BASIC
PARA MICROS PESSOAIS

Importante apresentação da linguagem BASIC, em forma extremamente didática. Cada capítulo traz um guia de estudo dirigido e muitos exercícios, fornecendo ao leitor as informações básicas para a perfeita utilização de microcomputadores pessoais, e também os de teclado plano.

Jorge da Cunha Pereira Filho é um conhecido profissional de Processamento de Dados, desde 1970. Graduou-se em Engenharia Civil, pela UERJ, e obteve o título de mestre em Ciências em Engenharia de Sistemas e Computação, pela COPPE. É autor de dois outros livros na área, inclusive o BASIC BÁSICO, *best-seller* na área de Computação.

ISBN 85-7001-320-5

Jorge da Cunha Pereira Filho — BASIC PARA MICROS PESSOAIS

Jorge da Cunha Pereira Filho

BASIC
PARA MICROS PESSOAIS

3ª EDIÇÃO

EDITORA
CAMPUS

TÍTULOS DE INTERESSE CORRELATO

BASIC

BASIC COM ESTILO: PROVÉRBIOS DE PROGRAMAÇÃO – *P. Nagin e H. Ledgard*

ENCICLOPÉDIA DA LINGUAGEM BASIC – *C. Pereira e R. Alcantara*

BASIC BÁSICO, 5ª Edição – *J. C. Pereira Filho*

Conheça toda a linha de Informática da Editora Campus, com títulos nas áreas de: Introdução à Computação; Teoria, Organização e Processamento de Dados; Linguagens; Programação; Programas e Aplicativos; Sistemas Operacionais e Compiladores; Arquitetura e Equipamentos; Interesse Especial.

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.

O COMPUTADOR ENGUIÇOU – *Gabor Geszti*
Livro de humor, fartamente ilustrado, analisando espirituosamente a relação homem x máquina.

MICROBITS
Macroinformações e Programas para Micros TK 82-83-85-90X, CP-200 e compatíveis. Uma publicação bimestral com análises de hardware, aplicativos, artigos sobre linguagem de máquina, dicas de programação e respostas para suas dúvidas.

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
20261 – Rio de Janeiro – RJ – Brasil
Telefone: (021) 284 8443

Atendemos também pelo reembolso postal

Jorge da Cunha Pereira Filho

Mestre em Ciências em Engenharia de Sistemas e Computação

BASIC
PARA MICROS PESSOAIS

3ª edição

EDITORA
CAMPUS LTDA.
Rio de Janeiro

2ª Edição, 1984.
3ª Edição, 1986.

Capa
Otavio Studart

Diagramação, composição, paginação 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
20261 Rio de Janeiro RJ Brasil
Endereço Telegráfico: CAMPUSRIO

ISBN 85-7001-320-5

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

Pereira Filho, Jorge da Cunha, 1937-
P491b BASIC para micros pessoais / Jorge da Cunha Pereira
3.ed. Filho. – 3.ed. – Rio de Janeiro: Campus, 1986.

Apêndice.
Bibliografia.
ISBN 85-7001-320-5

1. BASIC (Linguagem de programação para computadores). I. Título.

86-0151 CDD – 001.6424
CDU – 800.92BASIC

Aos meus mestres, com os quais tive o privilégio de aprender, o que tornou possível chegar a escrever este livro. Em particular referencio a Tabajara Pedroso, Felipe dos Santos Reis, João Cordeiro da Graça Filho, Otávio Galvão Ramos, Hélio dos Santos e Roberto Chaves.

SUMÁRIO

APRESENTAÇÃO

Criada em 1960, por John G. Kemeny e Thomas E. Kurtz, professores do Dartmouth College, Hanover, New Hampshire, EUA, inspirados na linguagem FORTRAN, com o objetivo de facilitar o aprendizado de programação de computadores e a execução de cálculos matemáticos, conversacionalmente, a linguagem BASIC, decorridas pouco mais de duas décadas, apresenta-se como a mais popular e utilizada em todo o mundo.

Inicialmente disponível em minicomputadores, seu uso espalhou-se muito mais com o aparecimento dos microcomputadores. Praticamente, não há míni ou microcomputador que não disponha de compilador da linguagem BASIC.

Para os microcomputadores de uso pessoal, a linguagem BASIC é obrigatória, por ser de fácil aprendizado e utilização.

Por este motivo o autor destina ao público mais uma publicação sobre BASIC, desta vez voltada para as versões da linguagem que se encontram em uso em microcomputadores pessoais.

Aproveita para sublinhar o descortínio da Editora, que já editou *BASIC Básico*, também de sua autoria, o primeiro livro em língua portuguesa sobre a linguagem, à qual agradece, nas pessoas de Cláudio Marcello Rothmüller, Carlos Hamilton Rocha e Antonio F. Carpinteiro, por mais esta oportunidade. Agradece também a sua esposa, Marlene, a tolerância e compreensão com os transtornos causados para a produção do livro.

Para um integral aproveitamento da matéria exposta, recomenda-se a utilização, durante o aprendizado, de um microcomputador pessoal.

O livro se desdobra em dez capítulos, com numerosos exemplos, abordando os principais tópicos sobre a programação em BASIC para os microcomputadores pessoais, conforme consta do sumário. Cada capítulo vem acompanhado de exercícios, que devem ser feitos em um microcomputador, para um efetivo aproveitamento da matéria exposta.

O AUTOR

CONVENÇÕES GERAIS

1. Na apresentação da forma geral das instruções, são usados metassímbolos, para indicar componentes opcionais, alternativos e repetidos. Essas convenções são:

TIPO DE COMPONENTE	METASSÍMBOLO
opcionais	[componente]
alternativos	{componente-1 | componente-2 | ... | componente-*n*}
opcionais e alternativos	[componente-1 | componente-2 | ... | componente-*n*]
repetidos	idem ...

2. As palavras reservadas da linguagem BASIC aparecem em caixa alta (maiúsculas) e, destas, as palavras-chave, que definem funções, aparecem em negrito.
3. Os nomes e elementos que devem ser supridos pelo programador aparecem em caixa baixa (minúsculas) e itálico (letras inclinadas), entre os sinais < e >.

PRIMEIRO CAPÍTULO
O MICROCOMPUTADOR

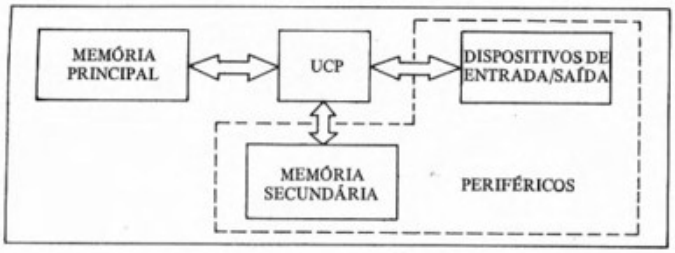
microcomputadores – o equipamento – os programas – as aplicações – microcomputadores pessoais

1.1 Microcomputadores

Microcomputadores são computadores eletrônicos digitais construídos com a mais avançada tecnologia eletrônica. Tornaram-se viáveis com o desenvolvimento da chamada *microeletrônica*, uma tecnologia que permite colocar em um único componente eletrônico (pastilha) centenas de milhares de circuitos eletrônicos microscópicos, a chamada *integração em larga escala*. A integração em larga escala é possível pelo uso de materiais *semicondutores*, principalmente o *silício*. Desta forma é possível construir toda a *unidade central de processamento* (UCP) de um computador dentro de uma única pastilha, um único componente, que é chamada de *microprocessador*. Portanto, o *microprocessador* é a UCP do *microcomputador*, seu órgão mais importante, onde se processa a execução das *instruções* e da qual resulta o funcionamento do microcomputador. Além da UCP, o microcomputador tem que dispor de *memórias*, onde possa armazenar *programas* e *dados*. A *memória principal* é um dispositivo de armazenamento *temporário*, de alta velocidade, também construído com circuitos integrados em larga escala, de semicondutores. Além da memória principal, temporária, os programas e dados podem ser armazenados de forma *permanente*, para uso posterior, em *memórias secundárias*, que são dispositivos externos, em geral eletromecânicos, relativamente mais lentos, como, por exemplo, as *unidades de fitas magnéticas* e *unidades de discos magnéticos*. Além disso, os microcomputadores possuem também dispositivos para permitir a comunicação com o *usuário*, isto é, dispositivos de *entrada e saída de dados*, como o *teclado*, que é um dispositivo de *entrada de dados*, e o *vídeo* e a *impressora*, que são dispositivos para a *saída de dados*. Os dispositivos externos, onde são realizadas as operações de entrada/saída, são denominados, em conjunto, de *periféricos*.

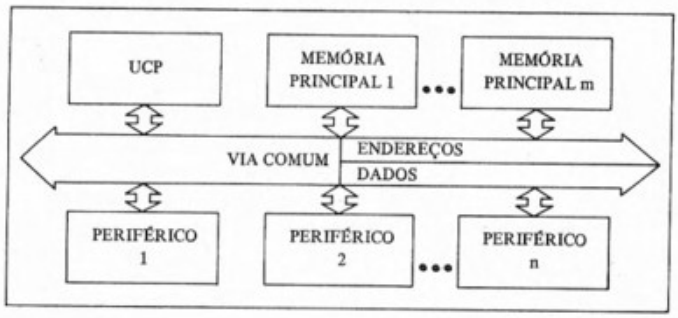
A figura 1-1 mostra o diagrama de grandes blocos que constituem a configuração de um microcomputador.

FIGURA 1-1: *CONFIGURAÇÃO GERAL DE UM MICROCOMPUTADOR*



Os microcomputadores, no que diz respeito ao *equipamento* (*hardware*), possuem uma arquitetura que pode ser chamada de *via comum*. Essa arquitetura consiste em colocar todos os módulos, como UCP, memórias e periféricos, ligados a uma via que conduz os sinais de endereços e dados a todos os pontos do sistema. Esquematicamente, essa arquitetura é apresentada na figura 1-2.

FIGURA 1-2: *ARQUITETURA GERAL DE MICROCOMPUTADORES*



Com relação ao tipo de via comum que possuem, os microcomputadores classificam-se em dois grandes tipos: *S-1ØØ* e *não S-1ØØ*. A via do tipo S-1ØØ recebeu esse nome por ser constituída de 1ØØ linhas de controle, endereços e dados. Foi criada por um fabricante de micros que foi logo imitado

por outros e, naturalmente, constitui um padrão para a indústria de micros. Todavia muitos preferem projetar sua própria via, fora do padrão, com a vantagem de conseguir microcomputadores de dimensões muito reduzidas. Apesar de serem maiores, os microcomputadores do tipo S-1ØØ apresentam como vantagem a possibilidade de adaptação de um grande número de módulos de diversos fabricantes diferentes (com alguns problemas). Todavia, mesmo os microcomputadores que *não* são do tipo S-1ØØ já costumam apresentar adaptadores para extensão da via, no padrão S-1ØØ, o que permite usar também módulos e periféricos do tipo S-1ØØ.

Uma outra distinção que costuma ser feita, com relação aos microcomputadores, é quanto à sua utilização ou mercado de aplicação. Uma primeira categoria é destinada às pessoas que vêem na atividade de construir computadores um *hobby*. Os microcomputadores desse tipo são fornecidos desmontados, em forma de *kits* ou em partes isoladas, de vários fabricantes. A maioria desses fabricantes só produz o equipamento. Neste caso, o usuário tem que desenvolver todos os programas. Não trataremos desse tipo de microcomputador. O microcomputador de que vamos tratar neste livro é o microcomputador montado, pronto para usar (*turn key*) destinado aos que querem apenas utilizá-lo, aplicá-lo na solução de problemas, sem ter que construí-lo. Os fabricantes desse tipo de microcomputadores em geral já fornecem os programas necessários ao funcionamento da máquina e até mesmo algumas aplicações padronizadas. Um grupo intermediário é apresentado também em forma de *kit*, mas o fabricante também fornece os programas essenciais à operação do microcomputador, como o de supervisão. Chama-se aos microcomputadores da primeira categoria, sem programas, de *1ª geração* de microcomputadores, pois foram os primeiros disponíveis no mercado, na década de 70. Na década de 80 surgiram os microcomputadores prontos para uso, chamados de *2ª geração*, que incluem, além do equipamento, os programas que permitem usar, aplicar o computador. Os microcomputadores de *2ª geração* são divididos conforme seu uso em *pessoais* e *comerciais*. Neste livro, em particular, trata-se dos microcomputadores pessoais. Além dessas categorias, existem outras como: *domésticos* (*home*), *de mesa* (*desk top*) etc.

1.2 O equipamento

A *UCP* ou *microprocessador* contém todas as unidades comuns aos computadores, como a *unidade de controle* (UC), a *unidade de lógica e aritmética* (ULA) e o *fluxo de dados* (ou *conjunto de registradores*).

A *unidade de controle* (UC) tem por função controlar todas as demais unidades do sistema e é quem decodifica e executa as instruções de programa, em primeira instância.

A *unidade de lógica e aritmética* (ULA) tem por função executar as ordens recebidas da UC, para realizar as operações e funções que correspondem

à instrução que está sendo correntemente executada, sejam lógicas ou aritméticas.

O *fluxo de dados* ou *conjunto de registradores* constitui a memória local e temporária da UCP onde são armazenados os dados, enquanto estão sendo realizadas as operações correspondentes à instrução que está sendo executada correntemente. Serve também para passar dados entre uma instrução e as seguintes.

Apesar do tamanho reduzido da UCP, graças à integração em larga escala, o número de registradores é, em geral, maior do que o dos minicomputadores ou mesmo computadores de maior porte. O conjunto de instruções também rivaliza com os conjuntos de instruções de máquinas de maior porte. A única grande diferença em relação aos computadores de maior porte estava no tamanho da *palavra*, isto é, o número máximo de sinais de dados que fluem na via entre UCP, memórias e periféricos. Cada linha de dados conduz uma unidade de informação correspondente a um *bit* (*binary digit*). Um *bit* de informação indica um entre dois *estados* possíveis: *0* ou *1*, *sim* ou *não*, *falso* ou *verdadeiro*, *ligado* ou *desligado*. Um conjunto de oito *bits* é chamado de *byte*. Nos microprocessadores, o tamanho de palavra, durante quase uma década, esteve restrito a um *byte*. Atualmente, já existem microprocessadores com tamanho de palavra de dois e até mesmo quatro *bytes*, o que caracteriza os minicomputadores e mesmo computadores de pequeno e médio porte. Portanto, embora a grande maioria dos microprocessadores tenha palavra de oito *bits*, progressivamente estão sendo substituídos por microprocessadores de 16 *bits*. A vantagem evidente é o aumento de velocidade e desempenho. A representação interna de dados no computador é feita caractere a caractere, cada um correspondendo a um *byte*. A cada *caractere* corresponde um *código binário* (constituído por *0s e 1s*) e existem vários códigos utilizados para isso, sendo os mais comuns conhecidos pelas siglas BCD, EBCDIC, ASCII etc. Uma *cadeia de caracteres* é representada na memória interna do computador por uma seqüência de posições de um *byte* cada uma. Todavia, para *valores numéricos* é usual outro tipo de representação, mais compacta. Os números *inteiros* são representados em uma palavra, através de um valor numérico binário (número representado no sistema de numeração de base 2). Os valores numéricos *fracionários* são representados em forma *exponencial*, separadamente, o *expoente* (*característica*) e a *base* (*mantissa*). Cada posição da memória pode ser referenciada pelo número da posição, que é o seu endereço.

