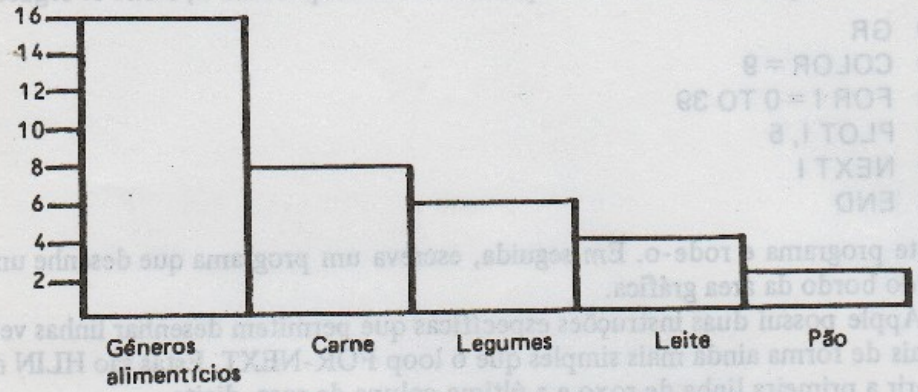


Fig. 3.1 Gráfico de barra das despesas domésticas

Um programa para representar esta informação é dado a seguir.

```
 10  REM GRAFICO DE BARRA COM LARGURA VARIAVEL
 20  INPUT "LARGURA DA BARRA? "; K
 30  INPUT "FATOR DE ALTURA? "; M
 40  HOME
 50  GR
 60  FOR I = 0 TO 4 * K STEP K
 70  READ A
 80  COLOR = A - 1
 90  DATA 16, 8, 6, 4, 2
100  FOR J = I TO I + K - 1
110  VLIN 39 - M * A, 39 AT J
120  NEXT J
130  NEXT I
140  VTAB 21
150  PRINT TAB (1); "G"; TAB (K + 1); "M"; TAB (2 * K + 1); "V";
     TAB (3 * K + 1); "M"; TAB (4 * K + 1); "B"
160  END
```

Primeiro o programa pede a especificação da largura da barra. Esta deve depender do número total de itens, visto que a largura x número total de barras deve ser menor que 39. Analogamente, a resposta para a segunda pergunta 'fator de altura?' depende do dado de maior valor.

Digite o programa e rode-o algumas vezes com diferentes valores para 'largura' e 'fator altura'. Se algum deles exceder o limite, você irá receber uma mensagem de erro indicando quantidade ilegal. Observe como as legendas que indicam o tipo de gasto abaixo de cada barra mudam de posição na tela para valores diferentes de largura K. Isto é obtido com o uso da instrução PRINT TAB, enquanto que o posicionamento vertical é obtido com o uso da instrução VTAB.

Da mesma maneira, um histograma pode ser desenhado para representar as notas obtidas pelos candidatos em um exame, após terem sido agrupados de forma adequada, tais como:

Notas	0 – 9	10 – 19	20 – 29	30 – 39	etc
Nº de candidatos	1	3	15	30	etc

Problema 3.1

Uma pesquisa das profissões dos sócios de um certo clube produziu as seguintes informações:

1º ano:	Contadores 1	Banqueiros 5	Médicos 8
	Engenheiros 10	Bombeiros 5	Advogados 6
	Enfermeiros 16	Policiais 15	Professores 9
2º ano:	Contadores 3	Banqueiros 3	Médicos 6
	Engenheiros 8	Bombeiros 0	Advogados 4
	Enfermeiros 18	Policiais 8	Professores 12

*Escreva um programa que irá desenhar um gráfico de barra para representar toda informação, escolhendo cores diferentes para representar cada ano. Lembre-se de que a segunda cor é "plotada" * em cima da anterior, logo será necessário plotar as barras mais longas em primeiro lugar, para cada profissão.*

Gráficos de alta resolução

O uso de gráficos de alta resolução permite que sejam desenhadas figuras bem mais detalhadas que as desenhadas em baixa resolução e constituídas de 280 por 160 pontos de plotagem. Assim como em baixa resolução, a coordenada X (número da coluna) começa em 0, mas agora termina em 270, enquanto que a coordenada Y (número de linha) começa em 0, mas agora termina em 159.

Os gráficos de alta resolução possuem comandos semelhantes aos de baixa resolução, sendo apenas adicionado um H antes de cada um deles. Logo,

```
HGR
```

coloca em modo de alta resolução e

```
HPLOT 140, 80
```

plota um pequeno ponto no centro da tela. Contudo o aumento na resolução resulta em uma diminuição de cores. As cores disponíveis são dadas na Tab. 3.2, apesar de estas tenderem a variar de televisor para televisor, assim como o seu posicionamento na tela.

* N. do T. – Referente a plotar.

Há ainda um comando para acionar a alta resolução que transforma toda a tela para o modo gráfico (280 por 192 pontos) sem deixar qualquer espaço reservado para texto. O comando é

```
HGR2
```

embora semelhante a HGR em todos os outros aspectos, ele não se encontra disponível nos sistemas com menos de 24K de memória.

Tabela 3.2 Cores para gráficos de alta resolução

0	Preto 1	4	Preto 2
1	Verde	5	Laranja
2	Violeta	6	Azul
3	Branco 1	7	Branco 2

ATENÇÃO: Uso dos comandos HGR e HGR2

HGR só está disponível em sistemas que possuem Applesoft em RAM, então deverá usar o comando alternativo

```
HGR2
```

e você deve assegurar-se de que seu programa e suas variáveis não estão nas áreas de memória usadas para o modo gráfico, visto que estas áreas de memória são limpadas ao executar os comandos HGR e HGR2. Para detalhes de como a memória do Apple é organizada, veja o Anexo D.

O método pelo qual você projeta o seu programa BASIC depende do tamanho de seu sistema. Para os sistemas de 24, 32 e 36K de memória, use o comando

```
5  HIMEM: 16384
```

enquanto que para um sistema de 48K de memória, use o comando

```
5  LOMEM: 24577
```

Os motivos destas diferenças são explicados no Anexo D, mas tudo que você precisa saber agora é que tal comando precisa aparecer na última linha de qualquer programa seguinte se, e somente se, você possuir um sistema em que o Applesoft tenha de ser carregado em RAM. Se este for o caso, não se esqueça de substituir HGR 2 por HGR no programa.

Desenhando linhas em alta resolução

Assim como nos gráficos de baixa resolução, o loop FOR-NEXT pode ser usado para desenhar uma linha contínua. Por exemplo, o programa abaixo desenha uma linha horizontal no meio da tela.

```
10 HGR
20 HCOLOR = 1
30 FOR I = 0 TO 279
40 HPLOT I, 80
50 NEXT I
60 END
```

Digite este programa e rode-o. Em seguida, tente mudá-lo de forma a imprimir uma linha vertical no meio da tela.

Desenhar uma linha torna-se ainda mais fácil plotando-se simplesmente de um ponto a outro da tela. Por exemplo, para desenhar uma linha vertical no centro da tela, escreva

```
HPLOT 0,80 TO 279,80
```

que substitui as linhas 30 a 50 do último programa. Com este método, desenhar linhas verticais, horizontais ou diagonais se torna mais fácil. As três linhas de HPLOT no programa abaixo desenham um Z relativamente grande.

```
10 HGR
20 HCOLOR = 5
30 HPLOT 0,0 TO 279,0
40 HPLOT 279,0 TO 0,159
50 HPLOT 0,159 TO 279,159
60 END
```

Note que a linha 40 faz com que uma linha seja desenhada a partir do fim do HPLOT anterior (linha 30) e analogamente a linha 50 faz com que uma linha seja desenhada a partir do fim do HPLOT anterior (linha 40). Uma outra forma de se escrever estas instruções, levando em consideração a continuidade da linha, é a seguinte:

```
30 HPLOT 0,0 TO 279,0
40 HPLOT TO 0,159
50 HPLOT TO 279,159
```

A linha 40 faz com que a plotagem continue a partir do fim da linha até o novo ponto na tela dado pelas coordenadas 0,159. Analogamente, a linha 50 continua a plotagem até o próximo ponto dado por 279,159.

Se você possuir uma versão do Apple II com Applesoft em ROM, você pode encadear as três últimas instruções em uma para obter

```
HPLOT 0,0 TO 279,0 TO 0,159 TO 279,159
```

que torna o desenho de figuras ainda mais fácil. É o equivalente a colocar uma caneta no papel e desenhar uma linha reta entre os pontos A e B e prosseguir com outra de B para C, e assim por diante.

Vejamos agora a possibilidade de se desenharem curvas. Visto que toda curva pode ser desenhada com uma série de pequenos segmentos de reta, é possível desenhar qualquer forma lembrando-nos de que, quanto menor a curvatura, mais curtos devem ser os segmentos para cada HPLOT. O programa a seguir resolve a equação Y = X ↑ 2 para valores de X de 1 a 12, e os plota em pequeno ponto na tela para cada valor calculado de Y.

```
10 HGR
20 HCOLOR = 6
30 HPLOT 0,0 TO 0,159
40 HPLOT TO 140,159
50 FOR X = 1 TO 12
60 Y = X ↑ 2
70 HPLOT 5 * X, 159 – Y
80 NEXT X
90 END
```

Digite este programa e rode-o. Obviamente, se ao invés de plotar pontos isolados plotamos um segmento de um ponto ao seguinte, o resultado seria algo com uma aparência de curva. Isto pode ser obtido adicionando-se a linha

```
45 HPLOT 0,159
```

para dar um ponto de partida para a linha, e mudando a linha 70 para

```
70 HPLOT TO 5 * X, 159 – Y
```

para dar um ponto de continuidade.

Plotando gráficos em alta resolução

O programa abaixo permite que a função Y = X ↑ 2 seja plotada tanto para valores positivos como negativos de X, com um fator multiplicativo para a ordenada X, XFACTOR, o qual você especifica. O valor de XFACTOR está limitado entre 1 e 17, pois valores maiores que 17 resultam em uma posição de plotagem fora do campo da tela. A linha 120 foi, portanto, incorporada ao programa para evitar a emissão do seguinte erro:

```
?ILLEGAL QUANTITY ERROR (quantidade ilegal)
```

Digite este programa e rode-o. Você pode parar o programa digitando 0 (zero)

```
 10  REM PLOTANDO A EQUAÇÃO Y = X ↑ 2
 20  HOME
 30  HGR
 40  HCOLOR = 5
 50  VTAB 24
 60  HPLOT 140,0 TO 140, 159
 70  HPLOT 0,159 TO 279, 159
 80  INPUT "FATOR X? "; XF
 90  IF XF = 0 THEN END
100  X = – 8
110  Y = X ↑ 2
120  IF XF * X + 140 < = 0 GOTO 80
130  HPLOT XF * X + 140, 159 – 2 * Y
140  FOR X = – 7 TO 8
150  Y = X ↑ 2
160  HPLOT TO XF * X + 140, 159 – 2 * Y
170  NEXT X
180  GOTO 50
```

Problema 3.2

Modifique o programa acima para que a função Y = X ↑ 3 possa ser plotada. Note que o eixo horizontal precisa ser erguido para que valores negativos de Y possam ser plotados, enquanto que o mesmo tipo de atenção deve ser dada aos limites das coordenadas, tanto para valores de X como de Y.

A aparência da curva do Problema 3.2 pode ser melhorada consideravelmente plotando-se somente os pontos calculados, ao invés de unir estes com segmentos de reta.

No Cap. 9 continuaremos com o estudo de técnicas gráficas avançadas tais como a criação de uso de uma tabela de formas, desenhos que transladam e giram, bem como guardar tais desenhos em disco. Contudo, antes de continuarmos com este assunto, faz-se necessária a familiarização com arquivos binários, disponibilidade de espaço na memória para armazenamento de tais arquivos e técnicas de manuseio de arquivos, que mostraremos nos próximos capítulos.

Capítulo 4
Strings e arrays*

Variáveis String

Uma seqüência de caracteres é chamada literal, e um literal entre aspas é chamado string. Por exemplo,

ABC12 é um literal, e
"ABC12" é um string.

Assim como os números, os strings podem ser designados por nomes de variáveis, que são distinguidos das variáveis numéricas pelo $ após o nome. Por exemplo, A$. O programa abaixo atribui um string à variável de nome B$ e em seguida imprime B$.

```
10 B$ = "ABC 123"
20 PRINT B$
30 END
```

O programa seguinte irá ler um string de uma instrução DATA e o atribuirá a uma variável.

```
10 READ AA$
20 PRINT AA$
30 DATA "A1B2"
40 END
```

Vários strings podem ser lidos e atribuídos a várias variáveis, desde que os strings contidos em uma instrução DATA estejam separados por vírgulas. Em geral, uma variável string pode ser usada em um programa BASIC nas seguintes instruções:

(a) A$ = "ABC" ou A$ = B$
(b) READ A$. O string deve estar na instrução DATA.
(c) INPUT A$
(d) PRINT A$
(e) IF A$ = "YES" THEN . . . ou IF A$ < B$ GOTO . . .
(f) C$ = A$ + B$. Isto é conhecido como concatenação.

O BASIC foi projetado para atribuir um código numérico a cada caractere do teclado, de acordo com o código ASCII, mostrado na Tab. 4.1. Deste modo, a cada letra do alfabeto é atribuído um valor numérico. Em conseqüência, a letra A possui um valor menor que B, a letra B um valor menor que C, e assim por diante.

Quando aparecem strings em instruções IF (como na letra (e) acima), são comparados caractere por caractere, da esquerda para direita, baseado na seqüência comparativa do ASCII, até que uma diferença seja encontrada. Se um caractere numa posição do string A$ possui um código ASCII de maior valor do que o caractere encontrado na mesma posição do string B$, então A$ é maior que B$. Se todos os caracteres na mesma posição

* Vide nota do tradutor.

forem idênticos, mas um string possuir mais caracteres que o outro, o mais longo é o maior dos dois. Portanto, se strings alfabéticos forem ordenados segundo o resultado dos testes relacionais, podem ser facilmente dispostos em ordem alfabética.

TABELA 4.1 Códigos de Conversão de Caracteres ASCII

carac.	*decimal*	*carac.*	*decimal*	*carac.*	*decimal*
espaço	32	4	52	H	72
!	33	5	53	I	73
"	34	6	54	J	74
#	35	7	55	K	75
$	36	8	56	L	76
%	37	9	57	M	77
&	38	:	58	N	78
'	39	;	59	O	79
(	40	<	60	P	80
)	41	=	61	Q	81
*	42	>	62	R	82
+	43	?	63	S	83
,	44	@	64	T	84
–	45	A	65	U	85
.	46	B	66	V	86
/	47	C	67	W	87
0	48	D	68	X	88
1	49	E	69	Y	89
2	50	F	70	Z	90
3	51	G	71	] *sft-M* ↑	93 94

Nas instruções consideradas até agora, as variáveis string não são subscritas. Isso faz com que a referência só possa ser feita ao comprimento total do string (comprimento máximo limitado em 256 caracteres). Mais adiante, contudo, veremos que com o auxílio de três instruções string especiais podemos acessar qualquer caractere de um dado string.

Arrays de strings

Vários strings podem ser armazenados sob um único nome em algo conhecido como array de strings. Vamos supor que tenhamos 4 nomes, a saber SMITH, JONES, BROWN e WILSON, que desejamos armazenar num array de strings. Em BASIC, toda vez que um array for usado num programa, isto deve ser declarado numa instrução de dimensionamento, como mostrado no programa a seguir, que permite leitura e armazenamento de nomes em uma única variável X$( ).

```
10 REM USO DE ARRAY DE STRINGS
20 DIM X$ (4)
30 FOR I = 1 TO 4
```

```
40 READ X$ (I)
50 NEXT I
60 DATA "SMITH", "JONES", "BROWN", "WILSON"
70 END
```

Uma maneira simples de se visualizar um array de strings é a seguinte:

SMITH	JONES	BROWN	WILSON

Os quatro nomes são armazenados em uma única caixa, que possui quatro compartimentos, cada qual contendo um nome. Portanto, X$(2) refere-se ao 2º compartimento do string X$( ), e X$(4) refere-se ao 4º compartimento. A instrução DIM diz ao BASIC que o string de arrays chamado X$( ) deve ser usado com o número máximo de compartimentos determinado pelo número entre parênteses após o nome do array (neste caso 4). Finalmente, qualquer referência a um nome de um array em um programa deve ser da forma

X$(I)

onde I tem um valor entre 0 e o número máximo dado na instrução DIM. Note que a instrução DIM X$(4) possui, na realidade, 5 compartimentos, começando com o compartimento X$(0). Referências somente a X$ não se referem ao array, mas à variável string X$ não subscrita, que, por acaso, esteja usando a mesma letra.

O programa seguinte irá ler de uma instrução DATA o nome, a sala e o ramal telefônico de cinco funcionários. Note que os dados foram estruturados de forma que as vírgulas que separam os nomes das salas e estas dos ramais tenham a mesma posição em cada string. Isto é obtido adicionando espaços em branco para compensar os diferentes comprimentos de nome etc. Fazemos isto propositadamente nesta fase, para possibilitar a manipulação destes strings posteriormente. No exemplo, dois loops FOR-NEXT foram usados meramente para demonstrar que, uma vez lidos os dados, estes são armazenados na memória (a não ser que sejam escritos por cima), do contrário um loop teria sido suficiente.

```
10  REM INFORMACOES SOBRE FUNCIONARIOS
20  DIM Y$(5)
30  FOR I = 1 TO 5
40  READ Y$(I)
50  NEXT I
60  FOR I = 1 TO 5
70  PRINT Y$(I)
80  NEXT I
310 DATA "WILSON M., SALA 1.24, 395"
320 DATA "SMITH M., SALA 2.6, 7315"
330 DATA "JONES B.M., SALA 6.19, 1698"
340 DATA "SMITH A.A., SALA 2.12, 456"
350 DATA "BROWN C., SALA 3.1, 432"
400 END
```

Digite este programa, salve-o e rode-o. O BASIC irá escrever os laterais na tela à medida que vão aparecendo nas instruções DATA, só que sem as aspas.

Funções string

Vamos apresentar agora certas funções que permitem a manipulação de strings. Por exemplo, suponhamos que queiramos extrair e imprimir somente os nomes dos funcionários existentes no array Y$( ). O Basic nos permite fazer isto facilmente com a função

```
LEFT$( )
```

As poucas linhas de programa dadas abaixo, quando adicionadas ao programa anterior, irão fazer isto.

```
140 PRINT
150 FOR I = 1 TO 5
160 PRINT LEFT$(Y$(I), 10)
170 NEXT I
```

Observe que a função LEFT$ possui dois argumentos entre parênteses; o primeiro é o i-ésimo string do array de string Y$( ) e o segundo é o numeral 10, que se refere ao número de caracteres desejados. Digite estas linhas adicionais e tente o programa.

Outra função string permite a manipulação dos caracteres do extremo direito de um string. Isto é obtido com o uso da função

```
RIGHT$( )
```

Para ilustrar isto, troque a linha 160 do programa por

```
160 PRINT LEFT$(Y$(I),10),RIGHT$(Y$(I),4)
```

e rode-o. Você verá que a segunda coluna da última instrução PRINT contém os quatro caracteres da extremidade direita de cada string.

Uma terceira função que permite a extração de informações do meio de um string é

```
MID$( )
```

Substitua a linha 160 do programa pela linha abaixo, que imprimirá a sala de cada funcionário.

```
160 PRINT MID$(Y$(I),12,9)
```

Note que esta função requer dois valores numéricos após o i-ésimo string. O primeiro é o ponto de início dentro do string (12º caractere neste caso) e o segundo é o número de caracteres a ser considerado (neste caso 9). Contudo, se o segundo número for omitido da lista de argumentos, então os caracteres considerados começarão no primeiro número e terminarão no final do string.

Como um exemplo do uso de array de string, considere o programa adiante, que faz com que informações sobre a quantidade e preços de vários itens em estoque sejam armazenados pelo BASIC. Para extrair informações a respeito dos itens estocados, simplesmente comece o programa e responda as perguntas apresentadas.

Inicialmente, o programa lê e armazena em um array de string X$( ) os nomes dos itens em estoque, enquanto que ao mesmo tempo as suas quantidades e preços são colocados em Y$( ). Em resposta à pergunta 'QUE ITEM', o nome do item, associado à variável string B$, é digitado. Se for o 'FIM', o programa parará. Se, no entanto, tiver qualquer outro nome, ele iniciará um loop (linha 90 a 130) que compara, então, o conteúdo de X$( ) com B$( ). Se estes forem iguais, ele imprime a informação desejada que está contida em Y$( ), se não, ele executa a linha 140.

```
10 REM CONTROLE DE ESTOQUE
```

```
 20 DIM X$(4),Y$(4)
 30 FOR I = 1 TO 4
 40 READ X$(I),Y$(I)
 50 NEXT I
 60 PRINT
 70 INPUT "WHICH ITEM? "; B$
 80 IF B$ = "END" THEN END
 90 FOR I = 1 TO 4
100 IF X$(I) <> B$ GOTO 130
110 PRINT ">>>>>>> "; LEFT$(Y$(I),3);" ESTOCADO EM # ";
    RIGHT$(Y$(I),3);" CADA"
120 GOTO 60
130 NEXT I
140 PRINT "ITEM NAO ESTOCADO"
150 GOTO 60
160 DATA "BORRACHA P/TINTA","200,.10"
170 DATA "BORRACHA P/LAPIS","320,.15"
180 DATA "BORRACHA DE DATILOGRAFIA","25,.25"
190 DATA "LIQUIDO DE CORRECAO","150,.50"
```

Digite cuidadosamente este programa, prestando atenção particularmente aos espaços inseridos nas instruções DATA e PRINT. Rode o programa e salve-o sob o nome de arquivo ESTOQUE.

Problema 4.1

(a) Altere o programa acima para que você só precise digitar as três primeiras letras de cada item toda vez que a pergunta "QUE ITEM" for feita. A saída de seu programa deve, entretanto, imprimir o nome completo de cada item.

(b) Feita a modificação do item (a), reestruture os dados de forma que cada linha de dados seja armazenada em cada elemento de um array de strings. Use as funções LEFT$( ), RIGHT$( ) e MID$( ) para extrair as informações apropriadas para a impressão de saída.

A necessidade de estruturação dos dados pode dar margem a erro ao digitar as informações em uma linha DATA, especialmente no caso de dados numéricos. De fato, dados numéricos podem ser armazenados em um array numérico sem a necessidade de estruturá-los na instrução DATA. Isto permite uma maior flexibilidade de programação, o que será estudado na seção que se segue.

Variáveis numéricas subscritas

As variáveis subscritas permitem a apresentação de várias quantidades com uma única variável. Uma quantidade específica é indicada escrevendo-se o subscrito entre parênteses após o nome da variável. Quantias individuais são chamadas elementos, enquanto que um conjunto de elementos é chamado array. Uma variável subscrita pode possuir um, dois ou três subscritos, representando arrays de uma, duas ou três dimensões. O número máximo de subscritos permitidos é 88.

Os elementos de um array unidimensional podem ser representados da seguinte forma:

A(0) A(1) A(2) A(3) A(4)

enquanto que os de um array bidimensional como:

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)

O primeiro dos dois subscritos refere-se ao número da linha, indo de 0 ao número máximo de linha declarado, e o segundo subscrito ao número de coluna, indo de 0 ao número máximo de colunas declarado.

Um array tridimensional pode ser pensado como arrays bidimensionais empilhados com um terceiro subscrito, indo de 0 até a altura máxima da pilha.

Entretanto, no computador, os arrays são armazenados com seus elementos em uma única linha, como mostrado abaixo.

A(0,0) A(1,0) A(2,0) A(0,1) A(1,1) A(2,1)

com o primeiro subscrito mostrado mais rapidamente que o segundo, e o segundo mais rapidamente que o terceiro (no caso de array tridimensional). Desde que isto esteja reconhecido e compreendido, podemos usar a forma ilustrada de representação anterior para fins de programação.

Arrays numéricos devem ser declarados previamente em uma instrução DIM assim como declaramos o array de string. O formato da instrução é mostrado abaixo.

```
DIM X(15),  Y(3,5),  Z(3,5,4)
```

onde o array X( ) foi declarado como sendo unidimensional, com no máximo 16 elementos (não se esqueça do elemento 0), o array Y(,) foi declarado bidimensional contendo 4 linhas e 6 colunas, e o array Z(,,) como um array tridimensional de 4 linhas, 6 colunas e 5 de altura.

O programa seguinte ilustra o uso dos arrays numéricos. Os dados são lidos e colocados em um array unidimensional e, logo após, os conteúdos dos elementos de número par são somados na variável A, enquanto que o conteúdo de todos os elementos de número ímpar são somados na variável B.

```
 10  REM ARRAY NUMERICO
 20  DIM X(15)
 30  REM LER E ARMAZENAR EM X( ) 16 NUMEROS
 40  FOR I = 0 TO 15
 50  READ X(I)
 60  NEXT I
 70  REM SOMAR ELEMENTOS PARES
 80  A = 0
 90  FOR I = 0 TO 14 STEP 2
100  A = A + X(I)
110  NEXT I
120  REM SOMAR ELEMENTOS IMPARES
130  B = 0
140  FOR I = 1 TO 15 STEP 2
```

```
150  B = B + X(I)
160  NEXT I
170  REM IMPRIMIR CONTEUDO DO ARRAY
180  FOR I = 0 TO 15
190  PRINT X(I)
200  NEXT I
210  PRINT A, B
220  DATA 4, 7, 6, 1, 9, 7, 14, 39, 24, 19, 32, 21, 8, 5, 15, 28
230  END
```

Existem três mensagens de erro relacionadas com o uso de array. São elas:

```
REDIMENSIONAR ARRAY
BAD SUBSCRIPT ERROR
ILLEGAL QUANTITY ERROR
```

A primeira é emitida se o array for dimensionado mais de uma vez; a segunda, se houver uma tentativa de se usar um elemento fora das dimensões declaradas do array, ou o produto dos subscritos exceder a memória disponível; e a terceira ao tentar usar um subscrito negativo.

Instruções múltiplas em uma linha

Vamos agora modificar o programa original de controle de estoque para que as partes numéricas dos dados sejam armazenadas em um array bidimensional. Após estudá-la, faça modificações semelhantes na sua versão do programa que resultou do item (a) do Problema 4.1.

Como o novo programa não difere muito do original, vamos apresentar uma nova técnica, para economia de espaço, de se escrever instruções múltiplas em uma única linha. Isto é feito simplesmente separando-se as instruções que aparecem na mesma linha com o sinal gráfico dois pontos (:). Para ilustrar isto, escreveremos o loop FOR-NEXT que contém a instrução READ (originalmente de três linhas) em uma única linha, como mostrada abaixo.

```
30  FOR I = 1 TO 4:READ X$(I),Y$(I):NEXT I
```

Contudo é bom lembrar que, quando se escrever programas usando este método, instruções que geram ramificações dentro do programa só podem ser direcionadas para o início de uma linha.

```
10   REM CONTROLE DE ESTOQUE USANDO STRINGS E ARRAYS
20   DIM X$(4),Y(4,2)
30   FOR I = 1 TO 4:READ X$(I),Y(I,1),Y(I,2):NEXT I
40   PRINT:INPUT "QUE ITEM? "; B$
50   IF B$ = "END" THEN END
60   FOR I = 1 TO 4:IF X$(I) <> B$ GOTO 80
70   PRINT " >>>>>> "; Y(I,1);" ESTOCADO EM # " ; Y(1,2);
     "CADA": GOTO 40
80   NEXT I
90   PRINT "ITEM NAO ESTOCADO": GOTO 40
100  DATA "BORRACHA P/TINTA",200,.1
110  DATA "BORRACHA P/LAPIS",320,.15
```

```
120 DATA "BORRACHA P/MAQUINA",25,.25
130 DATA "LIQUIDO DE CORRECAO",150,.5
```

Classificação alfabética

Muitas aplicações de programação, tais como manipulação de informações dos registros de funcionários, exigem uma classificação alfabética. Para conseguirmos isto, devemos nos basear na técnica desenvolvida anteriormente para classificar números (ver as Figs. 2.2 e 2.3, assim como o programa CLASSIF TRES NUM.)

A técnica que adotaremos é mais ou menos a mesma que a usada anteriormente, exceto que, em vez de variáveis individuais usamos arrays de strings. Isto reduz o número de instruções IF para somente uma. A técnica é ilustrada abaixo, aplicando-se no programa FUNCIONARIOS que deve estar em seu disquete de trabalho. O programa é ligeiramente reescrito para acomodar instruções múltiplas (linhas 20, 30 e 40) em uma linha. Adições ao programa foram incorporadas entre as linhas 50 e 310. Carregue (LOAD) o programa FUNCIONARIOS e faça estas alterações. Agora o seu programa deve estar como o seguinte:

```
 10  REM CLASSIFICACAO ALFABETICA
 20  READ N:DIM Y$(N)
 30  FOR I = 1 TO N:READ Y$(I):NEXT I
 40  FOR I = 1 TO N:PRINT Y$(I):NEXT I
 50  PRINT:PRINT:PRINT "INFORMACAO CLASSIFICADA"
110  FOR I = 1 TO N - 1
120  IF Y$(I) <= Y$(I + 1) GOTO 160
130  T$ = Y$(I + 1)
140  Y$(I + 1) = Y$(I)
150  Y$(I) = T$
160  NEXT I
180  FOR I = 1 TO N:PRINT Y$(I):NEXT I
300  DATA 5
310  DATA "WILSON M., SALA 1.24, 395"
320  DATA "SMITH M., SALA 2.6, 7315"
330  DATA "JONES B.M., SALA 6.19, 1698"
340  DATA "SMITH A.A., SALA 2.12, 456"
350  DATA "BROWN C., SALA 3.1, 432"
400  END
```

Certifique-se de que os números das linhas de seu programa correspondem ao mostrado acima, visto que linhas adicionais serão incorporadas brevemente.

Ao rodar este programa, você verá que as primeiras cinco linhas imprimem os funcionários na mesma ordem em que aparecem na instrução DATA. As próximas cinco linhas são o resultado da execução do loop FOR-NEXT das instruções 110 a 160. Neste loop, quando I = 1, o primeiro string é comparado com o segundo, e se este for menor, o controle é deslocado para linha 160, pois do contrário os dois strings seriam permutados (linhas 130 e 150). Quando I = 2, o segundo string é comparado com o terceiro, e assim por diante, até que I = N − 1, onde o (N − 1)-ésimo string é comparado ao N-ésimo. O resultado, no seu caso, foi que BROWN deslocou para uma posição acima e, a fim de colo-

cá-lo no topo da lista, precisamos repetir o loop FOR-NEXT das linhas 110 a 160 N – 1 vezes. Façamos isto adicionando um loop FOR-NEXT extra, como se segue:

```
100  FOR J=1 TO N-1
```

Digite estas duas linhas em seu computador e rode o programa. Você verá que, apesar de toda a informação aparecer na tela, é um tanto difícil distinguir o resultado de cada execução do loop FOR-NEXT externo. A adição das duas linhas abaixo deve consertar isto.

```
170  PRINT:PRINT:PRINT J
190  GET A$
```

A instrução GET obriga o computador a chamar (pedir) informações do teclado e, neste caso, as atribui a A$. O computador esperará até que você aperte uma tecla, e então continuará com a execução do programa. Isto significa que o resultado de cada iteração do loop J pode ser estudada cuidadosamente antes de dar procedimento à execução do programa (apertando qualquer tecla, exceto RESET). Adicione estas linhas e torne a rodar o programa.

Classificação pela técnica de bolha

Ao rodar o programa acima você irá notar duas coisas:
(*a*) Após a 1ª execução do loop J, WILSON cai para o fim da lista, e após cada iteração subseqüente o próximo nome de maior valor aparecerá acima de WILSON.
(*b*) Após cada iteração do loop J, BROWN desloca-se uma posição acima na lista de nomes.

Isto significa que existe espaço para aprimorar o programa de duas maneiras. Visto que o nome de maior valor é deslocado para o fim da lista, podemos reduzir o limite superior do loop I de um a cada execução do loop J. Além disso, embora na pior das hipóteses todas as N–1 iterações sejam necessárias, a lista muitas vezes estará classificada entre 0 e N–1. Isto pode ser superado incorporando-se uma "chave" (flag) no programa cujo valor é normalmente fixado em 0, mas todas as vezes que uma permutação for feita ele é alterado para 1. Testando-se o valor da "chave" ao final de cada iteração, podemos determinar se precisaremos ou não executar o loop J mais uma vez.

A inclusão das linhas 90, 105, 125 e 195 bem como a troca da variável que representa o limite superior da variável de controle I na linha 110 cobrem ambas as sugestões de aprimoramento da eficiência do programa. O programa resultante está listado a seguir e o fluxograma é mostrado na Fig. 4.1.

```
 10  REM CLASSIFICACAO DA BOLHA
 20  READ N: DIM Y$(N)
 30  FOR I=1 TO N: READ Y$(I): NEXT I
 40  FOR I=1 TO N: PRINT Y$(I): NEXT I
 50  PRINT: PRINT: PRINT "INFORMACAO CLASSIFICADA"
 90  M=N
100  FOR J=1 TO N-1
105  M=M-1: K=0
110  FOR I=1 TO M
120  IF Y$(I)<=Y$(I+1) GOTO 160
125  K=1
130  T$=Y$(I+1)
```

```
140  Y$(I + 1) = Y$(I)
150  Y$(I) = T$
160  NEXT I
170  PRINT: PRINT: PRINT J
180  FOR I = 1 TO N: PRINT Y$(I): NEXT I
190  GET A$
195  IF K = 0 GOTO 400
200  NEXT J
300  DATA 5
310  DATA "WILSON M., SALA 1.24, 395"
320  DATA "SMITH M., SALA 2.6, 7315"
330  DATA "JONES B.M., SALA 6.19, 1698"
340  DATA "SMITH A.A., SALA 2.12, 456"
350  DATA "BROWN C., SALA 3.1, 432"
400  END
```

A variável M é usada na linha 90 para armazenar temporariamente o número total de strings a serem manipulados em cada execução do loop J. O seu valor é inicialmente igualado a N e subseqüentemente é reduzido de um na linha 105, formando, então, o valor limite superior da variável de controle I na linha 110 a ser menor, o que também reduz ao mínimo o número total de comparações de string. A constante K nas linhas 105 e 125 é usada como "chave". O seu valor é estabelecido em 1 para indicar que uma permutação foi feita. Se K mantiver o valor 0 (zero), durante todo o loop I, significará que os strings já estão na ordem desejada.

Problema 4.2

Modifique o programa acima para que a parte numérica dos dados (omitindo a palavra SALA) sejam lidos e armazenados em um array numérico bidimensional. Em seguida, altere o programa de tal forma que as informações contidas nas instruções DATA sejam classificadas em ordem ascendente do número da sala. O trecho de classificação do programa deve ser alterado de forma a manipular informações ao array numérico.

Concatenação de strings

O BASIC permite a concatenação (união) de strings. Ilustraremos este artifício considerando o programa anterior em que a palavra SALA aparece em todas as linhas de dados. Poderíamos alocar esta palavra a uma variável string R$, por exemplo, na linha 25, e omiti-la das instruções DATA, o que irá encurtar consideravelmente o programa, especialmente se tivéssemos de cuidar de um número muito grande de funcionários. Vamos assumir, então, que uma linha de dados fosse tipicamente assim:

```
DATA "NAME,6.20,1234"
```

A fim de se obter a mesma impressão de saída que antes, devemos alterar a linha 180 para o seguinte:

```
180  FOR I = 1 TO N: PRINT
       LEFT$(Y$(I),11) + R$ + RIGHT$(Y$(I),9): NEXT I
```

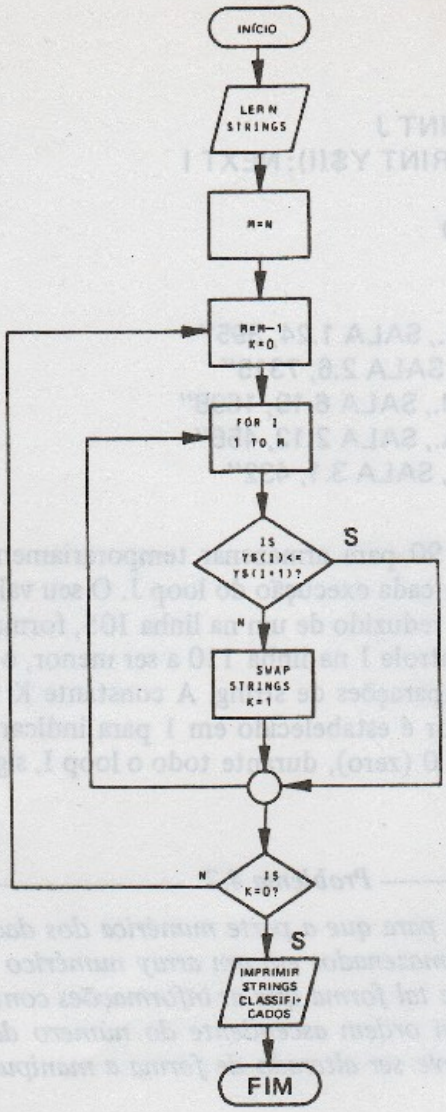


Fig. 4.1. Fluxograma do programa de classificação pelo método da bolha

Note que a primeira parte da instrução PRINT imprime os 11 primeiros caracteres do string; a segunda parte imprime o string R$, enquanto que a última parte imprime os 9 caracteres mais à direita do string. Espaços extras entre seções de impressão podem ser inseridos concatenando-se o string espaço " " a qualquer parte da instrução PRINT.

Mais algumas funções string

Nesta seção, apresentaremos as seguintes funções-string adicionais, que facilitam a programação

ASC( ), CHR$( ), LEN( ), STR$( ) e VAL( )

Exemplos do seu uso são dados a seguir.

Conversão ASCII

```
N = ASC("ABCD")
```

irá retornar * o código decimal ASCII do primeiro caractere do string contido entre os parênteses da função. Neste caso, o código retornado será G5 (veja Tabela 4.1).

Conversão de caracteres

```
C$ = CHR$(66)
```

irá retornar o caractere ASCII que corresponde ao valor do argumento, neste caso a letra B. O valor do argumento deve estar entre 0 e 255. Os caracteres cujos valores vão de 32 a 94 são dados na Tabela 4.1. A Tabela E-1, no Anexo E, dá os códigos ASCII para todos os caracteres, incluindo aqueles que podem ser gerados com a tecla CTRL.

Comprimento de string

```
L = LEN("XYZ")
```

irá retornar o valor do comprimento do string, i.e., o número de caracteres contidos no string, que neste caso é 3.