A *memória principal do microcomputador* é constituída por blocos ou bancos de memórias, cada um construído com pastilhas de semicondutores integrados em larga escala. Existem diversos tipos de memórias de semicondutores, como as do tipo *ROM* (*read only memories*, que só podem ser lidas), do tipo *PROM* (*programable ROM*, que só podem ser gravadas fora do computador) e do tipo *RAM* (*random access memories*, que podem ser lidas e gravadas). Esses diversos tipos de memórias são combinados de forma adequada, dependendo da função que têm, para constituírem os blocos de memórias. Na memória principal residem os programas e dados que estão sendo *correntemente* processados. Nos *endereços* que correspondem a memórias *RAM* são armazenados *temporariamente* os programas e dados do usuário. Nos endereços correspondentes às memórias *ROM* e *PROM* são armazenados *permanentemente* os programas de supervisão do microcomputador e programas de serviço, como o interpretador BASIC. A velocidade de acesso às memórias principais é da ordem de *nanossegundos*.

As *memórias auxiliares* aumentam de muito as facilidades que o sistema oferece ao usuário. Tornam possível armazenar *permanentemente* os programas e, eventualmente, dados, na forma de *arquivos*. As principais memórias auxiliares são as *fitas* e *discos magnéticos*. A velocidade de acesso às memórias auxiliares é da ordem de *microssegundos*.

Os microcomputadores geralmente utilizam como *unidade de fitas magnéticas* os *gravadores/reprodutores de som*, principalmente os de *fita cassete* (*de áudio*). Existem alguns, entretanto, que têm dispositivos especiais para leitura e gravação de *cartuchos de dados*, que são cassetes especiais para processamento de dados, com fitas de maior densidade e velocidade. As fitas magnéticas são feitas de material plástico flexível e revestidas em uma das faces com material magnetizável.

As *unidades de discos* mais usuais em microcomputadores são as de *discos flexíveis*. Os discos flexíveis são feitos de um plástico resistente (*mylar*), revestido de material magnetizável. Há uma tendência ao aumento do uso de discos rígidos (base de *alumínio*), em unidade selada e no vácuo – a chamada tecnologia *Winchester* –, mas são mais caros atualmente do que os flexíveis, embora de muito maior capacidade, densidade e velocidade.

Os principais *periféricos* de microcomputadores são as *impressoras* e as unidades de *vídeo* e o *teclado*.

As *impressoras* são dispositivos que imprimem caracteres, em forma gráfica e legível, sobre papel, o *formulário contínuo*. Basicamente, as impressoras podem ser *de impacto* ou *sem impacto*. As impressoras de impacto assemelham-se às máquinas de escrever comuns, no sentido de que os tipos batem sobre uma fita entintada, que marca o papel. Existem as impressoras de linhas e as de caracteres que, como o nome indica, imprimem, respectivamente, uma linha de uma vez e caractere a caractere. As impressoras sem impacto mais usuais operam termicamente. Impressoras térmicas requerem o uso de papel especial.

A *unidade de vídeo* dispensa maiores descrições por ser muito conhecida, principalmente dos televisores domésticos, onde se chama tela.

O *teclado* também é um dispositivo que dispensa apresentação porque está disponível em máquinas de escrever, telex etc.

1.3 Os programas

Um computador é uma máquina que se distingue das demais porque pode assumir os mais diversos *padrões de comportamento*. Essa mudança de comportamento é possível graças à sua característica única de executar um *programa*, ou seja, uma seqüência de *instruções*, que é armazenado em sua própria memória interna (principal). Substituindo-se os programas (*software*), obtêm-se diversos padrões de comportamento diferentes.

É possível distinguir duas camadas de programas, a primeira das quais está intimamente ligada à máquina ou equipamento e sem a qual a operação do computador tornar-se-ia senão impossível, pelo menos muito difícil, ineficiente e inútil. Essa primeira camada constitui o chamado *software básico* ou *programas básicos* e é aquele conjunto de programas que é fornecido pelo fabricante, como parte integrante do computador, com o qual o seu uso se torna viável e econômico. A camada mais externa, cuja utilização se tornou viável e econômica graças à mais interna, é chamada de *software de aplicação* ou *programas de aplicação* ou *aplicativos*. Os programas de *aplicação*, como seu nome indica, estão destinados a resolver os problemas do usuário, as chamadas *aplicações*. A figura 1-3 apresenta um diagrama com as três principais camadas de um *sistema de computação*.

FIGURA 1-3: *CAMADAS DE UM SISTEMA DE COMPUTAÇÃO*

Os *programas básicos* permitem uma divisão em dois tipos de programas diferentes: o *sistema operacional* e os *programas de serviço* ou *de processamento*. O *sistema operacional* é responsável pela gerência de todo o sistema e também é conhecido como *monitor, executivo, supervisor, núcleo* ou *programa de controle*. É o sistema operacional que permite o diálogo entre o usuário e o computador, tornando-o utilizável ou operacional.

Os sistemas operacionais, para fins meramente didáticos, podem ser divididos em três tipos básicos, quanto ao modo de operação, como mostra o quadro 1-1.

QUADRO 1.1: *TIPOS DE SISTEMAS OPERACIONAIS*

DESIGNAÇÃO	DESCRIÇÃO
processamento em batelada ou de lotes (*batch*)	destinado ao processamento de grandes massas de dados, que formam um fluxo contínuo de serviços (*job stream*).
tempo compartilhado (*time sharing*)	processamento interativo de serviços de diversos usuários, simultaneamente, a partir de terminais, conversacionalmente.
tempo real (*real time*)	processamento assíncrono de eventos aleatórios, originados em um processo externo ao sistema, o qual se pretende controlar.

Os *programas de serviço* ou *de processamento* são utilizáveis sob o controle do sistema operacional da mesma maneira que os programas do usuário, isto é, as aplicações. A diferença está em que os programas de serviço se destinam a auxiliar a operação do próprio computador, enquanto os programas de aplicação se destinam a resolver os problemas externos ao sistema de computação, as chamadas aplicações. Exemplos de aplicações são a contabilidade, a folha de pagamento, o cálculo estrutural, a programação linear etc.

Entre os programas de serviço encontram-se o *sistema de desenvolvimento de programas* e os *utilitários*. Os programas para desenvolvimento de programas são muito importantes e incluem os *editores, macroexpansores, montadores* (*assemblers*), *compiladores, interpretadores, ligadores, carregadores* etc. Em conjunto, os montadores, compiladores e interpretadores são chamados de programas *tradutores*. A função dos *tradutores* é fundamental para o desenvolvimento de programas e, como seu nome diz, traduzem programas escritos em uma linguagem de programação qualquer, como BASIC, para a linguagem binária ou *linguagem de máquina*, que é a que o computador entende. Entre os *utilitários* estão os programas de administração de arquivos como os de cópia, reorganização, classificação, intercalação etc.

Os programas de aplicação são divididos segundo sua área de aplicação: *técnico-científicos, administrativos* (ou *comerciais*) e *industriais*. Exemplo de programa técnico-científico é o de cálculo de estruturas. Já programas como o de contabilidade ou a folha de pagamento são considerados aplicações administrativas ou comerciais. As aplicações industriais incluem o controle de processos e o controle de produção.

Uma divisão geral dos sistemas de computação é apresentada no quadro 1-2.

QUADRO 1-2: *DIVISÃO DE UM SISTEMA DE COMPUTAÇÃO EM COMPONENTES*

<table>
<tr><td rowspan="9">Sistema de Computação</td><td rowspan="3">Equipamento</td><td colspan="3">UCP – Unidade central de processamento</td></tr>
<tr><td colspan="3">Memória principal</td></tr>
<tr><td colspan="3">Periféricos</td></tr>
<tr><td rowspan="6">Programas</td><td rowspan="3">Básicos</td><td colspan="2">Programa de controle</td></tr>
<tr><td rowspan="2">Programas de processamento</td><td>Desenvolvimento</td></tr>
<tr><td>Utilitários</td></tr>
<tr><td rowspan="3">De aplicação</td><td colspan="2">Técnico-científica</td></tr>
<tr><td colspan="2">Administrativa</td></tr>
<tr><td colspan="2">Industrial</td></tr>
</table>

Para que um programa seja executado, várias etapas devem ser cumpridas. A primeira delas, evidentemente, é desenvolver o *algoritmo* ou *lógica do programa*. A segunda etapa consiste em *codificar o programa*, isto é, transcrever o algoritmo utilizando uma *linguagem de programação* que seja mais fácil de utilizar do que a linguagem de máquina. As linguagens utilizadas pelos programadores podem ser classificadas em dois níveis: *baixo* e *alto*. As linguagens de *baixo nível* são as linguagens *simbólicas* ou *mnemônicas* ou de *montagem* (*assembly languages*) e devem o seu nome ao fato de estarem muito próximas da de máquina, isto é, estão voltadas para a máquina e existe, praticamente, uma relação de 1 para 1 entre as instruções de linguagem simbólica e de máquina. Além disso, as linguagens de montagem são particulares de cada computador. O programa tradutor de linguagem de montagem é o *montador* (*assembler*). As linguagens *de alto nível* são aquelas que *não* estão voltadas para a máquina mas sim para o problema. Entre as linguagens de alto nível mais usuais se encontram: FORTRAN, BASIC, COBOL, PL/I, ALGOL, PASCAL etc. As linguagens de alto nível são traduzidas para linguagem de máquina por tradutores chamados de *compiladores*. Eventualmente as linguagens de alto nível serão interpretadas pelo programa *interpretador* ou *intérprete*. O texto do programa escrito em linguagem de programação (que *não* a de máquina) é chamado de texto ou programa-*fonte*. Depois que esse texto ou programa é traduzido para a linguagem de máquina, chama-se de programa-*objeto*. A diferença entre a montagem ou compilação e a interpretação é que, nesta última, o programa *não* é traduzido para linguagem de máquina e sua execução é simulada por uma interpretação ou execução diretamente pelo interpretador. O interpretador toma todas as ações que o programa-fonte solicita, mas quem está sendo executado pela UCP é o interpretador e *não* o programa-fonte. Eventualmente, existem sistemas que fazem uma compilação parcial, isto é, o programa-fonte é traduzido para um código intermediário, mais fácil de ser interpretado mas que *não* é código de máquina. Os microcomputadores pessoais utilizam em geral a interpretação, diretamente a partir do programa em texto-fonte. Os microcomputadores para uso comercial geralmente utilizam a compilação do programa. A interpretação de programas-fonte é mais lenta do que a execução, diretamente a partir do código-objeto. Todavia oferece a enorme vantagem, principalmente ao usuário individual, de não ter que administrar todo o processo de produzir um programa executável. Em compensação, a execução do programa-objeto é muito mais eficiente (milhares de vezes) em relação à interpretação, embora seja um processo mais complicado, que comporta diversas etapas. No caso de compilação, o programa-objeto na verdade *não* é diretamente executável e requer mais uma etapa de processamento, para produzir uma *carga de memória*, que é a ligação, feita pelo programa *ligador*.

Após a produção da carga de memória, o programa poderá ser executado. Deverá ser, então, carregado na memória principal pelo programa *carregador*. Após o carregamento, o carregador passa o controle para o programa carregado e é iniciada a sua execução.

Na figura 1-4 são apresentados os diagramas que descrevem os dois processos, de interpretação e de compilação, até chegar à execução do programa.

Do ponto de vista do usuário, tanto usando a interpretação como a compilação, os resultados finais da execução são os mesmos. A execução do programa, do ponto de vista do usuário (o que ele vê), é representada na figura 1-5.

Os programas de controle e interpretadores BASIC dos microcomputadores podem ser baseados em disco (ou disquete), a partir de onde serão carregados na memória principal, ou residirem permanentemente na memória

FIGURA 1-4: *DIFERENÇA ENTRE A INTERPRETAÇÃO E A COMPILAÇÃO*

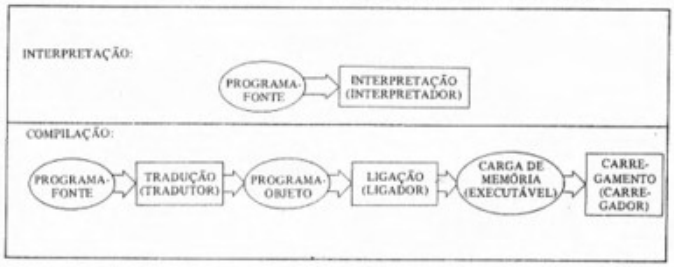
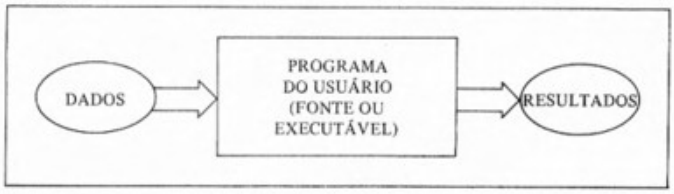


FIGURA 1-5: *EXECUÇÃO DE PROGRAMA*



principal, em endereços de memória correspondentes a um bloco de memória implementado em *ROM*. Os microcomputadores pessoais que *não* dispõem de disquete ou disco têm o programa de controle e o interpretador BASIC residentes na memória principal, em *ROM*. Alguns microcomputadores têm programas de controle particulares, mas muitos usam programas de controle gerais ou padrão, como *CP/M* (*Control Program/Monitor*), *UNIX* ou *DOS* (*Disk Operating System*), que podem estar implementados total ou parcialmente.

1.4 As aplicações

Quem adquire um microcomputador pessoal deseja utilizá-lo da maneira mais fácil e tão rapidamente quanto possível, o que é o caso de mais de 90% dos seus usuários. A grande motivação para a compra de um microcomputador pessoal é a possibilidade de sua aplicação na solução de problemas. Quais são os possíveis campos de aplicação dos microcomputadores pessoais?

As grandes áreas de aplicação de microcomputadores pessoais são: *educação, música, jogos* e *administração* da economia pessoal e doméstica. As alternativas de solucionar problemas nessas áreas incluem a compra de programas prontos ou o desenvolvimento dos programas pelo próprio usuário.

A aplicação do microcomputador na *educação* é um campo que está em crescente expansão. Pode-se aprender desde aritmética até aerodinâmica. Em geral os programas e lições estão disponíveis para venda pelas casas de programação que os produziram ou até mesmo em lojas de microcomputadores, mas a maioria ainda se encontra em idioma inglês. Como alternativa, existem programas que dispõem de uma linguagem especial, que permite ao próprio usuário escrever as lições e testes de aproveitamento. Após a utilização de cada lição é aplicado um teste de avaliação, sendo verificado o aproveitamento do aluno. Tanto o preparo como a aplicação das lições e testes são feitos em forma de *cardápio* ou *menu*, guiando todos os passos do usuário. Se o usuário dispuser de tempo, poderá desenvolver seus próprios programas de ensino.

A aplicação do microcomputador na *música* é outro campo em crescimento mas, evidentemente, é preciso que o microcomputador disponha de uma unidade geradora de som, quase sempre embutida dentro da própria caixa. O teclado do microcomputador é usado como o teclado de um instrumento musical, diretamente, ou a pauta musical pode ser armazenada para ser interpretada por um programa que a transforma em sons. Além disso, composições musicais podem ser produzidas com o auxílio do microcomputador ou pode-se programar o microcomputador para gerar (compor) a própria música. Existem programas prontos e composições musicais para microcomputador à venda em casas de programação e lojas de microcomputadores e muitos fabricantes, para demonstrar todas as possibilidades do seu produto, fornecem programas e composições de demonstração. Se o usuário tiver real interesse e dispuser de tempo suficiente, poderá desenvolver os seus próprios programas musicais.

O emprego de microcomputadores em *jogos* é talvez a mais popular das aplicações, por ser a mais utilizada em demonstrações pelos fabricantes, distribuidores e lojas de microcomputadores. Certamente, é a área que dispõe do maior número de programas prontos, os quais podem ser encontrados em casas de programação e lojas de microcomputadores. Todavia, produzir programas de jogos pode ser uma tarefa interessante, se o usuário deseja programar. A área é muito dinâmica e existem sempre novos jogos sendo oferecidos.

A aplicação de microcomputadores na *administração pessoal* e *doméstica* é talvez, do ponto de vista econômico, a mais importante das aplicações. Pode-se administrar a caderneta de endereços e telefones mais freqüentemente usados, a composição de dietas para diminuir, manter ou aumentar o peso ou, simplesmente, racionalizar a alimentação, a conta corrente bancária, as contas

a pagar do mês, o fluxo de caixa, a previsão orçamentária doméstica, a preparação do imposto de renda, a contabilidade pessoal, o calendário de compromissos (agenda) etc. Se se incluir o uso profissional entre as aplicações pessoais, o número de aplicações nesta área pode crescer bastante. Entre as aplicações de caráter mais pessoal se inclui o relacionamento de bibliografia técnica sobre a profissão, com possibilidade de consulta e pesquisa sob diversos ângulos. Evidentemente, não apenas neste caso, mas em todas as demais aplicações onde se deseja armazenar e recuperar informações, é indispensável a existência de memórias que permitam o seu armazenamento permanente, isto é, memórias de massa, principalmente discos e disquetes. Existem programas prontos para a execução da maioria dessas tarefas, mas o usuário tem sempre a alternativa de desenvolver ele mesmo o programa, sob medida, para resolver o seu caso específico.

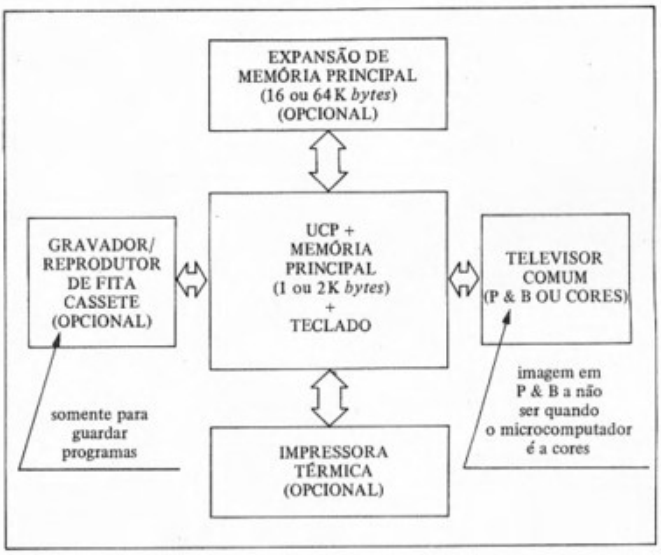
1.5 Microcomputadores pessoais

Os microcomputadores pessoais são os que oferecem a maior gama de alternativas em termos de facilidades, de configuração do equipamento e, conseqüentemente, de preço.

Um primeiro grupo é constituído pelos microcomputadores que só possuem UCP, memória principal e teclado plano instalados em uma única e pequena caixa. O programa de controle e o interpretador BASIC, neste caso, vêm gravados em *ROM*, na memória principal. A unidade de vídeo a ser utilizada é a tela de qualquer televisor doméstico, através da ligação por um cabo à antena do televisor. É possível ligar, opcionalmente, como memória auxiliar, um gravador/reprodutor de som de fita cassete, mas só é usado para guardar programas, evitando que o usuário tenha que digitá-los no teclado toda vez que queira usá-los. Eventualmente, a configuração acima poderá ser expandida pela inclusão de uma impressora térmica, opcional, para impressão de caracteres ou figuras. Em geral, esses microcomputadores são de tamanho muito reduzido e oferecem, também opcionalmente, expansão da memória principal que, originalmente, tem 1 ou 2K (1K = 1.Ø24) posições (ou caracteres ou *bytes*) e pode chegar a 16K posições ou mesmo 64K posições. Em relação aos demais microcomputadores, o primeiro grupo oferece ainda pouca possibilidade de expansão, mas em compensação o seu preço é muito reduzido e podem ser resolvidos os problemas mais comuns, que não requeiram o armazenamento de grandes volumes de dados, porque no momento eles ainda não dispõem de arquivos em memórias de massa, possíveis somente pela inclusão, na configuração, de discos e disquetes.

Um diagrama que mostra a configuração de um microcomputador pessoal pertencente ao primeiro grupo encontra-se na figura 1-6.

FIGURA 1-6: *EXEMPLOS DE CONFIGURAÇÃO DE MICROCOMPUTADOR PESSOAL DO PRIMEIRO GRUPO*



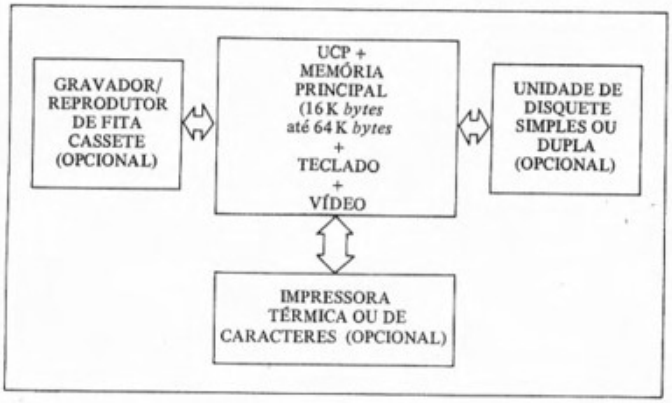
O exemplo de maior sucesso até o momento de microcomputador pertencente ao primeiro grupo é o da marca SINCLAIR, de origem inglesa, que vem sendo produzido e também imitado em vários países. No Brasil, constituem exemplos do modelo acima os microcomputadores pessoais da marca TK 82-C e NE-Z8ØØØ. O preço de lançamento desse tipo de microcomputador no mercado internacional foi de US$99, o da expansão de memória de 16K *bytes*, de US$39 e o da impressora térmica opcional, de US$99, em 1981/82.

O segundo grupo de microcomputadores pessoais é constituído pelos que possuem, além da UCP, memória principal e teclado, também a unidade de vídeo, instalados em uma única caixa, em geral com as dimensões de um aparelho de televisão doméstico. Esse tipo de microcomputador permite uma maior expansão da configuração básica. Em geral tem local reservado na pró-

pria caixa para abrigar até duas unidades de discos flexíveis ou unidade de fita cassete, que são oferecidas já embutidas, em modelos mais avançados. Além disso, quase sempre prevêem a expansão da memória auxiliar com o acréscimo de uma ou duas unidades de disquete, externas, ou um ou dois gravadores de fita cassete. Todos permitem a ligação opcional de uma impressora, que pode ser térmica ou de caracteres (impacto). Nas versões mais simples, sem disquetes, o programa de controle e o interpretador BASIC são residentes em *ROM*, na memória principal.

Um diagrama que mostra a configuração simples típica de um microcomputador pessoal pertencente ao segundo grupo é apresentado na figura 1-7.

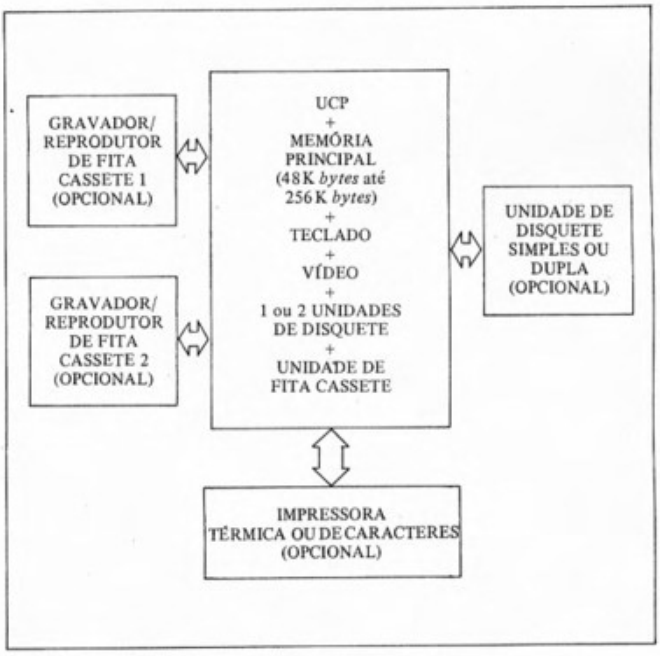
FIGURA 1-7: *EXEMPLO DE CONFIGURAÇÃO MAIS SIMPLES DE MICROCOMPUTADOR PESSOAL DO SEGUNDO GRUPO*



Nas versões mais completas, que incluem unidades de disquete, o programa de controle e o interpretador residem, geralmente, em disquete, a partir de onde serão carregados na memória principal. As versões mais completas permitem o uso de arquivos, em fitas cassete e em disquete, o que confere a esses microcomputadores a possibilidade de armazenar programas e grandes volumes de dados e portanto realizarem aplicações mais sofisticadas.

Um diagrama que mostra a configuração completa típica de um microcomputador pessoal pertencente ao segundo grupo é apresentado na figura 1-8.

FIGURA 1-8: *EXEMPLO DE CONFIGURAÇÃO COMPLETA DE MICROCOMPUTADOR PESSOAL DO SEGUNDO GRUPO*



O exemplo de microcomputador do segundo grupo de maior sucesso no mercado internacional é o da marca TRS-8Ø, modelo III. No Brasil, citam-se dentro do segundo grupo os microcomputadores Digitus DGT-1ØØ, Dismac D-8ØØØ e Prológica CP-5ØØ, entre outros.

A linguagem BASIC que se apresentará neste livro estará voltada principalmente para os microcomputadores pessoais do primeiro grupo mas também atenderá à maioria dos microcomputadores do segundo grupo, excluindo-se apenas o que diz respeito à manipulação de arquivos e ressalvando-se pequenas diferenças de sintaxe em uma ou outra instrução da linguagem.

Exercícios:

ESTUDO DIRIGIDO:

1) Que é microcomputador? Que é microprocessador? Qual a função da UCP? Que tecnologia é usada na construção da UCP? Qual a função das memórias? Que função tem a memória principal? Qual a tecnologia usada na construção da memória principal? Que é integração em larga escala? Qual a função das memórias secundárias? Quais os principais tipos de memórias secundárias? Que são dispositivos de entrada/saída? Qual a função dos dispositivos de entrada/saída? Que são periféricos? Que tipo de dispositivo é um teclado? E o vídeo, como se classifica? A impressora é de que tipo? Que é *hardware*? O que é via comum? Com relação ao tipo de via, como se classificam os microcomputadores? Qual a vantagem da via S-1ØØ? E a desvantagem? Qual a vantagem da via *não* S-1ØØ? E a desvantagem? É possível ter periféricos do tipo S-1ØØ ligados em microcomputadores do tipo *não* S-1ØØ? Como? Quanto à utilização ou mercado, como classificar os microcomputadores? Microcomputador para *hobby* é o mesmo que microcomputador montado (*turn key*)? Que é um *kit*? Que é microcomputador de 1ª geração? Que é microcomputador de 2ª geração? Que são microcomputadores pessoais? Que são microcomputadores comerciais? Existem outras categorias de microcomputadores?

2) Que unidades compõem a UCP? Qual a função da unidade de controle? Qual a função da ULA? Qual a função do fluxo de dados? Que informação contém um *bit*? Quantos *bits* tem o *byte*? A que corresponde uma palavra? Qual o tamanho de palavra dos microprocessadores mais comuns? Qual o tamanho de palavra dos novos microprocessadores? Qual a tendência do tamanho de palavra? Como são representados internamente os caracteres no computador? Para que servem os códigos BCD, EBCDIC e ASCII? Como é representada internamente no computador uma cadeia de caracteres? Como é feita a representação interna dos números inteiros? Como é feita a representação interna de números fracionários? Como se chama o número atribuído a uma posição de memória e que a identifica unicamente? Como é feita a referência a uma posição de memória? Qual o significado das siglas *ROM, PROM* e *RAM*? É possível combinar diversos tipos de memórias de semicondutores? Onde são colocados os programas e dados do usuário para a execução? Onde são armazenados o programa de controle e o interpretador BASIC para serem executados? Qual a diferença entre o armazenamento em *ROM* e *RAM*? Qual a ordem de grandeza da velocidade das memórias principais? Qual a ordem de grandeza da velocidade das memórias auxiliares? Quais são as unidades de fita magnética mais usadas em microcomputadores? Quais as unidades de discos magnéticos mais usadas em microcomputadores? De que materiais são feitas as fitas magnéticas? De que materiais são feitos os discos magnéticos flexíveis? De que materiais são feitos os discos magnéticos rígidos? Como se chama a tecnologia usada nos discos magnéticos rígidos em unidade selada e a vácuo? Existem vantagens no uso de discos flexíveis? E desvantagens? Existem vantagens na utilização dos discos rígidos? E desvantagens? Existe diferença entre um cartucho (cassete) de áudio e de dados? Quais os principais periféricos usados em microcomputadores? Qual o princípio de funcionamento da impressora? Que é a unidade de vídeo? Que é o teclado?

3) Que é um programa? Que são programas básicos? Que são programas de aplicação? Que é *software*? Quais são as três camadas de um sistema de computação? Que é o sistema operacional? Que são programas de serviço? Programa de controle é o mesmo que sistema operacional? Quais os principais tipos de sistemas operacionais? Para que servem os sistemas operacionais? Para que servem os programas de controle? Quais são os programas que constituem o sistema de desenvolvimento? Que são utilitários? Para que servem os programas tradutores? Quais as principais áreas de aplicações? Que são algoritmos? Que é codificar um programa? Que é linguagem de programação? Que é linguagem de máquina? Que é linguagem de baixo nível? Que é linguagem de montagem ou simbólica (*assembly language*)? O que é montador (*assembler*) e para que serve? Que são linguagens de alto nível? Quais são as principais linguagens de alto nível? Que são compiladores e para que servem? O que é programa-fonte? Que é programa-objeto? Que é um interpretador ou intérprete e para que serve? Qual a função do programa ligador? Qual a função do programa carregador? Qual a diferença entre a compilação e a interpretação de um programa? Quais as vantagens e desvantagens de interpretar ou de compilar? Onde podem estar baseados (ou armazenados) o programa de controle e o interpretador BASIC de um microcomputador? Quais os mais populares programas de controle padronizados?

4) Quais os objetivos de quem adquire um microcomputador pessoal? Quais as maiores áreas de aplicação? Quais as alternativas oferecidas pelos programas de educação? É possível usar programas prontos? É possível encontrar lições prontas? É possível preparar lições e testes? Qual a forma de apresentar as lições e testes? O usuário pode desenvolver seus próprios programas? Quais as alternativas de aplicação do microcomputador na música? É possível tocar música diretamente no teclado de um microcomputador? Existem programas que podem ler pautas musicais (codificadas)? É possível compor música com o auxílio de um microcomputador? Pode-se programar um microcomputador para compor música automaticamente? O usuário pode desenvolver seus próprios programas musicais? Qual a mais popular das aplicações dos microcomputadores? Que são programas de jogos? Existem programas de jogos prontos? É possível o próprio usuário desenvolver os seus programas de jogos? Qual a característica da área de aplicação de jogos? Quais as alternativas de aplicação dos microcomputadores na administração pessoal e doméstica? O uso profissional pode ser incluído entre as aplicações pessoais? Em que casos? Quais as aplicações mais típicas de uso pessoal do microcomputador?

5) Existe alguma distinção entre os microcomputadores pessoais? É possível grupá-los segundo algum critério? Quantos grupos é possível formar? Que características têm os microcomputadores do primeiro grupo? Qual a configuração típica do primeiro grupo? Quais as unidades essenciais e quais as opcionais? Que unidades são fornecidas pelo fabricante e que unidades são adaptadas pelo usuário (disponíveis no comércio de eletrodomésticos)? Quais os microcomputadores (marcas) que podem ser classificados no primeiro grupo? Que características têm os microcomputadores do segundo grupo? Quais as configurações típicas do segundo grupo? Quais as unidades essenciais e quais as opcionais? Que unidades são fornecidas pelo fabricante e que unidades são adaptadas pelo usuário? Quais os microcomputadores que podem ser classificados no segundo grupo?