Conversão de string

```
S$ = STR$(X)
```

onde X é uma variável numérica que pode ser o resultado de cálculos. Ao usar a função STR$( ) o valor do argumento será convertido em um string. Neste caso, se X tivesse o valor 98.56, por exemplo, então S$ passa a ser igual a "98.56".

Valor de string

Se R$ representa o string dado abaixo

```
R$ = "3.123E12 METRES"
```

então a instrução

```
X = VAL(R$)
```

irá retornar o valor do string até o primeiro caractere não-numérico, neste caso 3.123E + 12. Se o string começar com um caractere não-numérico, o valor 0 é retornado.

ATENÇÃO: A função VAL e valores negativos

Se você vier a usar a função VAL para avaliar um número negativo, a função retornará um zero, pois o sinal negativo é tratado como um caractere não-numérico. Isto causará erros, caso tente usá-la. O programa seguinte deve contornar esta dificuldade. Assumimos que cada string é lido na variável-string A$ e que ele pode ser tanto positivo como negativo.

```
50  IF LEFT$(A$,1) = "-" THEN L = LEN(A$)-1: B$ = MID$(A$,2,L):
      X = (-1) * VAL(B$): GOTO 70
60  X = VAL(A$)
70  PRINT X
80  END
```

* Vide nota do tradutor.

Na linha 50, o caractere mais à esquerda do string A$ é testado para saber se ele é o sinal negativo. Se for, o comprimento do string é avaliado, e o seu valor é atribuído à variável 1. A parte numérica do string (L caracteres, a partir do 2º) é atribuída à variável string B$, seu valor avaliado e multiplicado por (– 1).

Problema 4.3

Escreva um programa que use uma ou mais funções-string que permitam:

(a) a impressão de uma letra especificada pela entrada de um número entre 1 e 26;

(b) a impressão de um número correspondente à posição de uma dada letra no alfabeto, entrando-se qualquer letra.

Capítulo 5
Subprogramas

Funções aritméticas standard

O BASIC possui funções para executar operações matemáticas. Elas dispensam que o usuário tenha de programar as suas próprias rotinas para calcular funções tão comuns, tais como logaritmos, raízes, senos de ângulos etc. As funções matemáticas do BASIC possuem um nome de chamada de três letras, seguidas por um argumento entre parênteses. Elas já estão predefinidas e podem ser usadas em qualquer ponto do programa. Algumas das funções standard do BASIC mais comuns estão listadas a seguir.

TABELA 5.1 Funções standard do BASIC

NOME DE CHAMADA	FUNÇÃO
SIN(X)	Seno do ângulo X, onde X é dado em radianos
COS(X)	Cosseno do ângulo X, onde X é dado em radianos
TAN(X)	Tangente do ângulo X, onde X é dado em radianos
ATN(X)	Arco-tangente de X, onde o ângulo 'retornado' é dado em radianos na faixa de + 1.570796 a – 1.570796
EXP(X)	Eleva *e* à potência X
LOG(X)	'Retorna' o logaritmo natural de X, 1 n(X)
SQR(X)	'Retorna' a raiz quadrada de X
ABS(X)	'Retorna' o valor absoluto de X
SGN(X)	'Retorna' 1, 0, ou – 1 para indicar o sinal de X
INT(X)	'Retorna' a parte inteira truncada de X
RND(X)	Gera números randômicos entre 0 e + 1

Chamadas de função podem ser usadas como uma expressão ou como elemento de expressões, toda vez que estas forem legais. O argumento X da função pode ser uma constante, uma variável, uma expressão ou outra função. Uma explicação mais detalhada do uso destas funções é dada abaixo.

SIN(X), COS(X) e TAN(X)

As funções seno, cosseno e tangente requerem um ângulo expresso em radianos como argumento. Se este estiver em graus, a conversão para radianos é obtida com a identidade:

Radianos = Graus * 3.1415926/180

ATN(X)

A função arco-tangente retorna um valor em radianos, entre + 1.570796 e – 1.570796 correspondente ao valor da tangente fornecida como argumento X.

EXP(X)

A função exponencial eleva o número e a potência X. A função EXP( ) é a inversa da função LOG( ). A relação é:

```
LOG(EXP(X)) = X
```

LOG(X)

A função LOG( ) deriva o logaritmo na base *e* de um dado valor. Logaritmos da base *e* podem ser facilmente convertidos para qualquer outra base usando-se a seguinte identidade:

$\log_a(N) = LOG(N) / LOG(a)$

onde $\log_a(N)$ é o logaritmo desejado na base a, enquanto LOG(N) e LOG(a) são os logaritmos na base *e* de N e a respectivamente.

SQR(X)

A função SQR( ) retorna a raiz quadrada do número que lhe foi fornecido.

ABS(X)

A função ABS( ) retorna o valor absoluto (i.e., positivo) de um dado número. Por exemplo, ABS(1.234) é 1.234, enquanto que ABS(-2.345) é retornado com 2.345.

SGN(X)

A função sinal retorna 1 se X for positivo, 0 se X = 0, e – 1 se X for negativo.

INT(X)

A função inteiro retorna o valor de X arredondado para baixo do valor inteiro mais próximo (i.e., INT(6.97) e retorna ao valor de 6). O inteiro de um número negativo é um número negativo de mesmo valor absoluto (i.e., INT(–5.34) retorna o valor de –6).

A função INT( ) pode ser usada para aproximar números ao inteiro mais próximo, seja ele para cima ou para baixo, usando a identidade:

```
INT(X + 0.5)
```

ou para (*a*) qualquer casa decimal ou (*b*) potência de 10, através da seguinte expressão:

```
INT(X * 10 ↑ D + 0.5)/10 ↑ D
```

onde D é (*a*) um inteiro positivo ou (*b*) um inteiro negativo fornecido pelo usuário. (Para arredondar a primeira casa decimal, D = 1; para a centena mais próxima, D = – 2)

RND(X)

A função RND( ) produz um número randômico entre 0 e 1. Se X for repetidamente um

número positivo, a seqüência randômica produzida será diferente a cada rodada do programa. Se, no entanto, X possuir inicialmente um valor negativo, mas em seguida possuir um positivo, a seqüência randômica original será reproduzida a cada execução do programa.

Funções definidas pelo usuário

Em alguns programas pode ser necessário usar a mesma expressão matemática em vários pontos, geralmente com dados diferentes. As funções definidas pelo usuário do BASIC permitem a definição de expressões ou operações exclusivas. Desta forma estas podem ser chamadas ao programa na maneira das funções standard.

A função definida pelo usuário é identificada por três letras de chamada seguidas por um argumento entre parênteses. As duas primeiras letras do nome da função têm de ser FN e a terceira pode ser qualquer letra do alfabeto; logo, são permitidas até 26 funções exclusivas definidas pelo usuário. Contudo, tal função precisa ser definida antes de ser usada. Ela é definida usando-se a instrução DEF, e geralmente é colocada no início do programa. O programa seguinte ilustra o uso de uma função definida pelo usuário.

```
10 REM FUNCAO DEFINIDA PELO USUARIO
20 DEF FNX(R) = 3.1415926 * R ↑ 2
30 FOR I = 1 TO 10
40 A = FNX(I)
50 PRINT I,A
60 NEXT I
70 END
```

O programa calcula a área de círculos de raio iguais a inteiros de 1 a 10. A fórmula é dada pela instrução DEF FNX( ) na linha 20. O valor do raio é passado à função através de uma variável entre parênteses, conhecida como parâmetro da função. Note que o nome da variável que representa este parâmetro não precisa ser o mesmo que é usado no corpo do programa. Quando chamado, o argumento entre parênteses pode ser qualquer expressão legal, o seu valor é simplesmente substituído pela variável da função. O nome da função pode ser usado repetitivamente, mas só pode ser definido uma única vez. Se uma função for definida mais de uma vez, a primeira definição será usada e as subseqüentes ignoradas. A definição de uma função é limitada a uma linha; fórmulas mais longas podem ser definidas como sub-rotinas BASIC, como veremos brevemente.

Problema 5.1

Modifique o programa acima para que ele possua uma segunda função definida pelo usuário que arredonde o valor da área calculada para duas casas decimais. Use a fórmula dada na função INT( ) com o valor D = 2.

Funções matemáticas derivadas

Algumas funções matemáticas muito úteis, que podem ser derivadas de funções BASIC standard, estão listadas na Tab. 5.2.

Sub-rotinas

Geralmente é desejável escrever um grupo de instruções (uma sub-rotina) uma única vez em um programa e depois desviar-se para ela de vários pontos do programa.

TABELA 5.2 Funções Matemáticas Derivadas

Função	*Fórmula*
Trigonométricas	
Cosec	`CSC(X)=1/SEN(X)`
Cotan	`COT(X)=1/TAN(X)`
Sec	`SEC(X)=1/COS(X)`
Trigonométricas inversas	
Arc.Cos	`ACOS(X)=-ATN(X/SQR(-X*X+1))+PI/2`
Arc.Sen	`ASEN(X)=ATN(X/SQR(-X*X+1))`
Arc.Cosec	`ACSC(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*PI/2`
Arc.Cotan	`ACOT(X)=-ATN(X)+PI/2`
Arc.Sec	`ASEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*PI/2`
Hiperbólicas	
Cos Hip.	`COSH(X)=(EXP(X)+EXP(-X))/2`
Sen Hip.	`SENH(X)=(EXP(X)-EXP(-X))/2`
Tan Hip.	`TANH(X)=-EXP(-X)/(EXP(X)+EXP(-X))*2+1`
Cosec Hip.	`CSCH(X)=2/(EXP(X)-EXP(-X))`
Cotan Hip.	`COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1`
Sec Hip.	`SECH(X)=2/(EXP(X)+EXP(-X))`
Hiperbólicas inversas	
Arc.Cos	`ACOSH(X)=LOG(X+SQR(X*X-1))`
Arc.Senh	`ASENH(X)=LOG(X+SQR(X*X+1))`
Arc.Tanh	`ATANH(X)=LOG((1+X)/(1-X))/2`
Arc.Cosech	`ACSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)`
Arc.Cotanh	`ACOTH(X)=LOG((X+1)/(X-1))/2`
Arc.Sech	`ASECH(X)=LOG((SQR(-X*X+1)+1)/X)`

Nota: PI = 3,141592654 em todas as fórmulas acima.

As instruções GOSUB e RETURN

Na Fig. 5.1 um exemplo do fluxo de um programa que incorpora uma sub-rotina é mostrado em diagrama. Quando o BASIC encontra a instrução GOSUB no corpo principal do programa, ele desvia para a primeira instrução da sub-rotina, continua a executar as instruções da sub-rotina até que seja encontrada a instrução RETURN, que desvia o fluxo do programa para a instrução seguinte, a instrução GOSUB.

Quando há várias instruções GOSUB desviando para a mesma sub-rotina, cada vez que o comando RETURN for alcançado, o programa principal é retomado na mais recente instrução GOSUB de onde ele foi desviado. Nenhuma linha de referência é necessária.

O programa seguinte, que calcula os juros compostos de empréstimos em dinheiro, será usado para ilustrar o uso de sub-rotinas. O programa calcula os juros usando a fórmula

$$A = P * (1 + R/100) \uparrow N$$

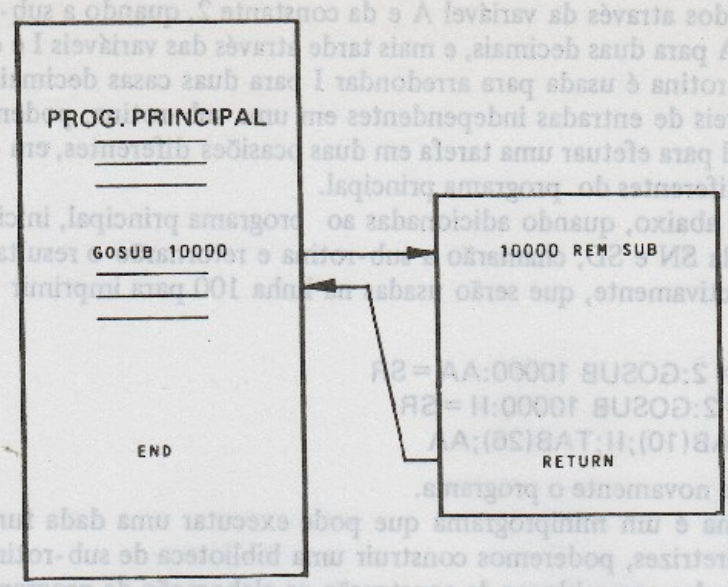


Fig. 5.1 Representação em diagrama de programa com sub-rotina.

onde P é a quantia emprestada e A é o valor total em N anos a R por cento ao ano.

```
10   REM JUROS
20   INPUT "QUANTIA EMPRESTADA ";P
30   INPUT "CORRECAO ";R
40   INPUT "NUMERO DE ANOS ";X
50   PRINT:PRINT
60   PRINT "ANOS";TAB(10);"JUROS";
     TAB(26);"QUANTIA":PRINT
70   FOR N = 1 TO X
80   A = P*(1 + R/100) ↑ N
90   I = A - P
100  PRINT N;TAB(10);I;TAB(26);A
110  NEXT N
120  END
```

Digite este programa e rode-o usando os valores 5000, 11 e 15 para P, R e X, respectivamente. Você verá imediatamente que existe uma necessidade de se arredondar para a segunda casa decimal tanto o resultado de I como o de A, pois, ao lidar com dinheiro, quaisquer dígitos extra são geralmente desprezados.

Parâmetros de sub-rotina

A sub-rotina seguinte irá arredondar números para duas casas decimais.

```
10000  REM SUB TO CONVERT NUMBERS TO ###.## FORMAT
10010  SR = INT (10 ↑ SD*SN + 0.5)/10 ↑ SD
10020  RETURN
```

Os valores de SN e SD devem ser fornecidos pelo programa principal, e o valor de SR é fornecido ao programa principal pela sub-rotina. Estas variáveis são chamadas de parâmetros, sendo SN e SD os parâmetros de entrada, e SR o parâmetro de saída. Os valores

de SN e SD são fornecidos através da variável A e da constante 2, quando a sub-rotina é usada para arredondar A para duas decimais, e mais tarde através das variáveis I e da constante 2, quando a sub-rotina é usada para arredondar I para duas casas decimais. Deste modo, usando-se variáveis de entradas independentes em uma sub-rotina, podemos usá-la no programa principal para efetuar uma tarefa em duas ocasiões diferentes, em duas variáveis completamente diferentes do programa principal.

As linhas 85 e 95 abaixo, quando adicionadas ao programa principal, inicializarão os parâmetros de entrada SN e SD, chamarão a sub-rotina e retornarão o resultado para variáveis AA e II, respectivamente, que serão usadas na linha 100 para imprimir os valores arredondados.

```
85   SN = A:SD = 2:GOSUB 10000:AA = SR
95   SN = I:SD = 2:GOSUB 10000:II = SR
100  PRINT N;TAB(10);II;TAB(26);AA
```

Inclua estas linhas e rode novamente o programa.

A sub-rotina acima é um miniprograma que pode executar uma dada função. Se respeitarmos algumas diretrizes, poderemos construir uma biblioteca de sub-rotinas standard, que poderão ser usadas como blocos de construção na elaboração de programas mais longos. As diretrizes a serem observadas neste livro são as seguintes:

(*a*) Todas as sub-rotinas terão seus números de linha com dígitos, sendo os dois primeiros comuns a todas as linhas de uma dada sub-rotina, i.e., 10000 a 10990 serão os números de linha de uma sub-rotina, enquanto que 11000 a 11990 serão os de outra.

(*b*) Todos os nomes das variáveis usadas como parâmetros de entrada ou saída em uma sub-rotina terão como prefixo a letra S, sendo o valor inicial de entrada inicializado no programa principal antes da chamada da sub-rotina.

Ambos os pontos foram seguidos na sub-rotina antes estudada.

Alinhamento de números à direita

Observe que os resultados do programa anterior não aparecem alinhados à direita, o que torna difícil, por exemplo, a adição dos números de cada coluna. A sub-rotina apresentada abaixo irá gerar zeros depois do ponto decimal e, em seguida, alinhar pela direita todos os números. Usa-la-emos ainda como outra ilustração do uso de sub-rotinas.

Já dissemos que sub-rotinas são miniprogramas que podem executar uma dada função. Como tais, eles podem ser escritos e testados independentemente do programa principal, o que é exatamente o que fizemos abaixo:

```
   10  REM PROGRAMA TESTE P/A SUB 11000
   20  REM SO USE INTEIROS OU NUMEROS COM UMA OU
         DUAS CASAS DECIMAIS
   30  INPUT "ENTRE UM NUMERO ";N
   40  SZ = INT(10 ↑ 2*N + .5)/10 ↑ 2:GOSUB 11000:N$ = SR$
   50  PRINT N$:END
11000  REM ALINHAMENTO DE NUMEROS A DIREITA
11010  Z$ = STR$(SZ):LZ = LEN(Z$)
11020  FOR J = 1 TO LZ:IF MID$(Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ + Z$ + ".00":GOTO 11070
```

```
11050  IF (LZ – J) = 2 GOTO 11070
11060  Z$ = Z$ + "0"
11070  SR$ = RIGHT$("          " + Z$,10)
11080  RETURN
```

O parâmetro de entrada para esta sub-rotina é SZ, enquanto que o parâmetro de saída é SR$. A linha 11010 converte SZ para um string e depois iguala LZ ao comprimento deste string. A linha 11020 procura no string um ponto decimal e, se não encontra um, concatena a ele o string '00'. Se, contudo, ela encontra um ponto decimal no string, o programa salta para a linha 11050 onde é verificado se há ou não 2 dígitos após o ponto decimal. Se houver, ele salta para a linha 11070, do contrário ele concatena um zero no final do string e depois vai à linha 11070, onde dez caracteres em branco são concatenados, e os dez caracteres da extremidade direita do string resultante são atribuídos à SR$.

Isto é importante, porque se Z$ for mais curto que 10 caracteres então caracteres em branco são necessários à sua frente, para propiciar um correto alinhamento à direita.

Problema 5.2

Modifique o programa acima para que a sub-rotina 11000 possa manipular números com qualquer quantidade específica de dígitos após o ponto decimal. Esta quantidade deve ser especificada pelo parâmetro de entrada SD.

Note que uma forma de se fazer isto é mudando a linha 11040 para que o ponto decimal seja concatenado primeiro, seguido de SD zeros. Similarmente, as linhas 11050 e 11060 devem agora ser mudadas de forma que o número correto de zeros seja concatenado ao final do string. O número de zeros varia de 1 a SD-1.

Vamos agora juntar o programa dos juros compostos com as duas sub-rotinas recém-desenvolvidas. O programa resultante está listado abaixo, seguido do seu fluxograma.

```
   10  REM JUROS COM DUAS SUB-ROTINAS
   20  INPUT "QUANTIA EMPRESTADA ";P
   30  INPUT "TAXA DE JUROS ";R
   40  INPUT "NUMERO DE ANOS ";X
   50  PRINT:PRINT
   60  PRINT "ANOS"; TAB(10);"JUROS"; TAB(26);"QUANTIA":
       PRINT
   70  FOR N = 1 TO X
   80  A = P*(1 + R/100) ↑ N
   85  SN = A:SD = 2:GOSUB 10000:AA = SR
   87  SZ = AA:GOSUB 11000:A$ = SR$
   90  I = A – P
   95  SN = I:SD = 2:GOSUB 10000:II = SR
   97  SZ = II:GOSUB 11000:I$ = SR$
  100  PRINT N;TAB(10);I$;TAB(26);A$
  110  NEXT N
  120  END
10000  REM SUB-ROTINA P/CONVERTER NUMEROS P/O FORMATO
       ###.##
```

```
10005  REM I/O PARAMETERS: INPUT SN, SD – OUTPUT SR
10010  SR = INT(10 ↑ SD*SN + 0.5)/10 ↑ SD
10020  RETURN
11000  REM SUB TO RIGHT JUSTIFY NUMBERS
11005  REM I/O PARAMETERS: INPUT SZ – OUTPUT SR$
11010  Z$ = STR$(SZ):LZ = LEN(Z$)
11020  FOR J = 1 TO LZ:IF MID$(Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ = Z$ +".00":GOTO 11070
11050  IF (LZ – J) = 2 GOTO 11070
11060  Z$ = Z$ +"0"
11070  SR$ = RIGHT$("          "+Z$,10)
11080  RETURN
```

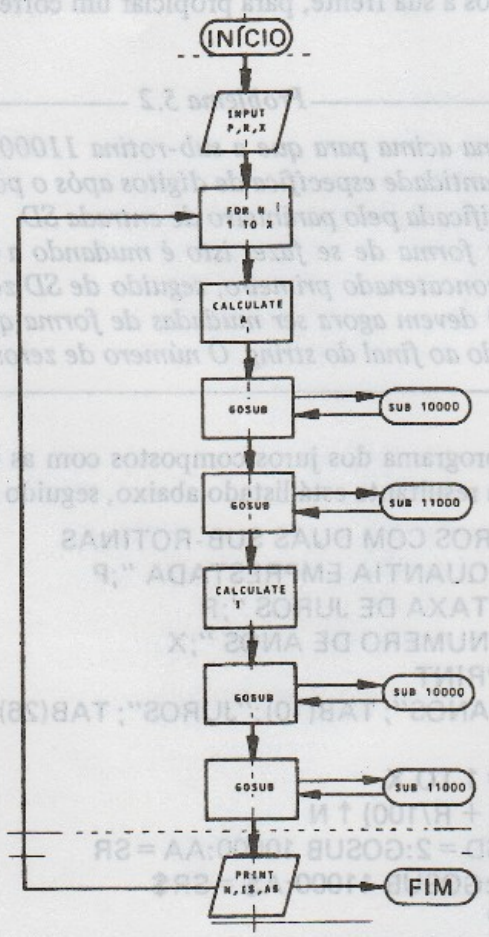


Fig. 5.2 Fluxograma do programa dos juros.

Sub-rotinas aninhadas

Pela análise do fluxograma e da listagem do programa dos juros compostos vê-se claramente que o fluxo desvia para a sub-rotina 10000, e retorna ao programa principal só pa-

ra desviar-se imediatamente para a sub-rotina 11000. Uma outra maneira de se fazer isto seria primeiro desviar para a sub-rotina 10000, depois, antes de voltar ao programa principal, desviar para a sub-rotina 11000 a partir da sub-rotina 10000. Isto é obtido adicionando-se uma linha extra à sub-rotina 10000, a saber:

```
10015  SZ=SR:GOSUB 11000
```

modificando as linhas 85 e 95, como mostrado abaixo, e deletando as linhas 87 e 97,

```
85  SN=A:SD=2:GOSUB 10000:A$=SR$
95  SN=I:SD=2:GOSUB 10000:I$=SR$
```

Representando em forma de blocos, o que fizemos foi o seguinte:

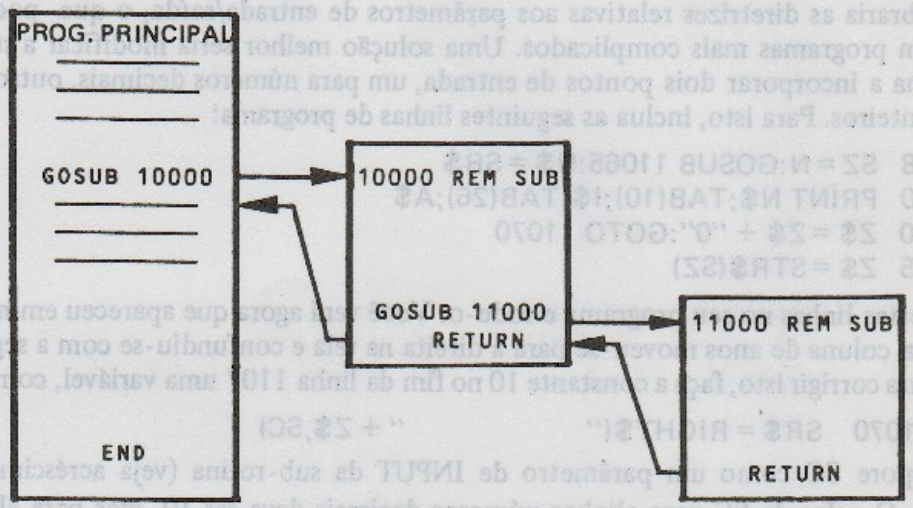


Fig. 5.3 Representação em diagrama de sub-rotinas aninhadas.

Apresentado acima está somente o resultado da execução da linha 85. A linha 95 simplesmente repete a chamada à sub-rotina 10000 um pouco mais abaixo no programa principal. Observe a trilha de retorno da sub-rotina 11000. Ela é inicialmente retornada à sub-rotina 10000 e subseqüentemente a instrução RETURN desta sub-rotina devolve o controle ao programa principal.

A sub-rotina 11000 na configuração acima é conhecida como uma sub-rotina aninhada. Aninhamentos múltiplos, i.e., chamados de sub-rotinas de dentro de uma sub-rotina que está dentro de outra, que por sua vez foi chamada pelo programa principal, são dispendiosos em termos de memória, além de poderem acarretar erros de programação devido a necessidade de se inicializarem os parâmetros de entrada de uma sub-rotina dentro de outra. Por estes motivos, não aconselhamos o aninhamento de sub-rotinas.

Problema 5.3

Modifique o programa dos juros compostos para que a sub-rotina 11000 seja substituída por aquela desenvolvida no Problema 5.2, lembrando-se também de tornar o parâmetro de entrada SD uma variável.

Sub-rotinas com múltiplos pontos de entrada

Por fim, ainda resta um problema a ser resolvido no exercício de Juros Compostos. Note que os números listados sob o cabeçalho ANOS não estão com alinhamento à direita. Isto é notado rodando-se o programa para 15 anos. A sub-rotina 11000 pode ser chamada para ajustar N (o número representando anos) antes da impressão. Isto, porém, gerará um ponto decimal e dois zeros após cada número, o que não é desejável.

Uma possível solução para alinhar inteiros à direita consiste em desviar o programa para a parte da sub-rotina que trata de alinhamento mas exclui a parte que adiciona ponto decimal e zeros. Com efeito, isto pode ser feito com a inclusão da instrução:

```
98  Z$ = STR$(N):GOSUB 11070
```

e alterando-se a primeira parte da linha 100 para PRINT Z$ em vez de N que, entretanto, quebraria as diretrizes relativas aos parâmetros de entrada/saída, o que pode levar a erros em programas mais complicados. Uma solução melhor seria modificar a sub-rotina de forma a incorporar dois pontos de entrada, um para números decimais, outro para números inteiros. Para isto, inclua as seguintes linhas de programa:

```
   98  SZ = N:GOSUB 11065:N$ = SR$
  100  PRINT N$;TAB(10);I$;TAB(26);A$
11060  Z$ = Z$ + "0":GOTO 11070
11065  Z$ = STR$(SZ)
```

Digite estas linhas no seu programa e rode-o. Você verá agora que apareceu em novo programa: a coluna de anos moveu-se para a direita na tela e confundiu-se com a segunda coluna. Para corrigir isto, faça a constante 10 no fim da linha 1107 uma variável, como segue,

```
11070  SR$ = RIGHT$("                    " + Z$,SC)
```

e incorpore SC como um parâmetro de INPUT da sub-rotina (veja acréscimo à linha 11005). O valor de SC para alinhar números decimais deve ser 10, mas para alinhar números inteiros deve ser, por exemplo, 3. SC é o número de caracteres de juros no string Z$ e, claramente, neste caso, 3 é bem suficiente. Observe que se SC é feito igual a 3, e se um número inteiro com mais de 3 dígitos for processado, o dígito mais significativo não será impresso. Portanto, o usuário deve escolher o valor de SC tendo em mente o tamanho do número a ser alinhado. As duas linhas afetadas são 98 e 10015.

A listagem do programa final, após todas as modificações sugeridas, fica assim:

```
   10  REM JUROS COM SUB-ROTINAS ANINHADAS
   20  INPUT "QUANTIA EMPRESTADA ";P
   30  INPUT "TAXA DE JUROS ";R
   40  INPUT "NUMERO DE ANOS ";X
   50  PRINT:PRINT
   60  PRINT "ANOS";TAB(10);"JUROS";
         TAB(26);"QUANTIA":PRINT
   70  FOR N = 1 TO X
   80  A = P*(1 + R/100) ↑ N
   85  SN = A:SD = 2:GOSUB 10000:A$ = SR$
   90  I = A - P
   95  SN = I:SD = 2:GOSUB 10000:I$ = SR$
   98  SZ = N:SC = 3:GOSUB 11065:N$ = SR$
  100  PRINT N$;TAB(10);I$;TAB(26);A$
```

```
  110  NEXT N
  120  END
10000  REM SUB P/CONVERTER NUMEROS P/O FORMATO ###.##
10005  REM I/O PARAMETERS: INPUT SN, SD – OUTPUT SR
10010  SR = INT(10 ↑ SD*SN + 0.5)/10 ↑ SD
10015  SZ = SR:SC = 10:GOSUB 11000
10020  RETURN
11000  REM SUB P/ALINHAR A DIREITA
11005  REM I/O PARAMETERS: INPUT SZ, SC – OUTPUT SR$
11010  Z$ = STR$(SZ):LZ = LEN(Z$)
11020  FOR J = 1 TO LZ:IF MID$(Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ = Z$ + ".00":GOTO 11070
11050  IF (LZ – J) = 2 GOTO 11070
11060  Z$ = Z$ + "0":GOTO 11070
11065  Z$ = STR$(SZ)
11070  SR$ = RIGHT$("                " + Z$,SC)
11080  RETURN
```

Problema 5.4

Modifique o programa dos Juros Compostos resultante do Problema 5.3, para incorporar dois pontos de entrada para a sub-rotina de alinhamento à direita, como discutido anteriormente.

CAPÍTULO 6

Comunicação com periféricos

Já vimos que informação pode ser enviada a um periférico, como uma impressora, digitando-se o comando DOS

PR#*s*

onde *s* é o número do soquete (SLOTO) ao qual o periférico é conectado.

Analogamente, se uma entrada foi requerida de um periférico conectado ao soquete *s*, o comando DOS

IN#*s*

levará o computador a procurar a entrada no soquete *s*. Isto é verificável digitando-se IN#6 para que o computador rechame ("re-boot") o sistema do disco, desde que o cartão controlador de disco esteja no soquete 6.

Comandos DOS a partir do BASIC

Comandos DOS podem ser usados a partir de um programa BASIC imprimindo-se um string que consiste de 'CTRL'D seguido pelo comando entre aspas, como se segue:

```
10 D$=CHR$(4)
20 PRINT D$;"PR#1"
```

Na linha 10 usamos a função CHR$( ) com um argumento de valor 4 que equivale a digitar 'CTRL'D. Alternativamente, poderíamos digitar 'CTRL'D entre aspas. Mas como o caractere gerado por 'CTRL'D é do tipo que não é impresso, poderia acarretar um erro de transcrição, erroneamente interpretado como um espaço entre as aspas. Por este motivo é dada preferência ao uso da função CHR$( ).

Para retornar a impressão ao monitor, adicione a instrução

```
100 PRINT D$;"PR#0"
```

que deve aparecer em um programa após a instrução que causou a impressão de saída.

O programa seguinte irá imprimir o string XYZ na impressora.

```
10 REM IMPRIMIR SAIDA NA IMPRESSORA
20 S$="XYZ"
30 INPUT "OUTPUT NA TELA OU IMPRESSORA? (S/P) ";A$
40 IF LEFT$(A$,1)="S" THEN PRINT S$:GOTO 90
50 IF LEFT$(A$,1)<>"P" GOTO 30
60 D$=CHR$(4)
70 PRINT D$;"PR#1":PRINT S$
80 PRINT D$;"PR#0"
90 END
```

Problema 6.1

Incorpore o recurso acima para imprimir o resultado dos cálculos do programa dos Juros Compostos (Veja Prob. 5.4) na impressora. Todas as linhas adicionais requeridas para isso devem ser colocadas entre as instruções 50 e 60 já existentes. O retorno do controle para a tela do monitor deve ser inserido entre as linhas 110 e 120.

O uso de comandos DOS a partir de um programa pode ser muito útil em vários casos, exceto para impressão dos resultados na impressora. Por exemplo, quando chamar o sistema (BOOT) em um dado disquete, a sua listagem de arquivo pode ser exposta na tela automaticamente alternando-se o programa ALO para o seguinte:

```
10 REM HELLO
20 D$=CHR$(4)
30 PRINT D$;"CATALOG"
40 END
```

Delete o programa ALO do disquete, limpe a memória do computador, digitando NEW, e em seguida digite o programa acima e salve-o no lugar do antigo programa ALO. Se você estiver usando um Apple com uma versão do DOS em que o comando CATALOG tenha sido abreviado para CAT, você terá de substituir a palavra CAT no lugar de CATALOG na linha 20. Note que se seu disquete não contiver o programa com o nome originalmente usado para inicializá-lo (no nosso caso ALO), este disquete não poderá chamar o sistema (boot).

Todos os comandos DOS podem ser usados em um programa na forma discutida acima. O único comando DOS que jamais deverá ser usado em um programa é o comando INIT, visto que inicializa um disquete e, portanto, apaga todos os programas nele armazenados.

O comando RUN pode ser usado eficazmente em um programa BASIC para carregar (LOAD) e rodar (RUN) um programa à sua escolha. De fato, você poderia tornar o Apple II em um sistema "turn-key" (i.e., entrar no meio de um programa simplesmente ligando o computador, tão simples quanto ligar uma chave) incorporando o comando RUN ao programa ALO, como mostrado a seguir.

```
10 REM HELLO
20 D$=CHR$(4)
30 PRINT D$;"RUN COMPOUND INTEREST"
40 END
```

Trocando o seu atual programa ALO pelo programa acima, ele não só chamará o sistema, mas irá ao mesmo tempo carregar e rodar automaticamente o programa dos Juros Compostos, ou qualquer outro programa de sua escolha.

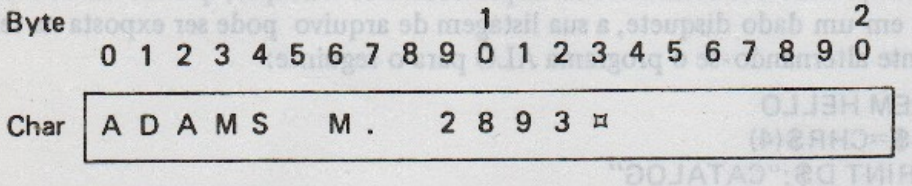
Arquivos-texto

Vimos que programas podem facilmente ser armazenados em disquetes. Com a mesma facilidade, podemos usar um disquete para armazenar dados em arquivos chamados texto ou arquivo de dados e que são identificados pela letra I no diretório do disco. Existem dois tipos de arquivos-texto que podem ser usados para armazenar informações. São eles o arquivo texto seqüencial e o arquivo-texto de acesso randômico. Estudaremos ambos,

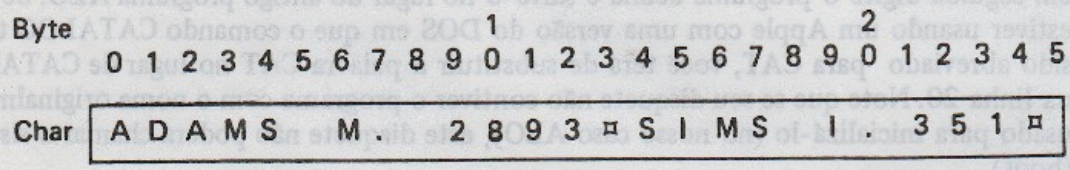
inicialmente analisando a sua estrutura e depois discutindo o processo utilizado para escrever e ler dados em cada um destes arquivos.

Arquivos-texto seqüenciais

Um arquivo-texto seqüencial pode ser pensado como um array unidimensional, com cada posição deste sendo um byte capaz de armazenar um caractere de um string. Por exemplo, o nome de um amigo juntamente com seu telefone poderiam ser guardados (no código ASCII) com a seguinte representação visual



onde o byte número 13 (que neste caso é o 11º byte, visto que o 1º byte começa em ∅) será o código ASCII para RETURN e que indica o fim do campo de dados, e, neste caso, também o fim do arquivo. O nome de dois amigos serão armazenados como segue:



onde os bytes de número 13 e 25 representam o código ASCII para RETURN e indicam os respectivos fins dos dois campos que compõem o arquivo. O RETURN final indica o fim do arquivo. Acréscimos a tal arquivo começarão a partir do byte de número 26. Para poder se escrever dados em um arquivo de texto seqüencial, precisamos escrever um pequeno programa BASIC que irá 'criar' tal arquivo e que irá 'escrever' nele os caracteres que representam a informação que gostaríamos de armazenar no disquete. Os strings que compõem cada campo do arquivo serão inicialmente lidos (READ) no computador e armazenados em um array de strings. Subseqüentemente, eles serão transferidos para o disquete. O programa abaixo satisfaz tais exigências.

```
 10  REM CRIACAO DE SEQ. TEXT FILE
 20  READ N:DIM A$(N)
 30  FOR I=1 TO N:READ A$(I):NEXT I
 40  DATA 2
 50  DATA "ADAMS M. 2893"
 60  DATA "SIMS I. 351"
100  INPUT "SALVAR SOB NOME? ";X$
110  D$=CHR$(4)
120  PRINT D$;"OPEN";X$
130  PRINT D$;"WRITE";X$
140  PRINT N
150  FOR I=1 TO N:PRINT A$(I):NEXT I
160  PRINT D$;"CLOSE";X$
170  END
```

As linhas 20 e 30 permitem a leitura de cada campo que está armazenado no array de

strings A$. A partir da linha 100, os comandos de controle OPEN, WRITE e CLOSE devem ser executados em modo "deferred", daí a existência da linha 110. A linha 120 abre (OPEN) um arquivo cujo nome é aquele que você deu em resposta à instrução INPUT da linha 100. Ao abrir um arquivo, o nome deste arquivo é automaticamente escrito no diretório do disquete, a não ser que este já exista no disquete, que permitirá o seu acesso a ele. A linha 130 permite que você escreva no arquivo, o qual acabou de abrir, usando as instruções PRINT das linhas 140 e 150. A linha 140 escreve no disquete (no byte 0) o valor de N, que é o número total de campos usados. O loop FOR-NEXT na linha 150 simplesmente força a impressão, no arquivo-texto, dos strings armazenados no array de strings. Quando a impressão de todos os strings estiver completa, o arquivo é fechado (CLOSE) e o programa termina.

Digite este programa em seu computador, salve-o e rode-o. Quando lhe for pedido um nome de arquivo no qual serão armazenadas as informações, digite TEL LIST. Ao término da execução deste programa, o arquivo TEL LIST é ligado no diretório do disquete com a letra de prefixo T, para indicar seu tipo.