PROBLEMAS:

1) Desenhar o diagrama de um microcomputador cuja arquitetura seja baseada na via única.

2) Desenhar o diagrama da configuração típica de um microcomputador do primeiro grupo, assinalando as unidades essenciais e opcionais e distinguindo as que são fornecidas pelo fabricante e as que são adaptadas pelo próprio usuário (estão disponíveis no comércio de eletrodomésticos).

3) Desenhar o diagrama de uma configuração típica de um microcomputador do segun-

do grupo, na sua versão mais simples. Fazer os mesmos assinalamentos que no problema 2 acima.

4) Desenhar o diagrama de uma configuração típica de um microcomputador do segundo grupo, na sua versão completa. Fazer os mesmos assinalamentos que no problema 2 acima.

5) Fazer um quadro da divisão de um microcomputador, incluindo o equipamento e os programas.

6) Desenhar o diagrama da seqüência a ser obedecida para a execução de um programa que seja compilado.

7) Desenhar o diagrama da seqüência a ser obedecida para a execução de um programa que seja interpretado.

SEGUNDO CAPÍTULO
O TECLADO E SEU USO

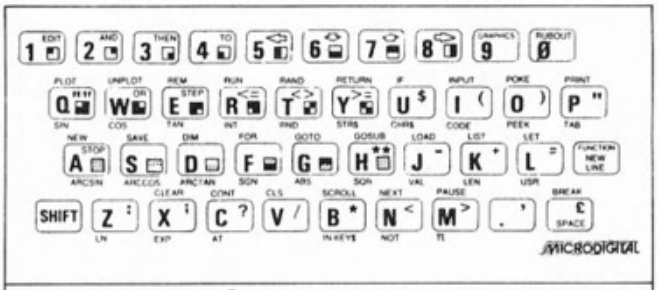
descrição do teclado – caracteres permitidos – funções do teclado – o computador como calculadora – o que é um programa – desenvolvimento e execução de programas – instruções de entrada e saída básicas – formato de dados na saída – edição de um programa

2.1 Descrição do teclado

O periférico com o qual o usuário de um microcomputador mais convive é o teclado. É o seu meio de entrada principal, mesmo que haja outros. Portanto, um conhecimento do teclado é fundamental para a comunicação com o microcomputador.

O teclado dos microcomputadores do primeiro grupo, como SINCLAIR, TK 82-C e NE-Z8000, são teclados planos. Esse tipo de teclado é apresentado na figura 2-1.

FIGURA 2-1: *TECLADO PLANO*



FONTE: MICRODIGITAL ®, manual de operação do microcomputador TK 82-C científico

O teclado plano não possui teclas no sentido estrito do termo, mas apenas as posições correspondentes, marcadas sobre uma placa plástica flexível, que deverão ser tocadas com os dedos e pressionadas. Basicamente, o teclado tem os símbolos numéricos (Ø a 9) e alfabéticos (A a Z) que correspondem às posições do teclado de uma máquina de escrever. O teclado plano tem as posições de toque marcadas em quatro fileiras ou linhas, cada uma com 1Ø posições, ou seja, 4Ø posições de toque (*teclas*). Esses caracteres básicos se encontram colocados na parte inferior esquerda de cada tecla, marcados em negrito. Além deles, há outros símbolos, especiais, gráficos e palavras reservadas, que vêm marcadas dentro da posição de toque, nas partes inferior direita e superior direita. Além desses, existem também palavras reservadas colocadas fora (exteriores) das posições de toque, na parte inferior e posterior. Como utilizar todos esses símbolos é o que se exporá nas próximas seções.

2.2 Caracteres permitidos

Apesar das diferenças existentes entre os teclados dos microcomputadores e também das diferenças entre as implementações da linguagem BASIC, um conjunto de caracteres é comum ao *alfabeto* da linguagem BASIC. Esses caracteres podem ser grupados conforme o quadro 2-1.

QUADRO 2-1: *ALFABETO BASIC*

GRUPO	CARACTERES
dígitos	Ø 1 2 3 4 5 6 7 8 9
letras	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
caracteres especiais	( ) + – / * ↑ = < > # (*cancela*) $ (*cifrão*) ! (*ponto de exclamação*) . (*ponto*) , (*vírgula*) ; (*ponto e vírgula*) " (*aspas*) ' (*apóstrofo*) (*branco*)

Os dígitos e letras estão presentes em todos os alfabetos BASIC, sem alterações. Onde ocorrem modificações é entre os caracteres especiais, cuja presença varia de uma implementação para outra. Por exemplo, o BASIC dos microcomputadores com teclado plano (primeiro grupo dos microcomputadores pessoais) não tem os símbolos # (cancela) e ! (ponto de exclamação) e o símbolo ↑ (potência) é substituído por **.

2.3 Funções do teclado

O mais incomum dos teclados é sem dúvida o teclado plano, onde um número considerável de funções (mais de 18Ø) está distribuído entre apenas 4Ø teclas. Por isso vai merecer atenção, em lugar dos teclados comuns, já bastante conhecidos.

Em geral, as teclas funcionam como nos teclados comuns, quer dizer, a um simples toque obtém-se o símbolo básico, isto é, o símbolo da parte inferior esquerda da tecla. Para se obter o símbolo que figura na parte superior direita da tecla, é preciso calcar a tecla SHIFT, mantendo-a calcada simultaneamente. A tecla SHIFT está na quarta e última fileira, à esquerda, na parte inferior do teclado.

Por exemplo, calcando-se a tecla 1, obtém-se o dígito 1. Todavia, calcando-se SHIFT e, simultaneamente, a mesma tecla 1, obtém-se a palavra-chave **EDIT** (canto superior direito da tecla). O símbolo **EDIT** *não* aparece na tela de vídeo. Outro exemplo, se se calcar a tecla Z, obtém-se o caractere Z. Todavia, se se calcar a tecla SHIFT e, simultaneamente, a tecla Z, obtém-se o caractere : (*dois pontos*). O leitor terá observado que a tecla SHIFT é a única que só tem um único símbolo e que serve para alterar a função de outras teclas.

No alfabeto BASIC o espaço em *branco* é um caractere, como qualquer outro símbolo. O *branco* está colocado na quarta e última fileira, na última tecla à direita, na parte inferior do teclado, onde aparece o símbolo SPACE. Obtém-se um branco com o simples toque da tecla SPACE. Se for calcada a tecla SHIFT, simultaneamente com a tecla SPACE, obtém-se o caractere £ (libra).

Outro símbolo que merece especial atenção é o de NEW LINE, colocado na terceira fileira, à direita. Este símbolo permite a entrada de dados no computador, sejam comandos, linhas de programa ou valores quaisquer. Para obter o símbolo NEW LINE, basta um simples toque desta tecla. Se se deseja o símbolo FUNCTION, a tecla SHIFT deverá ser calcada e, simultaneamente, a tecla NEW LINE. Após o símbolo FUNCTION, obtém-se o símbolo de *uma* função matemática correspondente à tecla que for calcada, entre as que têm nome de função matemática na parte inferior externa da posição de toque, como SIN, COS, TAN etc.

E os símbolos (palavras-chave) que aparecem na parte superior externa de várias posições de toque? Como ter acesso a eles? Basta um simples toque, igual ao que se dá para obter o caractere básico da tecla. Se o toque é o mesmo, como obter um ou outro símbolo? O interpretador BASIC, dependendo da sintaxe da instrução ou comando, selecionará *automaticamente* um símbolo ou outro. Se a sintaxe exige que ocorra uma palavra-chave da linguagem BASIC, o interpretador BASIC *automaticamente* selecionará o símbolo externo e *não* o símbolo básico da tecla. Por exemplo, se se vai iniciar um comando BASIC, o simples toque da tecla P dará o símbolo **PRINT** (e *não* P). Outro exemplo, quando se escreve uma instrução da linguagem BASIC, começa-se por um número de linha. Assim, o toque de teclas numéricas (Ø a 9), quando se começa a escrever a instrução, produz um número. A seguir, o simples toque de uma tecla com símbolo de palavra-chave, como a tecla L, produz o símbolo **LET** (e *não* L), porque em BASIC, após o número da linha, obrigatoriamente tem que vir uma palavra-chave. Essa seleção é feita pelo interpretador BASIC *automaticamente*, como já se disse. Portanto, não há como errar quanto a esse ponto. O usuário, quando emprega esse tipo de microcomputador com teclado plano, *não* precisa e *não* pode escrever as palavras-chave da linguagem BASIC, caractere por caractere. A vantagem evidente é uma escrita mais rápida de comandos e instruções da linguagem BASIC.

Os microcomputadores pessoais de teclado plano permitem escrever na tela com *inversão gráfica*, isto é, onde é *claro* passa a *escuro* e onde é *escuro* passa a *claro*. É também chamado de *vídeo invertido* ou *vídeo inverso*. Para se obter a inversão de vídeo para qualquer seqüência de símbolos basta, antes, entrar em *modo gráfico*, isto é, obter o símbolo GRAPHICS. Para obter o símbolo GRAPHICS, calca-se a tecla SHIFT e, simultaneamente, a tecla 9. A partir daí, todos os símbolos aparecerão em vídeo invertido, até que novamente se obtenha um símbolo GRAPHICS, quando se deixa ou sai do *modo gráfico* e o vídeo voltará ao normal.

Alguns símbolos têm particular importância para a edição de programas, como ⇦, ⇩, ⇧, ⇨ e RUBOUT. Esses símbolos são obtidos calcando-se a tecla SHIFT e, simultaneamente, as respectivas teclas 5, 6, 7, 8 e Ø. Seu uso será explicado posteriormente. Estes símbolos, como **EDIT**, *não* aparecem na tela do vídeo.

O leitor terá observado os *símbolos gráficos*, em pequenos quadradinhos, que aparecem na parte inferior direita de várias teclas. Servem para construir gráficos. Para a obtenção desses símbolos é preciso inicialmente, entrar em *modo gráfico*. O leitor já sabe entrar e sair do *modo gráfico* com o uso do símbolo GRAPHICS (teclas SHIFT e 9). Estando em *modo gráfico*, os símbolos gráficos (dentro dos quadradinhos) são obtidos calcando-se a tecla SHIFT e a tecla que contém o símbolo gráfico escolhido. Deve-se observar que *não* existe o quadrado inteiramente *escuro*. Para obtê-lo, o símbolo que

deve ser tocado é o SPACE, em *modo gráfico*, apenas com a inversão do vídeo.

Outros símbolos são particularmente úteis, como **NEW, CLEAR, RUN** e **CONT**. O símbolo **NEW** corresponde a um comando que inicializa a memória, isto é, apaga os programas e dados, deixando a memória limpa para receber novos programas e dados. O símbolo **CLEAR** limpa apenas os dados, isto é, libera as áreas de memória que contêm valores, permitindo sua nova utilização. O símbolo **RUN** será um dos mais utilizados pelo leitor, pois corresponde ao comando de execução de programas. O símbolo **CONT** permite a execução de um comando de *continuação*, isto é, prossegue-se a execução de um programa que tenha sido interrompida (por exemplo, por uma instrução **STOP**).

Finalmente, cabe chamar a atenção para o símbolo **BREAK** que se encontra na parte externa superior da tecla SPACE. Com esse símbolo é possível interromper a execução de qualquer programa, por meio de um simples toque. O símbolo **BREAK** só é obtido quando um programa está em execução e *não* aparece na tela do vídeo.

Outras funções serão apresentadas no restante do livro, à medida que o leitor for entendendo melhor a linguagem e o uso do microcomputador.

2.4 O computador como calculadora

Eventualmente, é preciso fazer *cálculos simples* para os quais *não* é necessário escrever um *programa*. Deseja-se saber *imediatamente* o resultado da avaliação de uma *seqüência de operações aritméticas*. É possível obter o resultado com o uso do *comando* **PRINT**, que dá saída na tela de vídeo.

Um *comando* difere de uma *instrução*, em BASIC, no sentido de que o *comando* é executado *imediatamente*, logo após ter-se calcado a tecla NEW LINE. Aliás, o leitor já sabe que nada entra no computador sem que se tecle o símbolo NEW LINE. A *instrução* em BASIC só é executada como parte integrante de um *programa*, isto é, com o *comando* **RUN**. O conceito de programa será apresentado na próxima seção.

Alternativamente, representa-se, neste livro, o símbolo NEW LINE por uma seta curva: ↵. Na maioria dos casos, entretanto, nenhuma das duas representações, NEW LINE ou ↵, aparece, porque está *implícita*, já que é obrigatória a presença deste símbolo. Seria cansativo repetir o símbolo em todos os comandos e instruções. Será representado apenas inicialmente, nesta seção, para que o leitor se habitue, ou quando puder haver alguma dúvida.

Usando-se o comando **PRINT**, podem-se executar as operações de adição (+), subtração (–), divisão (/), multiplicação (*) e potenciação (**).

A soma de 2 com 5, por exemplo, pode ser obtida teclando-se

```
PRINT 2 + 5↵
```

,

que produzirá, na tela, a saída do valor

```
7
```

.

A diferença entre os números 3.7639 (o *ponto* corresponde à *vírgula fracionária*) e 5.000387 pode ser obtida teclando-se

```
PRINT 3.7639 – 5.000387 ↵
```

,

que dará saída, na tela, do valor

```
–1.236487
```

.

O quociente entre os valores 52.46 e 1.733 pode ser obtido com o comando

```
PRINT 52.46 / 1.733 ↵
```

,

que produzirá o resultado

```
30.271206
```

.

O produto dos números –1.762 e 4.03 pode ser efetuado pelo comando

```
PRINT –1.762 * 4.03 ↵
```

,

que mostrará, na tela, o resultado

```
–7.10086
```

.

A potência de expoente 5 da base 3.45 é calculada pelo comando

```
PRINT 3.45 ** 5 ↵
```

,

produzindo como resultado o valor

```
488.7598
```

.

O leitor poderá fazer também o cálculo de funções matemáticas, utilizando as funções predefinidas do BASIC, que serão apresentadas posteriormente. Por exemplo, seja calcular a raiz quadrada do número 10.76543, com o comando **PRINT** e a função predefinida SQR, isto é,

```
PRINT SQR (10.76543) ↵
```

,

que produzirá o resultado

```
3.2810715
```

.

É possível, usando o comando **PRINT**, avaliar expressões que contêm diversas operações. Seja, por exemplo, avaliar a expressão numérica

```
2 + 3 / 1.5 * 1.2 – 3 ** 2
```

,

com o uso do comando **PRINT**, ou seja,

```
PRINT 2 + 3 / 1.5 * 1.2 – 3 ** 2 ↵
```

.

Existe uma *prioridade* entre os *operadores*, embora a expressão, como *regra geral*, seja avaliada da *esquerda para a direita*. A prioridade *mais alta* é da potenciação (**), seguida pelas operações de multiplicação e divisão (*, /) e

terminando pela prioridade mais baixa, da soma e da subtração (+, −). Todavia, os *parênteses* têm *a mais alta* prioridade, *maior* quanto *mais internos*, seguindo-se as *funções*. Portanto, o uso de parênteses altera qualquer outra prioridade. O mesmo comando acima pode ser escrito

```
PRINT ((2 + ((3 / 1.5) * 1.2)) − (3 ** 2))↵
```

,

cujo resultado é

```
−4.6
```

.

Já que neste caso particular a ordem de avaliação dos parênteses é a mesma que a determinada pela prioridade dos operadores, este resultado é igual ao da avaliação da expressão *sem* os parênteses. Mudando-se a posição dos parênteses, altera-se o resultado, isto é, será totalmente diferente.

Além do cálculo com o uso de valores numéricos *constantes*, é possível utilizar valores *variáveis*, que tenham um *nome*, como A, B1, C3 etc., constituído por uma *letra* ou uma *letra* e um *dígito*. Pode-se atribuir valor a esta variável com o comando **LET**. Como exemplo,

```
LET X = 2↵
LET Y = 3↵
PRINT X + Y↵
```

,

produzirá o resultado

```
5
```

.

É possível também incluir, no comando **PRINT**, constantes alfanuméricas (contendo letras, dígitos e caracteres especiais entre aspas), chamadas *literais*, que serão impressas exatamente como forem definidas. Um exemplo é a seqüência de comandos

```
LET A1 = 3↵
PRINT "A1=" ; A1↵
```

,

que produzirá a saída

```
A1 = 3
```

Os valores de variáveis definidas através de comandos BASIC permanecem na memória até que sejam alterados por novo comando **LET** ou sejam apagados por um comando como **NEW** ou **CLEAR**.

É possível incluir mais de uma expressão ou constante literal num comando **PRINT**, desde que as diversas expressões ou literais venham separadas por , (*vírgula*) ou ; (*ponto e vírgula*). O uso da , (*vírgula*) causa a impressão com vários espaços entre os valores das expressões e o uso do ; (*ponto e vírgula*) causa a impressão dos valores *sem* espaço entre eles. O *formato de dados na saída* é tratado numa das próximas seções com este mesmo título. A sintaxe do comando **PRINT** é igual à da instrução **PRINT**, que será abordada na seção sobre *instruções de entrada e saída*, uma das próximas. Sobre as regras para avaliação de expressões, deve o leitor se referir à seção intitulada *expressões*. O comando **LET** tem a mesma sintaxe que a instrução **LET**, que é apresentada na seção sob o título *instrução de atribuição*.

2.5 O que é um programa

Um programa na linguagem BASIC é uma seqüência ordenada de instruções da linguagem BASIC. Como exemplo, é apresentado, na figura 2-2, um texto de programa, bastante simples, em linguagem BASIC, utilizando instruções que estão disponíveis na maioria dos microcomputadores.

FIGURA 2-2: *EXEMPLO DE PROGRAMA BASIC*

```
10  INPUT A, B
20  LET C = A * B
30  PRINT "PRODUTO DE "; A; " POR "; B; " = "; C
40  END
```

A partir do exemplo da figura 2-2, a primeira observação é a de que as *linhas* são *numeradas*: 10, 20, 30 e 40. Em um programa, toda linha tem que ser numerada. Admitindo-se apenas uma *instrução* de programa por linha, cada instrução terá um *número de linha*, que a identificará unicamente.

A instrução na linha 10 tem a palavra-chave **INPUT** e serve para fazer a entrada de dados de dois valores numéricos, para as variáveis de nomes A e B.

A instrução na linha 20 é uma instrução de atribuição de valor que faz o cálculo da expressão A * B e atribui o resultado à variável C.

A instrução na linha 30 contém a palavra-chave **PRINT**, isto é, é uma instrução que faz a saída de valores na tela do vídeo. No caso, os valores são literais e variáveis, separados por ; (*ponto e vírgula*), indicando que serão apresentados sem espaço entre eles.

A última linha é a de número 40 e contém a palavra-chave **END**, que indica o *fim físico* do programa, ou seja, que não há mais instruções de programa depois desta. Se por acaso houver alguma instrução depois de **END**, será ignorada (é como se não existisse). Em alguns sistemas BASIC, a colocação da instrução **END** é obrigatória. Todavia, existem aqueles que dispensam o uso de **END** como última instrução. Quando o uso de **END** é opcional, por compatibilidade e para formar hábito, deve-se usar sempre, exceto se a implementação não tiver a instrução **END**, como é o caso dos pequenos microcomputadores.

A instrução **END** também indica *fim lógico* do programa, isto é, o *fim de execução*. Nos sistemas BASIC onde não é obrigatória, a execução termina quando for encontrada a instrução com o maior número de linha.

A *execução* do programa em questão se inicia com a instrução de número 10, prossegue com as instruções de números 20 e 30 e termina com a execução da instrução de número 40. A *execução* de um programa é feita, portanto, segundo a ordem de numeração das linhas, da de menor número para a seguinte de maior número. O *início* de *execução* de um programa é obtido pela execução de um *comando* **RUN**. Iniciada a execução, ela prossegue até que ocorra o *fim lógico* ou *de execução* do programa.

No exemplo apresentado, para que sejam fornecidos os valores 2.5 e 3, respectivamente, às variáveis A e B, quando da execução da instrução **INPUT** de número 10, esta provoca o aparecimento na tela do símbolo ? (*interrogação*). Os valores devem ser fornecidos separados por , (*vírgula*) e calcando-se a tecla NEW LINE. O valor da variável C será calculado na linha 20, pela instrução **LET**, isto é, C terá então o valor 7.5. Quando a execução do programa chegar à linha 30, será impresso

```
PRODUTO DE 2.5 POR 3 = 7.5
```

.

A execução do programa termina na linha 40.

A forma geral de uma instrução da linguagem BASIC é

```
< número de linha > < palavra-chave > [ < parâmetros > ]
```

.

A forma geral de um programa é

```
< instrução > [ < instrução > ] . . .
```

.

O *número de linha* é um valor inteiro positivo, entre 1 (*um*) e 9999 (*nove mil novecentos e noventa e nove*). A *palavra-chave* é uma palavra reservada da linguagem BASIC representativa de uma operação, como **INPUT**, **PRINT**, **LET** etc. Os *parâmetros* são constituídos por argumentos válidos para as operações, eventualmente separados por palavras reservadas da linguagem BASIC, ou outras instruções, de acordo com a sintaxe da instrução.

Os microcomputadores do primeiro grupo *não* possuem a instrução **END**. Além disso, nesses pequenos microcomputadores a instrução **INPUT** só pode fazer a entrada de um único valor, isto é, só admite uma única variável na lista de entrada. Assim, para pequenos microcomputadores, uma versão viável do mesmo programa apresentado na figura 2-2 é a que está na figura 2-3.

FIGURA 2-3: *EXEMPLO DE PROGRAMA BASIC (MODIFICADO PARA PEQUENOS MICROS)*

```
10 INPUT A
20 INPUT B
30 LET C = A * B
40 PRINT "PRODUTO DE "; A; " POR "; B; " = "; C
```

2.6 Desenvolvimento e execução de programas

Para desenvolver e executar programas em um microcomputador, o usuário precisa saber basicamente:

1) como iniciar uma seção;
2) como escrever um programa;
3) como executar um programa; e
4) como terminar a seção.

.

Para **iniciar uma seção** é preciso que o computador esteja alimentado (conectado na corrente elétrica e ligado), bem como os seus periféricos. Se

não estiver, é preciso fazê-lo. Uma vez alimentados, o computador e periféricos estão prontos para operar. Se o microcomputador só dispõe do sistema BASIC, não existe nenhum procedimento especial e a seção pode ser iniciada imediatamente. Se o microcomputador dispõe de sistema operacional (programa de controle) é preciso dar os comandos de inicialização do sistema e pedir o sistema BASIC. Uma vez sob controle do sistema BASIC, a seção pode ser iniciada imediatamente.

O usuário pode, agora, **escrever um programa**, instrução por instrução. Toda instrução é iniciada por um número de linha, para distingui-la de um comando. Os comandos *não* têm número de linha e são executados imediatamente. O número de linha é um valor inteiro positivo entre 1 (*um*) e 9999 (*nove mil novecentos e noventa e nove*).

Para **executar um programa**, os sistemas BASIC, ao que tudo indica, são unânimes em eleger o comando **RUN**. Após a entrada desse comando, é iniciada a execução do programa que se encontrar na memória, até o término, normal ou anormal.

O **término da seção** varia ligeiramente caso exista um sistema operacional ou apenas um sistema BASIC. Se existir um sistema operacional, é preciso retornar do sistema BASIC para o sistema operacional, com o que está encerrada a seção. No caso dos microcomputadores que só possuem o sistema BASIC, *não* existe um procedimento de encerramento de seção. Se não se deseja mais utilizar o microcomputador, este deverá ser desligado e desconectado das tomadas elétricas.

Alguns dos mais comuns comandos BASIC, com possíveis formas alternativas, são descritos no quadro 2-2.

QUADRO 2-2: *COMANDOS BASIC MAIS COMUNS EM MICROCOMPUTADORES PESSOAIS*

COMANDO	FUNÇÃO
BASIC	Entrar em modo BASIC (se existe sistema operacional).
BYE ou **EXIT** ou **SYSTEM**	Sair do modo BASIC (retornar ao sistema operacional, se existir).
RUN	Executar um programa na área de trabalho (memória).
BREAK	Parar a execução de um programa.
CONT	Continuar a execução de programa que foi interrompida por uma instrução **STOP** ou comando **BREAK**.

NEW	Inicializar a área de trabalho (apagar programas e dados).
CLEAR	Liberar área de trabalho ocupada por variáveis (dados).
CLS	Limpar a tela do vídeo.
LIST ou **LLIST**	Fornecer uma listagem ordenada de um programa da área de trabalho (na memória) para o vídeo ou impressora (alternativa: **LLIST**).
LOAD ou **CLOAD**	Carregar na memória um programa a partir de um arquivo do sistema (que tem o nome do programa) em disquete ou fita cassete (alternativa: **CLOAD**).
SAVE ou **CSAVE**	Salvar (armazenar) um programa que se encontra na área de trabalho (na memória) para um arquivo do sistema (que tem o nome do programa), em disquete ou cassete (alternativa: **CSAVE**).
EDIT	Selecionar uma linha para a área (modo) de edição.

2.7 Instruções de entrada e saída básicas

Na seção 2.5 o leitor ficou conhecendo duas instruções para realizar operações de entrada e de saída de dados, respectivamente as de números **10** e **30**. As palavras-chave dessas instruções são, respectivamente, **INPUT** e **PRINT**. A função da instrução de entrada de dados **INPUT** é ler dados a serem fornecidos pelo teclado. A função da instrução de saída de dados **PRINT** é escrever dados na tela da unidade de vídeo. Alternativamente, existe a instrução **LPRINT**, que faz a saída de dados na impressora.

Alguns exemplos da instrução de entrada de dados pelo teclado são apresentados na figura 2-4.

FIGURA 2-4: *EXEMPLOS DE INSTRUÇÃO DE ENTRADA DE DADOS PELO TECLADO*

EXEMPLO	EXPLICAÇÃO
10 INPUT X	A execução da instrução provocará o aparecimento de um ? (*ponto de interrogação*) no vídeo. O usuário deverá fornecer um valor numérico (e NEW LINE). É feita a leitura do valor numérico para a variável numérica X, que passará a ter o valor lido.

35 **INPUT** X, Y, Z	Após o aparecimento do ? (*ponto de interrogação*) na tela, o usuário fornecerá os três valores numéricos solicitados, separados por *vírgula*, (e NEW LINE). Se um valor for fornecido de cada vez (e NEW LINE), novos ?? (*pontos de interrogação*) surgirão na tela, solicitando o seguinte, até se esgotar a lista de variáveis. Os valores serão lidos, na ordem, para as variáveis X, Y e Z, que passarão a ter os valores lidos.
5 **INPUT** "X="; X	Aparecerá na tela a constante literal X=, seguida do ? (*ponto de interrogação*). O restante do processamento será feito normalmente, isto é, fornecido um valor numérico, será o mesmo atribuído à variável X.

A presença de diversas variáveis na lista de entrada, como no segundo exemplo da figura 2-4, não é permitida pela sintaxe da instrução **INPUT** nos pequenos microcomputadores (do primeiro grupo). Uma solução é o desdobramento da instrução em várias, cada uma fazendo a entrada de uma única variável. O segundo exemplo da figura 2-4, neste caso, se desdobra como na figura 2-5.

FIGURA 2-5: *EXEMPLO DE DESDOBRAMENTO DE UMA INSTRUÇÃO* **INPUT**

```
35  INPUT X
36  INPUT Y
37  INPUT Z
```

A inclusão de uma constante literal na primeira posição da lista de variáveis, para aparecer na tela, não é permitida em todas as versões de BASIC, como é o caso dos pequenos microcomputadores pessoais (do primeiro grupo). A solução é o uso de uma instrução **PRINT**, antes da instrução **INPUT**, colocando a constante literal na tela. Um exemplo aparece na figura 2-6.

FIGURA 2-6: *EXEMPLO DE COMBINAÇÃO DE INSTRUÇÕES DE SAÍDA E ENTRADA DE DADOS*

```
10  PRINT "X=";
20  INPUT X
```

A seqüência de instruções do exemplo da figura 2-6 substitui a instrução que é o terceiro exemplo da figura 2-4, nos microcomputadores pessoais do primeiro grupo.

A forma geral da instrução de *entrada de dados pelo teclado* é

INPUT < *lista de variáveis* >

A *lista de variáveis* é uma seqüência de nomes de variáveis, separados por , (*vírgula*).

Uma observação a fazer, com relação ao aparecimento do ? (*ponto de interrogação*) como forma de solicitar o fornecimento de dados durante a execução da instrução **INPUT**, é a de que nos pequenos microcomputadores pessoais isto *não* ocorre. Não se deve esperar o aparecimento de nenhum símbolo especial, nesse caso, a não ser a presença de um *cursor*.

Alguns exemplos da instrução de saída de dados na tela do vídeo são apresentados na figura 2-7.

FIGURA 2-7: *EXEMPLOS DE INSTRUÇÃO DE SAÍDA DE DADOS PELA TELA DO VÍDEO*

EXEMPLO	EXPLICAÇÃO
30 **PRINT** A1 + 3	A expressão A1 + 3 é avaliada e o resultado da avaliação aparece na tela do vídeo.
5 **PRINT** X; X ** 2	Aparecerão na tela do vídeo os valores contidos na variável X e resultado da avaliação da expressão X ** 2, *sem* espaços em *branco* de separação.
10 **PRINT** "X=", X	Aparecerão na tela do vídeo os valores da constante literal X= e da variável X, separados por vários espaços em *branco*.

O leitor deve observar, na figura 2-7, que são usados dois tipos de separadores para as expressões: , (*vírgula*) e ; (*ponto e vírgula*). O uso desses separadores corresponde, respectivamente, à impressão *com* vários espaços em branco entre os valores e *sem* nenhum espaço em *branco*. Esses separadores são símbolos de formatos de espaçamento de dados na saída. Podem ser usados também após a última expressão da lista para determinar a posição em que começará a saída do próximo valor, isto é, do primeiro valor da próxima instrução **PRINT**. Portanto, a posição de saída do primeiro valor de uma instrução **PRINT** depende do símbolo que aparece após a última expressão da instrução **PRINT** anterior. Para iniciar a próxima saída em uma nova linha, es-

ta posição (após a última expressão) deve ficar em branco, o que corresponde a um símbolo NEW LINE.

A forma geral de uma instrução de *saída de dados na tela do vídeo* é

```
PRINT <lista de expressões>
```

A *lista de expressões* é uma seqüência de expressões separadas por um símbolo de formato de espaçamento (, ou ;), podendo conter constantes literais (cadeias de caracteres, entre *aspas*).

2.8 Formato de dados na saída

O leitor já conhece o efeito do uso dos *separadores* ou *delimitadores* , (*vírgula*) e ; (*ponto e vírgula*) no espaçamento dos valores de saída, na instrução **PRINT**. Os diferentes tipos de espaçamento são chamados *formatos* de dados. No caso, o *formato* de dados é *implícito* porque é produzido automaticamente pelo sistema BASIC. Existem outras maneiras de formatar dados explicitamente, os *formatos explícitos* de dados. No momento, o objetivo é tratar somente do *formato implícito* de dados. O formato de dados produzido pela , (*vírgula*) é o *formato aberto* e o formato produzido pelo ; (*ponto e vírgula*) é o *formato fechado*.