Uma vez criados os arquivos-texto, devemos estar aptos a ler (READ) cada campo de volta ao computador para que as informações possam ser recuperadas novamente. Isto é obtido através de um pequeno programa BASIC como o dado abaixo.

```
10  REM RECUPERACAO DE ARQUIVO TEXTO SEQUENCIAL
20  D$=CHR$(4)
30  INPUT "NOME DO ARQUIVO A SER CARREGADO? ";X$
40  IF X$="" THEN PRINT D$;"CATALOG":GOTO30
50  PRINT D$;"OPEN";X$
60  PRINT D$;"READ";X$
70  INPUT N:DIM A$(N)
80  FOR I=1 TO N:INPUT A$(I):NEXT I
90  PRINT D$;"CLOSE";X$
100 FOR I=1 TO N:PRINT A$(I):NEXT I
110 END
```

A linha 40 nos permite examinar o disquete, a fim de consultar o nome de arquivo correto, simplesmente apertando a tecla RETURN. As linhas 50 e 60 abrem e lêem o arquivo designado. O primeiro item a ser lido é o valor de N (número total de string no arquivo), o que é obtido através da instrução PRINT da linha 70. Subseqüentemente, todos os strings são introduzidos no array A$ através do uso do loop FOR-NEXT da linha 80. Finalmente, o arquivo é fechado e o conteúdo do array A$ é impresso na tela.

Tanto os comandos WRITE como READ devem seguir um comando OPEN para um arquivo que já esteja fechado. Ao se tentar ler (READ) mais campos que os disponíveis no arquivo irá causar a emissão da mensagem

```
END OF DATA (fim de arquivo)
```

que aparece na tela.

Usando a instrução INPUT nos arquivos-texto

Uma outra maneira de se criar arquivos-texto seqüenciais seria substituindo a instrução READ-DATA do programa "criação" pela instrução INPUT. Este método também é útil para atualizar arquivos-texto. Faça as seguintes modificações:

```
10 REM INPUT EM ARQUIVO SEQ.
20 INPUT "NUMERO DE NOVAS ENTRADAS? ";NE:DIM A$(NE)
30 FOR N=1 TO NE:PRINT N;">";:INPUT "";A$(N)
40 IF A$(N)<>"" THEN NEXT N
50 N=N-1
```

A linha 60 não é necessária e deverá ser deletada da nova versão do programa "criação". Salve o programa final sob o nome de arquivo INPUT ARQUIVO SEQ. e em seguida rode-o. As aspas que seguem a instrução INPUT na linha 30 lá estão para eliminar o (?) emitido pelo BASIC quando executada a instrução e que estragaria a impressão de saída. A linha 40 diz que, se a última entrada não for um RETURN, então continue com a entrada do próximo campo. Isto lhe permite terminar as entradas antes do término do loop FOR-NEXT das linhas 30 e 40 simplesmente apertando RETURN. Em ambos os casos, o contador de controle N no loop terá sido incrementado de 1, que terá de ser subtraído antes de se escrever o seu valor no arquivo do disquete.

Note que o uso da vírgula no string de entrada irá causar a emissão da mensagem "? EXTRA IGNORED" na tela. Isto porque o BASIC assume que você deseja entrar dois strings em duas variáveis separadas por vírgulas.

Problema 6.2

Use a instrução ON-GOTO para escrever um programa que lhe permita tanto criar um novo arquivo (usando a instrução INPUT), ou recuperar um já existente.

Atualização de arquivos-texto seqüenciais

Até agora, já vimos como criar e recuperar arquivos-texto seqüenciais. Contudo, arquivos raramente são criados na sua íntegra e, por isto, faz-se necessário investigar a possibilidade de atualização de arquivos existentes. O programa abaixo irá abrir um arquivo existente, ler o primeiro campo, o qual contém o número total de registros no arquivo, e dimensionar um array A$ com número total de registros existentes mais os novos registros a serem incluídos no arquivo. A linha 90 coloca (INPUT) em A$ os registros já existentes no arquivo, enquanto a linha 100 fecha (CLOSE) o arquivo.

```
 10 REM ATUALIZACAO DE ARQUIVO SEQ.
 20 D$=CHR$(4)
 30 INPUT "ARQUIVO A SER ATUALIZADO? ";X$
 40 IF X$="" THEN PRINT D$;"CATALOG":GOTO 30
 50 INPUT "NUMERO DE NOVAS ENTRADAS? ";N
 60 PRINT D$;"OPEN";X$
 70 PRINT D$;"READ";X$
 80 INPUT K:DIM A$(K+N)
 90 FOR I=1 TO K:INPUT A$(I):NEXT I
100 PRINT D$;"CLOSE";X$
110 FOR I=K+1 TO K+N:PRINT I;">";:INPUT "";A$(I)
120 IF A$(I)<>"" THEN NEXT I
130 I=I-1
140 PRINT D$;"OPEN";X$
150 PRINT D$;"DELETE";X$
```

```
160 PRINT D$;"OPEN";X$
170 PRINT D$;"WRITE";X$
180 PRINT I
190 FOR L=1 TO I:PRINT A$(L):NEXT L
200 PRINT D$;"CLOSE";X$
210 PRINT "OPERACAO CONCLUÍDA":END
```

Novos registros são introduzidos em A$ na linha 110, enquanto que a linha 120 nos permite sair do loop de entradas máximas pressionando RETURN em resposta a uma nova entrada. Um contador I é incrementado de tal forma que ao sair do loop ele possui o valor correto de registros armazenados em A$. A partir da linha 140, o arquivo é aberto (OPEN), apagado (DELETE), reaberto e o número de registros do novo arquivo é escrito no primeiro campo, enquanto que os campos subseqüentes mantêm os registros atuais.

Observe o uso do comando DELETE. O seu uso pode ser muito importante, pois quando você está escrevendo em um arquivo existente, sem antes apagá-lo, você poderá estar escrevendo por cima do arquivo antigo e, a não ser que seu novo arquivo tenha mais caracteres que o velho, você acabará com uma mistura do novo seguido pela parte restante dos registros antigos. Neste exemplo, como o arquivo novo é mais longo que o antigo, não é estritamente necessário o uso do comando DELETE.

A maior desvantagem do programa de "atualização" acima é que a dimensão do array A$ deve ser suficientemente grande para armazenar todo o arquivo, e com arquivos grandes pode não haver espaço suficiente na memória RAM de seu computador. Além disso, tempo é gasto em se ler e escrever arquivos grandes.

O uso do comando APPEND

O programa abaixo supera as limitações do programa anterior, através do uso do comando APPEND. Note que a dimensão do array A$ está restrita somente ao número de novas entradas. Estas são introduzidas no array e seu número armazenado na variável I.

```
 10 REM APPEND ARQUIVO SEQ.
 20 D$=CHR$(4)
 30 INPUT "ARQUIVO A SER ATUALIZADO? ";X$
 40 IF X$="" THEN PRINT D$;"CATALOG":GOTO 30
 50 INPUT "NUMERO DE NOVAS ENTRADAS? ";N:DIM A$(N)
 60 FOR I=1 TO N:PRINT I;">";:INPUT "";A$(I)
 70 IF A$(I)<>"" THEN NEXT I
 80 I=I-1
 90 PRINT D$;"OPEN";X$
100 PRINT D$;"READ";X$
110 INPUT K
120 PRINT D$;"CLOSE";X$
130 K=K+I
140 PRINT D$;"OPEN";X$
150 PRINT D$;"WRITE";X$
160 PRINT K
170 PRINT D$;"CLOSE";X$
180 PRINT D$;"APPEND";X$
190 PRINT D$;"WRITE";X$
200 FOR L=1 TO I:PRINT A$(L):NEXT L
```

```
210 PRINT D$;"CLOSE";X$
220 PRINT "OPERACAO CONCLUIDA":END
```

Nas linhas 90 a 130, o arquivo é aberto, o número de registros existentes no arquivo (K) é lido e adicionado ao valor de novas entradas. Nas linhas 140 a 170 este número é escrito no primeiro campo do arquivo e o arquivo é fechado novamente.

O uso do comando APPEND na linha 180 primeiro abre o arquivo e em seguida procura o marcador (o posicionamento do RETURN final do arquivo). Qualquer uso subseqüente do comando WRITE irá escrever os novos registros a partir do final do arquivo existente. Digite este programa em seu computador e salve-o sob o nome de arquivo APPEND ARQUIVO SEQ.

Uso da instrução GET em arquivos-texto

O uso da instrução INPUT ao criar ou recuperar arquivos-texto não possui limitações. A vírgula, ponto-e-vírgula e aspas, os quais podem ser essenciais para o sentido do texto, são dados inaceitáveis. O uso da instrução GET no lugar de INPUT irá contornar este problema.

O programa abaixo irá criar um arquivo-texto usando a instrução GET. Um caractere é pegado de cada vez (linha 40) e examinado para ver se este é um RETURN (CHR$ (13)). Se não for, é acumulado em uma variável string A$ (linha 50) e impresso na tela para que possamos vê-lo (o GET extrai um caractere mas não o apresenta na tela). A linha 60 confere se o string contém 'nada'. Isto fornece uma saída do loop antes que ele seja executado NE vezes, simplesmente apertando RETURN. Depois disto, os strings criados são escritos no arquivo designado, da mesma maneira que antes.

```
 10 REM CRIACAO DE ARQUIVO SEQ. COM GET
 20 INPUT "NUMERO DE NOVAS ENTRADAS? ";NE:DIM A$(NE)
 30 FOR N=1 TO NE:PRINT:PRINT N;">";
 40 GET Z$:IF Z$=CHR$(13) GOTO 60
 50 A$(N)=A$(N)+Z$: Z$:PRINT Z$;:GOTO 40
 60 IF A$(N)="" GOTO 80
 70 NEXT N
 80 N=N-1:PRINT
 90 INPUT "NOME SOB O QUAL DEVE SER SALVADO? ";X$
100 D$=CHR$(4)
110 PRINT D$;"OPEN";X$
120 PRINT D$;"WRITE";X$
130 PRINT N
140 FOR I=1 TO N:PRINT A$(I):NEXT I
150 PRINT D$;"CLOSE";X$
160 PRINT "OPERACAO CONCLUIDA":END
```

O programa correspondente para recuperar arquivos existentes usando a instrução GET é mostrado abaixo. Os princípios são os mesmos usados no programa "criação".

```
 10 REM RECUPERACAO DE ARQUIVO SEQ COM GET
 20 D$=CHR$(4)
 30 INPUT "NOME DO ARQUIVO A SER CARREGADO? ";X$
 40 IF X$="" THEN PRINT D$;"CATALOG":GOTO 30
 50 PRINT D$;"OPEN";X$
```

```
60  PRINT D$;"READ";X$
70  INPUT N:DIM A$(N)
80  FOR I=1 TO N
90  GET Z$:IF Z$=CHR$(13) GOTO 110
100 A$(I)=A$(I)+Z$:GOTO 90
110 NEXT I
120 PRINT D$:PRINT D$;"CLOSE";X$
130 FOR I=1 TO N:PRINT A$(I):NEXT I
140 END
```

Digite ambos os programas anteriores em seu computador e salve-o sob nomes de arquivo apropriados.

ATENÇÃO: Problemas com o uso da instrução GET

(*i*) A instrução GET irá procurar um caractere sem exigir que a tecla RETURN Seja apertada. Como conseqüência disto, a parte delimitada da instrução INPUT da linha 90 do programa "criação' acima seria impressa no final da última entrada do arquivo-texto, se o PRINT (que faz saltar para a linha seguinte), na linha 80, não tivesse sido incluído. O uso de PRINT D$ isoladamente (onde D$=CHR$(4) foi especificado), como um comando para mudar de linha, teria a mesma eficiência.

(*ii*) Quando GET é usado para extrair caracteres de um arquivo em disco, tem o efeito de cancelamento do próximo comando DOS. Para retificar isto, um PRINT D$ extra deve ser inserido como na linha 120 do programa de "recuperação" acima.

(*iii*) Além disso, quando GET for usado para procurar caracteres de um disquete, se a primeira instrução PRINT que se seguir for pura (i.e., simplesmente PRINT), o primeiro caractere na impressão será perdido. Mais uma vez, se antes for inserido um PRINT D$, isto será evitado.

Note que se estes programas forem para ser usados para um número grande de registros, o primeiro campo do arquivo, no qual está armazenado o número total de registros, deve ser de bytes suficientes, para que possa acomodar um número grande, sem escrever por cima das entradas seguintes. A forma mais simples de se obter isto é entrar com o valor N (o número total de registros no primeiro campo) como um string precedido de caracteres em branco. Para implementar isto complemente a linha 130 do programa CRIACAO DE ARQUIVO SEQ com GET da seguinte forma:

```
130 N$=STR$(N):PRINT RIGHT$("    "+N$,5)
```

Ao ler o string do arquivo, será necessário usar a função VAL( ) a fim de se obter o seu valor aritmético. Para obter isto, faça o seguinte acréscimo no seu programa Recuperar ARQUIVO SEQ com GET

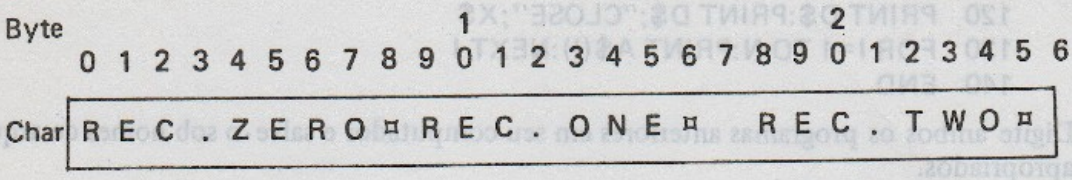
```
70 INPUT N$:N=VAL(N$):DIM A$(N)
```

Problema 6.3

Modifique o programa 'criação/recuperar' escrito no Problema 6.2 para incluir uma opção complementar. Incorpore a instrução GET em cada seção e dê espaço suficiente no primeiro campo para um número de registros totais com 5 dígitos.

Acesso randômico a arquivos-texto

Os arquivos de dados de acesso randômico são como uma coleção de arquivos seqüenciais de mesmo comprimento, o que significa que cada registro pode possuir um número de campos (cada um terminando com um RETURN). O comprimento de registro escolhido deve ser especificado pelo parâmetro L. Uma representação visual dos arquivos-texto de acesso randômico é dada abaixo.



Observe que o primeiro registro de um arquivo-texto de acesso randômico é o registro de número zero, o segundo é o registro de número um etc. O comprimento de cada registro na representação acima é nove bytes.

Arquivos de acesso randômico são ineficientes quando há economia de espaço. Por exemplo, o primeiro registro acima poderia conter um nome bem curto. Mesmo assim um comprimento standard é estabelecido para cada registro. Por outro lado, informações podem ser encontradas facilmente (o terceiro registro, por exemplo, começa no byte 3*L) onde L é o comprimento do registro.

Criação e recuperação de arquivos randômicos

A criação e recuperação de arquivos de acesso randômico seguem regras semelhantes às estabelecidas para arquivos-texto seqüenciais com as seguintes exigências adicionais: quando um arquivo de acesso randômico for aberto, o parâmetro de comprimento do registro, L, precisa ser especificado, e na execução de um READ ou WRITE, um parâmetro de número de registro, R, é necessário.

O exemplo seguinte irá ajudar a ilustrar como arquivos de acesso randômico podem ser criados usando a instrução INPUT.

```
10 REM CRIACAO DE ARQUIVO RANDOMICO COM INPUT
20 INPUT "NUMERO DE NOVAS ENTRADAS? ";NE:DIM A$(NE)
30 FOR N=1 TO NE:PRINT N;">";:INPUT "";A$(N)
40 IF A$(N)<>"" THEN NEXT N:N=N-1
50 D$=CHR$(4)
60 PRINT D$;"OPEN TEL LIST, L25"
70 FOR RN=1 TO N: PRINT D$;"WRITE TEL LIST, R";RN
80 PRINT A$(RN):NEXT RN
90 PRINT D$;"CLOSE TEL LIST":END
```

A linha 60 abre um arquivo chamado TEL LIST e especifica o comprimento do registro em 25 bytes. As linhas 70 e 80 escrevem em TEL LIST o conteúdo de A$(1) no registro número 1 (RN=1), o conteúdo de A$(2) no registro número 2, e assim por diante.

Obviamente, quando estiver usando arquivos de acesso randômico, os registros jamais deverão ultrapassar o comprimento estabelecido pelo parâmetro L, do contrário a superposição de caracteres será inevitável, gerando resultados confusos. Analogamente, o mesmo parâmetro L usado na criação dos registros deve ser usado para recuperá-los. Informações relacionadas ao comprimento dos registros podem ser armazenadas no registro

zero do arquivo. Se L não for exatamente conhecido, resultará em confusões ao tentar recuperar registros.

Assim como arquivos seqüenciais, o uso da instrução INPUT pode criar problema, se o texto digitado possuir vírgulas e ponto-e-vírgula. Para contornar esta dificuldade, os dois programas dados abaixo usam a instrução GET. Ao mesmo tempo, eles foram projetados de forma a permitir ao usuário especificar o nome do arquivo em que se deseja salvar informações, ou de onde se deseja extraí-las.

Ao se criar um arquivo de acesso randômico, pede-se ao usuário que especifique o parâmetro de comprimento L, que é subseqüentemente armazenado no registro zero juntamente com o número total de registros do arquivo (linha 160). Estes ocupam um campo no mesmo registro. Todos os demais registros foram restritos a um único campo para simplificar. Mais tarde, discutiremos métodos para incluir vários campos em um único registro, cada qual podendo começar em um ponto específico do registro. Isto é chamado de "opção de parâmetro de byte"

```
 10 REM CRIACAO DE UM ARQUIVO DE ACESSO RANDOMICO
 20 D$=CHR$(4):CR$=CHR$(13)
 30 INPUT "COMPRIMENTO MAX DE REGISTRO? ";RL:IF RL>256GOTO 30
 40 INPUT "NUMERO DE NOVAS ENTRADAS? ";NE:DIM A$(NE)
 50 FOR N=1 TO NE:PRINT
 60 HTAB 1:PRINT N;">";
 70 GET Z$
 80 IF Z$=CR$ GOTO 100
 90 A$(N)=A$(N)+Z$:PRINT Z$;:GOTO 70
100 IF A$(N)="" GOTO 120
110 NEXT N
120 N=N-1:PRINT
130 INPUT "NOME SOB O QUAL DEVE SER SALVO? ";X$
140 RN=0
150 PRINT D$;"OPEN";X$;",L";RL
160 PRINT D$;"WRITE";X$;",R";RN
170 PRINT N:PRINT RL
180 FOR RN=1 TO N:PRINT D$;"WRITE";X$;",R";RN
190 PRINT A$(RN):NEXT.RN
200 PRINT D$;"CLOSE";X$
210 PRINT "OPERACAO CONCLUIDA":END
```

O programa seguinte irá recuperar um arquivo de acesso randômico específico, e usará os parâmetros armazenados no registro 0.

```
 10 REM RECUPERACAO DE ARQUIVO DE ACESSO RANDOMICO
 20 D$=CHR$(4):CR$=CHR$(13)
 30 INPUT "NOME  DO ARQUIVO A SER CARREGADO? ";X$
 40 IF X$="" THEN PRINT D$;"CATALOG":GOTO 30
 50 PRINT D$;"OPEN";X$
 60 PRINT D$;"READ";X$
 70 INPUT N:DIM A$(N):PRINT "NUMERO DE REGISTROS = ";N
 80 INPUT RL:PRINT "PARAMETRO DE COMPRIMENTO = ";RL
 90 PRINT D$;"CLOSE";X$
100 PRINT D$;"OPEN";X$;",L";RL
```

```
110 FOR RN=1 TO N:A$=""
120 PRINT D$:PRINT D$;"READ";X$;",R";RN
130 GET Z$:IF Z$=CR$ GOTO 150
140 A$(RN)=A$(RN)+Z$:GOTO 130
150 NEXT RN
160 PRINT D$:PRINT D$;"CLOSE";X$
170 FOR RN=1 TO N:PRINT A$(RN):NEXT RN
180 END
```

A linha 30 nos permite especificar o nome do arquivo a ser recuperado. Em seguida, o arquivo é aberto e o número máximo de registros assim como o valor do parâmetro de comprimento são lidos e atribuídos às variáveis N e RL, respectivamente. O arquivo é subseqüentemente fechado na linha 90.

Nas linhas 100 a 160, o arquivo é reaberto, mas desta vez com o parâmetro de comprimento L (anteriormente lido do registro zero) e todos os registros de 1 a N são lidos usando a instrução GET na linha 130. Note mais uma vez o uso dos dois comandos PRINT D$ nas linhas 120 e 160 para contornar o efeito de cancelamento da instrução GET na primeira instrução DOS subseqüente.

Problema 6.4

Modifique o programa Recuperação Arquivo Randômico de forma que, após ele ter recuperado um arquivo, mas antes de fechá-lo, ele permita a opção para inclusão de mais registros (veja nota abaixo), e que, neste caso, a informação no registro zero seja atualizada.

Nota: Quando um arquivo é aberto (OPEN), e você está lendo informações dele, todas as instruções INPUT são desviadas para o disquete, enquanto que todas as instruções PRINT são desviadas para o monitor. Reciprocamente, se você usou o comando WRITE, todas as instruções PRINT são desviadas para o disquete, enquanto que todas as instruções INPUT são desviadas para o monitor. O uso de um PRINT D$ extra cancela qualquer comando (READ ou WRITE) que esteja ativo.

Comprimento ótimo de registro

Observe que o comprimento do registro no programa CRIAR ARQUIVO RANDÔMICO foi escolhido como 25 bytes. Esta escolha a esmo leva a um uso ineficiente do espaço disponível no disquete pelos seguintes motivos:

O Apple armazena informações no disquete em 35 trilhas concêntricas, com o comprimento de cada trilha sendo subdividido em 16 setores (para DOS 3.3) ou 13 setores (para DOS 3.2). Cada setor pode armazenar 256 bytes de informações. Antes de escrever as informações no disquete, o Apple junta as informações em arquivos especiais de sua memória, chamados de Arquivos do Buffer*, que geralmente são três. Quando um desses arquivos de Buffer fica cheio, o computador transfere a informação para um setor do disco.

* Vide nota da tradução.

Assim, como as informações são armazenadas em comprimentos de 256 bytes, o comprimento ideal de um registro seria de múltiplos ou submúltiplos de 256, tais como 128, 64, 32, 16, 8 bytes. Quando adotamos estes comprimentos de registro, estaremos seguros de estarmos maximizando o número de registros armazenados em cada setor, o que, por sua vez, irá maximizar o número total de registros armazenados em cada disquete, sem desperdícios de bytes.

Contudo, isto exposto, deve ser mencionado que, tratando-se de arquivos, o parâmetro de comprimento de seus registros deve estar na faixa 1-32767, enquanto que o parâmetro do número de registros deve estar na faixa 0-32767.

CAPÍTULO 7

Programas utilitários e arquivos binários

Rotinas de manipulação de erros

Quando você usa uma unidade de disco, o DOS pode detectar erros, tais como SYNTAX ERROR (erro de sintaxe), quando você soletrar mal um comando DOS em seu programa, ou FILE NOT FOUND (arquivo não encontrado) se você tentar carregar um arquivo que não conste no diretório do disquete. Diferentemente das mensagens emitidas pelo BASIC APPLESOFT, que são precedidas por um ponto de interrogação (?), as mensagens dos DOS não são precedidas por nenhum caractere, para que possam ser diferenciadas das mensagens emitidas devido a erros em APPLESOFT.

Os erros que ocorrem durante a execução (RUN) do programa irão parar a mesma, o que, no caso de alguns erros DOS, pode ser indesejável. Por exemplo, suponha que estejamos rodando um programa que cria um arquivo de acesso randômico e que já tenhamos digitado um número considerável de registros. Ao se tentar salvar (SAVE) estes registros em um arquivo bloqueado, a mensagem DOS, FILE LOSKED (arquivo bloqueado), irá aparecer na tela com a inevitável perda total de todos os novos registros que tomaram um longo tempo de digitação no computador. Erros podem ser detectados usando-se o comando ONERR GOTO em um programa, e escrevendo uma rotina de manipulação de erros que pode fazer parte de seu programa. Quando ocorrer um erro em um programa após o comando ONERR GOTO, o código deste erro é armazenado na posição de memória 222, e poderá ser examinado conferindo o conteúdo daquele local usando o comando

```
ER=PEEK(222)
```

A variável ER irá então conter o código correspondente ao erro que causou a ação do comando ONERR GOTO.

Os códigos de erros do DOS estão numerados de 1 a 15 e estão listados abaixo, e na Tabela B-2 do Apêndice B, onde uma explicação das mensagens menos óbvias são dadas, junto com as suas respectivas mensagens. Os erros de código e os da faixa 16 a 225 estão relacionados a erros no BASIC Applesoft, e uma lista destes códigos, acompanhados das respectivas mensagens, é dada na Tabela B-1 do Anexo B.

TABELA 7.1 Códigos de erros do DOS

Cód.	Mensagem	*Cód.*	Mensagem
1	LING. NÃO DISPONÍVEL	2,3	FAIXA ERRO
4	DISCO PROTEGIDO	5	FIM DE ARQUIVO
6	NÃO HÁ ARQUIVO	7	VOLUME ILEGAL
8	E/S ERRO	9	<LOTADO>
10	<TRAVADO>	11	GRAFIA-ERRO
12	TODOS BUFFERS CHEIOS	13	TIPO ARQ. ILEGAL
14	PROGRAMA LONGO	15	COMANDO NÃO DIRETO

A fim de analisar o uso de uma rotina de manipulação de erros, carregue o programa CRIAÇÃO DE ARQUIVO DE ACESSO RANDÔMICO (se você não o tiver salvado, redigite-o) e depois digite os seguintes acréscimos:

```
   15  ONERR GOTO 50000
50000  REM ROTINA DE MANIPULACAO DE ERROS
50010  ER=PEEK(222)
50020  PRINT "ERRO NR ";ER;" IN LINE ";
       PEEK(218)+PEEK(219)*256
50030  IF ER=4 THEN PRINT "DISCO PROTEGIDO!": GOTO
       50130
50040  IF ER=5 OR ER=6 THEN PRINT "ARQUIVO NAO
       ENCONTRADO!": PRINT D$;"DELETE";X$:GOTO 50130
50050  IF ER=7 THEN PRINT "VOLUME ERRADO!"
50060  IF ER=8 THEN PRINT "I/O ERRO!"
50070  IF ER=9 THEN PRINT "DISCO LOTADO!": PRINT
       D$;"DELETE";X$:GOTO 50130
50080  IF ER=10 THEN PRINT "ARQUIVO TRAVADO!": INPUT
       "UNLOCK?";UN$:IF LEN(UN$)=0 THEN PRINT
       D$;"CATALOG":GOTO 50080
50090  IF LEFT$(UN$,1)="Y" THEN PRINT D$; "UNLOCK";X$:GOTO
       50150
50100  IF LEN(UN$)>0 GOTO 50130
50110  IF ER=11 THEN PRINT "NOME ILEGAL!": GOTO 50130
50120  IF ER=13 THEN PRINT "TIPO DE ARQUIVO INCOMPATIVEL!"
50130  IF LD=1 GOTO 50160
50140  GOTO 130:REM PARA LINHA QUE NOMEIA ARQUIVO A SER SALVO
50150  GOTO 140:REM PARA LINHA SEGUINTE QUE NOMEIA ARQUIVO A
       SER SALVO
50160  REM GOTO XXX:REM PARA LINHA QUE DETERMINA ARQUIVO A
       SER CARREGADO
50170  IF ER=77 THEN PRINT "ERRO FORA DE MEMORIA"
50180  POKE 216,0:RESUME
```

Na rotina acima, a maioria dos códigos de erro DOS são testados e mensagens apropriadas são emitidas. Os códigos de erro 5, 6 e 13 estão incluídos aqui para uso posterior. Eles tratam de erros provenientes da tentativa de carregar arquivos e seu uso se tornará evidente mais adiante. Por ora, uma instrução REM é usada na linha 50160 que eventualmente terá de ser substituída. Outros erros, tais como os de código 5, 6, 7 e 8, podem ser provocados ao se tentar salvar ou carregar um arquivo. Por este motivo, um teste é incluído na linha 50130 a fim de distinguir os dois casos. Foi assumido que LD=1.

A última linha da rotina acima é muito importante, visto que ela introduz um zero (0) na posição de memória 216, o que limpa o sinal (flag) de erro criado pela execução do comando ONERR GOTO. Este sinal (flag) deve ser limpado, do contrário as mensagens de erro normais não serão emitidas.

Para você poder ver como funciona esta rotina, salve o programa sob o nome de arquivo CRIAÇÃO DE ARQUIVO RANDÔMICO COM ERR e rode-o enquanto cria erros para chamá-lo.

A opção de parâmetro de volume

Um número de volume pode ser desligado a cada disquete durante o processo de iniciali-

zação. Para designar um número de volume 9 a um disquete em branco, digite

```
INIT HELLO,V9
```

O número que segue V deve ser um inteiro na faixa 1 a 254. Todos os comandos DOS podem ser usados em uma opção de volume que pode proteger o seu disquete de ser regravado por engano. Se VO é especificado, o DOS irá ignorar o volume do disquete e se comportará como se uma opção de volume não tivesse sido dada. Se a opção V não for dada ao inicializar o disquete, o DOS atribui por sua conta um volume de número 254, dito volume "default" *.

A opção de volume pode ser muito útil em sistemas baseados em disco, onde semanalmente ou mensalmente são inseridas informações em disquetes diferentes, cada qual com um número de volume distinto. Se o volume do disquete não corresponder ao da semana ou mês ao qual foi designado, ao se tentar lê-lo ou gravá-lo a mensagem de erro VOLUME MISMATCH (volume não confere) será emitida.

Uso de sub-rotina do monitor

Já foi mencionado anteriormente neste livro que várias sub-rotinas em linguagem de máquina, conhecidas em conjunto como MONITOR, então disponíveis ao usuário ao ligar o computador, uma vez que residem em ROM. As mais fáceis de se demonstrar são as sub-rotinas que apresentam caracteres na tela, pois o resultado é visual. Uma lista das posições de entrada a estas sub-rotinas é dada na Tabela 7.2.

O comando CALL pode ser usado tanto em modo imediato como em um programa. Tente usá-lo em modo imediato, tendo antes preenchido a tela com algumas inscrições (catalogue o diretório do disquete), e em seguida use os comandos de escape de código para mover o cursor para uma posição adequada na tela.

TABELA 7.2 Sub-rotinas do monitor afetando textos

Comando	*Ação*
CALL –936	Limpa a tela e desloca o cursor para o extremo superior esquerdo da mesma.
CALL –958	Limpa tudo que estiver à direita e abaixo do cursor.
CALL –868	Limpa os caracteres que estiverem à direita e na mesma linha do cursor.
CALL –922	Muda para a linha seguinte.
CALL –912	Sobe o texto uma linha.

Para ilustrar como estas sub-rotinas podem ser incorporadas a um programa BASIC, carregue o programa CRIAÇÃO DE UM ARQUIVO RANDÔMICO COM ERR e rode-o. Note que, enquanto você digita um registro, a tecla flecha esquerda desloca o cursor para

* Vide nota da tradução.

a esquerda mas a flecha direita está inativa. Além disso, se você tivesse que dar retrocesso sobre informações já digitadas, apertar a tecla RETURN não eliminará as letras à direita do cursor. Ao IMPRIMIR o string apropriado A$, você verá que a posição do cursor no instante em que apertou RETURN não exerceu nenhuma influência no comprimento original do registro quando este apareceu na tela.

Edição de entrada da tecla sob BASIC

As seguintes alterações e/ou inclusões ao programa CRIAÇÃO DE UM ARQUIVO RANDÔMICO EM ERR irá permitir o uso da flecha esquerda e da flecha direita para que a edição dos registros, que estão sendo digitados durante a execução de um programa BASIC, possa ser facilmente obtida.

```
   25  LA$=CHR$(8):RA$=CHR$(21)
   50  FOR N=1 TO NE:PRINT:A$="":B$=""
   80  IF Z$=CR$ THEN CALL -868:GOTO 100
   82  IF Z$=LA$ GOTO 3000
   84  IF Z$=RA$ GOTO 4000
   90  A$(N)=A$(N)+Z$:PRINT Z$;
   92  LB=LEN(B$):IF LB>1 THEN B$=LEFT$(B$, LB-1):GOTO 70
   94  B$="":GOTO 70
 3000  REM ROTINA DE EDICAO P/FLECHA ESQUERDA
 3010  LA=LEN(A$(N):IF LA<1 THEN A$(N)="":GOTO 60
 3020  B$=B$+RIGHT$(A$(N),1):LB=LEN(B$)
 3030  IF LA>1 THEN A$(N)=LEFT$(A$(N), LA-1):PRINT Z$;:GOTO 70
 3040  A$(N)="":PRINT Z$;:GOTO 70
 4000  REM ROTINA DE EDICAO P/FLECHA DIREITA
 4010  LB=LEN(B$):IF LB<1 THEN B$="":GOTO 70
 4020  RB$=RIGHT$(B$,1):A$(N)=A$(N)+RB$:PRINT RB$;
 4030  IF LB>1 THEN B$=LEFT$(B$, LB-1):GOTO 70
 4040  B$="":GOTO 70
```

Salve este programa sob o nome de ARQUIVO RANDÔMICO COM ERR & EDIT e depois rode-o e teste os seus recursos de edição. A lógica desta rotina de edição é explicada abaixo; a Fig. 7.1 irá ajudar no entendimento do processo.

À medida que caracteres são digitados no teclado, eles não somente são expostos na tela, mas armazenados caractere por caractere em um array de string A$. O posicionamento do cursor é mostrado na Fig. 7.1 por dois parênteses envolvendo um espaço ou uma letra.

Pressionando-se a tecla flecha esquerda, o controle é desviado para a linha 3000 onde o caractere sob o cursor é armazenado em B$ e simultaneamente o conteúdo de A$ é reduzido deste caractere. A cada retrocesso sucessivo o string B$ aumenta e A$ diminui.

Pressionando-se a tecla flecha direita, o controle é desviado para a linha 4000 onde o caractere mais à direita contido em B$ é adicionado ao conteúdo de A$ e simultaneamente extraído do conteúdo de B$.

As linhas 92 e 94 asseguram que B$ é diminuído em um caractere caso você redigite um caractere para substituir um existente na tela, enquanto que ao mesmo tempo A$ é aumentado na linha 90 pelo novo caractere.

A linha 80 chama uma sub-rotina do monitor que apaga todos os caracteres na mesma linha e à direita do cursor.

Output na tela	Conteúdo de B$	Conteúdo de A$
A B C( )		A B C
A B(C)	C	A B
A(B)C	C B	A
(A)B C	C B A	
A(B)C	C B	A
A X(C)	C	A X
A X C( )		A X C

Fig. 7.1. Representação visual do processo de edição.

Problema 7.1

Modifique o programa RANDÔMICO EM ERR+EDIT para que todas as instruções relacionadas aos recursos de edição (linha 60-94, 3000-3040 e 4000-4040) sejam juntadas em uma sub-rotina que comece na linha 20000. Desta forma, a sub-rotina resultante passa a ser uma rotina de "entradas pelo teclado" que pode ser usada não somente para novos registros, mas para editar registros já existentes ou adicionar novos a um arquivo existente. Salve o programa resultante sob o nome de arquivo RANDÔMICO COM EDIT SUB.

A sub-rotina GETLN

O monitor contém uma sub-rotina chamada GETLN (GET LINE-pegar linha) que, sob condições normais, quase substitui a rotina 'entrada pelo teclado' desenvolvida acima. Para efetuar as modificações necessárias, carregue (delete) todas as linhas, a partir da linha da sub-rotina 'entrada pelo teclado', em seguida digite as seguintes linhas:

```
50  FOR N=1 TO NE:PRINT N;">";
60  CALL 64874:GOSUB 21000:A$(N)=SA$+CHR$(13)
21000  REM PEEK CONTEUDO DE ARQ. BUFFER
21010  REM I/O PARAMETERS: OUTPUT – SA$
21020  SA$="":FOR K=512 TO 768
21030  X=PEEK(K):IF X=141 THEN RETURN
21040  SA$=SA$+CHR$(X)
21050  NEXT K
```

O comando CALL 64874 na linha 60 envia o programa à posição de início da sub-rotina do monitor GETLN, que lhe permite digitar qualquer caractere no teclado que simultaneamente é exibido na tela e armazenado em um arquivo buffer de 256 bytes. Assim, o comprimento de cada registro é limitado a 256 bytes, que é o tamanho do arquivo buffer. Tão logo a tecla RETURN seja pressionada, o programa retorna da sub-rotina GETLN para o seu programa.

A instrução GOSUB na linha 60 desvia a execução do programa para a linha 21000 que retira (PEEK) o conteúdo do buffer, byte por byte, e o coloca em SA$. O retorno da sub-rotina ocorrerá somente quando o conteúdo de um dado byte seja igual ao código ASCII para a tecla RETURN (neste caso 141, que representa "volta do carro' dentro da faixa 96 a 256 do código da conversão ASCII). Os 256 bytes do buffer usados pela GETLN começam na posição de memória 512.

A sub-rotina GETLN permite o uso das teclas flecha esquerda e flecha direita, assim como o uso do comando de edição 'CTRL'X. Contudo, usar a flecha direita sem antes ter digitado qualquer outro caractere no teclado irá produzir um registro em branco ao pressionar a tecla RETURN.

Mais adiante, o uso da flecha esquerda em GETLN, para retornar o cursor para o início de uma entrada, poderá dar um salto para a linha seguinte, caso a primeira posição seja batida mais de uma vez. Mudança para nova linha também acontece quando se usa o comando 'CTRL'X, o que significa que a sub-rotina GETLN pode causar problema no caso de entradas em telas formatadas (i.e., quando entrar com informações de comprimentos predeterminados em pontos específicos da tela). Contudo, a sub-rotina GETLN pode ser usada (com cuidado) para entrar informações pelo teclado e como tal você deve salvar este programa sob o nome de arquivo RANDÔMICO COM GETLN.

O monitor possui uma quantidade de sub-rotinas que podem ser usadas por um programador ousado, mas como o seu uso está além do objetivo deste livro, os leitores interessados devem consultar o manual de referências do Apple II para mais informações.

Formatação da tela de janelas

O comando POKE do BASIC pode ser usado para entrar parâmetros, em posições específicas da memória, que controlam a altura e largura da tela para que ao entrar com informações pelo teclado estas possam ser limitadas em determinados campos (janelas). Normalmente a tela pode exibir 24 linhas de 40 caracteres cada.

Uma lista das posições de memória em que parâmetros específicos devem ser colocados (poke) para delimitar a tela é dada na Tabela 7.3.