A diferença entre os dois formatos implícitos pode ser observada no programa do exemplo que aparece na figura 2-8.

FIGURA 2-8: *EXEMPLOS DE FORMATOS IMPLÍCITOS COM DADOS ALFANUMÉRICOS*

```
10 PRINT "FORMATO" , "ABERTO"
20 PRINT "FORMATO "; "FECHADO"
```

A execução do programa da figura 2-8 produz a saída no vídeo, que aparece na figura 2-9.

FIGURA 2-9: *RESULTADO DA EXECUÇÃO DO PROGRAMA EXEMPLO DA FIGURA 2-8*

```
FORMATO          ABERTO
FORMATO FECHADO
```

O programa da figura 2-8 é um exemplo de saída de dados alfanuméricos, no caso, constantes literais. Todavia, um exemplo que mostra a diferença entre os dois formatos implícitos com dados numéricos aparece na figura 2-10.

FIGURA 2-10: *EXEMPLO DE FORMATOS IMPLÍCITOS COM DADOS NUMÉRICOS*

```
10 PRINT -1,0,1
20 PRINT -1;0;1
```

A execução do programa da figura 2-10 produz a exibição, no vídeo, dos dados, desde que a tela tenha tamanho suficiente, como apresentado na figura 2-11.

FIGURA 2-11: *RESULTADO DA EXECUÇÃO DO PROGRAMA EXEMPLO DA FIGURA 2-10*

```
-1          0          1
-1  0  1
```

Existem diferenças entre a saída de valores numéricos e alfanuméricos que o leitor terá observado pelos exemplos anteriores. Essas diferenças são sumariadas no quadro 2-3.

QUADRO 2-3: *FORMATO DE SAÍDA DE ITENS DE DADOS*

ITEM DE DADO	FORMATO
numérico	O valor numérico é precedido de um espaço que pode ser *branco*, se o valor é positivo, ou levar um sinal – (menos) se o valor é negativo, e é também seguido de um espaço, que é sempre *branco*. A forma geral é *sNb*, onde: *s* = se negativo: – (menos); se positivo: b (*branco*). *N* = representação de um número positivo. *b* = espaço em *branco*.
alfanumérico	O valor fica exatamente como se encontra, isto é, eliminadas as "" (*aspas*) que o delimitam, se for um literal.

Uma observação importante a fazer com relação aos menores micros (primeiro grupo) é que eles *não* obedecem às regras do quadro 2-3, isto é, não há espaços em *branco* precedendo e sucedendo aos valores numéricos.

Nos microcomputadores, a tela e também a linha da impressora são divididas em *zonas* para a realização do *formato aberto*. O número de caracteres por zona varia de 14 a 16 caracteres. Nos menores microcomputadores, cada zona tem 16 caracteres. Como a tela tem 32 colunas de caracteres por 22 linhas, cada linha tem duas zonas. As colunas são numeradas de Ø a 31 e as linhas, de Ø a 21. A divisão da tela de um pequeno microcomputador em zonas é mostrada na figura 2-12.

FIGURA 2-12: *DIVISÃO EM ZONAS DE UMA TELA DE PEQUENO MICROCOMPUTADOR PESSOAL*

COLUNAS	Ø 15	16 31
INTERVALO	← 16 colunas →	← 16 colunas →

A saída em formato aberto coloca cada dado alinhado à esquerda de cada zona. Se o tamanho de um dado ultrapassa o tamanho de uma zona (de uma ou mais zonas), o dado seguinte será alinhado à esquerda da primeira zona que estiver livre.

Um sumário do efeito dos delimitadores sobre a formatação dos dados é apresentado no quadro 2-4.

QUADRO 2-4: *FORMATO IMPLÍCITO DE SAÍDA EM LISTAS DE SAÍDA*

DELIMITADOR	FORMATO DE SAÍDA
, (*vírgula*)	*Aberto*: o *cursor* será movido para a primeira zona disponível na mesma linha ou, se necessário, na linha seguinte, no vídeo (ou impressora).
; (*ponto e vírgula*)	*Fechado*: *não* haverá movimento do *cursor* no vídeo (ou impressora).

A representação de números na tela ou impressora, isto é, na saída, tem suas limitações. Em geral, é fixado um limite máximo, em número de *dígitos significativos* que podem ser representados. Este número varia de sistema

BASIC para sistema BASIC, mas são típicos limites da ordem de 6, 8, 1Ø, 12 e 14 *dígitos significativos*. Nos menores microcomputadores pessoais este limite é de 8 dígitos significativos. O que fazer com valores muito grandes ou muito pequenos? A solução é adotar uma outra forma de representação. A representação dentro dos limites (número de dígitos significativos) é a *forma normal* ou *forma padrão*. A nova forma que permite a representação de valores fora dos limites é a chamada *forma exponencial* ou *notação científica*. A saída de números na forma exponencial ou notação científica também tem a forma geral *sNb*. Todavia, a representação de *N* é da forma *n.n...nEspp* onde *n.nn...n* é um número com ponto decimal (vírgula fracionária da matemática) que representa a base ou mantissa, a letra *E* indica o início do expoente, *s* é a posição do *sinal* (que pode ser omitido, se positivo) do expoente e *pp* é o próprio *expoente*. O número de posições do expoente varia de um sistema BASIC para outro. O expoente pode ter uma posição (*p*), duas (*pp*), três (*ppp*) e até mais. O número de dígitos da mantissa obedece à limitação do número de dígitos significativos. O significado da notação científica pode ser compreendido com um exemplo simples: 1E5 é o mesmo que 1 * 1Ø ** 5. Portanto, o valor correspondente à representação em notação científica é o produto da *mantissa* por uma potência de 1Ø cujo expoente é a *característica*. Outro exemplo: 4.53E-3 é o mesmo que 4.53 * 1Ø ** (−3).

Essas são as regras gerais. Todavia, cada sistema BASIC pode ter suas próprias idiossincrasias. É o caso de pequenos microcomputadores em que, por exemplo, onde o menor número, em valor absoluto, que pode ser representado na forma normal corresponde a Ø.ØØØØ1 (−Ø.ØØØØ1 ou Ø.ØØØØ1). Qualquer número, em valor absoluto, menor do que Ø.ØØØØ1 será convertido para a forma exponencial ou científica. Todavia, com os grandes números, em valor absoluto, o oitavo dígito significativo a partir da esquerda sofrerá arredondamento e os dígitos à sua direita serão substituídos por Øs (*zeros*). Se o valor é fracionário (com ponto fracionário), os Øs (*zeros*) sem significado, da parte fracionária, serão eliminados. Exemplos são apresentados na figura 2-13, mostrando comandos **PRINT** e o resultado da sua execução em um pequeno microcomputador pessoal.

FIGURA 2-13: *EXEMPLOS DE FORMAS DE REPRESENTAÇÃO DE NÚMEROS NA SAÍDA DE PEQUENOS MICROS*

COMANDO	RESULTADO (NO VÍDEO)
PRINT Ø.ØØØØ1	.ØØØØ1
PRINT −Ø.ØØØØ1	−.ØØØØ1
PRINT Ø.ØØØØØ1	1E-6
PRINT −Ø.ØØØØØ1	−1E-6

PRINT 12345678	12345678
PRINT −12345678	−12345678
PRINT 123456789	123456790
PRINT −123456789	−123456790
PRINT 1.23456789	1.2345679
PRINT 12345678.9	12345679

2.9 Edição de um programa

Em geral, toda linha que está sendo escrita se encontra no *modo de edição*, isto é, é possível alterá-la. Além disso, certas características do sistema BASIC permitem a *edição do programa*, na memória (*área de trabalho*).

A principal característica do sistema BASIC que facilita a edição de programas é a de que a execução de um programa é feita sempre na ordem numérica crescente das linhas, qualquer que tenha sido a ordem de entrada. Assim, as linhas podem ser escritas em qualquer ordem. Os resultados práticos permitem:

1) *substituir* qualquer instrução que tenha sido escrita anteriormente, bastando escrever a nova instrução, com o mesmo número de linha que a anterior;
2) *apagar* qualquer instrução que tenha sido escrita anteriormente, bastando apenas escrever o número de linha (*instrução vazia*);
3) *inserir* instruções, em pontos determinados do programa, atribuindo-se às novas instruções números de linha intermediários, no intervalo entre as já existentes;
4) *intercalar* e combinar textos de programas previamente escritos, de acordo com o número de linha das instruções e tendo em vista a ordem de chamada dos textos.

Nos pequenos microcomputadores de teclado plano, o *cursor* da área de edição, que aparece na tela do vídeo, pode ser movimentado para editar em qualquer ponto desta área. Os símbolos utilizados são ⇐ e ⇒, que servem para o deslocamento do *cursor* da linha de edição, respectivamente para a *esquerda* e para a *direita* na horizontal. Esses símbolos são obtidos calcando-se a tecla SHIFT e, simultaneamente, 5 e 8. Cada toque desloca o cursor de uma única posição. Esse *cursor* é normalmente representado por um símbolo □, um quadradinho, em vídeo invertido, contendo uma letra. Em relação ao *cursor* é

possível realizar as operações de *inserir* ou *apagar* caractere. A operação de *apagar* ou *remover* caractere é feita sobre o caractere à *esquerda* do *cursor*. A operação de *inserir* ou *incluir* é feita na *própria* posição do *cursor*, havendo um deslocamento de todos os caracteres à *direita*, para a *direita*. A operação de *remover* um caractere é feita com o uso do símbolo RUBOUT, que é obtido calcando-se a tecla SHIFT e, simultaneamente, a tecla Ø. A operação de *incluir* um caractere é feita com a tecla ou combinação de teclas que geram o próprio símbolo a ser incluído.

Os menores micros *não* possuem uma tecla para apagar uma instrução inteira, na área de edição. Para apagar uma instrução, em modo de edição, é preciso apagar caractere por caractere.

Se a linha que se deseja editar já faz parte do programa que está na memória, isto é, *não* se encontra na *área de edição*, é preciso trazê-la. Para trazer uma linha ao modo de edição usa-se o comando **EDIT**. Nos micros pessoais pequenos (primeiro grupo) esse comando é executado com o símbolo **EDIT**, que é obtido calcando-se a tecla SHIFT e, simultaneamente, a tecla 1. Que linha será selecionada? Aquela que estiver sendo apontada pelo *cursor* de programa, que aparece na tela representado pelo símbolo ⊠ em vídeo inverso, na listagem do programa. Esse *cursor* é movimentado pelos símbolos ⇓ e ⇑, que o deslocam na vertical, respectivamente para *baixo* e para *cima*. Esses símbolos são obtidos calcando-se a tecla SHIFT e, simultaneamente, 6 e 7. Cada toque desloca o cursor de *uma linha*, na listagem do programa. Se se está escrevendo um programa, o *cursor* de programa normalmente está colocado na linha que por último se incluiu no programa, a menos que o usuário o tenha deslocado.

Os símbolos ⇐, ⇑, ⇓, ⇒, RUBOUT e **EDIT** *não* são mostrados na tela do vídeo e executam as operações descritas imediatamente, com um simples toque da combinação de teclas correspondente.

Exercícios:

ESTUDO DIRIGIDO:

1) Qual o principal meio de entrada? Por quê? Que é teclado plano? Que tipos de microcomputadores usam teclados planos? Onde estão os símbolos numéricos e alfabéticos no teclado plano? Onde estão os demais símbolos? Além do teclado plano, que outros tipos de teclado conhece? Com que teclados eles se parecem?

2) A linguagem BASIC permite o uso de todos os caracteres do teclado na formação das instruções? Como se chama o conjunto de caracteres permitidos na linguagem BASIC? Como podem ser agrupados os símbolos do alfabeto BASIC? Que símbolos pertencem a cada grupo? Esses símbolos estão presentes em todas as implementações de BASIC? Quais estão sempre presentes? Quais os que são implementados de maneira variável?

3) Quantas teclas tem o teclado plano? Como são obtidos os símbolos alfabéticos (A a Z)

e os numéricos (Ø a 9)? Para que serve a tecla SHIFT? Como são obtidos os símbolos especiais? Como são obtidas as palavras-chave da linguagem BASIC? Como são obtidas as funções matemáticas? Para que serve a tecla NEW LINE? Como se entra em modo gráfico? Que acontece em modo gráfico? Que é vídeo inverso? Como se sai de modo gráfico? Como se obtêm os símbolos gráficos? Para que servem os símbolos **EDIT**, ←, ↓, ↑, → e RUBOUT? Como são obtidos? Como obter o símbolo espaço em *branco*? Como obter um quadrado preto? Que outros símbolos representam funções importantes? Quais são elas?

4) Qual o comando BASIC que permite usar o micro como calculadora? O comando **PRINT** pode conter expressões matemáticas? As expressões matemáticas podem conter variáveis? Como atribuir valor às variáveis? Qual o efeito do comando **LET**? Como são calculadas as expressões matemáticas? Vários cálculos simultâneos podem ser feitos com o mesmo comando **PRINT**? Como separar as expressões no mesmo comando **PRINT**? Qual o efeito de se usar , (*vírgula*) ou ; (*ponto e vírgula*) como separador?

5) Que é um programa BASIC? Que é uma instrução BASIC? Qual a diferença entre um comando e uma instrução BASIC? A mesma palavra-chave pode ser usada como comando e como instrução? Em todos os casos? Quais as palavras-chave que você conhece? Para que serve a instrução **INPUT**? Para que serve a instrução **LET**? Para que serve a instrução **PRINT**? Para que serve a instrução **END**? Todas as implementações de BASIC possuem a instrução **END**? Pode-se usar **INPUT** como comando? E **LET** e **PRINT**? Quais os limites da numeração das linhas (menor e maior valor)?

6) Como iniciar e terminar uma seção em um microcomputador? Como escrever um programa? Como executar o programa? Quais os principais comandos disponíveis em microcomputadores pessoais? Qual a função de cada comando?

7) Que instruções básicas conhece para entrada e saída? Qual a forma geral de uma instrução **INPUT**? Quais as diferenças de implementação na instrução **INPUT**? Qual a forma geral da instrução **PRINT**? Existem diferenças de implementação significativas da instrução **PRINT**?

8) Que é formatação implícita de dados na saída? Como é feita essa formatação? Que é formatação explícita? Que é formato aberto e como é obtido? Que é formato fechado e como é obtido? Existe diferença entre o formato de dados numéricos e alfanuméricos? Quantos caracteres tem uma zona em um vídeo ou impressora? A divisão em zonas é usada pelo formato aberto ou fechado? Como é feito o alinhamento dos dados dentro de uma zona? Qual o formato determinado pela , (*vírgula*)? Qual o formato determinado pelo ; (*ponto e vírgula*)? Que é forma normal ou padrão? Que é forma exponencial ou notação científica?

9) Que é editar um programa? Como pode ser feita a edição da linha que está sendo escrita? Como podem ser editadas outras linhas (inteiras) no programa que já se encontra na memória? Quais as operações que a numeração das linhas permite? Que é modo de edição? Como colocar uma linha qualquer do programa em modo de edição? Como substituir uma linha de programa? Como apagar uma linha de programa? Como inserir uma nova linha de programa? Como intercalar linhas de programas? Quais os símbolos usados para edição de uma linha? Qual o comando para colocar uma linha em modo de edição? Que é cursor de programa? Quais os símbolos usados para a edição de uma linha na área de edição? Que é cursor de linha? Para que serve o cursor de linha? Como inserir um caractere numa linha? Como remover um caractere de uma linha? Os símbolos de edição são mostrados na tela do vídeo? Quais são os símbolos usados para edição e qual o seu efeito?

PROBLEMAS:

1) Alimentar o microcomputador. Iniciar uma seção. Sem fazer nenhum trabalho durante a seção (seção vazia), encerrar a seção. Desalimentar o microcomputador. Repetir essas operações para se certificar que aprendeu. Deixar a seção aberta para realizar os exercícios que se seguem.

2) É fácil usar o terminal como calculadora eletrônica. Podem ser feitos vários cálculos simultaneamente, com o comando:

```
PRINT < lista de expressões >
```

Experimente:

```
PRINT 427 – 341                    (Resposta:  86)
PRINT 532.67 – 732.3               (Resposta:  –199.63)
PRINT 3Ø – 2.54 * 7.95/1.43 + 7    (Resposta:  22.879Ø21)
PRINT 5 + 3.5, 5 ** 2              (Resposta:  8.5 e 25)
```

Escreva suas próprias expressões e use o comando **PRINT** com uma, duas ou várias expressões simultaneamente. Experimente os separadores , (*vírgula*) e ; (*ponto e vírgula*). Compare o efeito dos separadores.

3) Escreva o programa da figura 2-3. Mande executar. Forneça valores de A e de B. Observe o resultado. Mande executar várias vezes, com diferentes valores. Verifique os resultados. Observe a forma que os dados têm na entrada e na saída.

4) Edite o programa do exercício anterior, para fazê-lo exibir mensagens A=? e B=?, solicitando a entrada dos valores de A e de B.

5) Inicialize a memória com o comando **NEW**. Escreva e execute o programa:

```
1Ø PRINT "A", "E", "I",
2Ø PRINT "O", "U"
3Ø PRINT "A"; "E"; "I";
4Ø PRINT "O"; "U"
```

Observe a diferença de formatos de saída com dados alfanuméricos.

6) Inicialize a memória. Escreva e execute o programa:

```
1Ø PRINT –2, –1, Ø,
2Ø PRINT 1, 2
```

Escreva (inclua) novas instruções (por exemplo: 3Ø e 4Ø) substituindo nas acima as „ (*vírgulas*) por ;; (*pontos e vírgulas*), e reexecute.

7) Inicialize a memória. Escreva e execute o programa:

```
10 PRINT "-2", "-1", "0", "1", "2"
20 PRINT -2, -1, 0, 1, 2
```

Escreva (inclua) novas instruções (por exemplo: 30 e 40), substituindo nas acima as vírgulas por pontos e vírgulas. Reexecute. Observe as diferenças.

8) Inicialize a memória. Reescreva todo o programa do exercício anterior, inclusive as linhas que foram acrescentadas, fazendo a exibição de um único dado por linha. Observe as diferenças na saída.

9) Escreva as instruções de programa:

```
30 PRINT "TCHAU"
10 PRINT "ALO AMIGO"
20 PRINT "TUDO BEM?"
```

Observe a ordem em que entraram. Execute o programa. Mande listar com o comando **LIST**. Alguma diferença? Mande reexecutar. Alguma diferença nos resultados? Modifique o programa para enviar a mensagem AQUELE ABRACO, entre as linhas 20 e 30. Reexecutar. Listar. Reexecutar. Elimine a mensagem TCHAU. Reexecute. Renumere as linhas. Reexecute.

10) Reescreva o programa anterior, em apenas duas linhas de programa. Execute. Faça as mesmas modificações. Reexecute a cada modificação.

11) Escreva um programa que faça a leitura de três valores a partir do teclado e imprima sua média aritmética. Inclua mensagens para a entrada e a saída.

12) Experimente:

```
10 PRINT "A"; "B"; "C"; "D"; "E"
20 PRINT "A"; "B";; "C";;; "D";;;; "E"
30 PRINT "A", "B", "C", "D", "E"
40 PRINT "A", "B",, "C",,, "D",,,, "E"
```

Repita a experiência com números. Combine *vírgulas* e *pontos e vírgulas* na mesma instrução **PRINT**.

TERCEIRO CAPÍTULO
ELEMENTOS DA LINGUAGEM

elementos da linguagem BASIC – constantes: numéricas e alfanuméricas – variáveis simples – comentários: instrução REM

3.1 Elementos da linguagem BASIC

Um *texto* de programa BASIC é constituído por *linhas numeradas*. Cada *linha* contém *uma instrução* da *linguagem* BASIC. As *instruções* da linguagem BASIC são compostas de *elementos* da linguagem e de outros *símbolos* que servem de *separadores* entre os *elementos*, como o espaço em *branco*, os *parênteses*, a *vírgula* (,) e a *igualdade* (=).

Em um programa, um valor que *não* se altera durante a execução do programa é chamado de *constante* e é representado pelo *próprio* valor, como por exemplo

```
534    0.00389    "NAO MUDAR DE VALOR"    "S+-A9↑3B"
```

.

Outro tipo de valor, que se altera durante a execução do programa, é chamado genericamente de *variável* e é representado por um *nome* ou *identificador*, como por exemplo

```
X        A7$        W3(2)        B$(J+3)
```

.

Esses dois tipos de *elementos* são as duas grandes divisões dos elementos da linguagem BASIC, com suas principais características, como apresentado no quadro 3-1.

QUADRO 3-1: *ELEMENTOS DA LINGUAGEM BASIC*

elemento da linguagem BASIC	*constante*:	*não* muda de valor durante a execução do programa e se representa pelo *próprio* valor
	variável:	muda de valor durante a execução do programa e se representa por um *nome* ou *identificador*

Constante que representa valor *numérico* é chamada de *constante numérica*, como por exemplo

```
534    Ø.ØØ389
```

Não são considerados *constantes* os *números de linhas*, pois *não* são elementos de composição das instruções, tendo uma função diferente, isto é, permitem *referenciar* uma determinada *linha*.

Constante numérica que *não* possui *ponto fracionário* (.), que é correspondente à *vírgula fracionária* (,) da matemática, é *constante numérica inteira* ou, simplesmente, *constante inteira*, como por exemplo

```
534
```

Se a *constante numérica* incluir *um* e *único ponto fracionário* (.), é uma *constante numérica fracionária de ponto fixo* ou *constante de ponto fixo*, ou simplesmente *constante fracionária*, como por exemplo

```
Ø.ØØ389
```

A *constante* que *não* representa valor *numérico* e sim um valor *alfanumérico* qualquer, colocado entre *aspas* ("), é a *constante alfanumérica* ou *constante literal* ou, simplesmente, *literal*, como por exemplo as duas *constantes*

```
"NAO MUDAR DE VALOR"    "S+-A9↑3B"
```


O *nome de variável* que representa *um único* valor, isto é, ao qual só é possível atribuir *um único* valor em cada instante, é chamado de *variável simples* ou, simplesmente, de *variável* e, embora *não* necessariamente representem este tipo de *variável*, podem ser considerados como tal os exemplos

```
X    A7$
```

Além disso, o *nome de variável* pode representar também *coleção* de valores, isto é, sob *um mesmo nome* existe um *conjunto* de valores e *não* apenas *um único*, sendo chamado *nome de conjunto* ou, simplesmente, *conjunto*.

Um problema que se apresenta é o de fazer *referência* a um valor que seja um *elemento* do *conjunto*, pois todos os *elementos* têm o *mesmo nome*. Como os *conjuntos* são *ordenados*, cada *elemento* possui um *número de ordem* ou *posição relativa* que é representado por um *índice* ou *subscrito*. Para *referenciar* um determinado *elemento* de um *conjunto* sob um determinado *nome* basta que o *nome do conjunto* seja acompanhado do *índice* ou *subscrito entre parênteses*, de que são exemplos

```
W3 (2)    B$ (J + 3)
```

Quando se utiliza o *nome* de um *conjunto sem* o *subscrito*, a *referência* é ao *conjunto* como um *todo* e *não* a qualquer elemento em particular.

Deve-se observar ainda que *nomes de variáveis* ou *conjuntos* como em

```
X    W3 (2)
```

constituídos por uma *letra* ou uma *letra* e um *dígito*, são *nomes* de *variáveis* e *conjuntos numéricos*, enquanto *nomes de variáveis* ou *conjuntos* como em

```
A7$    B$ (J + 3)
```

que *terminam* com um *cifrão* (**$**), são *nomes de variáveis* ou *conjuntos alfanuméricos*.

3.2 Constantes: numéricas e alfanuméricas

Da seção 3.1, o leitor já tem conhecimento sumário do que sejam *constantes*, elementos que se mantêm com o valor inalterado durante a execução do programa, bem como sua divisão, como apresentado no quadro 3-2.

QUADRO 3-2: *CONSTANTES*

constante	numérica	inteira
		fracionária
	alfanumérica ou literal	

Constante numerica é a representação de qualquer valor *numérico*, isto é, *número*, figurando em um texto de programa, excluídos os números de linhas.

Constante inteira é qualquer número *sem* ponto fracionário, isto é, representando valor numérico *inteiro*, como por exemplo

```
-5      +5      5      32767      Ø
```

.

Observando os números apresentados verifica-se que alguns têm *sinal* e outros *não* têm. Naqueles que vêm precedidos de *sinal*, este *sinal* faz parte integrante da constante. No caso de valor *inteiro negativo*, a presença de um *sinal menos* (–) precedendo o número é *obrigatória*. Quando se tratar de valor *inteiro positivo*, a presença do *sinal mais* (+) precedendo o número é *opcional*, isto é, tanto pode estar presente como estar omitido. *Não* se deve confundir o *sinal* de uma *constante inteira* (*operador* aritmético *unário*) com um dos *operadores* aritméticos (*binários*) de *adição* (+) e de *subtração* (–).

Assim, são *totalmente diferentes* os valores *numéricos* de *constantes numéricas inteiras* como

```
-5      +5
```

.

Porém, são *absolutamente idênticos* os valores representados por *constantes numéricas inteiras* como


```
+5      5
```

.

Deve-se evitar confundir o uso do *ponto fracionário* da linguagem BASIC com o uso que normalmente é feito na matemática, isto é, para *separar classes* de algarismos. É *proibido* o uso de qualquer caractere para separar *classes* de algarismos em *constantes numéricas*. Por exemplo, o valor que normalmente na matemática é representado como *32.767*, na linguagem BASIC aparece representado pela *constante*

```
32767
```

,

isto é, *sem ponto* (.) ou qualquer outro símbolo *separador* de *classes*.

Constante fracionária é qualquer *constante numérica* que inclua um *ponto fracionário*, isto é, represente um *número fracionário*, como por exemplo

```
-3.1416    +3.1416    3.1416    -Ø.ØØØØ125    367.421 Ø.Ø
```

.

Da mesma forma que com os *números inteiros*, no caso dos *fracionários* se observa que alguns vêm precedidos de *sinal* enquanto outros não vêm. O *sinal* é parte integrante da *constante* quando está presente em uma *constante*. No caso de valor *fracionário negativo*, a presença de um *sinal menos* (–) precedendo o número é *obrigatória*. Quando se tratar de valor *fracionário positivo*, a presença do *sinal mais* (+) precedendo o número é *opcional*, isto é, pode estar ou não presente. Da mesma forma que com os *inteiros*, *não* deve ser confundido o *sinal* de uma *constante fracionária* (*operador* aritmético *unário*) com um dos *operadores* aritméticos (*binários*) de *adição* (+) ou de *subtração* (–).

Portanto, são *totalmente diferentes* os valores *numéricos* de *constantes numéricas fracionárias* como

```
-3.1416    +3.1416
```

.

Mas são *absolutamente idênticos* os valores representados por *constantes numéricas inteiras* como

```
+3.1416      3.1416
```

Não se pode usar *vírgula fracionária* (,) da matemática em lugar do *ponto fracionário* (.), uma vez que é este último o símbolo correspondente na linguagem BASIC. Por exemplo, o valor que normalmente na matemática é representado como $-0,0000125$, na linguagem BASIC aparece representado pela *constante*

```
-0.0000125
```

Algumas implementações da linguagem BASIC permitem representar valores *numéricos fracionários* na *forma* mais conhecida como *notação científica*, ou seja, *exponencial* ou de *ponto flutuante*. Qualquer número, respeitados os limites da implementação, pode ser representado nesta *forma*. Todavia, é utilizada para valores *muito grandes* ou *muito pequenos*, caso em que se torna mais confortável o seu uso do que a *notação fracionária de ponto fixo*, já apresentada. São exemplos desta notação

```
-31416E-4  +31416E-4  31416E-4  -1.25E-5  3.67421E+2
```

Estes valores, em *notação de ponto fixo*, são representados por

```
-3.1416    +3.1416    3.1416    -0.0000125    367.421
```

O *nome da constante* reflete o que ocorre, isto é, o *ponto fracionário* (.) *flutua, desliza, se desloca*, dependendo da vontade do usuário na escolha do *expoente*. O *expoente* é o número (com *sinal*) que aparece após cada *letra*

```
E      E      E      E      E
```

ou seja, neste caso, os *números*

```
-4     -4     -4     -5     +2
```


Cada *expoente* representa o valor do expoente de uma *potência* de *base 10* que multiplica o número à esquerda, chamado *mantissa*.

Os valores representados como constantes correspondem, respectivamente, aos produtos (*notação da aritmética*)

```
-31416 x 10^-4  +31416 x 10^-4  31416 x 10^-4  -1.25x10^-5  3.6742x10^2
```

São válidas para as *constantes numéricas fracionárias de ponto flutuante*, ou em *notação científica*, todas as observações já feitas quanto a *valor* e *sinal* para as *constantes numéricas fracionárias de ponto fixo*.

Uma *constante alfanumérica* ou *literal* é qualquer seqüência *arbitrária* de caracteres do alfabeto BASIC, entre *aspas* ("), como por exemplo as duas *constantes*

```
"CONSTANTE"          "BOM DIA!!!"
```

Pode-se agora reformular o quadro 3-2 para considerar também as *constantes de ponto flutuante*, resultando o quadro 3-3.

QUADRO 3-3: *DIVISÃO DAS CONSTANTES*

constante	numérica	inteira	
		fracionária	de ponto fixo
			de ponto flutuante (notação científica)
	alfanumérica ou literal		

Exemplo de um programa que realiza diversas operações aritméticas com *constantes* e imprime o resultado dessas operações é o da figura 3-1.

FIGURA 3-1: *EXEMPLO DE PROGRAMA*

```
10 PRINT  1 + 1 - 1, 1 + 10 - 10
20 PRINT  1.0 + 1.0 - 1.0, 1.0 + 10.0 - 10.0
30 PRINT  1 * 1 / 1 , 1 / 1 * 1
```

```
40 PRINT  1.Ø * 1.Ø / 1.Ø, 1.Ø / 1.Ø * 1.Ø
50 PRINT  1E38, 1.Ø * 1Ø↑(38)
60 PRINT  1E-38, 1.Ø * 1Ø↑ (-38)
70 PRINT  1E-39, 1.Ø * 1Ø↑ (-39)
80 PRINT  1E+38 * 1E-38 , 1E+38 / 1E+38
```

3.3 Variáveis (simples)

Quando é necessário manipular um valor que se altera durante a execução do programa, a solução é atribuir a este valor um *nome* ou *identificador*.

Tomando-se para exemplo uma instrução de impressão como

```
PRINT A * B↑2
```

,

encontram-se nela duas *variáveis*, ou seja, os *elementos*

```
A   B
```

.

Ao ser executada esta instrução, presume-se que tenha sido atribuído às *variáveis* A e B algum valor. É possível, portanto, a cada execução da instrução, utilizar valores totalmente diferentes dos anteriores. Aí reside precisamente a grande flexibilidade e economia do uso de *variáveis*. Com uma única instrução, atribuindo previamente valores às *variáveis*, podem-se executar *cálculos* sobre um número praticamente *ilimitado* de valores, de acordo com uma *fórmula* qualquer, como é o caso de

```
A * B↑2
```

.