TABELA 7.3 Localizações de memória relacionadas à formatação da tela

Posição	*Parâmetro*	*Faixa*	*Ação*
32	LM	0-39	Estabelece margem esquerda
33	CL	1-40	Estabelece caracteres por linha
34	TM	0-23	Estabelece margem superior
35	BM	1-24	Estabelece margem inferior

ATENÇÃO: Erros causados por extensão fora da faixa

Os valores para os parâmetros acima não devem exceder a extensão dada. Estabelecer a margem esquerda não reduz automaticamente o número de caracteres por linha, o que pode estragar o seu programa se você tentar entrar com informações fora da janela estabelecida na tela. Para evitar este tipo de problema, reduza primeiro o número total de caracteres por linha e depois estabeleça a margem.

O programa seguinte irá abrir uma janela de uma única linha no centro da tela com o número de caracteres previamente determinado em 26. Para possibilitar uma verificação do comprimento do registro, o campo de uma única linha é exibido, em modo inverso (caracteres pretos em fundo branco).

```
10 REM ESTABELECENDO UMA JANELA
20 HOME:L$=""
```

```
30 FOR I=1 TO 5:HTAB 6:VTAB 13:PRINT I;">";
40 HTAB 8:VTAB 13:INVERSE:PRINT L$
50 POKE 35,13:POKE 34,12:POKE 33,26:POKE 32,7
60 HTAB 1:VTAB 13:INPUT "";A$(I)
70 POKE 32,0:POKE 33,40:POKE 34,0:POKE 35,24
80 NORMAL:PRINT A$(I):CALL-912
90 NEXT I:END
```

O programa inicialmente fixa 26 espaços em branco para string L$ para que, quando L$ for exibido em modo inverso (linha 40), uma linha branca de 26 caracteres apareça na tela, iniciando em uma posição específica determinada pelos comandos HTAB e VTAB da linha 40. Na linha 50, a janela é reduzida a uma linha de 26 caracteres, os comandos POKE são escritos em ordem descendente de posição de memória para que os números máximos de caracteres por linha sejam determinados antes de se estabelecer a margem esquerda.

A linha 60 nos permite introduzir (INPUT) informações em cada string do array A$. O HTAB e o VTAB desta linha são necessários para posicionar o cursor no início da janela. Poderíamos usar o comando CALL-936 para entrar na sub-rotina do monitor que posiciona o cursor na posição superior mais à esquerda da janela (neste caso, no começo da linha), mas infelizmente esta sub-rotina também limpa a tela, o que apagaria os brancos de L$ exibidos em modo INVERSO.

A linha 70 restabelece a tela ao tamanho normal. Aqui usamos o comando POKE em ordem ascendente de posições de memória, o que evita que o número total de caracteres ultrapasse os limites da tela. A linha 80 restabelece o modo de escrita NORMAL (caracteres em branco em fundo preto), imprime o conteúdo de A$ e entra em uma sub-rotina do monitor (com CALL-912), que desloca todo o texto uma linha para cima.

Rode este programa, salvando-o antes sob o nome de arquivo "estabelecer janela". Note que agora a tecla flecha esquerda e o comando 'CTRL'X funcionam corretamente. Existem, contudo, problemas com a tecla flecha direita, assim como para se digitar uma entrada que exceda o comprimento de registro especificado.

Problema 7.2

Incorpore a rotina de estabelecimento de janela acima ao programa RANDÔMICO COM GETLN que irá possibilitar entradas com vírgulas, e ao mesmo tempo corrigirá os efeitos indesejáveis do comando 'CTRL'X e de rebatida no começo de uma entrada com o uso da tecla flecha esquerda.

Carga de um programa em linguagem de máquina

O comando POKE pode ser usado para carregar um programa em linguagem de máquina em uma parte da memória especialmente separada para carga de programas de usuários. Independentemente da usada para carregar os programas do usuário. Consideremos que tais programas em linguagem de máquina já existam e nos preocuparemos somente em mostrar como usar o comando POKE para carregá-los na memória e em seguida chamá-los de dentro do BASIC.

O manual DOS do Apple II contém um programa em linguagem de máquina que, uma vez chamado (POKE), ativa uma sub-rotina que começa na posição de memória 768.

Esta sub-rotina elimina alguns dos problemas causados por ONERR GOTO, que podem ocorrer se uma rotina de manipulação de erros, que contenha o comando RESUME, for usada para manipular erros em uma instrução GET.

Para ilustrar como um programa em linguagem de máquina, mencionado acima, deve ser colocado (POKE) na memória e logo após chamado (CALL), carregue qualquer um dos últimos programas CRIAÇÃO RANDÔMICA que contenha a rotina de manipulação de erros já desenvolvida, e depois adicione as seguintes instruções nas linhas de programa que precedem a linha ONERR GOTO. Por exemplo, digite:

```
11  FOR I=0 TO 9:READ K:POKE 768+I,K:NEST I
12  DATA 104,168,104,166,223,154,72,152,72,96
```

depois modifique a primeira linha executável da rotina de manipulação de erros para:

```
50010  ER=PEEK(222):CALL 768
```

Tais sub-rotinas em linguagem de máquina devem ser colocadas (POKE) na memória toda vez que um programa for rodado, e só poderá ser usada pelo mesmo se fizer parte dele.

Como usar arquivos binários

Programas em linguagem de máquina grandes, geralmente escritos com auxílio de outro programa chamado ASSEMBLER, são armazenados no disquete como arquivos binários. A letra B precede o seu nome de arquivo no diretório do disquete para que possa ser diferenciado dos demais arquivos.

Arquivos binários podem ser carregados (BLOAD) de um disquete em que foram salvos (BSAVE), e em seguida chamados (CALL) para dar início à execução do programa, tanto em modo imediato como a partir do BASIC.

Em modo imediato, digite

```
BLOAD 'nome de arquivo'
CALL 'endereco'
```

onde 'nome arquivo' é o nome de arquivo binário no qual o programa em linguagem de máquina foi salvo (BSAVE), e 'endereço' o local de memória em que a execução deve iniciar. Isto significa que o usuário deve saber de memória este endereço específico. Se este endereço não for sabido, o programa ainda pode ser executado digitando-se

```
BRUN 'nome de arquivo'
```

que inicialmente irá carregar o programa chamado e depois executá-lo a partir da primeira posição de memória na qual foi originalmente salvo (BSAVE).

Para poder usar um programa em linguagem de máquina, a partir de um programa BASIC, o endereço de partida para a execução deve ser conhecido. Supondo que tenhamos um arquivo chamado TEST, que começa na posição 777, então a seguinte instrução será requerida:

```
100  D$=CHR$(4):PRINT D$,"BLOAD TEST":CALL 777
```

Um arquivo binário só pode ser salvo (BSAVE) em um disquete se a posição inicial de memória assim como o seu comprimento (em bytes) sejam conhecidos. Igualmente, para ilustrar o comando BSAVE, suponha que o comprimento do programa TEST acima seja de 99 bytes. Para transferir tal arquivo para um outro disquete, coloque na unidade o disquete que contém o arquivo armazenado e digite:

```
BLOAD TEST
```

depois substitua este disquete pelo outro em que se deseja gravar o arquivo e digite

```
BSAVE TEST,A777,L99
```

Você verá que algumas vezes tanto o endereço (A) como o comprimento (L) são dados no sistema hexadecimal (na base 16). Em tais casos, tanto A como L aparecem como A$ e L$ para serem distinguidos do sistema decimal de numeração. Um exemplo disto seria: BSAVE TEST,A$800,L$9E.

A notação hexadecimal utiliza os dígitos 0-9 para representar os números hexadecimais de 0 a 9, e as seis primeiras letras do alfabeto (A-F) para representar os números hexadecimais de 10 a 15. Qualquer valor entre 0 e 225 pode ser representado por um par de dígitos hexadecimais, enquanto um grupo de quatro dígitos pode denotar um número entre 0 e 65.535, que é o número total de locais na memória de seu computador. Este conceito será discutido mais adiante no Capítulo 8.

Programas utilitários em linguagem de máquina

O "disquete mestre" DOS contém vários programas úteis em linguagem de máquina, entre eles um programa chamado FID. Este programa é capaz de transferir qualquer tipo de arquivo, incluindo a si mesmo, de um disquete para outro, desde que o disquete destinado já tenha sido inicializado. O programa está igualmente apto a ser usado com uma ou duas unidades de disco. Primeiro tente transferir FID do 'disquete mestre' para o seu disquete de trabalho e depois use-o para transferir o programa em linguagem de máquina CHAIN.

O programa CHAIN é muito útil se você desejar encadear dois programas de tal forma que o primeiro seja executado, e em seguida o segundo seja carregado e executado automaticamente, sem apagar os valores correntes armazenados na memória como resultados do primeiro.

Para ilustrar como se encadeiam programas, carregue o programa RECUPERAÇÃO DE ARQUIVO RANDÔMICO e troque a variável de nome A$ para RA$ nas linhas 70, 150 e 180. Salve então o programa sob o nome de ARQUIVO RECUPERAR. Agora carregue o programa RANDÔMICO COM EDIT SUB (se você não tiver este programa, o RANDÔMICO COM ERR+EDIT também serve) e digite as seguintes linhas:

```
210 PRINT "DONE"
220 PRINT D$;"BLOAD CHAIN,A520"
230 CALL 520"RETRIEVE"
```

e salve-o sob o nome de arquivo CRIAR/RET CAM ENCADEAMENTO.

Note que as linhas 220 e 230 devem ser consecutivas, sem qualquer instrução entre si. Além disso, não deve haver um espaço entre o número 520 na instrução CALL e as aspas na linha 230. Se tudo estiver em ordem e se todos os três programas estiverem no mesmo disquete, digite RUN para ver a execução do programa.

Para ilustrar que os valores das variáveis são encadeados para o programa seguinte, você pode apagar as linhas 20 e 30 do programa RECUPERAÇÃO. Não se esqueça de salvar o programa alterado sob o mesmo nome de arquivo (RECUPERAÇÃO) antes de rodar novamente o programa CRIAR/RET COM ENCADEAMENTO.

Algumas linguagens de máquina solicitam que o endereço de partida (início) seja especificado quando carregadas (BLOAD) a partir do BASIC, como é o caso do programa ENCADEAMENTO. Isto é necessário, pois tais programas podem ter sido originalmente salvos (BSAVE) no disquete com um endereço de partida diferente. Se o endereço da par-

tida não for especificado ao carregar (BLOAD) um programa, este será carregado na memória a partir do endereço usado originalmente para gravá-lo (BSAVE).

Às vezes é necessário saber o comprimento de um programa em linguagem de máquina para que possa ser carregado (BLOAD) na memória do computador em um endereço de memória diferente do qual ele foi gravado (BSAVE). Isto pressupõe, é claro, que desvios em programas em linguagem de máquina só se referem às posições de memória relativas ao endereço de partida. O programa abaixo, quando rodado, irá carregar (BLOAD) um programa de linguagem de máquina especificado e em seguida chamará (PEEK) as posições de memória adequadas (dadas na linha 90), primeiro para o endereço de partida, SA, e depois para o comprimento do programa, LE. O programa pode operar em três tipos de RAM, as de 16, 32 e 48 bytes. O programa inicialmente pede o tamanho da RAM de seu computador.

```
10 REM LOCALIZADOR DE ENDERECO BSAVE
20 HOME:INPUT "TAMANHO DO SISTEMA EM KB? (16/32/48) ";
     V$:V=VAL(V$)
30 FOR I=1 TO 3:READ S:IF S<>V THEN NEXT I: RESTORE:GOTO 20
40 FOR J=1 TO I:READ SA,LE:NEXT J
50 PRINT "":INPUT "QUE ARQUIVO BINARIO? ";X$: D$=CHR$(4):IF
     X$="" THEN PRINT D$; "CATALOG": GOTO 50
60 PRINT D$;"BLOAD";X$
70 PRINT "":PRINT X$;",A"; PEEK(SA)+PEEK(SA+1)*256;",L";
     PEEK(LE)+PEEK(LE+1)*256: END
80 DATA 16,32,48
90 DATA 10866,10848,27250,27232,43634,43616
```

Digite o programa acima e salve-o sob o nome de arquivo BSALVADOR (ou BSAVER). Para poder testá-lo, digite RUN e insira o "disquete mestre" do DOS no drive e digite BOOT13, que é o nome do arquivo binário em seu disquete. O arquivo binário será carregado automaticamente e a seguinte mensagem aparecerá na tela

```
BOOT13,A5888,L2288
```

Você pode usar esta informação para salvar (BASAVE) o programa BOOT13, em qualquer outro disquete.

Analogamente, você pode usar o programa BSALVADOR para encontrar o endereço de partida e o comprimento do arquivo binário COPY-OBJ0, que também está no seu disquete mestre do DOS. O resultado será:

```
COPY.OBJ0,A704,L267
```

Para saber o comprimento do encadeamento, você deverá alterar a linha 60 para

```
60 PRINT D$;"BLOAD";X$;",A520"
```

do contrário, uma mensagem de erro será emitida pois o programa deve ser carregado a partir do endereço 520 e não a partir do qual foi originalmente salvo. Se agora você rodar o programa, receberá a mensagem:

```
CHAIN,A520,L456
```

CUIDADO: Se você pretende salvar o programa ENCADEAMENTO em outro disquete, deve usar o endereço de partida A2056, e não o dado acima. O endereço de partida A520 só deve ser usado para carregar o programa ENCADEAMENTO, como discutido anterior-

mente. O programa BSALVADOR não funciona nem com os programas FID nem com o MASTER CREATE, que aparecem no disquete mestre de seu sistema.

CAPÍTULO 8

Criação de uma seqüência de programas

O programa Menu

Um menu é um programa que mostra as diferentes opções disponíveis ao usuário em uma seqüência de programas. Ao selecionar uma opção, o usuário faz com que o programa salte para o ponto de partida da opção escolhida e, ao final da execução, ele retorna o controle do programa ao menu.

O programa seguinte servirá como um menu para criar, recuperar e atualizar arquivos de acesso randômico

```
 10 HOME:VTAB 5:PRINT "***CRIACAO/RECUPERACAO/ATUALIZACAO
    DE ARQ. RANDOMICOS
    FILES***"
 80 CM$="PRESS <RETURN> TO CONTINUE"
 90 VTAB 23:PRINT CM$:GET Q$
100 HOME:REM *****MENU*****
110 HTAB 15:VTAB 8:PRINT "M E N U"
120 HTAB 15:PRINT " _ ":PRINT
130 HTAB 5:PRINT "1.  CRIAR NOVO ARQUIVO RANDOMICO"
140 HTAB 5:PRINT "2.  RECUPERAR ARQUIVO RANDOMICO"
150 HTAB 5:PRINT "3.  ATUALIZAR ARQUIVO RANDOMICO"
160 HTAB 5:PRINT "4.  ENCERRAR PROGRAMA RANDOMICO"
270 VTAB 21:INPUT"CHOOSE(1/2/3/4/)";M$
280 M=VAL(M$):IF M<1 OR M>4 GOTO 270
290 ON M GOTO 1000,2000,3000,300
300 END
1000 REM *****CRIAR NOVO ARQUIVO RANDOMICO******
1010 VTAB 23:PRINT "CRIAR NOVOS ARQUIVOS":PRINT:PRINT
    CM$:GET
    Q$:GOTO 100
2000 REM *****RECUPERAR ARQUIVO RANDOMICO*****
2010 VTAB 23:PRINT "RECUPERAR ARQUIVOS":PRINT:PRINT
    CM$:GET Q$:GOTO 100
3000 REM*****ATUALIZAR ARQUIVO RANDOMICO*****
3010 VTAB 23:PRINT "ATUALIZAR ARQUIVOS":PRINT:PRINT CM$:GET
    Q$:GOTO 100
```

Digite este programa adotando o mesmo número de linhas usado acima, visto que mais linhas serão adicionadas brevemente. Salve o programa sob o nome de arquivo MENU e rode-o. As instruções temporárias nas linhas 1010, 2010 e 3010 só mostrarão o fluxo da execução do programa. A instrução GET Q$ funciona simplesmente como uma pausa. Ao

pressionar qualquer tecla (exceto RESET), a próxima instrução, no caso GOTO 100, será executada, o que devolve o controle do programa ao MENU.

A fim de se construir a seqüência de programas desejada, devemos ou redigitar todas as partes individuais (neste caso os programas CRIAR ARQUIVOS RANDÔMICOS, RECUPERAR ARQUIVOS RANDÔMICOS e ATUALIZAR ARQUIVOS RANDÔMICOS), um trabalho árduo, ou recorrer a qualquer outro meio pelo qual tais programas (que já estão salvos em disco) possam ser grupados. Adotaremos a técnica de transformar porções de programas BASIC em arquivos-texto que possam então ser executados na memória, de forma a serem intercalados com linhas de programação já existentes.

Arquivos executáveis

As linhas de programa seguintes, quando salvadas como um ARQUIVO-TEXTO, servirão para modificar porções de programas BASIC em ARQUIVOS-TEXTO. Para criar este programa como ARQUIVO-TEXTO, carregue e rode o programa CRIAR ARQUIVO SEQ. COM GET e digite as seguintes linhas como entrada de registros.

```
    0  GOTO 63991:REM ***CRIADOR DE ARQ TEXTO***
63991  D$=CHR$(4)
63992  PRINT D$;"OPEN CAP.TEXT"
63993  PRINT D$;"DELETE CAP.TEXT"
63994  PRINT D$;"OPEN CAP.TEXT"
63995  PRINT D$;"WRITE CAP.TEXT"
63996  POKE 33,33:LIST 1,63990
63997  PRINT D$;"CLOSE CAP.TEXT"
63998  PRINT:PRINT "NOW TYPE. . .":PRINT
63999  PRINT "RENAME CAP.TEXT,'FILENAME'":END
```

Salve o ARQUIVO-TEXTO sob o nome de arquivo CRIADOR DE ARQUIVO-TEXTO. Agora limpe a memória do computador digitando NEW e em seguida digite o comando

```
EXEC 'nome do arquivo'
```

onde "nome do arquivo' é neste caso o arquivo CRIADOR DE ARQUIVO-TEXTO. A unidade de disco (DRIVE) irá operar por algum tempo e o arquivo-texto designado será executado na memória. Se você quiser ver na tela o que está acontecendo, digite o comando

```
MON C,I,O
```

que corresponde aos Comandos Monitor, Input e Output ao disquete e execute novamente o arquivo CRIADOR DE ARQUIVO TEXTO. O MON C,I,O permanecerá em funcionamento até que o comando de cancelamento .

```
NOMON C,I,O
```

seja dado.

Ao listar, após executar o arquivo CRIADOR DE ARQUIVO-TEXTO, você verá que existe agora um programa BASIC na memória cujas linhas são o conteúdo do arquivo-texto CRIADOR DE ARQUIVO TEXTO.

Para ilustrar o uso do programa CRIADOR DE ARQUIVO-TEXTO, carregue o programa CRIADOR RANDÔMICO mais recente que contém a rotina de manipulação de erros e depois apague todas as linhas entre 1 e 49999. O que resta agora na memória é a rotina de manipulação de erros que começa na linha 50000. Confira isto e depois digite

```
EXEC TEXT MAKER
```

e quando o cursor retornar a tela, digite RUN.

O programa CRIADOR DE ARQUIVO-TEXTO abre (OPEN) um arquivo chamado CAP.TEXT, em seguida o apaga (para apagar o conteúdo do arquivo deixado por uso anterior deste programa), depois reabre o arquivo e nele escreve (WRITE) todas as linhas de programa entre 1 e 63990. Neste caso, como as únicas instruções dentro desta faixa são aquelas relacionadas à rotina de manipulação de erros, ele transforma esta parte do programa em um arquivo texto. Quando a mensagem

```
NOW TYPE. . . (digite agora)
RENAME*CAP.TEXT. 'nome do arquivo
```

aparecer na tela, redenomine o arquivo CAP.TEXT como ROTINA ERR.

Para intercalar a rotina de manipulação de erros no programa MENU, primeiro carregue o programa MENU e depois digite

```
EXEC ROTINA ERR
```

Ao listar, você verá que os dois programas foram fundidos em um único programa, que poderia ser salvado sob um nome de arquivo apropriado

Problema 8.1

Use o programa CRIADOR DE ARQUIVO-TEXTO para criar um arquivo-texto com as sub-rotinas de entrada pelo teclado (a partir da linha 20000) que resultou da solução do Problema 7.1

Renumeração e intercalação de programas

Um procedimento análogo pode ser seguido para criar arquivos-texto contendo quaisquer programas, mas deve-se ter o cuidado de renumerar tais programas numa seqüência coerente antes de torná-los programas em arquivos-texto. A maneira mais fácil de se renumerar programas é através do uso do programa RENUMBER (renumerar), que é encontrado no 'disquete mestre' do DOS. Primeiro rode o programa RENUMBER INSTRUCTIONS (renumerar instruções) e tome notas de como renumerar programas, particularmente como renumerar apenas uma porção de um dado programa, deixando o restante com a numeração original. Para praticar o uso do programa RENUMBER, carregue-o e rode-o, e quando receber uma mensagem (um "prompt"), carregue o programa MENU e digite

```
& F5,S10,E300,I5
```

e a seguir LISTE todas as linhas entre 0 e 300. Você verá que a primeira linha do programa (com F de "FIRST") começa agora na linha 5 (F5 no comando), e todas as demais linhas entre 10 e 300 do programa antigo (começando –S de "START" – em 10 e terminando –E de "ENDING" – em 300), foram renumeradas com um 'I'ncremento de 5. Note também que as linhas 1000-3010 mantiveram a numeração original, mas os números de linha nas instruções GOTO das linhas 1010, 2010 e 3010 foram modificados apropriadamente. Não salve esta versão do programa MENU visto que ele foi renumerado unicamente para fins de demonstração.

Agora estamos prontos para começarmos a intercalar o programa MENU com o programa RANDÔMICO COM EDIT SUB. Porém, precisamos primeiro renumerar este últi-

* RENAME (coloque outro nome para).

mo para que ele comece na linha 1000 com a sub-rotina de entradas por teclado e a rotina de manipulação de erros começando nas linhas 20000 e 50000, respectivamente. Salve o programa intercalado final sob o nome de arquivo MENU/CRIAR. Em seguida, renumere o programa RECUPERAR ARQUIVO RANDÔMICO para que comece na linha 2000 e intercale-o com o programa MENU/CRIAR. Finalmente, digite as instruções das linhas 20-70 e 3000-3060 do programa listado abaixo. Além disso, você terá de modificar algumas linhas, apagar outras e talvez adicionar algumas linhas extras antes de obter um programa idêntico ao dado abaixo. Confira cada linha com a listagem e quando estiver certo de que estão idênticas (renumere parte do programa, se necessário), então, e somente então, salve-o sob o nome de arquivo CR/RE/UP RANDÔMICO 20 e apague do disquete todos os programas BASIC intermediários (não o arquivo-texto) que foram salvados para obter o resultado final.

```
 10 HOME:VTAB 5:PRINT "***CREATE/RETRIEVE/UPDATE RANDOM
    FILES***"
 20 VTAB 10:HTAB 8:PRINT"TO EXIT ENTRY MODE PRESS:
    VTAB 2:HTAB 16:PRINT"<RETURN>"
 30 NE=20:DIM A$(NE)
 40 FOR I=0 TO 0:READ K:POKE 768+I,K:NEXT I
 50 DATA 104,168,104,166,223,154,72,152,72,96
 60 ONERR GOTO 50000
 70 DR=CHR$(4):CR$=CHR$(13):LA$=CHR$(8):
    RA$=CHR$(21):G$=CHR$(7)
 80 CM$="APERTE <RETURN>PARA PROSSEGUIR"
 90 VTAB 23:PRINT CM$:GET Q$
100 HOME:REM *****MENU*****
110 HTAB 15:VTAB 8:PRINT "M E N U"
120 HTAB 15:PRINT "  _ ":PRINT
130 HTAB 5:PRINT "1.   CRIAR NOVOS ARQ. RANDOMICOS"
140 HTAB 5:PRINT "2.   RECUPERAR ARQ RANDOMICO"
150 HTAB 5:PRINT "3.   ATUALIZAR ARQ. RANDOMICO"
160 HTAB 5:PRINT "4.   ENCERRAR PROGRAMA RANDOMICO"
270 VTAB 21:INPUT "CHOOSE (1/2/3/4/) ";M$
280 M=VAL(M$):IF M<1 OR M>5 GOTO 270
290 ON M GOTO 1000,2000,3000,300
300 END
1000 REM *****CRIAR NOVOS ARQUIVOS RANDOMICOS*****
1010 VTAB 23:INPUT "COMPRIMENTO DE REGISTRO MAX? ";RL$
1020 RL=VAL(RL$):IF RL<1 OU RL>256 GOTO 1010
1030 PRINT "NUMERO MAX. DE NOVAS ENTRADAS = ";NE:SL=RL
1040 FOR N=1 TO NE:PRINT:SN=N: GOSUB 20000:A$(N)=SR$
1050 IF A$(N)="" GOTO 1070
1060 NEXT N
1070 N=N-1:LD=0:PRINT:IF M=3 THEN N=N+NN: GOTO 1090
1080 PRINT G$:INPUT "NOME SOB O QUAL DEVE SER SALVADO? ";X$
1090 RN=0
1100 PRINT D$;"OPEN";X$;",L";RL
1110 PRINT D$;"WRITE";X$;",R";RN
```

```
1120  PRINT N:PRINT RL:IF M=3 GOTO 3020
1130  FOR RN=1 TO N:PRINT D$;"WRITE";X$;",R";RN
1140  PRINT A$(RN):NEXT RN
1150  PRINT D$;"CLOSE";X$
1160  PRINT:PRINT CM$:GET Q$:GOTO 100
2000  REM *****RECUPERAR ARQ. RANDOMICO*****
2010  PRINT G$:LD=1:INPUT "NOME DO ARQUIVO A SER CARREGADO?
        ";X$
2020  IFX$="" THEN PRINT D$;"CATALOG":GOTO 2010
2030  PRINT D$;"OPEN";X$
2040  PRINT D$;"READ";X$
2050  INPUT N:PRINT "MAX REGISTROS ";N
2060  INPUT RL:PRINT "COMPRIMENTO ";RL
2070  PRINT D$;"CLOSE";X$
2080  PRINT:IF M=3 GOTO 1030
2090  PRINT D$;"OPEN";X$;",L";RL
2100  FOR RN=1 TO N:AA$=""
2110  PRINT D$:PRINT D$;"READ";X$;",R";RN
2120  GET Z$:IF Z$=CR$ GOTO 2140
2130  AA$=AA$+Z$:GOTO 2120
2140  PRINT" ";RN;">";AA$
2150  NEXT RN
2160  PRINT D$:PRINT D$;"CLOSE";X$
2170  PRINT:PRINT CM$:GET Q$:GOTO 100
3000  REM *****ATUALIZAR ARQUIVO RANDOMICO*****
3010  GOTO 2000
3020  FOR RN=NN+1 TO N
3030  PRINT D$;"WRITE";X$;",R";RN
3040  PRINT A$(RN-NN):NEXT RN
3050  PRINT D$;"CLOSE";X$
3060  PRINT:PRINT CM$:GET Q$:GOTO 100
20000 SR$="":B$="":REM *****SUB-ROTINA DO TECLADO*****I/O
        PAR:
      INPUT SL,SN; OUTPUT SR$. STRINGS DEFINIDOS
      CR$,RA$,LA$ & G$ COM CODIGOS CHR$( ) APROPRIADOS.
20010  K=1:HTAB 1:PRINT SN;">";
20020  GET Z$
20030  IF Z$=CR$ THEN CALL-868:RETURN
20040  IF Z$=RA$ GOTO 20160
20050  IF Z$=LA$ GOTO 20110
20060  IF K=SL THEN PRINT G$;:GOTO 20020
20070  SR$=SR$;Z$:PRINT Z$;:K=K+1
20080  LB=LEN(B$):IF LB>1 THEN B$=LEFT$(B$,LB-1):GOTO
       20020
20090  B$="":GOTO 20020
20100  REM EDICAO DA TECLA DA FLECHA ESQUERDA
20110  LA=LEN(SR$):IF LA<1 THEN SR$="":GOTO 20010
20120  B$=B$+RIGHT$(SR$,1):LB=LEN(B$):K=K-1
```

```
20130  IF LA>1 THEN SR$=LEFT$(SR$,LA-1):PRINT Z$;:GOTO
       20020
20140  SR$="":PRINT Z$;:GOTO 20020
20150  REM EDICAO DE TECLA DA FLECHA DIREITA
20160  LB=LEN(B$):IF LB<1 THEN B$="":GOTO 20020
20170  RB$=RIGHT$(B$,1):SR$=SR$+RB$:PRINT RB$;:K=K+1
20180  IF LB>1 THEN B$=LEFT$(B$,LB-1):GOTO 20020
20190  B$="":GOTO 20020
50000  PRINT G$:REM *****ROTINA DE MANIPULACAO DE ERROS*****
50010  ER=PEEK(222):CALL 768
50020  PRINT "ERROR NR ";ER;" IN LINE ";
       PEEK(218)+PEEK(219)*256
50030  IF ER=4 THEN PRINT "DISCO PROTEGIDO!":GOTO
       50130
50040  IF ER=5 OU ER=6 THEN PRINT "ARQ. NAO ENCONTRADO!":
       PRINT
       D$;"DELETE";X$:GOTO 50130
50050  IF ER=7 THEN PRINT "VOLUME ILEGAL!":GOTO 50130
50060  IF ER=8 THEN PRINT "I/O ERRO":GOTO 50130
50070  IF ER=9 THEN PRINT "DISCO LOTADO!":PRINT
       D$;"DELETE";X$:GOTO 50130
50080  IF ER=10 THEN PRINT"ARQUIVO TRAVADO!":INPUT
       "UNLOCK? ";FU$:IF LEN(FU$)=0 THEN PRINT D$;"CATALOG":
       GOTO 50080
50090  IF LEFT$(FU$,1)="Y" THEN PRINT D$;"UNLOCK";X$:GOTO
       50150
50100  IF LEN(FU$)>0 GOTO 50130
50110  IF ER=11 THEN PRINT "NOME ILEGAL":GOTO 50130
50120  IF ER=13 THEN PRINT "TIPO DE ARQUIVO INCOMPATIVEL!"
50130  IF LD=1 GOTO 50160
50140  GOTO 1080:REM PARA LINHA QUE NOMEIA ARQUIVO A SER
       SALVO
50150  GOTO 1090:REM PARA LINHA SEGUINTE QUE NOMEIA ARQUIVO
       A SER SALVO
50160  GOTO 2010:REM PARA LINHA QUE DETERMINA ARQUIVO A SER
       CARREGADO
50170  IF ERR=77 THEN PRINT "ERRO FORA DE MEMÓRIA"
50180  POKE 216,0:RESUME
```

O programa acima usa um único array de strings de dimensão 20 (estabelecido na linha 30) para restringir o tamanho da memória por ele utilizado. Isto significa que o programa só permitirá 20 novas entradas de cada vez. Se você precisar entrar mais, escolha a opção 3 do MENU tantas vezes quanto necessário.

Como resultado da limitação do tamanho do array de strings A$, a seção 'Recuperar' do programa teve de ser modificada para que agora registros recuperados sejam impressos na tela, um registro completo de cada vez. Note também que como a instrução GET (linha 2120) está sendo usada para acumular todos os caracteres de cada registro em AA$, a instrução PRINT " " da linha 2140 é necessária, do contrário o primeiro caractere de cada registro não será impresso.

Finalmente, um contador K foi incorporado à sub-rotina "entrada por teclado" para saber o número total de caracteres à esquerda do cursor na tela. Quando K for igual ou maior que SL-1, onde SL é o comprimento do registro, a campainha soará. O último caractere de cada registro deve ser o código ASCII para a tecla RETURN. Rode o programa usando todas as suas opções.

Problema 8.2

Modifique a porção 'recuperar' da seqüência de programas acima, para que os registros recuperados sejam armazenados no array A$(N), com o valor de N não excedendo 20. Se o número máximo de registros contidos em um arquivo for maior que 20, o seu programa deverá ler os 20 primeiros em A$( ), imprimidos na tela, e ao pressionar a tecla RETURN ler e imprimir os próximos 20 registros, e assim por diante.

Pesquisa e edição de registros

Da forma como o programa CR/RE/UP RANDÔMICO 20 se encontra no presente, após registros terem entrado num arquivo, não poderão mais ser editados. Isto é um inconveniente que precisa ser corrigido.

Contudo, antes de continuar com esta seção, você deve ter resolvido o Problema 8.2, visto que a 'rotina de edição' (linhas 4000-4220 abaixo) depende dele. As linhas 2080 a 2168 estão relacionadas à solução do Problema 8.2, assim como outras inclusões necessárias.

A 'rotina de edição' a ser incorporada ao arquivo CR/RE/UP RANDÔMICO 20 usa os números de linha 4000 a 4220 e está listada abaixo. Mas, primeiro, listamos as modificações que devem ser feitas à parte do MENU do programa.

```
 160  HTAB 5:PRINT "4.  "EDITAR ARQUIVOS RANDOMICOS ANTIGOS"
 170  HTAB 5:PRINT "5.  ENCERRAR PROGRAMA"
 270  VTAB 21:INPUT "CHOOSE (1/2/3/4/5) ";M$
 280  M=VAL(M$):IF M<1 OU M>5 GOTO 270
 290  ON M GOTO 1000,2000,3000,4000,300
2080  PRINT:IF M=3 THEN NN=N:GOTO 1030
2082  IF M=4 THEN INPUT "EDIT QUE REC NR? ";B$:GOTO 2088
2085  INPUT "RETRIEVE FROM QUE REC NR? ";B$
2088  B=VAL(B$):IF B<1 OU B>N THEN PRINT "NAO PODE SER FEITO"
      :GOTO 2082
2095  E=B+NE-1:IF E>N THEN E=N
2100  FOR RN=B TO E:I=RN-B+1:A$(I)=""
2130  A$(I)=A$(I)+Z$:GOTO 2120
2140  PRINT "";RN;">";A$(I):IF M=4 THEN VTAB 23:GOTO 2160
2160  PRINT D$:PRINT D$;"CLOSE";X$: IF M=4 GOTO 4020
2162  IF N=RN-1 THEN PRINT "END OF FILE": GOTO 2170
2164  INPUT "CONTINUE LISTING? (Y/N) ";AA$: IF AA$="N" GOTO
      2170
2168  B=E+1:IF B<=N GOTO 2090
4000  REM *****EDITAR ARQ RANDOMICOS ANTIGOS*****
4010  GOTO 2000
```

```
4020 B$="":SR$="":SN=RN:SL=RL:L2=LEN(A$(I))
4030 FOR L1=1 TO L2:L3=L2-L1+1:B$=B$+MID$(A$(I),L3,1):NEXT L1
4040 GOSUB 20010:A$(I)=SR$
4050 DE=0:IF LEN(A$(I))=0 AND RN=N THEN DE=1:PRINT:
     INPUT "APAGAR ESTE REGISTRO? (Y/N) ";AA$:IF AA$="N"
     GOTO 4210
4060 IF DE<>1 GOTO 4130
4070 PRINT D$:PRINT D$;"OPEN";X$;",L";RL
4080 PRINT D$;"WRITE";X$;",R";0
4090 PRINT N-1:PRINT RL
4100 PRINT D$;"WRITE";X$;",R";RN
4110 PRINT
4120 PRINT D$;"CLOSE",X$:GOTO 4200
4130 BL=0:IF LEN(A$(I))=0 THEN BL=1:PRINT:INPUT "APAGAR ESTE
     REGISTRO? (Y/N) ";AA$:IF AA$="N" GOTO 4210
4140 PRINT D$:PRINT D$;"OPEN";X$;",L";RL
4150 PRINT D$;"WRITE";X$;",R";RN
4160 IF BL=0 GOTO 4180
4170 PRINT " ":GOTO 4190
4180 PRINT A$(I)
4190 PRINT D$;"CLOSE";X$
4200 PRINT "ARQ. ATUALIZADO":GOTO 4220
4210 PRINT "ARQ. INALTERADO"
4220 PRINT:PRINT CM$:GET Q$:GOTO 100
```

Ao escolher a ação A do MENU, o controle do programa é desviado para a seção 'recuperar' do programa para poder recuperar e dispor na tela os registros apropriados a serem editados. A linha 4020 inicializa a sub-rotina 20000 para que a edição do registro escolhido possa ser efetuada. Note que na linha 4030, o conteúdo de A$(I), que é o registro desejado, é inicialmente armazenado, de detrás para a frente, em B$ para que a tecla flecha direita possa ser usada para copiar entradas existentes. A sub-rotina 'entrada por teclado' é entrada na linha 20010.

A rotina de 'edição' permite que você apague um registro, se este for o último registro do arquivo, ou coloque brancos em qualquer outro registro que não seja o último. Essas duas alternativas podem ser efetuadas automaticamente ao se pressionar a tecla RETURN sem copiar qualquer parte do registro exibido na tela.

Se a chave (flag) DE é colocada em 1, os registros são apagados subtraindo-se 1 ao seu número no arquivo. Inicialmente o valor N-1 é escrito em um registro de número zero (linha 4090), e depois 'nada' é impresso no último registro. Se a chave BL for colocada em 1, os registros são apagados simplesmente imprimindo um 'espaço' neles (linha 4170). Isto permite a reutilização destes registros vazios. Todos os outros registros editados são recopiados nos lugares adequados do arquivo (linha 4180).

Digite estas inclusões, rode o programa, e salve-o sob o nome de arquivo EDIÇÃO DE ARQUIVOS RANDÔMICOS.

Finalmente, note que quando um registro é apagado de um arquivo, não só o valor de N-1 é reimpresso no registro número zero, mas também o 'comprimento do registro', RL (linha 4090 do programa). Isto é necessário, pois podemos escrever números em um arquivo tanto convertendo o número em um string e depois imprimindo o string no arqui-

vo (caso em que cada número será armazenado em seu código equivalente ASCII), como simplesmente armazenando o número em seu formato de ponto flutuante, caso em que o número de bytes necessários para caber tal número pode variar de acordo com o seu valor.

Em ambos os casos, não podemos saber exatamente em que byte, em um dado registro, um determinado número termina e outro começa, especialmente como em nosso exemplo em que o número de registros pode mudar de um instante para o seguinte à medida que adicionamos ou apagamos registros do arquivo. Se, por exemplo, o número de registros aumentar de 9 para 10, um número de um dígito RL será reimpresso também, parte dele será impresso por cima ("acavalado").

Problema 8.3

Modifique o programa EDIÇÃO DE ARQUIVO RANDÔMICO incluindo-lhe mais um recurso (na opção 2 do MENU), para que todos os registros mantidos em um dado arquivo possam tanto ser exibidos na tela como impressos no papel. Use o programa para criar um arquivo com no mínimo 25 registros a fim de testar todos os aspectos do programa, inclusive o recurso de impressão no papel. Salve o programa resultante sob o nome de arquivo IMPRIMIR RANDÔMICO.

Classificação de grandes arquivos

Existem várias técnicas para classificar alfabeticamente registros que estão armazenados em disquete. Muitas destas movem os registros fisicamente, o que representa consumo de tempo. Estas técnicas usam uma rotina de classificação que compara o i-ésimo registro com o j-ésimo e, se necessário, permuta os dois registros. Existem vários algarismos para escolher qual registro será o i-ésimo e qual será o j-ésimo. Se os dois registros forem adjacentes, temos então a técnica de classificação da bolha, enquanto que, se forem escolhidos de acordo com um algoritmo específico, obedecendo certos critérios, poderemos ter a chamada técnica de classificação da concha. Não obstante, as técnicas que necessitam de movimentos físicos dos registros são muito lentas e ineficientes.

Um método melhor de classificação de grandes arquivos é rotular cada registro com um número que indica a sua posição alfabética no arquivo. Vamos supor que registros foram entrados em um arquivo na seguinte ordem física.

1 SMITH	2 JONES	3 BROWN	4 WILSON

Ao classificar tal arquivo, ao invés de movimentar os registros em si, só precisamos mudar o número que precede cada registro, para dar:

3 SMITH	2 JONES	1 BROWN	4 WILSON

onde os números indicam a posição alfabética de cada registro.

Embora classificar ponteiros (os rótulos) ao invés de movimentar fisicamente registros seja mais eficiente, o método possui várias desvantagens. Um dos mais graves obstáculos é que a cada vez que um nome é adicionado, apagado ou editado de tal arquivo, todo o arquivo precisa ser reclassificado.

Ligação de registros com ponteiros

Um método mais eficiente de classificação é o de ligar registros em uma 'corrente' com cada ponteiro sendo estabelecido logo após a criação do registro. Para o funcionamento

de tal técnica, precisamos criar um inicial registro fantasma que contenha um simples A. Obviamente, qualquer outro registro deve seguir o registro inicial, como mostrado abaixo.

1 A	2 SMITH

Quando um terceiro registro é introduzido no arquivo, ele é imediatamente comparado com o registro existente e a sua posição alfabética encontrada. Supondo que tal registro seja JONES, a sua posição alfabética estará entre os registros 1 e 2, como mostrado a seguir,

1 A	3 SMITH	2 JONES

com o ponteiro do registro SMITH devidamente modificado, adicionando-se 1 a ele.

Analogamente, quando um registro posterior for introduzido no arquivo será comparado com o primeiro e todos os registros subseqüentes, até que sua posição alfabética seja encontrada. O seu ponteiro é então determinado e todos os ponteiros dos registros que o sucedem terão de ser atualizados adicionando-se 1 a cada. Uma vez que, de fato, a classificação dos ponteiros não é necessária, o método é bem mais eficiente do que qualquer outro descrito até agora.

A maior desvantagem em ligar registros em tal 'corrente' (com um único ponteiro) é a de que, a cada vez que um novo registro é estabelecido (criado, apagado ou editado), todos os ponteiros de ligação daquele registro em diante, até o fim da corrente alfabética, precisam ser atualizados. Isto pode levar um tempo muito longo, para um arquivo muito grande, o que pode ser inaceitável.

Ligações para trás e para frente

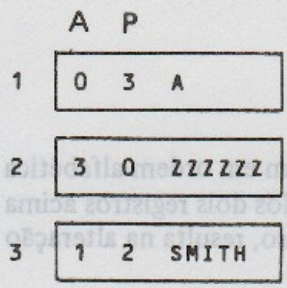
Para superar a desvantagem acima, adotaremos um método novo e diferente que necessita de dois ponteiros para ligar registros. O primeiro ponteiro liga o registro corrente ao seu "anterior" (na corrente alfabética), enquanto que o segundo ponteiro liga o registro corrente ao seu "posterior". Para obter tal ligação, contudo, precisamos criar dois registros-fantasmas antes de começarmos. O primeiro dos dois é preenchido com um simples A, enquanto que o segundo é preenchido com Zs. O motivo para isto ficará claro à medida que o método for explicado.

Para explicar o que acontece quando criamos registros temos de representá-los esquematicamente desenhando-os de forma empilhada. Isto só é feito devido às limitações impostas pela largura do papel. Indicaremos em seguida, na frente de cada registro, o número do registro. Isto só é mostrado por conveniência e não faz parte do arquivo, visto que o computador já sabe a posição física de cada registro através do comprimento de registro predefinido que tem de ser declarado antes de se criar o arquivo. Assim, de início, teremos:

	A	P
1	0	2 A
2	1	0 ZZZZZZ

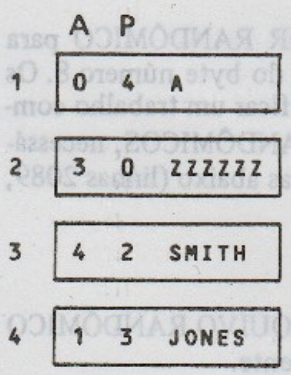
onde os rótulos dos ponteiros A e P correspondem a 'anterior' e 'posterior', respectivamente.

Para o primeiro registro, fixamos A=0 para indicar que não existe registro anterior no arquivo, enquanto P=2 indica que o próximo registro na corrente é o segundo registro. Analogamente, para o segundo registro, A=1 que indica que o seu registro anterior é o primeiro registro, enquanto P=0 indica que não existe registro seguinte. Desta maneira todos os registros criados serão estabelecidos (em ordem alfabética) entre os ponteiros destes dois registros. Logo, quando entrar um novo registro no arquivo, a sua posição alfabética no arquivo será indicada como se segue:

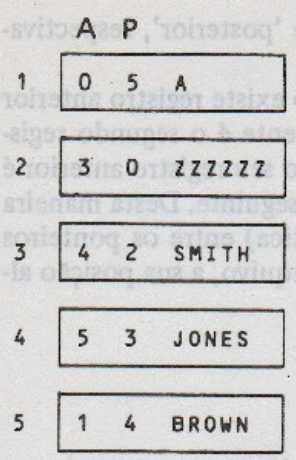


Note que o registro SMITH deve encaixar (alfabeticamente) entre o primeiro e o segundo registro. A para o primeiro registro e P para o segundo registro permanecem inalterados. Se não tivéssemos estes dois registros fantasmas, o processo resultaria em registros "anteriores" negativos, caso registros criados mais recentemente tivessem de ser posicionados alfabeticamente na frente dos já existentes.

Portanto, o registro SMITH deve possuir ponteiros que o posicionem alfabeticamente entre o primeiro e o segundo registro. Assim, para o primeiro registro, P passa a ser 3, enquanto que para o segundo registro A passa a ser 3. O terceiro registro físico tem A=1 e P=2..Suponha agora que o próximo registro é introduzido. Então:



Após a comparação alfabética, o registro JONES deve receber ponteiros que o liguem entre o registro A e o registro SMITH. Note que os únicos ponteiros a serem modificados são os do registro anterior (A) e o posterior (SMITH). Os ponteiros de ZZZZZZ permanecem inalterados. Adicionando-se mais um registro, ficamos com:



O quinto registro (BROWN) deve receber ponteiros que o posicionem em ordem alfabética entre os dois registros A e JONES. Desta maneira, só os ponteiros dos dois registros acima são alterados, e juntamente com os ponteiros, dados ao novo registro, resulta na alteração de três registros a cada vez que um novo registro é entrado.

Antes que possamos implementar as idéias acima, devemos investigar a facilidade de se escrever em um byte específico em um dado registro.

A opção de parâmetro de byte

Podemos usar a opção de parâmetro de byte para escrever num byte específico do r-ésimo registro com o uso do seguinte comando:

```
WRITE   "nome do arquivo",R5,B2
```

que é interpretado como 'escreva no byte número 2 de registro número 5 do arquivo especificado'. Analogamente,

```
READ   'nome do arquivo',R6,B3
```

é interpretado como 'leia o byte número 3 do registro número 6 do arquivo especificado'. O número seguindo B deve ser um inteiro entre 0 e 32767.

Podemos agora continuar, alterando o programa IMPRIMIR RANDÔMICO para que os registros possam ser criados e escritos no disquete a partir do byte número 8. Os bytes 0 a 7 serão reservados para escrever os dois ponteiros. Para ficar um trabalho completo, as modificações no programa EDIÇÃO DE ARQUIVOS RANDÔMICOS, necessárias para implementar a solução do Problema 8.3, são também dadas abaixo (linhas 2089, 2170 e 2180).

Criação de ponteiros para ligar arquivos

As seguintes alterações e acréscimos ao programa EDIÇÃO DE ARQUIVO RANDÔMICO irão criar os ponteiros apropriados que ligarão registros alfabeticamente.

```
   30  NE=20:DIM A$(NE+2),PN(NE+2,2):BY=8
 1020  RL=VAL(RL$):IF RL<(BY+1) OR RL>256 GOTO 1010
 1025  SL=RL:SB=BY:GOSUB 30000
 1030  PRINT "NUMERO MAXIMO DE NOVAS ENTRADAS = ";NE:SL=RL-BY
 1040  FOR N=1 TO NE:PRINT:SN=N:GOSUB 20000: A$(N+2)=SR$
```

```
1050  IF A$(N+2)="" GOTO 1070
1060  GOSUB 40000:NEXT N
1070  N=N+1:LD=0:PRINT:IF M=3 THEN N=N+NN: GOTO 1090:REM N
      INCLUDES FIRST TWO RECORDS
1135  PRINT PN(RN,1):PRINT PN(RN,2): PRINT D$;"WRITE"; X$;
      ",R"; RN; ",B"; BY
2050  INPUT N:PRINT "REGISTROS "; N-2
2088  B=VAL(B$):IF B<1 OR B> N-2 THEN PRINT "CAN'T BE
      DONE":GOTO 2082
2089  INPUT "OUTPUT P/I IMPRESSORA? S/N) ";T$:IF
      LEFT$(T$,1)="S THEN PRINT D$: PRINT D$;"PR#1": PR=1
2095  B=B+2: E=B+ NE-3: IF E>N THEN E=N
2115  PRINT D$;"READ";X$;",R";RN;",B";BY
2140  LN=RN-2: PRINT " ";LN;";">";A$(I): IF M=4 THEN VTAB
      23: GOTO 2160
2170  IF PR=1 THEN PRINT D$: PRINT D$;"PR#0": PR=0
2180  PRINT:PRINT CM$:GET Q$:GOTO 100
30000   REM *****SUB-ROTINA PARA INICIALIZAR NOVOS ARQUIVOS***** I/
    0 PAR:-INPUT SL,SB; OUTPUT A$( ),PN(,)
30010   PS=1:NT=0:S$=CHR$(255):SS$=""
30020   FOR I=2 TO 1 STEP-1:PN(I,1)=PS: PN(I,2)=NT
30030   IF I=1 GOTO 30060
30040   FOR J=1 TO SL-SB-1:SS$=SS$+S$: NEXT J:
    A$(I)=SS$+CR$
30050   PS=0:NT=2:SS$="":NEXT I
30060   FOR J=1 TO SL-SB-1:SS$=SS$+CHR$(0): NEXT J:
    A$(I)=SS$+CR$: RETURN
40000   NT=1:REM ***** SUB-ROTINA PONTEIROS ***** I/0 PAR:-
    INPUT N,A$( ),PN(,); OUTPUT PN(,)
40010   FOR J=1 TO N+1: PS=PN(NT,1): IF A$(NT)<=A$(N+2)
    THEN NT=PN(NT,2): NEXT J
40020   PN(N+2,1)=PN(NT,1)
40030   PN(N+2,2)=NT
40040   PN(PS,2)=N+2
40050   PN(NT,1)=N+2
40060   RETURN
```

A linha 1025 chama a sub-rotina 30000 que inicializa tanto o array de strings A$( ) como o array numérico PN(,). O primeiro string em A$( ) é composto de um único caractere CHR$(0) (que é o caractere de menor código ASCII), enquanto que o segundo string é totalmente preenchido com o caractere CHR$(255) (que é o maior código ASCII). O comprimento do string é determinado pelo registro de maior comprimento (RL=BY), onde BY foi estabelecido em 8 bytes na linha 30. Quando lhe for perguntado o 'comprimento máximo do registro', você deve se lembrar de que 8 bytes do número que você escolher serão reservados para os dois ponteiros que agora precedem cada registro.

A sub-rotina 40000 compara alfabeticamente o registro entrado A$(N+2) com todos os registros existentes no array de strings A$( ), inclusive os registros CHR$(0) e CHR$(255). As linhas 40020 a 40050 determinam os ponteiros apropriados e os armazena no array PN(,).

Digite estes acréscimos e salve o programa resultante sob o nome de arquivo BYTE RANDÔMICO. Para poder observar o que acontece cada vez que um registro é entrado, digite as seguintes linhas extras que não precisam ser salvadas com o resto do programa.

```
40055  PRINT: FOR J=1 TO N+2: PRINT PN(J,1);" ";
       PN(J,2);" ";A$(J): NEXT J
```

Problema 8.4

Implemente as seções de 'Atualização' e "Edição" do programa BYTE RANDÔMICO para que, à medida que dado registro seja acrescentado ou editado, os ponteiros de seus registros alfabeticamente adjacentes sejam modificados apropriadamente. Sugerimos que reescreva as seções relevantes do programa, fazendo-as auto-suficientes, eliminando todos os saltos de programa para outras seções.

Finalmente, adicione uma outra opção ao programa para que registros possam ser recuperados em ordem alfabética.

Classificar números inteiros no disquete, como estamos fazendo com os ponteiros, limita a exploração total da técnica mencionada acima, pois à medida que um inteiro aumenta, ele necessita de mais bytes para armazenamento. Com 4 bytes por ponteiro, como escolhido acima, podemos criar 999 registros, e embora possamos superar esta limitação aumentando o número de bytes designados para cada número, se alguma vez desejarmos ligar os registros acima a outros arquivos, devemos armazenar inteiros no disquete em formato tal que a sua posição no registro seja conhecida com precisão, independentemente de seu comprimento. Isto só pode ser obtido desmontando-se cada número em um número predefinido de bytes, antes de armazená-los em disco.

Desmontagem e remontagem de inteiros

Exploremos agora os fatos mencionados anteriormente e no Anexo E (leia a parte que diz respeito à representação numérica de dados), para podermos converter inteiros de tal forma que eles caibam em um número específico de bytes. Isto, juntamente com a opção de escrever num byte específico de um registro, irá ajudar aquele programador que quer aventurar-se nas profundezas dos 'Bancos de Dados':

O programa seguinte irá desmontar qualquer número inteiro em um número prefixado de bytes, imprimir o valor armazenado em cada byte (com o byte mais baixo primeiro), e em seguida remontar o número original a partir do conteúdo destes bytes.

```
10  REM DESMONTANDO E REMONTANDO INTEIROS
20  INPUT "DESMONTAR EM QUANTOS BYTES? ";K: DIM
    B(K):Z=0
30  INPUT "ENTRE QUALQUER INTEIRO ";N
40  IF N>256↑K-1 THEN PRINT "NAO PODE SER FEITO EM "; K;"
    BYTE(S)": GOTO 20
50  PRINT:FOR I=1 TO K:X=N/256:N=INT(X): B(I)= (X-N)*256
60  PRINT "BYTE ";I;" = ";B(I):NEXT I
70  FOR I=1 TO K:Z=Z+B(I)*256↑(I-1):NEXT I
80  PRINT:PRINT "NUMERO MONTADO ";Z: END
```

Vamos agora modificar o programa acima e escrevê-lo em termos de duas sub-rotinas, uma para desmontar inteiros em um ou dois bytes, e a outra para remontá-lo novamente no seu valor original. Ao escrever estas sub-rotinas, devemos ter em mente que, ao final, o inteiro desmontado será armazenado no disquete. Esta condição torna as sub-rotinas bem mais complicadas que o programa anterior.

Os números são armazenados no disquete em seu código ASCII. Por isto, é necessário saber se o número que representa um dado byte de um inteiro desmontado está entre 0 e 127 ou 128 e 255 (veja Anexo G). Além disso, escrever 0 num disquete indica "fim de dados" ao tentar lê-lo de novo no disco, escrever 4 cria confusão, pois é idêntico a PRINT D$ (o comando de acesso ao disco). Para distinguir entre estas variações, uma chave (flag) de um byte (SF no programa abaixo) é aplicada para cada byte do número desmontado.

Para ilustrar as idéias discutidas acima, digite o seguinte programa, deixando as linhas 60 a 280 em branco, pois serão usadas mais tarde para acréscimos ao programa.

```
   10  REM DESMONTANDO E REMONTANDO NUMEROS PARA O DISCO
   20  INPUT "DESMONTAR EM QUANTOS BYTES? (1/2) ";K: IF
         K<1 OR K>2 GOTO 20
   30  INPUT "ENTER ANY INTEGER ";N
   40  IF N>256↑K-1 THEN PRINT "NAO PODE SER FEITO EM ";K;"
         BYTE(S)": GOTO 20
   50  PRINT:SK=K:SN=N:GOSUB 10000:FOR I=1 TO K: B(I)=SB(I):
         F(I)=SF(I): NEXT I
  290  SK=K:FOR I=1 TO K:SB(I)=B(I):SF(I)=F(I): NEXT I:
         GOSUB 11000: Z=SZ
  300  PRINT "LB=";SB((1);"  FL1=";SF(1); "      HB=";SB(2);"
         FL2=";SF(2)
  310  PRINT Z;TAB(10);Z$:PRINT:PRINT: PRINT
  320  IF N<>Z THEN PRINT CHR$(7):END
  330  GOTO 20:END

10000   REM ***** SUB-ROTINA P/DESMONTAR***** I/0 PAR:-
      INPUT SK,SN; OUTPUT SB(I),SF(I)
10010   SB(2)=0:SF(2)=0:FOR I=1 TO SK:X=SN/256: SN=INT(X):
      SB(I)=(X-SN)*256
10020   IF SB(I)=0 THEN SF(I)=10:SB(I)=SB(I)+1: GOTO 10080
10030   IF SB(I)=4 THEN SF(I)=11:SB(I)=SB(I)+1: GOTO 10080
10040   IF SB(I)<128 THEN SF(I)=12:GOTO 10080
10050   SB(I)=SB (I)-128:IF SB(I)=0 THEN SF(I)=13:
      SB(I)=SB(I)+1: GOTO 10080
10060   IF SB(I)=4 THEN SF(I)=14:SB(I)=SB(I)+1: GOTO 10080
10070   SF(I)=15
10080   NEXT I:RETURN

11000   REM ***** SUB-ROTINA P/REMONTAR***** I/0 PAR:-
      INPUT SB(I),SF(I);OUTPUT SZ
11010   FOR I=1 TO SK
11020   IF SF(I)=10 OR SF(I)=11 THEN SB(I)=SUB(I)-1: GOTO
      11050
```

```
11030  IF SF(I)=13 OU SF(I)=14 THEN SB(I)=SB(I)+127: GOTO
       11050
11040  IF SF(I)=15 THEN SB(I)=SB(I)+128
11050  NEXT I:SZ=SB(1)+SB(2)*256:RETURN
```

Observação: A chave SF, para os bytes (SF(1) abaixo e SF(2) acima), recebe os valores mostrados na Tabela 8.1. Se os conteúdos do byte (SB(1) abaixo e/ou SB(2) acima) forem 0 ou 4, some-lhe 1 e estabeleça a chave SF apropriadamente. Se, por outro lado, o número em SB for maior que 127, subtraia 128 de seu valor e repita o teste acima para os valores 0 e 4, novamente determinando o conteúdo da chave apropriadamente.

Ao remontar um número, a chave é primeiro testada e, de acordo com seu valor, o número original é remontado, adicionando-se ou subtraindo-se apropriadamente os valores corretos. Por exemplo, se a chave está com o valor 15, adicione 128 ao conteúdo do byte ou, se estiver com o valor 13, adicione 128-1, e assim por diante.

Se um número for maior que 255, mas menor que 65535 (veja Anexo E), dois bytes são necessários para representá-lo, cada um deles com uma faixa de 0 a 255. Neste caso, cada byte tem uma chave associada e o procedimento descrito acima é aplicável a cada byte de cada vez:

TABELA 8.1 Representação de byte

Original	*Conteúdo do byte*	*Chave*
Faixa 0-127		
0	SB+1	10
4	SB+1	11
<128	SB	12
Faixa 128-255		
0	SB-128+1	13
4	SB-128+1	14
<256	SB-128	15

Podemos agora desmontar números em um ou dois bytes, dependendo se estão na faixa de 0-255 ou 256-65535, respectivamente, e escrevê-los no disquete em um byte específico. Subseqüentemente, vamos ler (READ) o conteúdo destes bytes específicos e remontar o número original. Para ilustrar este processo, carregue o último programa e digite as seguintes linhas adicionais.

```
 60  X$="TEMP":LE=16
 70  D$=CHR$(4):BY=2
 80  PRINT D$;"OPEN";X$;",L";LE
 90  FOR I=1 TO K
100  PRINT D$;"WRITE";X$;",B";BY
110  PRINT CHR$(B(I)):BY=BY+1
120  PRINT D$;"WRITE";X$;",B";BY
```

```
130  PRINT CHR$(F(I)):BY=BY+1
140  NEXT I
150  PRINT D$;"WRITE";X$;",B";BY
160  PRINT "DONE"
170  BY=2:FOR I=1 TO K
180  PRINT D$:PRINT D$;"READ";X$;",B";BY
190  GET B$
200  B(I)=ASC(B$):BY=BY+1
210  PRINT D$:PRINT D$;"READ";X$;",B";BY
220  GET F$
230  F(I)=ASC(F$):BY=BY+1
240  NEXT I
250  PRINT D$:PRINT D$;"READ";X$;",B";BY
260  INPUT Z$
270  PRINT D$;"CLOSE";X$
280  PRINT D$;"DELETE";X$
```

Na linha 60 o nome TEMP é atribuído ao arquivo X$, em vez de o computador perguntar ao usuário 'Nome sob o qual deve ser salvo?'. O programa foi escrito de forma que você possa testá-lo para diferentes entradas quantas vezes desejar sem acabar com um disquete cheio de arquivos criados. É por este motivo que X$ é apagado (DELETE) na linha 280, para poder ser usado novamente para fins de teste. Na linha 80, o arquivo TEMP (de comprimento 16 – como especificado na linha 60) é aberto (OPEN) e o byte baixo de um número (quando I=1) é escrito no segundo byte, enquanto que a chave associada a ela é escrita imediatamente após, no terceiro byte. Analogamente, o byte alto (quando I=2) é escrito no quarto byte, enquanto a sua chave é escrita no quinto byte. Finalmente, o string 'DONE' ('feito') é escrito a partir do quarto byte, se for um número desmontado de um byte, ou a partir do sexto byte, se for um número desmontado de dois bytes.

As linhas 170 a 270 lêem do disquete a informação armazenada e atribuem o conteúdo de cada byte à variável apropriada, (B(1) para o byte baixo, B(2) para o byte alto, enquanto F(1) e F(2) são suas chaves respectivas). A remontagem do número é então obtida a partir da linha 290 como explicado anteriormente.

Rode este programa e teste-o com os números mencionados na Tabela 8.1. Não se esqueça de salvar o programa completo sob o nome de arquivo NR DISASS TO DISK*.

* NR DISASS TO DISK = NR DECOMPOR PARA DISCO

Capítulo 9
Gráficos — 2.ª parte

Tabelas de formas

Formas gráficas podem ser criadas a partir do teclado do Apple com uma série de vetores de plotagem conhecidos como 'definição de formas'. Uma ou mais dessas definições de formas, junto com um índice que deve precedê-las, constitui uma 'Tabela de Formas'. O Applesoft possui quatro comandos gráficos de alta resolução que podem manipular os desenhos especificados por tais tabelas. Estes são:

DRAW, XDRAW, ROT and SCALE.

Estes comandos só podem ser usados se existir uma tabela de formas na memória do Apple. Para ilustrar o seu uso, vamos criar e introduzir (poke) na memória a tabela de formas mais simples possível, aquela de uma linha reta. Vamos usá-la então para desenhar esta forma começando no meio da tela, dirigida para baixo. Isto pode ser obtido com o seguinte programa:

```
 10 REM CRIACAO DE UMA TABELA DE FORMAS DE UMA LINHA RETA
 20 HOME: TEXT
 30 POKE 232,32:POKE 233,3
 40 FOR I=800 TO 805
 50 READ K:POKE I,K:NEXT I
 60 HGR:HCOLOR=3
 70 SCALE=30:ROT=0
 80 DRAW 1 AT 139,95
 90 DATA 1,0,4,0,6,0
100 END
```

O programa acima irá criar e armazenar a tabela de formas (dada pelos números da instrução DATA – que será explicada brevemente) na memória do Apple, através do comando POKE (linhas 40 e 50), a partir do local de memória 800. O Apple tem disponível uma pequena área de memória protegida (256 bytes) para guardar programas em linguagem de máquina de usuários. Esta área começa na posição de memória 800 para um sistema Apple com Applesoft em ROM, e no local 300 para sistemas com Applesoft em RAM. Para tabelas de formas, que necessitam uma memória maior que esta, você deve armazená-la logo abaixo de HIMEM (alta memória, do inglês HI-MEMORY) e restabelecer (RESET*) HIMEM logo abaixo dela para poder protegê-la (Veja Anexo D).

A linha 30 do programa acima dá informações ao BASIC sobre o endereço de partida na memória onde a tabela de formas é encontrada. Esta informação deve ser colocada (POKE) nos locais de memória 232 e 233 (sendo dado primeiro o byte mais baixo do endereço i.e., 32+256*3=800). No caso do sistema Apple com Applesoft em ROM, as li-

* Vide nota da tradução.

nhas 30 e 40 devem ser modificadas para refletir os diferentes endereços de partida das tabelas de formas como se segue:

```
30 POKE 232,4:POKE 233,1
40 FOR I=300 TO 305
```

Finalmente, a linha 60 ativa a área HGR e estabelece a cor, enquanto que a linha 70 estabelece a escala e rotação do desenho. Na linha 80 o Applesoft consulta as posições 232 e 233 para os endereços de partida da tabela de formas que, em nosso exemplo, é o local de memória 800. Depois ele anda através da tabela de formas, byte por byte, executando os comandos nela armazenados.

Geralmente, os conteúdos dos bytes em uma tabela de formas fornece a seguinte informação:

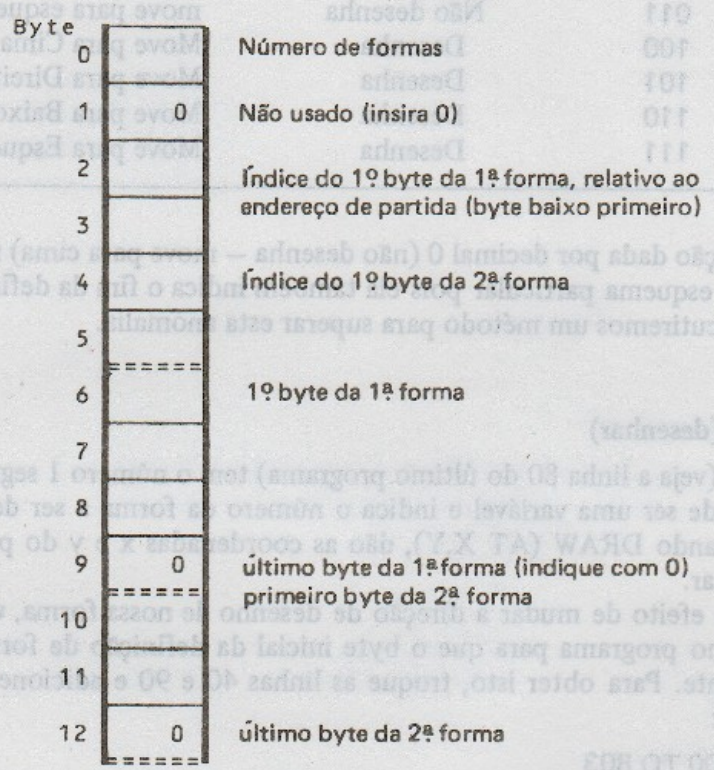
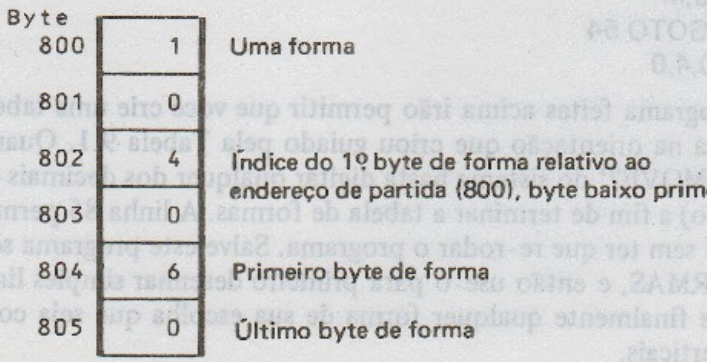


Fig. 9.1 Representação esquemática de uma tabela de formas.

Em nosso exemplo, a Fig. 9.1 é reduzida a:



Os conteúdos desses bytes aparecem na instrução DATA do programa anterior. No quarto byte da tabela de formas, que é o primeiro byte de nossa definição de forma da linha reta, encontramos o número decimal 6. A Tabela 9.1 lista os possíveis números que descrevem os movimentos do vetor, tanto quando desenhado ou não.

TABELA 9.1 Movimentos do vetor

Decimal	*Binário*	*Interpretação*	
0	000	Não desenha	move para cima
1	001	Não desenha	move para direita
2	010	Não desenha	move para baixo
3	011	Não desenha	move para esquerda
4	100	Desenha	Move para Cima
5	101	Desenha	Move para Direita
6	110	Desenha	Move para Baixo
7	111	Desenha	Move para Esquerda

Observe que a instrução dada por decimal 0 (não desenha – move para cima) não pode ser obedecida com este esquema particular pois ela também indica o fim da definição de forma. Mais adiante discutiremos um método para superar esta anomalia.

O comando DRAW (desenhar)

O comando DRAW (veja a linha 80 do último programa) tem o número 1 seguido da palavra DRAW. Isto pode ser uma variável e indica o número da forma a ser desenhada. Os parâmetros do comando DRAW (AT X,Y), dão as coordenadas x e y do ponto onde o desenho deve começar.

Para ilustrar o efeito de mudar a direção de desenho de nossa forma, vamos alterar ligeiramente o último programa para que o byte inicial da definição de forma possa ser modificado facilmente. Para obter isto, troque as linhas 40 e 90 e adicione mais três linhas, como se segue:

```
40 FOR I=800 TO 803
54 VTAB 21: FOR J=0 TO 9:INPUT "MOVER? ";M:
   IF M<>0 THEN POKE I+J,M:NEXT J
58 POKE I+J,M
85 GET Q$:GOTO 54
90 DATA 1,0,4,0
```

As alterações no programa feitas acima irão permitir que você crie uma tabela de formas, e desenhe esta forma na orientação que criou guiado pela Tabela 9.1. Quando receber a pergunta (prompt) 'MOVE?' do sistema basta digitar qualquer dos decimais da Tabela 9.1 com um final 0 (zero) a fim de terminar a tabela de formas. A linha 85 permite-lhe modificar o parâmetro M sem ter que re-rodar o programa. Salve este programa sob o nome de arquivo CRIAR FORMAS, e então use-o para primeiro desenhar simples linhas retas, depois um quadrado, e finalmente qualquer forma de sua escolha que seja composta de linhas horizontais e verticais.

O comando DRAW pode ser usado de forma um pouco diferente. Podemos, por exemplo, substituir as linhas 80 e 85 (a última linha não é mais necessária para o que vamos fazer) pelas seguintes:

```
80 HPLOT 139,95
85 DRAW 1
```

Isto tem o efeito de fixar a posição de partida de desenho nas coordenadas 139,95 através do comando HPLOT e depois desenhar a forma 1 com o auxílio do comando DRAW. A única desvantagem deste método sob o anterior é que qualquer execução subseqüente da linha 85 irá desenhar a forma 1 a partir do último ponto plotado na tela. Assim, o desenho completo de uma forma pode ser unido à forma que se seguir.

Problema 9.1

Escreva um programa que use uma tabela de formas com uma definição de forma nele constituída de duas linhas (nas direções para cima e para a direita). Use o comando DRAW, omitindo as coordenadas para desenhar degraus a partir do canto inferior esquerdo de sua tela.

Tendo resolvido o Problema 9.1, troque a linha 85 do programa CRIAR FORMAS pela seguinte:

```
85 FOR I=1 TO 1000=DRAW 1:NEXT I
```

e rode o programa. Você deverá ver a tela ser totalmente preenchida com o desenho da forma 1. Outro comando Applesoft nos permite desenhar uma dada forma no complemento da cor já existente em cada cor plotada. Este comando é:

```
XDRAW n
```

onde *n* é o número da forma. O comando nos demais aspectos é igual ao comando DRAW n, mas permite uma maneira fácil de apagar o desenho. Tudo que precisamos fazer é primeiro desenhá-lo (XDRAW), depois redesenhá-lo (XDRAW), o que tem o efeito de desenhar a mesma forma mas só que com a cor complementar da primeira, portanto apagando a forma mas não o fundo.

Para ilustrar o uso do comando XDRAW, troque DRAW por XDRAW na linha 85 do último programa rodado e acrescente a seguinte linha

```
88 HPLOT 139,95:FOR I=1 TO 1000=XDRAW 1:NEXT I
```

Observe que para apagar a última forma XDRAW n é necessário começar o segundo comando XDRAW nas mesmas coordenadas do ponto inicial de desenho. Salve este programa sob o nome de arquivo CRIAR E APAGAR FORMAS, e depois rode-o.

ATENÇÃO: Se tanto o comando DRAW como XDRAW forem emitidos e não existir uma tabela de formas, o sistema irá 'parar' e poderá causar um apagamento de parte de seu programa.

Escala e direção das formas

O comando SCALE é usado para determinar o tamanho em que será desenhada a forma na tela com os comandos DRAW e XDRAW. A forma geral do comando é

```
SCALE=n
```

onde *n* é uma variável que está entre 1 e 255. Portanto, SCALE=1 reproduz o tamanho da definição de forma na tela na base um-para-um, enquanto SCALE=255 multiplica cada vetor da tabela de definição de formas por 255 antes de desenhá-la na tela.

O comando ROT é usado para mudar a orientação em que a definição de forma é desenhada na tela com os comandos DRAW ou XDRAW. A sua forma geral é

```
ROT=n
```

onde *n* é uma variável entre 0 e 255. ROT=0 reproduz a definição de forma na tela na mesma orientação em que foi desenhada, enquanto que os valores 16, 32 e 48 fazem com que a forma seja desenhada no sentido horário de 90, 180 e 270 graus, respectivamente. O processo se repete em ROT=64 e em qualquer múltiplo inteiro deste (i.e., 128 e 192).

Ângulos intermediários de rotação dependem da escala (SCALE) do desenho. Para SCALE=1 só as quatro rotações básicas (0, 16, 32 e 48) são possíveis. Para SCALE=2, oito rotações (0, 8, 16, 24, 32, 40, 48 e 56) são possíveis, dando um ângulo incremental de 45 graus. Para SCALE=3, dezesseis rotações são possíveis, permitindo um ângulo incremental de 30 graus, e assim por diante.

Para ilustrar a interação entre SCALE e ROTation, carregue o programa CRIAR E APAGAR FORMAS e faça as seguintes alterações:

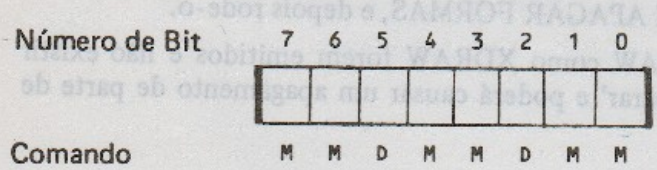
```
70 INPUT "SCALE? ";SC:SCALE=SC
80 FOR R=0 TO 255:ROT=R
85 XDRAW 1 AT 139,79:XDRAW 1:NEXT R
```

e apague a linha 88. Rode o programa usando diferentes valores para SCALE, tentando para uma forma quadrada e uma figura em forma de U.

Tabelas de formas complexas

Até agora, pudemos construir tabelas de formas simples usando as informações contidas na Tabela 9.1. Entretanto, ao tentarmos criar tabelas de formas mais complicadas que as que usamos até agora, deparamos com duas desvantagens. São elas: (*a*) iríamos usar rapidamente um grande número de bytes ao criar uma tabela de formas (visto que a técnica que adotamos utiliza um byte para cada movimento do vetor), e (*b*) não podemos usar o comando que 'move para cima sem desenhar'.

Ambas as desvantagens podem ser contornadas facilmente com o uso de uma tabela do tipo da 9.1, só que ligeiramente ampliadas. Antes, porém, faz-se necessário entender como as informações relacionadas a tabelas de formas são armazenadas em cada byte. Cada byte contém 8 bits numerados de 0 a 7, da direita para a esquerda, como mostrado abaixo.



Nas tabelas de formas, cada byte é dividido em três seções, as duas primeiras contendo três bits cada, enquanto que a terceira só contém dois, como indicado acima.

TABELA 9.2 Tabela para Definição de Formas

	m.c.	*m.d.*	*m.b.*	*m.e.*	*M.C.*	*M.D.*	*M.B.*	*M.E.*
m.c.	–	8	16	24	32	40	48	56
m.d.	1	9	17	25	33	41	49	57
m.b.	2	10	18	26	34	42	50	58
m.e.	3	11	19	27	35	43	51	59
M.C.	4	12	20	28	36	44	52	60
M.D.	5	13	21	29	37	45	53	61
M.B.	6	14	22	30	38	46	54	62
M.E.	7	15	23	31	39	47	55	63

O conteúdo de cada uma das três seções de cada byte de uma tabela de formas tem sua função indicada pela letra que aparece abaixo de nossa apresentação, onde M significa 'Movimento' e D 'Desenhe/ou não'. Até agora só utilizamos os três primeiros bits de cada byte armazenando decimais de 0 a 7 (como na Tabela 9.1) em um dado byte. A tradução destes números para notação binária (também fornecida na Tabela 9.1) dá uma visão mais aprofundada de como uma tabela de formas é construída.

Na Tabela 9.1, você verá que o bit mais à esquerda de cada representação binária dos números 0 a 7 é 0 (para os quatro primeiros números) ou 1 (para os quatro últimos). Este bit corresponde à parte de 'Desenhe/ou não' do comando contido em cada uma das duas primeiras seções de nossa representação do byte. Portanto, D=0 significa 'não desenhe', enquanto D=1 significa 'Desenhe'. Analogamente, os dois bits binários, representando a parte de "movimento" do comando, podem indicar os seguintes comandos:

00 move para cima
01 move para direita
10 move para baixo
11 move para esquerda

De acordo com a informação que pode ser mantida em cada byte, podemos acomodar dois comandos de movimento com 'Desenhe/ou não', assim como um terceiro comando só de movimento, por byte. O bit mais à esquerda da terceira seção que está faltando é interpretado como 'Não desenhe' (i.e., D é assumido como zero). Ao construir a tabela expandida, ignoramos a existência da terceira seção de cada byte.

A Tabela 9.2 pode ser usada para traduzir dois vetores de movimento de cada byte. Nela, as letras minúsculas indicam movimento sem desenhar, enquanto que as letras maiúsculas indicam movimento do desenho. Para ilustrar o uso da tabela, suponhamos que queiramos mover para cima sem desenhar 'e depois' mover para direita sem desenhar. Consultando a primeira linha indicada por M.C. e depois a coluna indicada por M.D., obteremos o número 8. Em termos de representação do byte isto é traduzido por:

Número de bit	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	0	0
Comando	M	M	D	M	M	D	M	M

que são os dois vetores de movimento desejados.

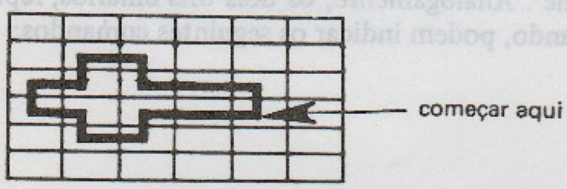
Analogamente, se desejarmos 'mover para cima sem desenhar' e depois 'mover para cima enquanto desenha', obteremos 32 da tabela, que em termos de representação do byte é traduzido por

Número de bit	7	6	5	4	3	2	1	0
	0	0	1	0	0	0	0	0
Comando	M	M	D	M	M	D	M	M

que novamente são os dois vetores de movimento desejados.

A primeira coluna da tabela representa os comandos traduzidos usando a Tabela 9.1 e representam uma única tradução de vetor por byte. Observe que ainda não podemos 'mover para cima sem desenhar', pois um 0 em todo um byte indica o fim da tabela de formas. Contudo, podemos 'mover para cima sem desenhar' se em seguida 'movermos para esquerda ou direita desenhando ou não'.

Para construir uma tabela de forma (de qualquer complexidade) precisamos esboçar a forma desejada em papel quadriculado e depois traduzir cada movimento em uma série de vetores de 'desenhe ou não'. Por exemplo, com a forma dada abaixo, e um ponto de partida no canto inferior direito mais à direita, podemos traduzi-la na seguinte série de movimentos.



(M.E. M.E.) (M.B. M.E.) (M.C. M.E.) (M.C. M.D.)
(M.C. M.D.) (M.B. M.D.) (M.D. M.B.)

Consultando a Tabela 9.2, são obtidos os seguintes números:

63, 62, 60, 44, 44, 46, 53, 0

onde o 0 final termina a tabela de formas.

Use o programa CRIAR FORMAS para criar uma tabela de formas do desenho acima usando os números dados.

Salvando tabelas de formas em disco

Uma vez criadas as tabelas de formas, seria útil poder armazená-las em disquete. Isto pode ser feito facilmente digitando-se:

```
BSAVE 'nome de arquivo',A800,LI
```

onde *l* é o comprimento do arquivo a ser salvado (BSAVE). No último exemplo, 1=12, pois teríamos os 8 bytes fornecidos pela lista de números obtidos da Tabela 9.2 mais 4 bytes que são necessários para definir o número total de definições de formas desta tabela (no nosso caso, 1) e os seus respectivos índices (2 bytes por definição de forma). O endereço de partida de 800 terá de ser modificado caso você decida armazenar a tabela em local de memória diferente.

O seguinte programa pode ser usado para carregar (BLOAD) uma tabela de formas que foi previamente salvada (BSAVE) em um disquete.

```
10 REM CARREGADOR DE FORMAS (BLOAD)
20 TEXT:HOME:D$=CHR$(4)
30 INPUT "TABELA DE FORMAS A CARREGAR? ";A$
40 PRINT D$;"BLOAD";A$
50 POKE 232,32:POKE 233,3
60 VTAB 21:INPUT "SCALA? ";S
70 HGR:HCOLOR=3:SCALE=S:R=0
80 DRAW 1 AT 139,95
90 END
```

Problema 9.2

Escreva um programa que permita a criação de uma tabela de formas com um número variável de definições de formas, número esse que deve ser especificado no início do programa. Após ter criado a tabela de formas completa, o programa deve plotar, uma a uma, as definições de forma, para fins de checagem, antes de salvá-la em disco. Salve o programa sob o nome de arquivo CRIAR DEFINIÇÕES DE FORMAS.

Animação de formas

Podemos simular um objeto voador criando uma tabela de formas com duas definições de formas praticamente idênticas a menos de um comando extra de "movimento sem desenhar". Para ilustrar este fenômeno, use o programa CRIAR DEFINIÇÕES DE FORMAS para criar uma tabela de formas com a primeira definição de formas sendo idêntica à discutida anteriormente. Depois repita a mesma definição de formas, mas antes de digitar 0, digite 1. Assim, a única diferença entre as duas formas é que a segunda possui um comando de "movimento para direita sem desenhar".

Use agora o seguinte programa para fazer a sua forma voar em qualquer direção que escolher. Para sair do programa pressione RESET.

```
10 REM OBJETOS VOADORES
20 HOME:TEXT:POKE 232,32:POKE 233,3:
   D$=CHR$(4)
```

```
30 INPUT "TABELA DE FORMAS A SER CARREGADA? ";ST$:
   PRINT D$;"BLOAD";ST$
40 VTAB 23:PRINT "P/MUDAR O CURSO PRESSIONE
   <RETURN> + REPT"
50 HGR:ROT =0:SCALE=3:HPLOT 20,20
60 XDRAW 1:IF PEEK (-16384) > 127 THEN
   POKE-16368,0:GOTO 80
70 XDRAW 2:GOTO 60
80 XDRAW 2:R=R+1:IF R > 255 THEN R=0
90 ROT=R:GOTO 60
```

Na linha 60 a primeira definição de forma é desenhada (XDRAW) seguida pela definição de forma 2 na linha 70. Quando a primeira definição de forma for desenhada (XDRAW) novamente, o desenho aparecerá um ponto à direita da sua posição anterior, pois a segunda definição de forma é mais longa. Isto cria o efeito de um objeto em movimento. Na linha 60 o teclado é monitorado por PEEK na posição de memória -16384. Se uma tecla for pressionada, seu valor passa a ser maior que 127 mas menor que 255, cada número indicando uma determinada tecla. O valor da posição acima permanecerá o mesmo até que outra tecla seja pressionada ou um comando seja enviado ao computador para liberar o conteúdo da posição. Esta última ação pode ser obtida referenciando a posição de memória -16368 (como na linha 60) que faz com que o conteúdo da posição -16368 caia abaixo de 127. Desta forma podemos interromper o computador simplesmente pressionando qualquer tecla, e passando o controle do programa para uma outra linha (no nosso caso linha 80), onde R é aumentado, modificando assim a rotação da forma.

ANEXO A

Resumo dos comandos de acesso e comandos DOS

Comandos de acesso

Os seguintes comandos permitem o acesso ao sistema.

s'CTRL'P	Chama o sistema (BOOT) do encaixe (SOQUETE) *s* para todas as versões do Apple II, exceto a versão de acionamento automático, que irá chamar o sistema automaticamente se o disquete estiver no drive, antes de se ligar a força. Se o Applesoft tiver de ser carregado em RAM e estiver armazenado no disquete em uso, o comando acima irá carregá-lo automaticamente.
IN#*s*	Recebe entrada (INPUT) do encaixe (soquete) *s*, e se estiver conectado a uma unidade de disco (DRIVE) o sistema será "rechamado" (REBOOTED).
PR#s	Envia saída (OUTPUT) ao soquete *s*.
PR#0	Comanda saída à tela.
FP	Coloca o sistema em BASIC Applesoft.
INT	Coloca o sistema em Integer BASIC.
RESET	Todas as versões do Apple II entram em monitor, exceto o Apple II Plus que inicia um ciclo de recomposição retornando ao Applesoft BASIC.
CALL – 151	Todas as versões do Apple II, incluindo o Apple II Plus, entram em monitor.
3D0G	Reentra na linguagem que estava em uso antes de entrar em monitor para todas as versões do Apple II, exceto o Apple II Plus.
'CTRL' C	Reentra na linguagem que estava em uso antes de entrar em monitor para a versão Apple II Plus.

Comandos DOS

No seguinte resumo dos comandos DOS, só os parâmetros compulsórios dos comandos são especificados. Todos os parâmetros opcionais ou parte deles (veja a próxima seção) podem seguir os parâmetros mostrados.

CATALOG	Exibe os nomes dos arquivos armazenados num disquete.
INIT X	Inicializa um disquete em branco e atribui-lhe X como nome de arquivo de apresentação.

LOAD X	Carrega o arquivo BASIC X do disco na memória RAM do computador.
RUN	Executa o programa correntemente na memória RAM do computador.
RUN X	Carrega o arquivo BASIC X do disco na memória RAM do computador e executa o programa.
SAVE X	Salva o programa BASIC corrente na memória RAM do computador em disco sob o nome de arquivo X.
VERIFY X	Verifica se o último programa salvado, ainda na memória, foi corretamente transferido para o disco.
LOCK X	Trava o arquivo X para que não seja apagado acidentalmente. Um arquivo travado possui um asterisco ao lado de seu nome no catálogo do disquete.
UNLOCK X	Destrava o arquivo X.
RENAME X,Y	Troca o nome do arquivo de X para Y. Um arquivo travado não pode ser renomeado.
DELETE X	Apaga o arquivo X do disquete. Um arquivo travado não pode ser apagado.
BLOAD X	Carrega o arquivo binário X do disquete na memória RAM e executa o programa.
BSAVE X,Aa,Ll	Salva em disco programa binário corrente na memória RAM sob o nome de arquivo X, com endereços de partida dados pelo decimal 'a' (entre 1 e 65535), e o comprimento dado pelo decimal 'l' (entre 1 – 32767).

Parâmetros dos comandos DOS

Os comandos DOS podem ser seguidos de uma lista de parâmetros opcionais, mostrados entre colchetes abaixo. Por exemplo:

```
LOAD  'nome de arquivo' [,Ss] [,Dd] [,Vv]
```

onde S, D e V significam soquete, DRIVE e Volume e as letras minúsculas s, d e v representam números que precisam ser fornecidos pelos usuários.

Se o comando DOS não for seguido de algo, o primeiro item precisa ser o nome do arquivo. Outros parâmetros devem estar separados do nome de arquivo e entre si por vírgula, como mostrado acima.

A ordem dos parâmetros de opção que seguem o comando DOS não é importante. As letras maiúsculas e vírgulas no comando devem ser digitadas como mostrado, enquanto que as letras minúsculas (que representam números) são fornecidas pelo usuário de acordo com a seguinte lista:

s é o número do soquete (ou encaixe) a que o cartão de controle da unidade de disco (DRIVE) está conectado. Este número deve estar entre 1 e 7. O soquete usual é o número 6 que é também o valor assumido se a opção s não estiver contida (DEFAULT) no comando.

d é o número do drive com o qual se deseja comunicar. Este número deve estar entre 1 e o número máximo de drives disponíveis em seu sistema. Inicialmente é assumido o valor 1 quando da chamada do sistema (BOOT), mas de imediato é assumido o drive mais recentemente especificado.

v é o número de volume designado ao disquete durante a inicialização. Deve estar na faixa de 0 a 254. O valor assumido, caso este não seja especificado, é o do volume do disquete utilizado para chamar o sistema (BOOT), geralmente 254, pois este é o número de volume do disquete mestre. Um disquete não deve ser inicializado com o número de volume 0 (zero). VO é um comando DOS que faz com que o DOS procure pelo volume do disquete não especificado e use-o.

Por exemplo:

```
LOAD TESTE,S6,D1,V0
```

irá carregar o arquivo TESTE de um disquete de volume não especificado que está no drive 1, cujo cartão de controle está conectado ao soquete 6. Todos os parâmetros usados acima são os valores de "DEFAULT" *, o comando é análogo a LOAD TESTE.

Comandos DOS a partir do BASIC

Em todos os exemplos dados a seguir, supomos que cada comando é precedido por PRINT D$, onde D$ foi atribuída a CHR$(A), e que o nome do arquivo tenha sido atribuído à variável string X$. É suposto também que os números de linha apropriados aparecem na frente de cada instrução em todos os exemplos que se seguem.

Arquivos-texto seqüenciais

"OPEN";X$	Faz com que o arquivo especificado seja aberto. Se este não existir, um novo será criado.
"READ":X$	Faz com que todas as instruções GET e INPUT subseqüentes obtenham suas respostas do arquivo aberto designado.
"WRITE";X$	Faz com que todas as instruções PRINT subseqüentes sejam desviadas para o arquivo aberto designado.
"APPEND";X$	Faz com que todas as instruções PRINT agreguem informações ao arquivo aberto designado.
"DELETE";X$	Apaga o arquivo aberto determinado.
"CLOSE";X$	Fecha arquivo especificado.
"POSITION";X$;",R";RF	Faz com que a próxima instrução READ, ou WRITE, mova RF campos para a frente em relação a sua presente posição.
"READ";X$;",B";BY	Faz com que a leitura comece no BY-ésimo byte.
"WRITE";X$;",B";BY	Faz com que a gravação comece no BY-ésimo byte.

Arquivos de acesso randômico

Os arquivos de acesso randômico requerem mais dois parâmetros além dos usados anteriormente. São eles:

* Vide nota do tradutor.

L o comprimento fixo para cada registro do arquivo, o qual deve ser especificado quando 'abrir' o arquivo, e

R o r-ésimo registro que deve ser especificado em todas as instruções READ e WRITE, como se segue:

```
"OPEN";X$;",L";RL
"READ";X$;",R";RN
"WRITE";X$;",R";RN
"DELETE";X$;",R";RN
"CLOSE";X$
```

Tais instruções farão com que o arquivo designado seja aberto, com cada registro especificado pelo parâmetro de comprimento L tendo RL bytes de comprimento, e a leitura, gravação ou destruição do RN-ésimo registro. Em caso de omissão do valor R, será assumido 0 (DEFAULT=0).

Se L for omitido do comando OPEN de um arquivo de acesso randômico, o DOS assume que tal arquivo seja seqüencial. Este fato é muito útil para acessar o 0-ésimo registro no qual, geralmente, se armazena o próprio valor de L.

A opção de parâmetro do byte também pode ser usada com as instruções READ, WRITE ou DELETE de forma que informações possam ser lidas, escritas ou apagadas a partir do byte especificado do RN-ésimo registro, como se segue:

```
"READ";X$;",R";RN;",B";BY
```

O valor de B, em caso de omissão, é assumido como sendo 0.

ANEXO B

Mensagens de erro do APPLESOFT e do DOS

TABELA B-1 Códigos de Erro do Applesoft

Código	*Mensagem*	*Código*	*Mensagem*
0	NEXT WITHOUT FOR	16	SYNTAX ERROR
22	RETURN WITHOUT GOSUB	42	OUT OF DATA
53	ILLEGAL QUANTITY	69	OVERFLOW
77	OUT OF MEMORY	90	UNDEF'D STATEMENT
107	BAD SUBSCRIPT	120	REDIM'D ARRAY
133	DIVISION BY ZERO	163	TYPE MISMATCH
176	STRING TOO LONG	191	FORMULA TOO COMPLEX
224	UNDEF'D FUNCTION	254	BAD INPUT RESPONSE
255	CTRL C ATTEMPTED		

TABELA B-2 Códigos de Erro do DOS

Código	*Mensagem*	*Código*	*Mensagem*
1	LANGUAGE NOT AVAILABLE (LING. NÃO DISPONÍVEL)	2,3	RANGE ERROR (FAIXA ERRO)
4	WRITE PROTECTED (DISCO PROTEGIDO)	5	END OF DATA (FIM DE ARQUIVO)
6	FILE NOT FOUND (NÃO HÁ ARQUIVO)	7	VOLUME MISMATCH (VOLUME ILEGAL)
8	I/O ERROR (E/S ERRO)	9	DISK FULL (< LOTADO)
10	FLE LOCKED (< TRAVADO >)	11	SINTAX ERROR (GRAFIA-ERRO)
12	NO BUFFERS AVAILABLE (TODOS BUFFERS CHEIOS)	13	FILE TYPE MISMATCH (TIPO ARQ. ILEGAL)
14	PROGRAM TOO LARGE (PROGRAMA LONGO)	15	NOT DIRECT COMMAND (COMANDO NÃO DIRETO)

A maioria das mensagens do DOS são auto-explicativas, exceto as seguintes:

LINGUAGEM NÃO DISPONÍVEL	Causada por não estar disponível no disco o Applesoft ou o INTERGER BASIC.
FAIXA ERRO	causada por um parâmetro de um comando fora de faixa.
FIM DE ARQUIVO	causada ao se tentar ler além do fim de um arquivo-texto.
VOLUME ILEGAL	causada pelo uso do volume errado de disco.

E/S ERRO	causada por uma porta aberta de drive, um disquete danificado ou em branco.
TODOS OS BUFFERS CHEIOS	causada por haver muitos arquivos abertos simultaneamente.
TIPO DE ARQUIVO ILEGAL	causada ao tentar carregar ou rodar um arquivo de diferente tipo do especificado no comando.
COMANDO NÃO DIRETO	causada ao tentar usar em modo direto um comando que deve fazer parte de um programa, como OPEN, WRITE ou CLOSE.

ANEXO C

Mensagens de erro

Observação: Estas mensagens são emitidas pelo microcomputador UNITRON. A maioria dos microcomputadores emite-as em inglês. Caso isto não aconteça, recorra ao manual do usuário.

Todas as mensagens de erro que podem ser geradas na linguagem "basic" estão listadas a seguir, com seus respectivos significados.

Quando ocorre um erro, o basic volta ao nível de comando, indicado pelo caractere "prompt" (1) e o cursor intermitente.

Os valores das variáveis e o texto do programa permanecem intactos, mas o programa não pode CONTinuar e todos os contadores de "loop" de GOSUB e FOR são zerados.

Quando ocorre um erro num comando de execução imediata, não aparece número de linha.

Formato das mensagens de erro:

Comando de execução imediata ?XX – ERRO

Comando de execução no programa ?XX – ERRO EM YY

Em ambos os casos exemplificados, "XX" é o nome do erro, e "YY" é o número da linha de programa onde o erro ocorreu. Erros no programa não são detectados até que o programa seja executado.

São os seguintes os possíveis erros e seus significados:

?S/ CONTINUACAO – ERRO

Tentativa de continuar um programa quando não existe nenhuma continuação, ou depois que ocorreu um erro, ou depois que uma linha foi apagada ou adicionada ao programa.

?DIVISAO POR ZERO – ERRO

Dividir por zero é um erro!

?FORMULA COMPLEXA – ERRO

Mais de dois comandos do tipo IF "XX" THEN foram executados.

?DIRETI ILEGAL – ERRO

Você não pode usar INPUT, DEF FN, GET ou DATA como comando de execução imediata.

?QUANTI ILEGAL – ERRO

O parâmetro, passando por uma função matemática ou string, cai fora da faixa de alcance.

Pode ocorrer devido a:

a) SUBSCRIPT negativo (Ex.: A(−1)=0)
b) Usar LOG com argumento 0 ou negativo
c) Usar SQR com argumento negativo
d) A ^ B com A negativo e B não-inteiro
e) Uso de MD$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC, ON ... GOTO, ou qualquer função gráfica com argumento não apropriado.

?"NEXT" SEM "FOR" −ERRO

A variável num comando NEXT não corresponde à variável num comando FOR que ainda estava válido.

?FALTA DADOS −ERRO

Um comando READ foi executado, mas todos os comandos DATA no E-2 programa já foram lidos. O programa tentou ler muitos dados ou foram fornecidos poucos dados a ele.

?FALTA MEMORIA −ERRO

Pode ser causado por: programa muito extenso; muitas variáveis; mais de 10 "loops" de FOR; mais de 24 comandos GOSUB; expressão muito complexa; mais de 36 níveis de parênteses; tentativa de comandar LOMEM: menor que o valor presente; tentativa de comandar HIMEM: muito baixo.

?S/ESPACO −ERRO

O resultado de um cálculo foi muito grande para ser apresentado no formato de números do BASIC. Se o erro ocorre por ser o número muito pequeno (muito próximo a 0), o zero é dado como resultado e a execução continua sem que seja apresentada mensagem de erro.

?REDIMENSIONAR −ERRO

Depois que um array foi dimensionado, outro comando de dimensão, para o mesmo array, foi encontrado. Este erro geralmente ocorre se é dada a um array a dimensão errada 10 devido a que um comando como A(I)=3 é seguido mais tarde num programa por DIM A(100). Esta mensagem de erro mostrar-se-á útil se você desejar descobrir em que linha do programa um certo array foi dimensionado: coloque a dimensão do array em questão na primeira linha, rode o programa, e o BASIC lhe dirá onde está o comando original de dimensão.

?"RETURN" SEM "GOSUB" −ERRO

Um comando RETURN foi encontrado sem que haja o correspondente GOSUB.

?"STRING" LONGO – ERRO

Foi feita uma tentativa, através de um comando de concatenação, para criar um "string" com mais de 255 caracteres.

?SUBSCRITO – ERRO

Foi feita uma tentativa de consultar um elemento de um array que está fora das dimensões do mesmo. Este erro pode ocorrer se o número errado da dimensão é usado numa referência a um array; por exemplo, LET A(11)=Z quando A foi dimensionado usando DIM A(2).

?GRAFIA – ERRO

Falta de parênteses numa expressão, caractere ilegal numa linha, pontuação incorreta etc.

?DIFERE O TIPO – ERRO

O lado esquerdo de um comando foi uma variável numérica e o lado direito foi um "string", ou vice-versa; ou a uma função que esperava um argumento de "string" foi dado um número, ou vice-versa.

?DECLARACAO INDEF. – ERRO

Tentativa de GOTO, GOSUB ou THEN a um número de linha que não existe.

?FUNCAO INDEF. – ERRO

Refere-se a uma função criada pelo usuário, que ainda não foi definida.

ANEXO D

Técnicas para depuração de erros

Existem várias técnicas, à disposição dos programadores, que podem ser usadas para isolar erros de programa (conhecidos como "bugs"). Tais bugs podem ter sido colocados em um programa seja pressionando uma tecla errada ao digitá-lo, seja por erro de lógica. Examinaremos ambos os casos e indicaremos técnicas adequadas para depurar tais "bugs".

(a) Erros de sintaxe

Alguns erros são revelados logo que a tecla RETURN é pressionada, enquanto outros não são constatados até que o programa seja executado. Para ilustrar, digite as seguintes linhas com os erros indicados (em grifo)

```
10 REN USO DA INSTRUÇÃO ON–GOTO
20 INPUT "NUMERO POR FAVOR" ";N
END
```

No instante que pressionar RETURN, após a última entrada, o BASIC responderá com

```
SINTAX ERROR (GRAFIA ERRO)
```

que é devido à falta do número de linha antes da instrução END. Redigite esta linha, mas introduza outro erro, como se segue:

```
30 END
```

onde após a letra D, a tecla CTRL, junto com a letra A, são pressionadas 'CTRL'A (ou qualquer outra letra no lugar de A) resultando em uma não impressão do caractere, de forma que nada errado aparece na tela. Ao rodar este programa, o BASIC responderá com

```
SINTAX ERROR IN 10 (GRAFIA ERRO EM 10)
```

e ao listar 10 deparará com

```
10 RENUSODAINSTRUCAO ON – GOTO
```

Porque a linha 10 não é reconhecida como uma instrução de lembrete (comentário (REM)), o BASIC tenta interpretá-la como uma atribuição BASIC no que resultou na separação das palavras-chave (ON e GOTO) do resto da linha. No entanto, como o resultado ainda estava irreconhecível, uma mensagem de erro foi emitida.

Redigite a linha 10 corretamente e re-rode o programa. Na execução, o BASIC pede por um número e, após recebê-lo responde com

```
SINTAX ERROR IN 30 (GRAFIA ERRO EM 30)
```

que é provocado pela presença do caractere 'CTRL'A que foi digitado, mas não impresso, no final da última instrução. Este tipo de erro pode ser confuso, pois não é de imediato evidente. Use o código ESC I,J,K,M para copiar a linha, visto que ele copia tudo exceto os caracteres de não impressão que podem ter sido colocados acidentalmente. Quando

usar o código ESC I,J,K,M para copiar linhas, você deve tomar cuidado para não copiar o sinal indicativo de mensagem do Applesoft, do contrário receberá outro erro de sintaxe na execução.

Outros erros, particularmente em programas longos, podem resultar em outras mensagens de erro que não seja "SINTAX ERROR"*. Estas mensagens estão listadas nas Tabelas B-1 e B-2 no Anexo B, com seus respectivos números de código. Cada erro resulta em um número de código que é mantido na posição de memória 222 e está na faixa de 0-255. Para descobrir o número de código de um erro, digite

```
PRINT PEEK(222)
```

Isto é particularmente útil no caso em que um erro fez com que o sistema parasse (não respondendo a qualquer indagação externa, tal como pressionar RETURN de qualquer outra tecla). Neste caso, pressione RESET, e uma vez de volta ao BASIC, chame (PEEK) o local de memória 222 para descobrir que erro gerou o problema.

À medida que você progride em programação, e começa a usar comandos que fazem com que INPUT e OUTPUT (entrada e saída) sejam passadas para o disquete, verá que é útil monitorar o que se passa. Isto pode ser obtido digitando-se

```
MON C,I,O
```

antes de executar tal programa. O comando significa monitorar comandos, entrada e saída (do inglês MONitor Comands, Input and Output). Para cancelar tal comando, digite

```
NOMON C,I,O
```

(b) Erros de lógica

Erros resultantes de má lógica podem causar uma imensa perda de tempo, a não ser que você aprenda como investigar o seu programa. Tais erros podem às vezes fazer o sistema parar ou resultar em respostas totalmente inesperadas. Em ambos os casos, você pode trilhar através dos passos da execução de seu programa digitando

```
TRACE
```

antes de rodar o programa. Isto causará a impressão do número de linha de cada instrução na tela, à medida que elas forem sendo executadas, para que possa seguir (trilhar) o programa até a reta onde a falha de lógica começa. Os números de linha aparecem na tela precedidos do símbolo I. Para cancelar o comando TRACE, digite

```
NOTRACE
```

Atenção: Enquanto TRACE estiver ativado, todos os comandos dirigidos ao disquete não são executados. Logo, se seu problema estiver nesta área, ele não será constatado por este método.

Existem mais três comandos que podem ser usados para conferir o desenvolvimento de um programa. São eles:

```
END, STOP e 'CTRL'C
```

que interrompem a execução sem perder os valores corretamente armazenados nas variáveis. Os dois primeiros comandos podem ser dispostos em pontos estratégicos do programa (vários deles podem ser inseridos em um mesmo programa) para fazer com que o

* Grafia erro.

computador interrompa a execução em um determinado ponto. O uso do comando END irá terminar com a execução sem reportar o número da linha em que isto ocorreu, enquanto que o comando STOP pára a execução naquele ponto e também reporta o número da linha em que isto ocorreu. Se o comando foi colocado, por exemplo, na linha 150, o BASIC irá responder com

```
BREAK 150
```

O comando 'CTRL'C pode ser usado enquanto um programa imprime resultados, caso este em que irá reportar o número da linha em que a execução foi interrompida, da mesma forma que no uso do comando STOP. Tendo interrompido a execução de um programa usando qualquer um destes três comandos, pode-se reiniciá-lo a partir do ponto em que foi interrompido, digitando-se

```
CONT
```

que significa 'continue'. Se um erro de sintaxe for encontrado antes de usar o comando CONT, em conseqüência de erro de grafia do próprio comando CONT, o seu uso subseqüente resultará em um

```
CAN'T CONTINUE ERROR (NAO POSSO CONTINUAR)
```

como resposta.

Finalmente, tendo interrompido a execução do programa, você pode examinar o conteúdo das variáveis digitando

```
PRINT X,Y
```

em modo imediato, onde X e Y são as variáveis que você deseja examinar. A execução do programa pode continuar, digitando-se o comando CONT, após você ter se certificado de que as variáveis que examinou contêm os valores esperados.

Finalmente, para trilhar (TRACE) erros de lógica, você tem de efetivamente trabalhar através do programa passo a passo (um fluxograma é de grande utilidade), com lápis e papel, usando variáveis que lhe forneçam respostas conhecidas. Afinal de contas, pode haver grande diferença entre o que você pensa que o programa está fazendo e o que ele realmente está fazendo.

O programa seguinte contém erros de lógica que são interativos, causando formatação incorreta das informações impressas na tela. Ele é dado de tal forma que você possa praticar o rastreamento da lógica através de problemas e adquirir alguma experiência em detectar erros. Este problema só deve ser tentado após ter completado todo o Capítulo 5, pois ele usa conceitos ali desenvolvidos.

Problema D-1

Encontre e corrija os erros de lógica do programa abaixo.

```
 10   REM ESTE PROGRAMA CONTEM ERROS QUE
 20   CAUSAM FORMATACAO INCORRETA NA SAIDA
 30   DEF FN Y(N)=12000*1.125↑N
 40   PRINT "N";TAB(10);"I";TAB(26);"A":PRINT
 50   FOR N=1 TO 15:A=FN Y(N):I= A-12000
 60   SN=A:SD=2:GOSUB 10000:A$=SR$
 70   SN=1:SD=2:GOSUB 10000:I$=SR$
 80   SZ=N:SC=3:GOSUB 11070:N$=SR$
 90   PRINT N$;TAB(10);I$;TAB(26);A$:NEXT N
100   END
10000 REM SUB P/ARREDONDAR NUMEROS
10010 SR=INT(10↑SD*SN+0.5)/10↑SD
10020 SZ=SR:SC=5+SD:GOSUB 11000:RETURN
11000 REM SUB TO CONVERT NRS TO ###.## FORMAT E
      ALINHÁ-LOS
11010 Z$=STR$(SZ):LZ=LEN(Z$)
11020 FOR J=1 TO LZ:IF MID$(Z$,J,1)="." GOTO 11040:NEXT J
11030 Z$=Z$+".":SX=0:GOTO 11060
11040 SX=LEN(MID$(Z$,J+1))
11050 IF SX=SD GOTO 11080
11060 FOR K=1 TO SD-SX:Z$=Z$+"0":NEXT K
11070 Z$=STR$(SZ)
11080 SR$=RIGHT$("          "+Z$,SC):RETURN
```

ANEXO E

O mapa de memória do APPLE

Representação de dados numéricos

Informações numéricas são armazenadas em computadores na forma de grupos de dígitos binários (bits, para abreviar), isto é, 0 e 1. Em quase todos os computadores atuais as informações são estruturadas em grupos de 8 bits (chamados bytes). Os números podem ser representados em formato binário direto em que o dígito mais à direita representa 2 elevado à potência 0 (zero). O próximo à esquerda, 2 elevado à potência 1, o próximo, 2 elevado à potência 2, e assim por diante, até chegarmos ao bit mais à esquerda que é 2 elevado à potência 7 para uma estrutura de 8 bits. Isto pode ser representado da seguinte forma:

$$2\uparrow 7 + 2\uparrow 6 + 2\uparrow 5 + 2\uparrow 4 + 2\uparrow 3 + 2\uparrow 2 + 2\uparrow 1 + 2\uparrow 0$$

Um número binário pode ser convertido do seu decimal equivalente multiplicando-se o valor do i-ésimo bit apropriado (que pode ser 0 ou 1) com o resultado de $2\uparrow i$. Por exemplo, o número binário 00010101 é equivalente a (0*128) + (0*64) + (0*32) + (1*16) + (0*8) + (1*4) + (0*2) + (1*1) = 21 decimal.

Pode-se demonstrar facilmente que com n bits disponíveis, números inteiros na faixa 0 a ($2\uparrow$ n-1) podem ser representados. Portanto,

4 bits (chamado nibble) podem representar a faixa 0-15
8 bits (chamado byte) podem representar a faixa 0-255
16 bits (chamado palavra) podem representar a faixa 0-65535

Observe o caso especial dos dígitos de 10 binários, que podem ir ao máximo 1024 decimal. Representamos isto pelo símbolo K. Então, um computador de 64K tem (64*1024-1) = 65535 posições de memória disponíveis.

O programa seguinte fornece toda a seqüência binária para um determinado número de bits.

```
10  REM GERADOR DE SEQUENCIAS BINARIAS
20  INPUT "ENTRE NUMERO DE BITS ";N:DIM T(N)
30  T1=2↑N-1:PRINT:PRINT"NUMERO MAX DE DECIMAIS PERMITIDO
    ";N;" BITS = ";T1:PRINT
40  PRINT "DECIM. NR";TAB(29);"NUMERO BINARIO":PRINT
50  FOR T2=0 TO T1:T5=T2:PRINT T5;TAB(20);
60  FOR I=N TO-1:T3=T5/2:T4=INT(T3):IF T3-T4<0.001 THEN
      T(I)=0:GOTO 80
70  T(I)=1
80  T5=T4:NEXT I
90  NEXT T2:END
```

Digite o programa e rode-o. Tente com 2, 4 e 8 bits para ver como números decimais são representados na forma binária. Se tiver um tempinho disponível tente o programa com 16 bits. Lembre-se de que em 16 bits existem 65536 combinações possíveis de 0s e 1s.

Problema E-1

Modifique o programa acima para que o binário equivalente a qualquer decimal seja calculado e impresso sem que nem seja necessário entrar o número de bits nem imprimir toda a seqüência binária.

O microprocessador 6502 do Apple é capaz de endereçar 65.536 posições de memória distintas começando em 0 até a posição 65.535. Como cada 1024 posições de memória são chamadas de 1K (1000 bytes) de memória, o Apple pode endereçar um total de 64K de memória. Destes, somente 48K estão disponíveis ao usuário, sendo o restante usado pelo próprio sistema.

Organização geral de memória

A memória do Apple é organizada em 256 partes chamadas 'páginas' (começando na página 0 até a 255) com cada página contendo 256 posições (0 a 255). Assim, para se referir a qualquer um dos 64K de memória, cada instrução é composta de três bytes de 8 bits. O primeiro byte descreve a operação a ser executada (conhecida como OPCODE) e os dois seguintes fornecem a posição absoluta da célula de memória, com os 8 bits baixos do endereço precedendo os 8 bits altos, como mostrado abaixo

OPCODE	BAIXO	ALTO

Os bits ALTOS indicam que página de memória está sendo acessada, enquanto que os bits BAIXOS indicam o byte desejado da página. Ter os 8 bits BAIXOS precedendo os 8 ALTOS não é convencional, mas possui as suas vantagens para acessar a página 0 onde o byte ALTO é desnecessário, como mostrado abaixo.

OPCODE	BAIXO

Este encurtamento da instrução faz com que a procura de informações na página 0 possa ser executada com maior rapidez que nas demais. Com isto, a página 0 passa a ser a página ideal para armazenar sub-rotinas importantes do sistema.

As posições de memória que vão da base da página 0 (zero) até o topo da página 191 estão alocadas na memória de acesso randômico (RAM), enquanto que as posições entre as páginas 192 e 207 estão alocadas para as operações de INPUT/OUTPUT (I/O). Finalmente, as páginas 208 a 255 estão reservadas para a memória só de leitura (ROM) – Read Only Memory).

Podemos representar a memória do Apple conforme mostra a Fig. E.1.

A memória I/O de 4K que está entre o 48º e 52º K byte é igualmente dividida com 2K em ROM e 2K em RAM.

Organização da memória RAM

Na página 0 ficam o *M*onitor, o interpretador BASIC (Integer e Applesoft) e o *S*istema *O*peracional em *D*isco (DOS).

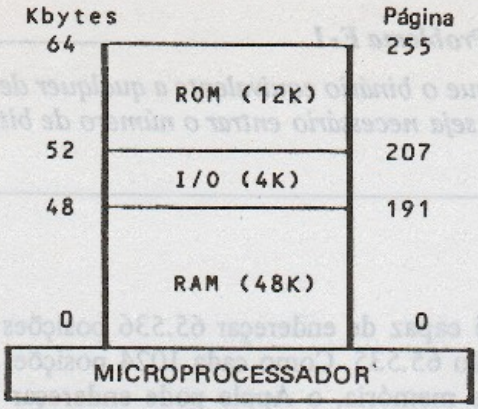


Fig. E-1 Representação das memórias RAM e ROM do Apple.

A página 1 é onde a próxima instrução a ser executada pelo microprocessador é armazenada.

As páginas 2 a 7 estão reservadas para textos e gráficos de baixa resolução, para o sistema BASIC e para o uso de cartões de 'interface' para periféricos.

O mapa da memória de 48K do sistema Apple II com Applesoft BASIC em "firmware" (seja com ROM ou num cartão) é mostrado em diagrama na Fig. E-2.

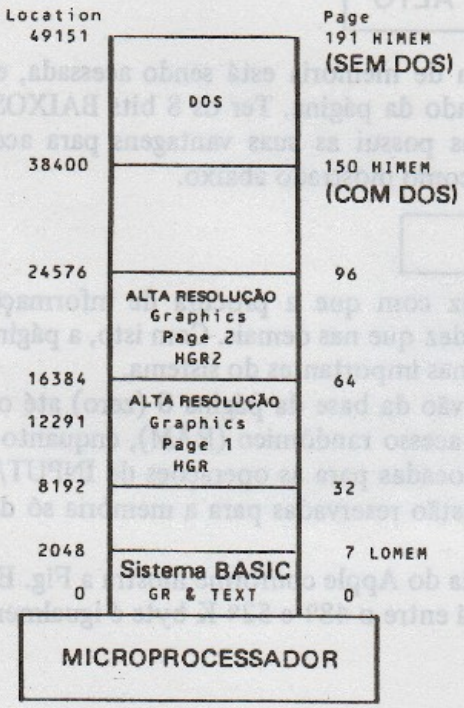


Fig. E-2 Mapa da memória RAM do Apple.

O endereço de memória RAM mais baixo disponível ao usuário (chamada LOMEM-low memory) está acima do topo da página 7, ou na posição decimal 8x256= 2048. Esta posição LOMEM é estabelecida automaticamente pelo Applesoft BASIC "firmware".

O endereço mais alto disponível ao usuário (HIMEM-highest memory) é estabelecido pelo Applesoft BASIC e depende do tamanho do sistema, como mostrado na Tabela E-1. Entretanto, no momento em que o DOS é chamado (BOOT), o HIMEM é "resetado" pelo DOS numa posição 10752 bytes mais baixa que a posição mais alta possível do sistema.

TABELA E-1 HIMEM valores

Sistema	*HIMEM estabelecido pelo APPLESOFT*	*HIMEM estabelecido pelo DOS*
48 K	49151	38400
36 K	36863	26112
32 K	32767	22016
24 K	24575	13824

Quando um programa BASIC do usuário é carregado na memória RAM do Apple, a LOMEM é elevada pelas linhas do programa à área usada para gráficos de alta resolução. Um programa longo irá inicialmente elevar a LOMEM à página 1 de gráficos (posições 12291 a 16383) que é acionada pelo comando HGR, enquanto que um programa BASIC ainda maior poderá elevar a LOMEM ainda mais adiante na área de gráficos da página 2 (posições 16384 a 24575), que é acionada pelo comando HGR2.

Variáveis numéricas são armazenadas a partir da posição corrente da LOMEM (estabelecida pelo programa BASIC do usuário), enquanto que as variáveis string são armazenadas em posições que começam em HIMEM e vão decrescendo.

Você pode descobrir os valores correntes da LOMEM e HIMEM digitando

```
PRINT PEEK(105)+PEEK(116)*256
```

e

```
PRINT PEEK(115)+PEEK(116)*256
```

respectivamente.

Para descobrir o espaço disponível após ter armazenado um programa e ambos os tipos variáveis, digite

```
PRINT FRE(0)
```

Observação: Se a resposta para qualquer um dos comandos acima for um número negativo, adicione simplesmente ao seu valor 65536 para obter o decimal positivo equivalente a tal número.

No caso de Applesoft BASIC em RAM (não em "firmware"), a LOMEM é elevada para a posição 1291 (Veja Fig. E-2) que é o meio da página 1 de gráficos de alta resolução. Como o comando HGR limpa automaticamente a área de memória entre as

posições 8192 e 16383, o seu uso afetará o Applesoft BASIC e o sistema irá falhar ("cair"). É por este motivo que o comando HGR2 deve ser usado ao invés de HGR.

A fim de evitar que programas invadam a área de gráficos de alta resolução na página 2, podemos estabelecer o HIMEM a partir de um programa BASIC para qualquer posição desejada, com o comando

```
HIMEM:16384
```

Isto nos permite utilizar aproximadamente 4K de memória (16384-12291) para carregar nossos programas BASIC e armazenar os dois tipos de variáveis. Contudo, acima da página 2 de gráficos de alta resolução existe também algum espaço disponível, começando na posição 24577 e estendendo-se à posição do HIMEM estabelecida pelo DOS. Entretanto, este espaço é variável e depende do tamanho da memória do seu sistema. A Tabela E-2 fornece o tamanho disponível de memória acima e abaixo da página 2 para diferentes tamanhos de sistemas.

TABELA E-2 Espaço disponível na memória

Sistema	*Acima da página 2*	*Abaixo da página 2*
48 K	13.5 K	4 K
36 K	1.5 K	4 K
32 K	———	4 K

Fica claro então que, em sistemas com 32K, gráficos em alta resolução não podem ser usados. Com o sistema de 36K, para se usar HGR2, o HIMEM deve ser fixado em 16384 que dará 4K de espaço disponível para um programa BASIC mais variáveis. Por outro lado, em um sistema com 48K é melhor fixar a LOMEM em 24577 e deixar que o HIMEM seja fixado pelo DOS na sua posição usual de 38400. Isto resulta em aproximadamente 13,5K de memória de trabalho para um programa BASIC.

ANEXO F

Resumo das características de edição

Entrar no modo de edição	Tecle [ESC]
Sair do modo de edição	Tecle barra espaço
Mover o cursor	Tecle [I] [J] [K] ou [M]
Eliminar um carácter	Tecle [←]
Reescrever um carácter	Tecle [→]
Limpar do cursor até o final da linha	Tecle [ESC] então [E]
Limpar do cursor até o final da tela	Tecle [ESC] então [F]
Limpar a tela toda	Tecle [ESC] então [SHIFT] e [P]
Parar a listagem	Tecle [CTRL] e [S]
Continuar a listagem	Tecle [CTRL] e [S]

ANEXO G

Códigos de conversão ASCII

Tabela G-1 Códigos ASCII de conversão decimal

Car.	*abr dec*	*Car.*	*abr dec*	*Car.*	*abr dec*
Códigos com chave CTRL					
CTRL @	nul 0	CTRL J	lf 10	CTRL T	dc4 20
CTRL A	soh 1	CTRL K	vt 11	CTRL U	nak 21
CTRL B	stx 2	CTRL L	ff 12	CTRL V	syn 22
CTRL C	etx 3	CTRL M	cr 13	CTRL W	etb 23
CTRL D	et 4	CTRL N	so 14	CTRL X	can 24
CTRL E	enq 5	CTRL O	si 15	CTRL Y	em 25
CTRL F	ack 6	CTRL P	dle 16	CTRL Z	sub 26
CTRL G	bel 7	CTRL Q	dc1 17	ESC	escape 27
CTRL H	bs 8	CTRL R	dc2 18	CTRL'ST'M	gs 29
CTRL I	ht 9	CTRL S	dc3 19	CTRL ↑	rs 30
Códigos do conjunto de caracteres					
espaço	32	4	52	H	72
!	33	5	53	I	73
"	34	6	54	J	74
#	35	7	55	K	75
$	36	8	56	L	76
%	37	9	57	M	77
&	38	:	58	N	78
'	39	;	59	O	79
(	40	<	60	P	80
)	41	=	61	Q	81
*	42	>	62	R	82
+	43	?	63	S	83
,	44	@	64	T	84
—	45	A	65	U	85
.	46	B	66	V	86
/	47	C	67	W	87
0	48	D	68	X	88
1	49	E	69	Y	89
2	50	F	70	Z	90
3	51	G	71	] *sft-M*	93
				↑	94

Observação: Os códigos 28, 31, 91, 92 e 95 não são acessíveis pelo teclado do Apple II. Os códigos na faixa de 96 a 127 repetem os caracteres fornecidos pelos códigos 32 a 63, enquanto que os códigos entre 128 e 255 repetem novamente toda a série. O grupo de duas ou três letras minúsculas são abreviações de caracteres de controle ASCII standard.

Códigos de ESCape

Todos os códigos de escape dados abaixo deixam a entrada ao computador inalterada.

ESC A	Move cursor um espaço para a direita
ESC B	Move cursor um espaço para a esquerda
ESC C	Move cursor uma linha para baixo
ESC D	Move cursor uma linha para cima
ESC E	Apaga todos os caracteres da linha onde se encontra o cursor e o coloca no fim da mesma
ESC F	Apaga todos os caracteres na tela após a posição corrente do cursor
ESC @	Limpa a tela e coloca o cursor no canto superior esquerdo da tela.

Códigos de ESCape adicionais para ROM com partida automática

Para os códigos dados abaixo, a tecla ESC só precisa ser pressionada uma única vez. A execução do código ESC não desativa o modo ESC, efetuando portanto movimentos sucessivos de cursor pressionando-se repetidamente a tecla ESC. O modo ESC só é desativado ao se pressionar qualquer tecla que não seja ESC.

ESC	Entra em modo ESC.
I	Move uma linha para cima.
J	Move um caractere à esquerda.
K	Move um caractere à direita.
M	Move uma linha para baixo.

Anexo H
Soluções dos problemas

As soluções dos problemas apresentados neste anexo, juntamente com todos os programas utilitários usados neste livro, podem ser obtidos em disco escrevendo para o autor na "Camborne School of Mines, Pool, Redruth, Cornwall".

Problema 1.1

```
  5 REM PROBLEMA 1.1
 10 REM MEDIA
 20 A = 26.9
 30 B = 28.2
 40 C = 27.3
 60 D = A + B + C
 70 X = D / 3
 80 PRINT "VALORES: A B C MEDIA"
 85 PRINT TAB( 8);A; TAB( 14);B; TAB( 20);C; TAB( 26);X
 90 END
```

Problema 1.2

```
  5 REM PROBLEMA 1.2
 10 REM CONVERSAO TEMP.
 20 INPUT "ENTRE VALOR ";F
 30 C = (F – 32) * 5 / 9
 40 PRINT F;" F = ";C;" C "
 50 END
```

Problema 1.3

```
  5 REM PROBLEMA 1.3
 10 REM MINUTOS
 20 PRINT "DIAS"; TAB( 10);"HORAS "; TAB( 20);"MINUTOS"; TAB( 30); "TOTAL
    MIN"
 30 READ D,H,M
 40 IF D < 0 GOTO 120
 50 T = D * 24 * 120 + H * 120 + M
 60 PRINT D; TAB( 10);H; TAB(20);M; TAB( 30);T
 70 GOTO 30
 80 DATA 2,10,30
 90 DATA 4,20,25
100 DATA 8,23,5
110 DATA –1,–1,–1
120 END
```

Problema 1.4

```
  5 REM  PROBLEMA 1.4
 10 REM PRIORIDADES
 20 READ A,B,C,D,E,F,G
 30 DATA 5,3,8,4,7,2,6
 40 X1 = A * B ↑ E + F
 50 X2 = A * B ↑ (E + F)
 60 X3 = A * B / C * D
 70 X4 = A * B / (C * D)
 80 X5 = A + B * G + C * G ↑ 2 + D * G ↑ 3
 90 X6 = (A + B) * G + C * G ↑ 2 + D * G ↑ 3
100 X7 = (A ↑ F + (B – 1 / C) ↑ F) ↑ 0.5
110 X8 = (A ↑ F + B – 1 / C ↑ F) ↑ 0.5
120 X9 = A / B ↑ 2 – C * D / ((E + F) + G ↑ 3)
130 PRINT "X1"; TAB( 14);"X2"; TAB ( 28);"X3"
140 PRINT X1; TAB( 14);X2; TAB( 28);X3
150 PRINT "X4"; TAB( 14;"X5"; TAB( 28);"X6"
160 PRINT X4; TAB( 14);X5; TAB( 28);X6
170 PRINT "X7"; TAB( 14);"X8"; TAB( 28);"X9"
180 PRINT X7; TAB( 14);X8; TAB( 28);X9
```

Problema 2.1

```
  5 REM PROBLEMA 2.1
 10 REM CLASS 3 NUM.
 20 PRINT
 30 INPUT A,B,C
 40 PRINT
 50 IF A < 0 THEN END
 60 IF A > B AND B > C THEN PRINT A,B,C
 70 IF A > B AND B > C GOTO 20
 80 IF A > B AND B < C GOTO 120
 90 IF B > C AND A > C THEN PRINT B,A,C
100 IF B > C AND A > C GOTO 20
110 GOTO 160
120 IF A > C THEN PRINT A,C,B
130 IF A > C GOTO 20
140 PRINT C,A,B
150 GOTO 20
160 IF B > C AND A < C THEN PRINT B,C,A
170 IF B > C AND A < C GOTO 20
180 PRINT C,B,A
190 GOTO 20
```

Problema 2.2

```
  5 REM PROBLEMA 2.2
 10 REM CONVERSOES
```

```
 20  READ A,B,C
 30  DATA 4.54609,0.3048,0.453592
 40  PRINT
 50  INPUT " GALOES/PES/LIBRAS(1/2/3)";X
 60  PRINT
 70  ON X GOTO 90,140,190
 80  END
 90  INPUT "GALOES ";G
100  L = A * G
110  PRINT G;" GALOES=";L;" LITROS"
120  PRINT
130  GOTO 50
140  INPUT " PES ";F
150  M = B * F
160  PRINT F;" PES = ";M;" METROS "
170  PRINT
180  GOTO 50
190  INPUT " LIBRAS ";P
200  K = C * P
210  PRINT P;" LIBRAS = ";K;" KILOS"
220  PRINT
230  GOTO 50
```

Problema 2.3

```
  5  REM PROBLEMA 2.3
 10  REM RAIZ X
 20  HOME
 30  VTAB 5
 40  FOR I = 1 TO 15
 50  FOR J = 1 TO 15
 60  PRINT TAB( 13);"X";
 70  NEXT J
 80  PRINT
 90  NEXT I
100  END
```

Problema 2.4

```
  5  REM PROBLEMA 2.4
 10  REM GRANDE E
 20  HOME
 30  VTAB 8
 40  FOR A = 1 TO 3
 50  FOR B = 1 TO 2
 60  FOR C = 1 TO 9
 70  PRINT TAB( 16);"E";
 80  NEXT C
 90  PRINT
```

```
100  NEXT B
110  IF A = 3 THEN END
120  FOR D = 1 TO 2
130  PRINT TAB( 16);"EE"
140  NEXT D
150  NEXT A
```

Problema 3.1

```
  5  REM PROBLEMA 3.1
 10  REM QUADRO DE SOCIOS
 20  HOME
 30  GR
 35  COLOR= 15
 40  FOR A = 39 TO 0 STEP – 2
 50  HLIN 0,3 AT A
 60  NEXT A
 65  VLIN 0,39 AT 3
 70  FOR D = 1 TO 9
 80  READ B
 90  READ E
100  FOR C = 0 TO 3
110  DATA 1,3,5,3,8,6,10,8,1,0,6,4,16,18,15,8,9,12
120  IF E > B GOTO 160
130  COLOR= 1
140  VLIN 39 – 2 * B,39 AT 4 * D + C
150  IF E > B GOTO 190
160  COLOR= 11
170  VLIN 39 – 2 * E,39 AT 4 * D + C
180  IF E > B GOTO 130
190  NEXT C
200  NEXT D
205  COLOR= 15: HLIN 0,39 AT 39
210  VTAB 21
220  PRINT TAB( 5);"ACC"; TAB( 9);"BNK"; TAB( 13);"DR"; TAB(
     17);"ENG"; TAB( 21);"FIR"; TAB( 25);"LAW"; TAB( 29);"NRS";
     TAB( 33);"PC"; TAB( 37);"TCH"
230  PRINT TAB( 4);"YEAR 1 = MAGENTA"; TAB( 24);"YEAR 2 = PINK"
260  END
```

Problema 3.2

```
  5  REM PROBLEMA 3.2
 10  REM PLOTAR GRAFICO (Y=Y↑3)
 20  HOME
 30  HGR
 40  COLOR= 5
 50  VTAB 24
 60  HPLOT 14,0 TO 140,159
```

```
 70  HPLOT 0,80 TO 279,80
 80  PRINT
 90  INPUT "FATORX? ";XF
100  PRINT
110  IF X = 0 THEN END
120  X = – 8
130  Y = X ↑ 3
140  IF XF * X + 140 < = 0 GOTO 80
150  HPLOT XF * X + 140,80 – Y / 7
160  FOR X = – 7.5 TO 8.0 STEP 0.5
170  Y = X ↑ 3
180  HPLOT TO XF * X + 140,80 – Y / 7
190  NEXT X
200  GOTO 50
```

Problema 4.1(a)

```
  5  REM PROBLEMA 4.1(A)
 10  REM CONTROLE DE ESTOQUE
 20  DIM X$(4),Y$(4)
 30  FOR I = 1 TO 4
 40  READ X$(I),Y$(I)
 50  NEXT I
 60  PRINT
 70  INPUT "QUE ITEM? ";B$
 80  PRINT
 90  IF B$ = "END" THEN END
100  FOR I = 1 TO 4
110  IF LEFT$ (X$(I),3) < > B$ GOTO 150
120  PRINT LEFT$ (Y$(I),3);" ";X$(I);"S";" ESTOCADO EM #";
     RIGHT$ (Y$(I),4);" CADA"
130  PRINT
140  GOTO 60
150  NEXT I
160  PRINT "ITEM NAO ESTOCADO"
170  PRINT
180  GOTO 60
190  DATA "BORRACHA P/TINTA","200,0.10"
200  DATA "BORRACHA P/LAPIS","320,0.15"
210  DATA "BORRACHA P/MAQUINA","25 ,0.25"
220  DATA "LIQUIDO DE CORRECAO","150,0.50"
```

Problema 4.1(b)

```
  5  REM PROBLEMA 4.1(B)
 10  REM CONTROLE DE ESTOQUE
 20  DIM X$(4)
 30  FOR I = 1 TO 4
 40  READ X$(I)
```

```
 50 NEXT I
 60 PRINT
 70 INPUT "QUE ITEM? ";B$
 80 PRINT
 90 IF B$ = "END" THEN END
100 FOR I = 1 TO 4
110 IF LEFT$ (X$(I),3) <> B$ GOTO 150
120 PRINT LEFT$ (X$(I),16);": "; MID$ (X$(I),18,3);" ESTOCADO EM
      #"; RIGHT$ (X$(I),4);" CADA"
130 PRINT
140 GOTO 60
150 NEXT I
160 PRINT "ITEM NAO ESTOCADO"
170 PRINT
180 GOTO 60
190 DATA " BORRACHA P/TINTA      ,200,0.10"
200 DATA " BORRACHA P/LÁPIS      ,320,0.15"
210 DATA " BORRACHA P/MÁQUINA  ,  25,0.25"
220 DATA " LÍQUIDO DE CORREÇÃO ,150,0.50"
```

Problema 4.2

```
  5 REM PROBLEMA 4.2
 10 REM CLASSIFICACAO POR BOLHA DAS SALAS
 20 READ N: DIM Z$(N),Y(N)
 30 FOR I = 1 TO N: READ Z$(I): NEXT I
 40 FOR I = 1 TO N: PRINT Z$(I): NEXT I
 50 PRINT : PRINT: PRINT "INFORMACAO CLASSIF."
 60 M = N
 70 FOR J = 1 TO N – 1
 80 M = M – 1:K = 0
 90 FOR I = 1 TO M
100 Y(I) = VAL ( MID$ (Z$(I),17,4))
110 Y(I + 1) = VAL ( MID$ (Z$(I + 1),17,4))
120 IF Y(I) <= Y(I + 1) GOTO 170
130 K = 1
140 T$ = Z$ (I + 1)
150 Z$(I + 1) = Z$(I)
160 Z$(I) = T$
170 NEXT I
180 PRINT : PRINT : PRINT J
190 FOR I = 1 TO N: PRINT Z$(I): NEXT I
200 GET A$
210 IF K = 0 GOTO 290
220 NEXT J
230 DATA 5
240 DATA "WILSON M. ,SALA 1.24,395"
250 DATA "SMITH M.  ,SALA 2.6 ,7315"
260 DATA "JONES B.M.,SALA 6.19,1698"
```

```
270 DATA "SMITH A.A.,SALA 2.12,456"
280 DATA "BROWN C. ,SALA  3.1 ,432"
290 END
```

Problema 4.3(a)

```
  5 REM PROBLEMA 4.3(A)
 10 REM ESPECIFIQUE NUM. P/IMPRIMIR LETRAS
 20 INPUT "ENTRE VALOR 0 A 26 ";N
 30 IF N < 0 OR N > 26 GOTO 20
 40 IF N = 0 THEN END
 50 PRINT CHR$ (64 + N)
 60 GOTO 20
```

Problema 4.3(b)

```
  5 REM PROBLEMA 4.3(B)
 10 REM ESPECIFIQUE NUM. P/IMPRIMIR LETRAS
 20 PRINT" PRESSIONE? P/ENCERRAR"
 30 INPUT "ENTRE LETRA";X$
 40 IF ASC (X$) = 63 THEN END
 50 IF ASC (X$) > 90 OR ASC (X$) < 64 GOTO 30
 60 K = ASC (X$) – 64
 70 PRINT K
 80 GOTO 30
```

Problema 5.1

```
  5 REM PROBLEMA 5.1
 10 REM AREAS CIRCULARES 2 DEC. PL.
 20 DEF FN X(R) = 3.1415926 * R ↑ 2
 30 DEF FN Y(A) = INT (A * 10 ↑ 2 + 0.5) / 10 ↑ 2
 40 FOR I = 1 TO 10
 50 A = FN X(I)
 60 PRINT I; TAB( 10);A; TAB( 28); FN Y(A)
 70 NEXT I
 80 END
```

Problema 5.2

```
  5 REM PROBLEMA 5.2
 10 REM PROGRAMA TESTE P/SUB 11000
 20 REM ESPECIFIQUE NUM. P/IMPRIMIR LETRAS
 30 INPUT "ENTRE UM NÚMERO ";N
 40 INPUT "Nº DE CASAS DECIMAIS? ";D
 50 SZ = INT (10 ↑ D * N + .5) / 10 ↑ D:SD = D: GOSUB 11000: N$ = SR$
 60 PRINT N$: END
11000 REM ALINHAMENTO DE NUMS A DIREITA.
      I/O PAR: – INPUT 57, 50, OUTPUT SR$
11010 Z$ = STR$ (SZ):LZ = LEN (Z$)
11020 FOR J = 1 TO LZ: IF MID$ (Z$,J,1) = "." GOTO 11050
11030 NEXT J
11040 Z$ = Z$ + ".":SX = 0: GOTO 11080
```

```
11050  SX = LEN ( MID$ (Z$,J + 1))
11060  IF SX = SD GOTO 11090
11080  FOR K = 1 TO SD – SX:Z$ = Z$ + "0": NEXT K
11090  SR$ = RIGHT$ ("              " + Z$,10 + SD)
11100  RETURN
```

Problema 5.3

```
    5  REM PROBLEMA 5.3
   10  REM JUROS COM DUAS SUB.
   20  INPUT "QUANTIA EMPRESTADA ";P
   30  INPUT "TAXA DE JUROS ";R
   40  INPUT "Nº DE ANOS ";X
   50  INPUT "Nº DE CASAS DECIMAIS? ";D: PRINT
   60  PRINT "ANOS"; TAB( 10);"JUROS"; TAB( 26);"QUANTIA": PRINT
   70  FOR N = 1 TO X
   80  A = P * (1 + R / 100) ↑ N
   85  SN = A:SD = D: GOSUB 10000:AA = SR
   87  SZ = AA: GOSUB 11000:A$ = SR$
   90  I = A – P
   95  SN = I:SD = D: GOSUB 10000:II = SR
   97  SZ = II: GOSUB 11000:I$ = SR$
  100  PRINT N; TAB( 6);I$; TAB( 22);A$
  110  NEXT N
  120  END
10000  REM SUB P/CONVERTER NUMEROS P/O FORMATO # # # . # #
10005  REM I/O PARAMETERS: INPUT SN, SD – OUTPUT SR
10010  SR = INT (10 ↑ SD * SN + 0.5) / 10 ↑ SD
10020  RETURN
11000  REM ALINHAMENTO A DIREITA DE NUM. I/O PAR:– INPUT SZ, SD;
       OUTPUT SR$
11010  Z$ = STR$ (SZ):LZ = LEN (Z$)
11020  FOR J = 1 TO LZ: IF MID$ (Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ = Z$ + ".":SX = 0: GOTO 11080
11050  SX = LEN ( MID$ (Z$,J + 1))
11060  IF SX = SD GOTO 11090
11080  FOR K = 1 TO SD – SX:Z$ = Z$ + "0": NEXT K
11090  SR$ = RIGHT$ ("              " + Z$,10 + SD)
11100  RETURN
```

Problema 5.4

```
    5  REM PROBLEMA 5.4
   10  REM JUROS COM DUAS SUBS
   20  INPUT "QUANTIA EMPRESTADA ";P
   30  INPUT "TAXA DE JUROS ";R
   40  INPUT "Nº DE ANOS ";X
   50  INPUT "Nº DE CASAS DECIMAIS "; D:PRINT
```

```
 60  PRINT "ANOS"; TAB( 10);"JUROS"; TAB( 26);"QUANTIA": PRINT
 70  FOR N = 1 TO X
 80  A = P * (1 + R / 100) ↑ N
 85  SN = A:SD = D: GOSUB 10000:AA = SR
 87  SZ = AA:SC = 10: GOSUB 11000: A$ = SR$
 90  I = A – P
 95  SN = I:SD = D: GOSUB 10000:II = SR
 97  SZ = II:SC = 10: GOSUB 11000:I$ = SR$
 98  SZ = N:SC = 3: GOSUB 11085:N$ = SR$
100  PRINT N$; TAB( 8);I$; TAB( 24);A$
110  NEXT N
120  END
10000  REM SUB P/CONVERTER NUMEROS P/O FORMATO ###.##
10005  REM I/O PARAMETERS: INPUT SN, SD – OUTPUT SR
10010  SR = INT (10 ↑ SD * SN + 0.5) / 10 ↑ SD
10020  RETURN
11000  REM ALINHAMENTO A DIREITA DE NUM. I/O PAR: – INPUT SZ, SD, SC;
       OUTPUT SR$
11010  Z$ = STR$ (SZ):LZ = LEN (Z$)
11020  FOR J = 1 TO LZ: IF MID$ (Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ = Z$ + ".":SX = 0: GOTO 11080
11050  SX = LEN ( MID$ (Z$,J + 1))
11060  IF SX = SD GOTO 11090
11080  FOR K = 1 TO SD – SX: Z$ = Z$ + "0": NEXT K: GOTO 11090
11085  Z$ = STR$ (SZ)
11090  SR$ = RIGHT$ ("          " + Z$,SC)
11100  RETURN
```

Problema 6.1

```
 5  REM PROBLEMA 6.1
10  REM JUROS COM DUAS SUBS"
20  INPUT QUANTIA EMPRESTADA"
30  INPUT TAXA DE JUROS"; R
40  INPUT Nº DE ANOS"; X
50  INPUT Nº DE CASAS DECIMAIS"; D
52  INPUT "SAIDA NA TELA OU IMPRESSORA? (S = T) ";A$
54  IF LEFT$(A$,1) = "T" GOTO 60
56  IF LEFT$ (A$,1) <> "I" GOTO 52
58  D$ = CHR$ (4): PRINT D$;"PR#1"
60  PRINT "ANOS"; TAB( 10);"JUROS"; TAB( 26);"QUANTIA": PRINT
70  FOR N = 1 TO X
80  A = P * (1 + R / 100) ↑ N
85  SN = A:SD = D: GOSUB 10000:AA = SR
87  SZ = AA:SC = 10: GOSUB 11000:A$ = SR$
90  I = A – P
95  SN = I:SD = D: GOSUB 10000:II = SR
```

```
   97  SZ = II:SC = 10: GOSUB 11000:I$ = SR$
   98  SZ = N:SC = 3: GOSUB 11085:N$ = SR$
  100  PRINT N$; TAB( 8);I$; TAB( 24);A$
  110  NEXT N
  115  IF LEFT$ (A$,1) = "I" THEN PRINT D$;PR#0"
  120  END
10000  REM SUB P/CONVERTER NUMEROS P/O FORMATO ###.##
10005  REM I/O PARAMETERS: INPUT SN, SD – OUTPUT SR
10010  SR = INT (10 ↑ SD * SN + 0.5) / 10 ↑ SD
10020  RETURN
11000  REM ALINHAMENTO A DIREITA DE NUM. I/O PAR: – INPUT SZ, SD, SC:
       OUTPUT SR$
11010  Z$ = STR$ (SZ):LZ = LEN (Z$)
11020  FOR J = 1 TO LZ: IF MID$ (Z$,J,1) = "." GOTO 11050
11030  NEXT J
11040  Z$ = Z$ + ".":SX = 0: GOTO 11080
11050  SX = LEN ( MID$ (Z$,J + 1))
11060  IF SX = SD GOTO 11090
11080  FOR K = 1 TO SD – SX: Z$ = Z$ + "0": NEXT K: GOTO 11090
11085  Z$ = STR$ (SZ)
11090  SR$ = RIGHT$ ("                " + Z$,SC)
11100  RETURN
```

Problema 6.2

```
    5  REM  PROBLEMA 6.2
   10  REM CRIACAO/RECUPERACAO DE ARQUIVO SEQ.
   20  D$ = CHR$ (4)
   30  INPUT "NOME DO ARQUIVO? ";X$
   40  IF X$ = "" THEN PRINT D$;"CATALOG": GOTO 30
   50  INPUT "CRIAR OU RECUPERAR? (1/2) ";K
   60  ON K GOTO 70,150
   70  INPUT "Nº DE ENTRADAS ";E: DIM A$(E): PRINT
   80  FOR N = 1 TO E: PRINT N;">";: INPUT "";A$(N)
   90  IF A$(N) <> "" THEN NEXT N
  100  N = N – 1
  110  PRINT D$;"OPEN";X$
  120  PRINT D$;"WRITE";X$
  130  PRINT N
  140  FOR I = 1 TO N: PRINT A$(I): NEXT I: GOTO 190
  150  PRINT D$;"OPEN";X$
  160  PRINT D$;"READ";X$
  170  INPUT N: DIM A$(N)
  180  FOR I = 1 TO N: INPUT A$(I): NEXT I
  190  PRINT D$;"CLOSE";X$
  200  IF K = 1 THEN END
  210  PRINT : FOR I = 1 TO N: PRINT A$(I): NEXT I: PRINT : END
```

Problema 6.3

```
  5 REM PROBLEMA 6.3
 10 REM CRIAÇÃO/RECUPERAÇÃO DE ARQUIVO SEQ.
 20 D$ = CHR$ (4)
 30 INPUT "NOME DO ARQUIVO? ";X$
 40 IF X$ = "" THEN PRINT D$;"CATALOG": GOTO 30
 50 INPUT "CRIACAO / RECUPERACAO  / ATUALIZACAO (1/2/3) ";K
 60 ON K GOTO 70,150,220
 70 INPUT "Nº DE ENTRADAS ";E: DIM A$(E): PRINT
 80 FOR N = 1 TO E: PRINT N;">";
 84 GET Z$: IF Z$ = CHR$ (13) GOTO 90
 86 A$(N) = A$(N) + Z$: PRINT Z$;: GOTO 84
 90 IF A$(N) <> "" THEN PRINT : NEXT N
100 N = N – 1: PRINT : IF K = 3 GOTO 270
110 PRINT D$;"OPEN";X$
120 PRINT D$;"WRITE";X$
130 N$ = STR$ (N): PRINT RIGHT$ ("      " + N$,5)
140 FOR I = 1 TO N: PRINT A$(I): NEXT I: GOTO 190
150 PRINT D$;"OPEN";X$
160 PRINT D$;"READ";X$
170 INPUT N$:N = VAL (N$): DIM A$(N)
180 FOR I = 1 TO N
184 GET Z$: IF Z$ = CHR$ (13) GOTO 188
186 A$(I) = A$(I) + Z$: GOTO 184
188 NEXT I
190 PRINT D$: PRINT D$;"CLOSE";X$
200 IF K = 1 THEN END
210 PRINT : FOR I = 1 TO N: PRINT A$(I): NEXT I: PRINT : END
220 PRINT D$;"OPEN";X$
230 PRINT D$;"READ";X$
240 INPUT N$:N = VAL (N$)
250 PRINT D$;"CLOSE";X$
260 NN = N: GOTO 70
270 NN = NN + N: PRINT D$;"OPEN";X$
280 PRINT D$;"WRITE";X$
300 NN$ = STR$ (NN): PRINT RIGHT$ ("      " + NN$,5)
310 PRINT D$;"CLOSE";X$
320 PRINT D$;"APPEND";X$
330 PRINT D$;"WRITE";X$
340 FOR I = 1 TO N: PRINT A$(I): NEXT I
350 PRINT D$;"CLOSE";X$
360 PRINT : PRINT "OPERACAO CONCLUIDA": END
```

Problema 6.4

```
  5 REM PROBLEMA 6.4
 10 REM RECUPERACAO E ATUALIZACAO DE ARQ. RANDOMICOS
 20 D$ = CHR$ (4):CR$ = CHR$ (13)
```

```
 30 INPUT "NOME DO ARQ. A SER CARREGADO? ";X$
 40 IF X$ = "" THEN PRINT D$;"CATALOG": GOTO 30
 50 PRINT D$;"OPEN";X$
 60 PRINT D$;"READ";X$
 70 INPUT N: DIM A$(N): PRINT "Nº DE REGISTROS = ";N
 80 INPUT RL: PRINT "PARAMETRO DE COMPRIMENTO = ";RL
 90 PRINT D$;"CLOSE";X$
100 RN = 0
110 PRINT D$;"OPEN";X$;",L";RL
120 FOR RN = 1 TO N
130 PRINT D$: PRINT D$;"READ";X$;",R";RN
140 GET Z$: IF Z$ = CR$ GOTO 160
150 A$(RN) = A$(RN) + Z$: GOTO 140
160 PRINT : PRINT A$(RN): NEXT RN
165 PRINT : PRINT "ATUALIZAR ARQUIVO? (Y/N) "
167 PRINT D$: PRINT D$;"WRITE";X$: INPUT T$: IF T$ = "Y" GOTO 190
170 PRINT D$;"CLOSE";X$
180 END
190 INPUT "Nº DE NOVAS ENTRADAS? ";NE: DIM B$(NE)
200 FOR NN = 1 TO NE: PRINT
210 HTAB 1: PRINT NN;">";
220 GET Z$
230 IF Z$ = CR$ GOTO 250
240 B$(NN) = B$(NN) + Z$: PRINT Z$;: GOTO 220
250 IF B$(NN) = "" GOTO 270
260 NEXT NN
270 NN = NN - 1: M = N + NN: RN = 0: PRINT
280 PRINT D$;"WRITE";X$;",R";RN
290 PRINT M: PRINT RL
300 FOR RN = N + 1 TO M: PRINT D$;"WRITE";X$;",R";RN
310 PRINT B$(RN - N): NEXT RN: GOTO 170
```

Problema 7.1

```
  5 REM PROBLEMA 7.1
 10 REM CRIACAO DE ARQ. RANDOMICO COM SUB. DE EDICAO
 15 ONERR GOTO 50000
 20 D$ = CHR$ (4): CR$ = CHR$ (13): LA$ = CHR$ (8): RA$ = CHR$ (21): G$ =
    CHR$ (7)
 30 INPUT "COMPRIMENTO MAX DE REGISTRO? ";RL: IF RL > 256 GOTO 30
 40 INPUT "Nº DE NOVAS ENTRADAS? ";NE: DIM A$(NE): SL = RL
 50 FOR N = 1 TO NE: PRINT :SN = N: GOSUB 20000: A$(N) = SR$
100 IF A$(N) = "" GOTO 120
110 NEXT N
120 N = N - 1: PRINT
130 INPUT "NOME SOB O QUAL DEVE SER SALVO? ";X$
140 RN = 0
150 PRINT D$;"OPEN";X$;",L";RL
```

```
160  PRINT D$;"WRITE";X$;",R";RN
170  PRINT N:PRINT RL
180  FOR RN = 1 TO N: PRINT D$;"WRITE";X$;",R";RN
190  PRINT A$(RN): NEXT RN
200  PRINT D$;"CLOSE";X$
210  PRINT "DONE": END
20000 SR$ = "" B$ = "": REM *****SUB-ROTINA DE ENTRADA PELO
       TECLADO***** I/O PAR:– INPUT SL,SN; OUTPUT SR$
20010 K = 1: HTAB 1: PRINT SN;">";
20020 GET Z$
20030 IF Z$ = CR$ THEN CALL – 868: RETURN
20040 IF Z$ = RA$ GOTO 20160
20050 IF Z$ = LA$ GOTO 20110
20060 IF K = SL THEN PRINT G$;: GOTO 20020
20070 SR$ = SR$ + Z$: PRINT Z$;:K = K + 1
20080 LB = LEN (B$): IF LB > 1 THEN B$ = LEFT$ (B$,LB – 1): GOTO 20020
20090 B$ = "": GOTO 20020
20100 REM EDICAO DA TECLA DA FLECHA ESQUERDA
20110 LA = LEN (SR$): IF LA < 1 THEN SR$ = "": GOTO 20010
20120 B$ = B$ + RIGHT$ (SR$,1):LB = LEN (B$):K = K – 1
20130 IF LA > 1 THEN SR$ = LEFT$ (SR$,LA – 1): PRINT Z$;: GOTO 20020
20140 SR$ = "": PRINT Z$;: GOTO 20020
20150 REM EDICAO DA TECLA DA FLECHA DIREITA
20160 LB = LEN (B$): IF LB < 1 THEN B$ = "": GOTO 20020
20170 RB$ = RIGHT$ (B$,1):SR$ = SR$ + RB$: PRINT RB$;:K = K + 1
20180 IF LB > 1 THEN B$ = LEFT$ (B$,LB – 1): GOTO 20020
20190 B$ = "": GOTO 20020
50000 REM LINHAS 50000 – 50180 OCUPADAS POR ROTINA DE ERRO USUAL
```

Problema 7.2

```
5    REM PROBLEMA 7.2
10   REM CRIACAO DE ARQ. RANDOMICOS COM SUB. GETLN
15   ONERR GOTO 50000
20   D$ = CHR$ (4)
30   INPUT "COMPRIMENTO MAX. DO REGISTRO? ";RL: IF RL > 256 GOTO 30
40   INPUT "NUMERO DE NOVAS ENTRADAS? ";NE: DIM A$(NE)
50   FOR N = 1 TO NE: VTAB 24: PRINT N;">";
60   SL = RL: GOSUB 20500:A$(N) = SR$: CALL – 912
100  IF A$(N) = "" GOTO 120
110  NEXT N
120  N = N – 1: PRINT
130  INPUT "NOME SOB O QUAL DEVE SER SALVO? ";X$
140  RN = 0
150  PRINT D$;"OPEN";X$;",L";RL
160  PRINT D$;"WRITE";X$;",R";RN
170  PRINT N:PRINT RL
180  FOR RN = 1 TO N: PRINT D$;"WRITE";X$;",R";RN
```

```
  190   PRINT A$(RN): NEXT RN
  200   PRINT D$;"CLOSE";X$
  210   PRINT "DONE": END
20500   REM *****SUB. P/CRIAR JANELA***** I/O PAR:– SL, SA; OUTPUT SR$
20510   L$ = " ": FOR I = 1 TO SL:L$ = L$ + " ": NEXT I
20520   INVERSE : HTAB 3: PRINT L$
20530   POKE 35,24: POKE 34,23: POKE 33,SL: POKE 32,2
20540   HTAB 1: VTAB 23: CALL 6487
qm  4:  GOSUB 21000:SR$ = SA$ + CHR$ (13)
20550   POKE 32,0: POKE 33,40: POKE 34,0: POKE 35,24
20560   NORMAL : RETURN
21000   REM PEEK CONTEUDO DO ARQ BUFFER
21010   REM I/O PAR:– OUTPUT SA$
21020   SA$ = "": FOR K = 512 TO 768
21030   X = PEEK (K): IF X = 141 THEN RETURN
21040   SA$ = SA$ + CHR$ (X)
21050   NEXT K
50000   REM LINHAS 50000 – 50180 OCUPADAS  POR ROTINA DE ERRO USUAL
```

Problema 8.1

Carregue o Problema 7.1 no computador e apague as linhas 5-201 e 50000 em diante. EXECute o programa CRIADOR DE TEXT e digite RUN.

Problema 8.2

A implementação da solução é dada em um programa no Capítulo 8 na seção PESQUISA E EDIÇÃO DE REGISTROS (linhas 2080 a 2168).

Problema 8.3

A implementação da solução é dada em um programa no Capítulo 8 na seção CRIAÇÃO DE PONTEIROS PARA LIGAR ARQUIVOS (linhas 2089, 2170 e 2180).

Problema 8.4

A listagem completa da solução deste problema é dada nas próximas páginas. Os seguintes pontos precisam de esclarecimentos.

(a) Quando terminar de digitar os acréscimos a um arquivo existente, usando a Opção de Atualização, o programa irá automaticamente concatenar os novos registros alfabeticamente ao arquivo existente. A instrução POKE na linha 3060 estabelece uma "janela" na tela a fim de parar o movimento do texto (SCROLL) na tela. A margem superior da tela é restabelecida na linha 3160 para que o movimento normal recomece.

(b) As duas instruções PRINT D$ na linha 3100 lá estão para permitir a alteração dos comandos do disco de READ para WRITE sem ter de fechar (CLOSE) e reabrir (OPEN) o arquivo.

(c) Registros incorretos podem ser editados usando-se a opção EDIT. O uso da flecha direita irá mostrar, letra por letra, o conteúdo do registro que está sendo editado.

(d) O espaço ocupado por um registro em branco pode ser utilizado simplesmente através da opção EDIT e digitando-se um novo registro no número de registro especificado.

```
  5 REM PROBLEMA 8.4
 10 HOME : VTAB 5: PRINT "***CRIACAO/RECUPERACAO/ATUALIZACAO
    RANDOMICA***"
 20 VTAB 10: HTAB 8: PRINT "P/SAIR MODO DE ENTRADA PRESSIONE": VTAB
    12: HTAB 16: PRINT "<RETURN>"
 30 NE = 20: DIM A$(NE + 2),PN(NE + 2,2):BY = 8
 40 FOR I = 0 TO 9: READ K: POKE 768 + I,K: NEXT I
 50 DATA 104,168,104,166,223,154,72,152,72,96
 60 ONERR GOTO 50000
 70 D$ = CHR$ (4):CR$ = CHR$ (13):LA$ = CHR$ (8):RA$ = CHR$ (21):G$ =
    CHR$ (7)
 80 CM$ = "PRESSIONE <RETURN> P/CONTINUAR"
 90 VTAB 23: PRINT CM$: GET Q$
100 HOME : REM *****MENU*****
110 HTAB 15: VTAB 8: PRINT "M E N U"
120 HTAB 15: PRINT "          ": PRINT
130 HTAB 5: PRINT "1. CRIAR NOVO ARQ RANDOMICO"
140 HTAB 5: PRINT "2. RECUPERAR ARQ RANDOMICO"
150 HTAB 5: PRINT "3. ATUALIZAR ARQ RANDOMICO"
160 HTAB 5: PRINT "4. ATUALIZAR ARQ RANDOMICO"
170 HTAB 5: PRINT "5. RECUPERAR ALFABETICAMENTE"
180 HTAB 5: PRINT "6. ENCERRAR PROGRAMA"
270 VTAB 21: INPUT "CHOOSE (1/2/3/4/5/6) ";M$
280 M = VAL (M$): IF M < 1 OU M > 6 GOTO 270
290 ON M GOTO 1000,2000,3000,4000,5000,300
300 END
999 REM
1000 REM *****CRIACAO DE NOVO ARQ. RANDOMICO*****
1010 VTAB 23: INPUT "MAX RECORD LENGTH? ";RL$
1020 RL = VAL (RL$): IF RL < (BY + 1) OR RL > 256 GOTO 1010
1030 SL = RL:SB = BY: GOSUB 30000
1040 PRINT "NUM. MAX. DE NOVAS ENTRADAS = ";NE:SL = RL – BY
1050 FOR N = 1 TO NE: PRINT :SN = N: GOSUB 20000:A$(N + 2) = SR$
1060 IF A$(N + 2) = "" THEN PRINT D$: GOTO 1080
1070 GOSUB 40000: NEXT N
1080 N = N + 1:LD = 0: PRINT : REM N INCLUI OS DOIS PRIMEIROS
     REGISTROS
1090 PRINT G$: INPUT "NOME SOB O QUAL DEVE SER SALVO? ";X$
1100 RN = 0
1110 PRINT D$;"OPEN";X$;",L";RL
1120 PRINT D$;"WRITE";X$;",R";RN
1130 PRINT N: PRINT RL
1140 FOR RN = 1 TO N: PRINT D$;"WRITE";X$;",R";RN
```

```
1150  PRINT PN(RN,1): PRINT PN(RN,2): PRINT
      D$;"WRITE";X$;",R";RN;",B";BY
1160  PRINT A$(RN): NEXT RN
1170  PRINT D$;"CLOSE";X$
1180  PRINT : PRINT CM$: GET Q$: GOTO 100
1999  REM
2000  REM *****RECUPERACAO DE ARQ. RANDOMICO*****
2010  GOSUB 12000: PRINT
2020  INPUT "RECUPERAR DE QUE NUMERO DE REGISTRO? ";B$
2030  B = VAL (B$): IF B < 1 OR B > N – 2 THEN PRINT "CAN'T
      BE DONE": GOTO 2020
2040  INPUT "OUTPUT P/IMPRESSORA? (Y/N) ";T$: IF LEFT$ (T$,1) =
      "Y" THEN PRINT D$: PRINT D$;"PR#1": PR = 1
2050  PRINT : PRINT D$;"OPEN";X$;",L";RL
2060  B = B + 2: E = B + NE – 3: IF E > N THEN E = N
2070  FOR RN = B TO E: I = RN – B + 1: A$(I) = ""
2080  PRINT D$: PRINT D$;"READ";X$;",R";RN;",B";BY
2090  GET Z$: IF Z$ = CR$ GOTO 2110
2100  A$(I) = A$(I) + Z$: GOTO 2090
2110  PRINT : PRINT RN – 2;">";A$(I),: NEXT RN
2120  PRINT D$: PRINT D$;"CLOSE";X$
2130  PRINT : IF N = RN – 1 THEN PRINT "FIM DE ARQUIVO": GOTO 2160
2140  INPUT "CONTINUAR LISTAGEM? (S/N) ";AA$: IF AA$ = "N" GOTO
      2160
2150  B = E + 1: IF B <= N THEN B = B – 2: GOTO 2050
2160  IF PR = 1 THEN PRINT D$: PRINT D$;"PR#0": PR = 0
2170  PRINT : PRINT CM$: GET Q$: GOTO 100
2999  REM
3000  REM *****UPDATE AN EXISTING RANDOM FILE*****
3010  GOSUB 12000: NN = N
3020  PRINT "NUMERO MAXIMO DE ENDERECOS DE ENTRADA = ";NE: SL =
      RL – BY
3030  FOR N = 1 TO NE: PRINT : SN = NN + N – 2: GOSUB
      20000: A$(N) = SR$
3040  IF A$(N) = "" GOTO 3060
3050  NEXT N: N = N – 1
3060  PRINT : PRINT : PRINT "ARQUIVO DE ATUALIZACAO . . .": POKE 34,23
3070  PRINT D$: PRINT D$;"OPEN";X$;",L";RL
3080  FOR II = 1 TO N – 1: Z(2) = NN + II: Y$(2) = ""
3090  Y$(2) = A$(II): FOR J = 1 TO 2: PP(2,J) = Z(2): NEXT J
3100  GOSUB 14000: PRINT D$: PRINT D$
3110  Z(1) = PP(2,1): Z(3) = PP(2,2)
3120  FOR I = 1 TO 3: GOSUB 15000: NEXT I
3130  NEXT II: RN = NN + II – 1
3140  PRINT D$;"WRITE";X$;",R";0
3150  PRINT RN: PRINT RL
3160  PRINT D$;"CLOSE";X$: POKE 34,0
3170  PRINT : PRINT "ARQUIVO ATUALIZADO"
```

```
3180  PRINT : PRINT CM$: GET Q$: GOTO 100
3999  REM
4000  REM *****EDITANDO ARQ. RANDOMICO*****
4010  GOSUB 12000: PRINT
4020  INPUT "EDITAR QUE NUMERO DE REGISTRO? ";T$:RN = VAL (T$):
      IF RN < 1 OR RN > N – 2 THEN PRINT "CAN'T BE DONE": GOTO 4020
4030  GOSUB 13000:I = 2
4040  B$ = "":SR$ = "":SN = RN:SL = RL:L2 = LEN (Y$(I))
4050  IF Z(1) = Z(3) GOTO 4070
4060  FOR L1 = 1 TO L2:L3 = L2 – L1 + 1:B$ = B$ + MID$
      (Y$(I),L3,1): NEXT L1
4070  GOSUB 20010:Y$(I) = SR$
4080  DE = 0: IF LEN (Y$(I)) = 0 AND RN = N – 2 THEN DE = 1:
      PRINT : INPUT "APAGAR ESTE REGISTRO (S/N) ";AA$: IF AA$ =
      "N" GOTO 4230
4090  IF DE <> 1 GOTO 4170
4100  POKE 34,23: PRINT D$: PRINT D$;"OPEN";X$;",L";RL
4110  PRINT D$;"WRITE";X$;",R";0
4120  PRINT N –1: PRINT RL
4130  PRINT D$;"WRITE";X$;",R";R
4140  PRINT
4150  FOR I = 1 TO 3 STEP 2: GOSUB 15000: NEXT I
4160  PRINT D$;"CLOSE";X$: GOTO 4220
4170  BL = 0: IF LEN (Y$(I)) = 0 THEN BL = 1: PRINT : INPUT
      "BLANK THIS RECORD? (Y/N) ";AA$: IF LEFT$ (AA$,1) = "N"
      GOTO 4230
4180  PRINT D$: PRINT D$;"OPEN";X$;",L";RL
4190  FOR I = 1 TO 3: GOSUB 15000: NEXT I
4200  PRINT D$;"CLOSE";X$: IF BL = 0 AND Z(1) <> Z(3) THEN
      GOSUB 16000: POKE 34,23: PRINT D$: PRINT
      D$;"OPEN";X$;",L";RL: GOSUB 14000: FOR I = 1 TO 3: GOSUB
      15000: NEXT I: PRINT D$;"CLOSE";X$
4210  IF BL = 0 AND Z(1) = Z(3) THEN POKE 34,23: PRINT D$:
      PRINT D$;"OPEN";X$;",L";RL: GOSUB 14000: PRINT D$: PRINT
      D$: Z(1) = PP(2,1):Z(3) = PP(2,2): FOR I = 1 TO 3: GOSUB
      15000: NEXT I: PRINT D$;"CLOSE";X$
4220  POKE 34,0: PRINT : PRINT "FILE UPDATED": GOTO 4240
4230  PRINT : PRINT "ARQUIVO ALTERADO"
4240  PRINT : PRINT CM$: GET Q$: GOTO 100
4999  REM
5000  REM *****RECUPERACAO ALFABETICA*****
5010  RN = 1: GOSUB 12000: PRINT : INPUT "OUTPUT P/IMPRESSORA?
      (S/N) ";T$
5020  IF LEFT$ (T$,1) = "S" THEN PRINT D$;"PR#1":PT = 1
5030  PRINT: PRINT D$;"OPEN";X$;",L";RL
5040  PRINT D$;"READ";X$;",R";RN
5050  INPUT PR: INPUT NT
5060  B$ = "":RN = NT: IF RN = 2 GOTO 5130
```

```
 5070  PRINT D$: PRINT D$;"READ";X$;",R";RN;",B";0
 5080  INPUT PR: INPUT NT
 5090  PRINT D$;"READ";X$;",R";RN;",B";BY
 5100  GET Z$: IF Z$ = CR$ GOTO 5120
 5110  B$ = B$ + Z$: GOTO 5100
 5120  PRINT : PRINT B$: GOTO 5060
 5130  PRINT D$: PRINT D$;"CLOSE";X$
 5140  IF PT = 1 THEN PRINT D$;"PR#0":PT = 0
 5150  PRINT : PRINT CM$: GET Q$: GOTO 100
 5999   REM
12000  REM *****SUB-ROTINA P/LER ARQUIVO***** I/O PAR:--
     OUTPUT N,RL
12010  PRINT G$:LD = 1: INPUT "NOME DO ARQUIVO A CARREGAR? ";X$
12020  IF X$ = "" THEN PRINT D$;"CATALOG": GOTO 12010
12030  PRINT D$;"OPEN";X$
12040  PRINT D$;"READ";X$
12050  INPUT N: PRINT "REGISTROS ";N - 2
12060  INPUT RL: PRINT "LENGTH ";RL
12070  PRINT D$;"CLOSE";X$: RETURN
13000  REM *****SUB-ROTINA PARA DESUNIR REGISTROS A SEREM
     EDITADOS***** I/O PAR:-- INPUT RL,RN
13010  FOR I = 1 TO 3:Y$(I) = "": NEXT I
13020  PRINT D$;"OPEN";X$;",L";RL
13030  R = RN + 2: PRINT D$;"READ";X$;"R";R
13040  FOR J = 1 TO 2: INPUT PP(2,J): NEXT J:Z(1) = PP(2,1):
     Z(2) = R:Z(3) = PP(2,2)
13050  IF (Z(1) = Z(3) GOTO 13170
13060  PRINT D$;"READ";X$;",R";R;",B";BY
13070  GET Z$: IF Z$ = CR$ THEN PRINT D$: GOTO 13090
13080  Y$(2) = Y$(2) + Z$: GOTO 13070
13090  FOR I = 1 TO 3 STEP 2:II = Z(I)
13100  PRINT D$;"READ";X$;",R";II
13110  FOR J = 1 TO 2: INPUT PP(I,J): NEXT J
13120  PRINT D$;"READ";X$;",R";II;",B";BY
13130  GET Z$: IF Z$ = CR$ THEN PRINT D$: GOTO 13150
13140  Y$(I) = Y$(I) + Z$: GOTO 13130
13150  NEXT I:PP(1,2) = PP(2,2):PP(3,1) = PP(2,1)
13160  PP(2,1) = R:PP(2,2) = R
13170  PRINT D$;"CLOSE";X$: RETURN
13999  REM
14000  REM *****SUB-ROTINA P/FORMAR NOVAS LIGACOES*****
14010  I = 3:RN + 1
14020  Y$(I) = "": PRINT D$;"READ";X$;",R";RN;",B";0
14030  FOR J = 1 TO 2: INPUT PP(I,J): NEXT J
14040  PRINT D$;"READ";X$;",R";RN;",B";BY
14050  GET Z$: IF Z$ = CR$ THEN PRINT D$: GOTO 14070
14060  Y$(I) = Y$(I) + Z$: GOTO 14050
14070  IF Y$(I) <= Y$(2) THEN RN = PP(I,2): GOTO 14020
```

```
14080  Z(I) = RN:RN = PP(I,1): PRINT D$;"READ";X$;",R";RN;",B";0
14090  I = I – 2: FOR J = 1 TO 2: INPUT PP(I,J): NEXT J:Y$(I)
       = "": Z(I) = RN
14100  PRINT D$;"READ";X$;",R";RN;",B";BY
14110  GET Z$: IF Z$ = CR$ THEN PRINT D$: GOTO 14130
14120  Y$(I) = Y$(I) + Z$: GOTO 14110
14130  PP(2,1) = PP(3,1):PP(2,2) = PP(1,2):R = Z(2)
14140  PP(1,2) = R:PP(3,1) = R: RETURN
15000  REM ***** SUB-ROTINA P/ESCREVER REGISTROS EDITADOS*****
15010  R = Z(I): PRINT D$;"WRITE";X$;",R";R;",B";0
15020  FOR J = 1 TO 2: PRINT PP(I,J): NEXT J
15030  PRINT D$;"WRITE";X$;",R";R;",B";BY

15040  PRINT Y$(I): RETURN
15999  REM
16000  REM ***** SUB-ROTINA P/APAGAR REGISTROS DE ARQUIVOS
       PRÉ-EDITADOS RECS*****
16010  PRINT D$;"OPEN";X$;",L";RL
16020  RN = Z(2): PRINT D$;"WRITE";X$;",R";RN;",B";BY
16030  PRINT ""
16040  PRINT D$;"CLOSE";X$: RETURN
16999  REM
20000  SR$ = "":B$ = "": REM *****SUB-ROTINA DE ENTRADAS DO TECLADO
       SUBROTINE*****I/O PAR: INPUT SL,SN; OUTPUT SR$. STRINGS TAMBEM
       DEFINIDOS CR$, RA$, LA$ & G$, COM OS CODIGOS CHR$ ( ) APROPRIADOS
20010  K = 1: HTAB 1: PRINT SN;">";
20020  GET Z$
20030  IF Z$ = CR$ THEN CALL – 868: RETURN
20040  IF Z$ = RA$ GOTO 20160
20050  IF Z$ = LA$ GOTO 20110
20060  IF K = SL THEN PRINT G$;: GOTO 20020
20070  SR$ = SR$ + Z$: PRINT Z$;:K = K + 1
20080  LB = LEN (B$): IF LB > 1 THEN B$ = LEFT$ (B$,LB – 1):
       GOTO 20020
20090  B$ = "": GOTO 20020
20100  REM EDICAO DA TECLA FLECHA ESQUERDA*
20110  LA = LEN (SR$): IF LA < 1 THEN SR$ = "": GOTO 20010
20120  B$ = B$ + RIGHT$ (SR$,1):LB = LEN (B$):K = K – 1
20130  IF LA > 1 THEN SR$ = LEFT$ (SR$,LA – 1): PRINT Z$;:
       GOTO 20020
20140  SR$ = "": PRINT Z$;: GOTO 20020
20150  REM EDICAO DA TECLA FLECHA DIREITA
20160  LB = LEN (B$): IF LB < 1 THEN B$ = "": GOTO 20020
20170  RB$ = RIGHT$ (B$,1):SR$ = SR$ + RB$: PRINT RB$;:K = K + 1
20180  IF LB > 1 THEN B$ = LEFT$ (B$,LB – 1): GOTO 20020
```

*EDIÇÃO → É ACONSELHÁVEL ABREVIAR POR EDIT. POR NÃO HAVER Ç NOS MICRO-COMPUTADORES.

```
20190  B$ = "": GOTO 20020
20999  REM
30000  REM ***** SUB-ROTINA P/INICIALIZAR NOVOS ARQUIVOS***** I/O
     PAR:– INPUT SL,SB; OUTPUT A$( ), PN(,)
30010  PS = 1:NT = 0:S$ = CHR$ (255):SS$ = ""
30020  FOR I = 2 TO 1 STEP – 1:PN(I,1) = PS:PN(I,2) = NT
30030  IF I = 1 GOTO 30060
30040  FOR J = 1 TO SL – SB – 1: SS$ = SS$ + S$: NEXT J:A$(I)
     = SS$ + CHR$ (13)
30050  PS = 0:NT = 2:SS$ = "": NEXT I
30060  FOR J = 1 TO SL – SB – 1:SS$ = SS$ + CHR$ (0): NEXT
     J: A$(I) = SS$ + CHR$ (13): RETURN
30999  REM
40000  NT = 1: REM ***** SUB-ROTINA PONTEIROS***** I/O PAR:–
     INPUT N, A$( ), PN(,); OUTPUT PN(,)
40010  FOR J = 1 TO N + 1:PS = PN(NT,1): IF A$(NT) <= A$(N + 2)
     THEN NT = PN(NT,2): NEXT J
40020  PN(N + 2,1) = PN(NT,1)
40030  PN(N + 2,2) = NT
40040  PN(PS,2) = N + 2
40050  PN(NT,1) = N + 2
40060  RETURN
40999  REM
50000  PRINT G$: REM *****ROTINA DE MANIPULACAO DE
     ERROS*****
50010  ER = PEEK (222): CALL 768
50020  IF ER = 4 THEN PRINT "DISCO PROTEGIDO!": GOTO 50120
50030  IF ER = 5 OR ER = 6 THEN PRINT "ARQUIVO NAO ENCONTRADO!":
     PRINT D$;"DELETE";X$: GOTO 50120
50040  IF ER = 7 THEN PRINT "VOLUME ILEGAL!": GOTO 50120
50050  IF ER = 8 THEN PRINT "I/O ERRO!": GOTO 50120
50060  IF ER = 9 THEN PRINT "DISCO LOTADO": PRINT
     D$;"DELETE";X$: GOTO 50120
50070  IF ER = 10 THEN PRINT "ARQUIVO TRAVADO!": INPUT "UNLOCK?
     ";FU$: IF LEN (FU$) = 0 THEN PRINT D$;"CATALOG": GOTO 50070
50080  IF LEFT$ (FU$,1) = "Y" THEN PRINT D$;"UNLOCK";X$: GOTO
     50140
50090  IF LEN (FU$) > 0 GOTO 50120
50100  IF ER = 11 THEN PRINT "NOME ILEGAL!": GOTO 50120
50110  IF ER = 13 THEN PRINT "TIPO DE ARQUIVO NÃO CONFERE!"
50120  IF LD = 1 GOTO 50150
50130  GOTO 1090: REM PARA LINHA QUE NOMEIA ARQUIVO A SER SALVO
50140  GOTO 1100: REM PARA LINHA SEGUINTE QUE NOMEIA ARQUIVO A
     SER SALVO
50150  ON M GOTO 50160,2010,3010: REM PARA LINHA QUE DETERMINA
     ARQUIVO A SER CARREGADO
50160  IF ERR = 77 THEN PRINT "ERRO FORA DE MEMORIA": GOTO 50180
50170  PRINT "ERROR NR ";ER;" IN LINE "; PEEK (218) + PEEK (219) * 256
50180  POKE 216,0: RESUME
```

Problema 9.1

```
5  REM PROBLEMA 9.1
10  REM TABELAS DE FORMA
20  TEXT : HOME
30  POKE 232,32: POKE 233,3
40  FOR I = 800 TO 803
50  READ K: POKE I,K: NEXT I
60  FOR J = 0 TO 10: INPUT "MOVE?"; M: IF M <> 0 THEN POKE I + J,M: NEXT J
70  POKE I + J,M
80  HGR : HCOLOR= 3
90  VTAB 21: ROT= 0: INPUT "SCALA? ";S: SCALE= S
100  HPLOT 0,159
110  INPUT "QUANTAS VEZES? ";N
120  FOR I = 1 TO N: DRAW 1: NEXT I
130  DATA 1,0,4,0
```

Problema 9.2

```
5  REM PROBLEMA 9.2
10  REM CRIACAO DE UM NUMERO DE UMA DEFINICAO DE FORMA
20  HOME : TEXT : VTAB 21:D$ = CHR$ (4)
30  POKE 232,32: POKE 233,3
40  INPUT "NUMERO DE DEFINICOES DE FORMA? ";D:K = 2 * D + 3
50  POKE 800,D: POKE 801,0: PRINT
60  FOR N = 1 TO D: PRINT "DEFINICAO DE FORMA ";N
70  FOR I = 800 + K TO 1056
80  INPUT "MOVER? ";M: IF M = 0 GOTO 100
90  POKE I,M: NEXT I: PRINT "TABELA DE FORMA > 256 BYTES"
100  P = 800 + 2 * N: POKE I,M: POKE P,K: POKE P + 1,0:K = I - 800 + 1: PRINT :
     NEXT N
110  PRINT "FIM DE TABELA DE FORMA"
120  INPUT "ESCALA? ";S: HGR :  HCOLOR = 3: SCALE=S: ROT=0
130  FOR I = 1 TO D: PRINT "DEFINICAO DE FORMA ";I
140  XDRAW I AT 139,95: GET T$: XDRAW I: NEXT I
150  TEXT : HOME :  INPUT "QUER SALVAR? (S/N) ";Q$: IF LEFT$ (Q$,1) =
     "N" GOTO 180
160  INPUT "NOME SOB O QUAL DEVE SER SALVO? ";A$
170  PRINT D$;BSAVE";A$;",A800";",L";K
180  TEXT : HOME : END
```

Nota do tradutor

Certas palavras, ao longo desta obra, foram mantidas em inglês por não haver tradução correspondente em português ou por terem sido consagradas em inglês, no Brasil, pelos profissionais de processamento de dados. Para facilitar a sua compreensão no texto, eis uma breve explicação dos termos e palavras mais usadas.

Acessar = Ter acesso a.
Array = Seqüência contínua de palavras de memória; agregados homogêneos.
Boot = Chamar o DOS, processo de introduzir qualquer forma operacional a partir de um disquete.
Buffer = Memória intermediária (termo consagrado no Brasil em inglês).
Debug = Depurar erros em um programa.
Default = Valor ou opção assumido(a) pelo computador quando o usuário deixa de fornecer um parâmetro.
Deletar = Nos meios de processamento de dados o anglicismo "deletar" tem sido consagrado na tradução de Delete (apagar, eliminar). Em alguns pontos da obra mantivemos esta forma, embora não se trate de bom português.
Drive = Adotado do inglês DISK-DRIVE pelos usuários de microcomputadores para se referirem às unidades de disco.
Imprimir = Exibir na tela.
Indent = Alinhar para dentro.
Loop = Laço.
Plotar = Marcar pontos em locais específicos da tela.
Retornar (RETURN) = Termo técnico de processamento de dados que indica o fornecimento pelo computador ao usuário de um código, uma frase etc. Resultado de uma situação específica (ex: mensagem de erro, uma conversão de códigos etc).
Soquete = Encaixe para placa de interfaces de periféricos que se encontra na parte traseira do microcomputador.
String = Seqüência de caracteres.

Índice analítico

ABS( ) (função), 67, 68
Acesso (comando), 127
Animação, 125
Append (comando), 83
Apple soft, 16
  BASIC, 15, 18
  Mensagens de erro, 131
Aritmética
  Funções, 67, 69
  Operadores, 32
  Prioridades, 31, 32
Arquivos binários, 20
Arquivos Buffer, 88
Arquivos Randômicos, 86
  Campo, 87
  Close (comando), 81
  Comandos (resumo), 129, 130
  Comprimento de Registro, 80-89
  Criação, 86
  Edição, 107, 108
  GET (instrução), 85
  INPUT (instrução), 81
  OPEN (comando), 81
  PRINT (instrução), 81
  READ (instrução), 81
  Recuperação, 84
  WRITE (instrução), 81
Arquivos seqüenciais, 80
  APPEND (comando), 83
  Atualização, 82
  Comandos (resumo), 129
  Criação, 80
  Delete (comando), 83
  GET (instrução), 84
  INPUT (instrução), 81
  OPEN (comando), 81
  PRINT (instrução), 81
  READ (instrução), 81
  Recuperação, 84
  WRITE (comando), 81
Arquivos TEXT, 20, 79, 86
  Banco de dados, 80, 86
  Randômico, 86
  Seqüencial, 80
ARRAYs, 58
  Erros, 58
ASC ( ) (função), 64
ASCII
  Código, 54, 55
  Código de conversão, 140
Assembler, 17
ATN ( ) (função), 67, 68

Banco de Dados, 114
Binário, 20
  Arquivos, 97
  Parâmetros de arquivo, 97
  Representação, 136
BLOAD
  comando, 20, 97
  Tabelas de forma, 125, 126
Brun (comando), 97
BSAVE
  Comando, 97
  Tabelas de forma, 125
BUFFER (arquivo), 88
BYTE, 15, 21, 80, 81
  Alto, 114-116
  Baixo, 114-116
  Parâmetros, 112
  Definição de formato, 119
  Representação de inteiro, 114-116

Cabeçalhos, 27
CALL (comando), 92, 97
CATALOG
Comando, 17, 18
A partir do BASIC, 79
Chamada do DOS (BOOT), 17, 18
CHR#( ) (função), 65
Classificação alfabética, 61, 109
Classificação de dados, 37, 38, 109
Grandes arquivos com ponteiros, 109 111
Grandes arquivos em disquete, 109
Close (comando), 79, 81, 86
Comandos
Acesso, 127
DOS, 127
Comparação alfabética de strings, 54
Concatenação, 54, 63
Constantes, 26
CONT (comando), 133
Cores, 50
COS ( ) (função), 67
Criar
Arquivos Randômicos, 86
Arquivos Seqüenciais, 80
Ponteiros, 111
CTRL-TECLA, 16
CTRL C (comando), 19, 34
CTRL D (comando), 78
CTRL G (comando), 16
CTRL X (comando), 22

Dados (data)
Arquivos, 79, 86
Campos, 79, 86
Classificação, 37, 38, 109
Data (instrução), 30
DEF FN ( ) (instrução), 69
Delete (comando), 20
a partir de arquivos text, 82
Desmontagem de números inteiros, 114
Diagramas de bloco, 35
DIM(instrução), 55, 56, 58
Disquete
Diretório, 21
Drive, 17
Pistas, 88
Sistema Operacional, 17
Setores, 21, 88
DOS, 17
Código de erro, 90, 129
Comando(s), 127
A partir do BASIC, 79, 129
Mensagens de erro, 90, 129
Parâmetros, 128, 129
DRAW (comando), 118, 120, 121

Edição de Arquivos Randômicos, 107
Edição de Tela, 21
Eliminação de erros ("DEBUG"), 133
END (instrução), 24
Erro
Mensagens do APPLESOFT, 131
Mensagens do DOS, 131
Rotinas, 90, 91
ESC (Código), 22, 147
ESC – Tecla, 16
Espaçamento em BASIC, 37
EXEC (comando), 102, 103
EXP ( ) (função), 67
Expressões, 26

Flecha-direita – tecla, 16
Flecha-esquerda – tecla, 16
Fluxogramas, 35, 36
Formato dos números de I/O, 26
FOR-NEXT (instruções), 41, 44
aninhamento de loops, 41, 44-45
FP (comando), 18
FRE ( ) (comando), 143

FUNÇÃO
ABS ( ), 67, 68
ASC ( ), 64
ATN ( ), 67, 68
CHR$ ( ), 65
COS ( ), 67
Definida por usuário, 69
Derivadas, 69
EXP ( ), 67
INT ( ), 67, 69
LEFT$ ( ), 57
LEN ( ), 64, 65
LOG ( ), 67, 68
MID$ ( ), 57
Parâmetro, 69

RIGHT$ ( ), 57
RND ( ), 67, 68
SGN ( ), 67, 68
SIN ( ), 67, 68
SQR ( ), 67, 68
STR$ ( ), 64, 65
TAN ( ), 67, 68
VAL ( ), 65

GET (instruçâo), 62
em arquivos TEX, 84, 86
GOSUB-RETURN (instrução), 70
GOTO (instrução), 34
GOTO Computado (instrução), 40
GR, (comando), 46
Gráfico de Barra, 48
Gráficos
Alta-resolução, 49
Baixa-resolução, 48
Comandos
DRAW, 118, 120, 121
GR, 46
HGR, 49
HGR2, 50
HLIN, 47
H PLOT, 49, 51
PLOT, 47
ROT, 118, 122
SCALE, 118, 122
TEXT, 46
VLIN, 47
XDRAW, 118, 121
Formas, 118, 120, 121, 125
Graus para radianos
conversão, 68

I/O encaixe, 15
IF (instrução), 34
operador lógico, 34, 38
Impressão formatada, 27, 44, 49
INH (comando), 17
Inicializando um disquete, 18
INIT (comando), 18, 92
INPUT (instrução), 28
em arquivos TEXT, 81, 86
INPUT/OUTPUT
encaixe, 15
subparâmetros, 71, 76

Instruções de atribuição, 25
INT ( ) (função), 67-69
INTEGER BASIC, 17, 20
INVERSE (comando), 96

LEFT$ ( ) (função), 57
LEN ( ) (função), 64, 65
Limpeza de tela, 16, 28
Linguagem de máquina, 20, 96
LIST (comando), 19, 25
Listagem de programa, 19, 20, 21
Literais, 54, 56
LOAD (comando), 19, 20
LOCK (comando), 20
LOG (conversão), 68
LOG ( ) (função), 67, 68
LOMEM (comando), 50
LOOP ("CASO"), 43

Mapa de memória do Apple, 140, 144
Margeamento (programa), 101, 104
Memória
Espaço disponível, 143
Posições para programas binários, 118
RAM, 15
ROM, 15
MENU (programa), 101, 104
MID$ ( ) (função), 57
Modo imediato, 24
Modo não-imediato, 24
MON C, I, O (comando), 102, 137
Monitor, 16, 19
Comando MON C, I, O, 102, 137
Comando NOMON C, I, O, 102, 137
Comando 3DØG, 16, 19
Comando 6'CTRL'P, 17
Entradas pelo teclado, 126
Sub-rotina GETLN, 94
SUB-ROTINA p/tela, 92

NEW (comando), 18, 26
NOMON C, I, O, 102, 137
Normal (comando), 96
Notação Hexadecimal, 98
NOTRACE (comando), 137
Número de linha, 25
Número de ponto flutuante, 26
Números decimais, 26

para conversões a SCII, 146
Números exponenciais, 26
Números inteiros, 25

ONERR GOTO (comando), 90, 91
Opções de Parâmetro
byte, 112
volume, 91, 92
OPEN (comando), 81
Operador Lógico, 34, 38

Palavras reservadas, 37
Parâmetro de comprimento em arquivos TEX randômicos, 86-89
Parâmetros
Em arquivos binários, 97
Em arquivos randômicos, 86
Comando DOS, 128, 129
Funções, 69
Em SUB-rotinas, 69
PEEK código de erro, 90, 91
comando, 90
Periféricos
Comunicação, 78
Encaixes, 15
PLOT (instrução), 47
POKE
Comando, 23
Parâmetro da ('Janela'), 95
Programas em Linguagem de máquina de memória, 96
PR#
A partir do BASIC, 78
Comando, 21
PRINT (instrução), 23, 24, 25
Com Tab., 27, 44, 49
Em arquivos Randômicos, 86
Em arquivos Seqüenciais, 80
Prioridade – Aritmética, 31, 32
Programa ALO, 18, 79
Programa de Apresentação, 18
Programas utilitários
Encadeamento, 99
FID, 98
Renumeração, 103

RAM, 15
Organização de memória, 141
READ
Em arquivos TEXT, 80, 86
Instrução, 30, 31
Recuperação
Arquivos Randômicos, 88
Arquivos Seqüenciais, 80
REM (instrução), 27
Remontando números inteiros, 114
RENAME (comando), 20
Representação de dados numéricos, 140
RESET
tecla, 16
contornando, 19
RESTORE (instrução), 31
RETURN-TECLA, 16
RIGHT$ ( ) (função), 57
RND ( ) (função), 67, 68
ROM, 15
ROT (comando), 118, 122
RUN (comando), 25
a partir do BASIC, 79

Salvando
Programas BASIC como arquivo TEXT, 102
Tabelas de formas, 124
SAVE (comando), 18
SCALE (comando), 118, 122
SGN ( ) (função), 67, 68
SHIFT –, tecla, 16
SIN ( ) (função), 67, 68
SQR ( ) (função), 67, 68
STEP (modificador), 42
STR (função), 64, 65
STRING
Arrays, 55, 56
Concatenação, 54, 63
Funções, 57, 64, 65
Subprogramas, 67, 69, 76
Sub-rotinas, 69, 76
Aninhadas, 74, 77
Entradas múltiplas, 76, 77
Parâmetros, 71, 76

Tabulação, 27
TAN ( ) (função), 67, 68
Teclado, 16
Entrada de Monitorização, 126

Tela
Edição, 21
"Janela", 95
"Parâmetros da Janela", 95
TEXT (comando), 46
TRACE (comando), 137

UNLOCK (comando), 20

VAL ( ) (função), 65
Variáveis, 26
Subscritas, 58
Vetor movimento p/tabelas, 120
VLIN (instrução), 47
VTAB (instrução), 28

WRITE (comando), 81

XDRAW (comando), 118, 121

Índice de figuras

2.1 Fluxograma do programa PERCENT 36
2.2 Classificação de cartas em ordem decrescente 38
2.3 Fluxograma para classificar três números 39
2.4 Algumas configurações de loop 44
3.1 Gráfico de barra das despesas domésticas 48
4.1 Fluxograma do programa de classificação pelo método da bolha 64
5.1 Representação em diagrama de um programa com sub-rotina 71
5.2 Fluxograma do programa dos juros 74
5.3 Representação em diagrama de sub-rotinas aninhadas 75
7.1 Representação visual do processo de edição 94
9.1 Representação esquemática da tabela de formas 119
E.1 Representação das memórias ROM e RAM do Apple 142
E.2 Mapa da memória RAM do Apple 142

Índice de tabelas

1.1 Operadores aritméticos e suas prioridades 32
2.1 Operadores lógicos 36
3.1 Cores em gráficos de baixa-resolução 46
3.2 Cores em gráficos de alta-resolução 50
4.1 Conversão de caracteres para o código ASCII 55
5.1 Funções BASIC standard 67
5.2 Funções matemáticas derivadas 70
7.1 Código de erro do DOS 90
7.2 Sub-rotinas do monitor afetando texto 92
7.3 Endereços de memória relacionados às janelas na tela 95
8.1 Representação de byte 116
9.1 Movimentos de vetor 120
9.2 Tabela p/definições de formas 123
B-1 Códigos de erro do Applesoft 131
B-2 Códigos de erro do DOS 131
E.1 Valores do HIMEM 143
E.2 Espaço de memória disponível 144
G.1 Códigos de conversão decimal ASCII 146

Impressão e Acabamento



"MARQUES-SARAIVA"
ESTABELECIMENTOS GRÁFICOS S. A.
Rua Santos Rodrigues, 240 –
Estácio de Sá – 20250
Tels.: 273-9498 – 273-9447
– 273-9449 – RJ

MANUAL APPLE INCLUINDO BASIC

Este é um manual especificamente orientado para microcomputadores Apple II equipados com uma unidade de disco.

A primeira parte se destina ao leitor iniciante, sem qualquer conhecimento a respeito de computadores e suas linguagens. A Linguagem BASIC lhe será apresentada e explicada com a ajuda de programas simples, que ele será encorajado a digitar, preservar no disco e continuar a aperfeiçoar à proporção que se for familiarizando com novos comandos e instruções mais complexos.

Desta forma, irá organizando sua própria biblioteca com programas e sub-rotinas que lhe servirão de base para as técnicas de programação mais avançadas e as aplicações de arquivos em disco que serão analisadas na segunda parte do livro.

O leitor mais experiente se beneficiará, portanto, dessa segunda parte, onde receberá orientação sobre tópicos tais como programas utilitários, rotinas de erro, sub-rotinas do Monitor, manuseio da tela, etc. Quanto às aplicações dos arquivos em disco e, em particular, às técnicas de bancos de dados, são analisados alguns de seus aspectos, entre os quais o uso do disco, arquivos de acesso seqüencial e aleatório, menus, gráficos de alta resolução, etc.

As soluções dos problemas propostos ao longo do livro encontram-se no último anexo.

ISBN 85-7001-402-3
(Edição original: ISBN 0-905104-26-9 Sigma Technical Press, Cheshire, U.K.)

KANTARIS
MANUAL APPLE INCLUINDO BASIC