Da seção 3.1 o leitor já adquiriu também uma noção elementar do que é uma *variável* e como ela se distingue de uma *constante*, quanto à sua função. Ainda com relação a *variáveis*, pode distinguir entre uma *variável simples* ou um *conjunto*. Além disso, sabe que tanto uma *variável* como um *conjunto* podem ser *numéricos* ou *alfanuméricos*. A divisão geral de *variáveis* é apresentada no quadro 3-4.

QUADRO 3-4: *VARIÁVEIS*

variável	simples	numérica
		alfanumérica
	conjunto	numérico
		alfanumérico

Uma *variável simples* é referenciada simplesmente pelo uso do seu *nome*.

A forma geral de uma *referência à variável simples*, isto é, de uma *variável não-subscrita*, é:

```
< nome de variável >
```

.

Da mesma forma que as *constantes numéricas*, as *variáveis numéricas* podem representar valores *numéricos* de duas naturezas: *inteiros* ou *fracionários*.

Uma *variável numérica inteira* representa somente valores *inteiros*, isto é, a ela só se podem atribuir valores *numéricos inteiros*.

Uma *variável numérica fracionária* representa valores *fracionários*, isto é, podem-se-lhe atribuir valores *numéricos fracionários*.

Se o leitor *não* declarar a *variável* como *inteira*, na falta da declaração, em geral, o sistema BASIC adotará para ela o tipo *fracionário*. Isto *não* quer dizer que o tipo *fracionário não* possa representar valores *inteiros*. Pelo contrário, o tipo *fracionário* é mais abrangente que o tipo *inteiro*. Enquanto o tipo *inteiro* só representa valores *inteiros*, o tipo *fracionário* representa valores *inteiros* e *fracionários*. Do ponto de vista de economia, *variáveis inteiras* gastam *menos* memória de computador do que as *variáveis fracionárias*. Todavia, este *não* é, em geral, um problema relevante e portanto o leitor não precisa se preocupar com a declaração do tipo. Na verdade, muitos sistemas BASIC *não* fazem nenhuma distinção entre os dois tipos de *variáveis*, isto é, a *variável numérica* aceita *qualquer* dos tipos de valor, *inteiro* ou *fracionário*.

Um exemplo de um programa que mostra o uso de *variáveis numéricas* para representar, *indiferentemente*, valores *inteiros* ou *fracionários* é apresentado na figura 3-2.

FIGURA 3-2: *PROGRAMA EXEMPLO DE USO DE VARIÁVEIS NUMÉRICAS*

```
10  LET    X = 3
20  PRINT  "X=" ; X
30  LET    X = 3.1416
40  PRINT  "X=" ; X
50  END
```

No programa da figura 3-2, inicialmente é atribuído à *variável* X um valor *numérico inteiro*, 3, que será impresso pela instrução seguinte. Posteriormente à *mesma variável* X é atribuído um valor *numérico fracionário*, 3.1416, o qual é impresso pela próxima instrução, após o que o programa termina, com a instrução seguinte.

O aspecto da *saída* impressa pelo programa da figura 3-2 é

```
X = 3
X = 3.1416
```

Exemplos de *nomes* de *variáveis numéricas* são

```
A     I9     B3     O1     PØ     Z
```

Observando-se a formação de *nomes* de *variáveis numéricas* podem-se deduzir as *regras* de sua formação.

As *regras* para a formação de *nomes* de *variáveis numéricas simples* são as constantes do quadro 3-5.

QUADRO 3-5: *REGRAS PARA A FORMAÇÃO DE NOMES DE VARIÁVEIS NUMÉRICAS SIMPLES*

1. O primeiro caractere tem que ser, obrigatoriamente, uma *letra* (maiúscula: A a Z).
2. Opcionalmente, o primeiro caractere pode ser seguido de um *dígito decimal* (Ø a 9).
3. Caso o nome tenha dois caracteres, não pode haver espaço em *branco* entre eles.

OBS.: O *nome* deve ser *único*, isto é, *não* pode haver, no mesmo programa, outro *nome* de *variável numérica simples* igual.

Considerando-se que são *26* as *letras* do *alfabeto* e *1Ø* os *dígitos decimais*, existe a possibilidade de *26Ø* combinações na formação de *nomes*. Todavia, apesar disso, muitos sistemas têm limitações de memória e só permitem criar um número limitado de *variáveis*, o qual, em alguns casos, está em torno de *9Ø nomes*, incluindo os *nomes de conjuntos*. Esse limite todavia é mais que suficiente para atender à maioria dos casos e, praticamente, *não* é preciso ter qualquer preocupação com a possibilidade de ultrapassá-lo, que é muito remota.

Para utilizar *variáveis numéricas simples* em um programa, em geral *não* é preciso *declará-las*. Todavia, em muitas implementações, para maior clareza do programa, é *opcional* a *declaração*, através da *instrução* **DIM**, que normalmente se destina à declaração de *variáveis alfanuméricas* e de *conjuntos*.

Em geral, *não* é preciso que *variáveis numéricas simples* estejam *definidas*, isto é, um valor lhes tenha sido atribuído, para que possam ser usadas. Isto porque, sempre que o sistema BASIC *inicia* a execução de um programa, *automaticamente* atribui a *todas* as *variáveis numéricas simples* o valor *zero* (Ø). Portanto, somente para *variáveis numéricas* que representem valores *diferentes* de *zero* (Ø) é necessário atribuir valores, através de *instruções* como **LET**, **READ** e **INPUT**.

Pelo visto, é mais importante distinguir entre *variáveis numéricas* e *alfanuméricas* do que entre as *numéricas inteiras* e *numéricas fracionárias*.

Da mesma forma que as *constantes alfanuméricas* representam valores *alfanuméricos*, também as *variáveis alfanuméricas* podem fazê-lo. A diferença é que a *constante não* muda de valor, isto é, permanece a mesma durante toda a execução do programa, enquanto a *variável* pode assumir diversos valores *alfanuméricos* durante a execução do programa.

Um exemplo de programa que mostra o uso de *variáveis alfanuméricas* para representar valores *alfanuméricos* é apresentado na figura 3-3.

FIGURA 3-3: *PROGRAMA EXEMPLO DE USO DE VARIÁVEIS ALFANUMÉRICAS*

```
10  LET    X$ = "VARIAVEL"
20  PRINT  X$ ;
30  LET    X$ = " ALFA"
40  PRINT  X$ ;
50  LET    X$ = "NUMERICA"
60  PRINT  X$
70  END
```

No programa da figura 3-3 inicialmente é atribuído à *variável* X$ o valor *alfanumérico* VARIAVEL, que será impresso pela instrução seguinte. A

seguir, à mesma *variável* X$ é atribuído o valor *alfanumérico* ALFA (precedido de um espaço em *branco*), que é impresso na *mesma linha*. Novamente é atribuído, à mesma *variável* X$, o valor *alfanumérico* NUMERICA, que também é impresso na *mesma linha*.

O aspecto da *saída impressa* pelo programa da figura 3-3 é

```
VARIAVEL ALFANUMERICA
```

.

Exemplos de *nomes* de *variáveis alfanuméricas* são

```
A$      I9$      B3$      O1$      PØ$      Z$
```

.

Observando-se a formação de *nomes* de *variáveis alfanuméricas*, podem-se deduzir as *regras* para sua formação.

As *regras* para a formação de *nomes* de *variáveis alfanuméricas* são as constantes do quadro 3-6.

QUADRO 3-6: *REGRAS PARA A FORMAÇÃO DE NOMES DE VARIÁVEIS ALFANUMÉRICAS*

1. *Iniciar* a formação do nome segundo as mesmas regras usadas para as variáveis numéricas simples.
2. *Terminar* o nome por um caractere *cifrão* ($), obrigatoriamente.
3. Entre o caractere *cifrão* ($) e o(s) anterior(es) *não* pode haver espaço em *branco*.

OBS.: O *nome* deve ser *único*, isto é, *não* pode haver no mesmo programa outro nome de *variável alfanumérica* igual.

Todos os sistemas BASIC têm um *tamanho máximo padrão* para *variáveis alfanuméricas* que *não* tenham sido previamente *declaradas* em uma *instrução* **DIM**, na falta de uma especificação do tamanho. É muito variável o *tamanho máximo padrão*, sendo o *menor* tamanho de *1Ø* caracteres. Todavia, há sistemas BASIC com *tamanho máximo padrão* de *variável alfanumérica* da ordem de *18*, *4Ø* e até mesmo *6Ø* caracteres. Por outro lado há aqueles que *exigem* uma *declaração* prévia das *variáveis alfanuméricas* através da *instrução* **DIM**. Para se ter a garantia de que o programa será aceito na grande maioria dos sistemas BASIC, o recomendável é *fazer a declaração*. Por exemplo, o programa da figura 3-3 pode ser alterado e ficar com o aspecto apresentado na figura 3-4.

FIGURA 3-4: *PROGRAMA EXEMPLO (MODIFICADO) DE USO DE VARIÁVEIS ALFANUMÉRICAS*

```
1Ø  DIM    X$ (4Ø)
2Ø  LET    X$ = "VARIAVEL ALFANUMERICA"
3Ø  PRINT  X$
4Ø  END
```

No programa da figura 3-4, inicialmente é definida uma *variável alfanumérica* X$ com *tamanho máximo* de 4Ø caracteres. A seguir é atribuído à *variável* X$, de uma única vez, o valor alfanumérico VARIAVEL ALFANUMERICA, o qual é impresso pela instrução seguinte. O aspecto da *saída impressa* é exatamente o *mesmo* que o do programa da figura 3-3.

Variáveis numéricas simples, *conjuntos* e *variáveis alfanuméricas simples* que se *iniciam* pelos *mesmos* caracteres são variáveis *totalmente diferentes* e *não* serão confundidas pelo compilador. Assim, são *totalmente diferentes* as *variáveis* A1 e A1$, B e B$, e assim por diante.

O *número máximo de caracteres* (*tamanho*) que uma *variável alfanumérica* pode ter é extremamente variável de sistema para sistema, variando este limite desde *255* até *32.767* caracteres.

3.4 Comentários: instrução REM

A documentação de programas tem um valioso auxílio na *instrução* **REM**, que é uma abreviatura de **REMARKS**, ou seja, *observações*. É uma instrução *não* executável. Os *comentários* que se deseja sejam feitos em um programa, para maior clareza, são feitos com o uso dessa *instrução*.

Os *comentários*, *narrativas*, *observações* ou *mensagens* que o usuário quer *incluir* na listagem de programa devem ser colocados logo *após* a *palavra-chave* **REM**. Exemplo do uso da *instrução de comentários* é apresentado no programa que consta da figura 3-5, que é o mesmo programa da figura 3-4, porém *comentado*.

FIGURA 3-5: *PROGRAMA EXEMPLO COM COMENTÁRIOS*

```
1Ø  REM  ********** PROGRAMA EXEMPLO **********
2Ø  REM  ********** DE USO DE COMENTARIOS ********
3Ø  REM  DIMENSIONA VARIAVEL X$ COM 4Ø CARACTERES
```

```
40  DIM    X$ (40)
50  REM    ATRIBUI VALOR ALFANUMERICO A VARIAVEL
60  LET    X$ = "VARIAVEL ALFANUMERICA"
70  REM    IMPRIME VALOR DA VARIAVEL ALFANUMERICA
80  PRINT  X$
90  REM    TERMINO DE EXECUCAO DO PROGRAMA
100 END
```

Deve-se observar que a *instrução* **REM** é colocada em *qualquer* parte do programa. É bastante compreensível que os *comentários* gastem *área de memória*, o equivalente ao número de caracteres mais uma constante (em torno de 5). Há também um gasto em *tempo de execução*, para interpretar a instrução, embora nenhuma ação resulte disso. Em virtude desse *custo*, em *memória* e em *tempo de execução*, os comentários devem ser utilizados de maneira econômica, isto é, o mínimo indispensável para tornar bem claro o texto do programa.

Pode-se aumentar a visibilidade do comentário dentro do texto do programa colocando-se uma *instrução* **REM** *vazia* (*sem* comentários, isto é, seguida de espaços em *branco*) antes e outra depois da *instrução* **REM** em que se encontra um comentário, como mostrado na figura 3-6.

FIGURA 3-6: *ESPAÇAMENTO DE UMA INSTRUÇÃO* **REM**

```
--  ----------
60  REM
70  REM   COMENTARIO COM ESPACAMENTO
80  REM
--  ----------
```

A forma geral de uma *instrução de comentários ou observações* é dada pela expressão:

REM [< *caractere* >] . . .

Uma segunda maneira de incluir *comentários* nos programas, que está se generalizando, é o uso de um *sinal de comentário*, na mesma linha e após uma instrução qualquer. A dificuldade é poder fazer uso de sinal que seja universalmente aceito. O mais utilizado é o *ponto de exclamação* (!), se bem que também se encontrem o *apóstrofo* (') e a combinação *barra-asterisco* (/*). No caso de *apóstrofo* ('), *não* pode ser usado na mesma linha de uma *instrução* do tipo **IMAGE** (*formato de dados* ou *imagem*). Nem todas as implementações da linguagem permitem esta segunda forma de comentários, pelo que deve ser *evitada* se se deseja ter uma maior portabilidade.

Um exemplo de um texto de programa utilizando esta segunda forma de *comentários* é apresentado na figura 3-7, que é o *mesmo* programa da figura 3-4, *comentado*.

FIGURA 3-7: *PROGRAMA EXEMPLO DO USO DO SINAL DE COMENTÁRIO*

```
10  DIM    X$ (40) ! DIMENSIONA VARIAVEL X$
20  LET    X$ = "VARIAVEL ALFANUMERICA" ! ATRIBUI VALOR A X$
30  PRINT  X$ ! IMPRIME VALOR DE X$
40  END    ! TERMINA EXECUCAO DO PROGRAMA
```

A forma geral do *comentário* com o uso de *sinal de comentário* é a da expressão:

! [< *caractere* >] . . .

Exercícios:

ESTUDO DIRIGIDO:

1) Quais são os principais elementos da linguagem BASIC? Que é uma constante? Que é uma variável?

2) Como se representa uma constante? Quais são os tipos básicos de constantes? Como se representam valores numéricos constantes? Como se representam valores numéricos inteiros constantes? Como se representam valores numéricos fracionários constantes? Que é ponto fixo? O que é ponto flutuante? Que é notação científica? Que é uma constante alfanumérica? Que caracteres pode conter uma constante alfanumérica? Que é constante literal?

3) Como se representa uma variável? Que é variável simples? Quantos nomes pode ter uma variável? Para que serve o nome da variável? Um nome de variável pode representar mais de um valor? Como se chama a variável que representa vários valores do mesmo tipo? Quais os tipos de valores que podem ser representados por uma variável? Como se chamam as variáveis que representam valores numéricos? E alfanuméricos? Como fazer uma referência a uma variável? Variáveis podem representar valores numéricos? E alfanuméricos? Como se chamam essas variáveis? Que tipos de valores são representados por variáveis inteiras? E por variáveis fracionárias? Como são formados os nomes de variáveis numéricas? Como são formados os nomes de variáveis alfanuméricas? Que é declaração

de variável? Para que serve? Qual a instrução para declaração de variáveis? É preciso declarar todas as variáveis? E as alfanuméricas?

4) Para que servem os comentários em um programa? Os comentários são feitos com que instrução? Essa instrução é executável? Qual a forma geral da instrução de comentários? A instrução **REM** consome tempo de interpretação? Deve ser usada intensivamente ou com parcimônia? Por quê? Alguns sistemas BASIC permitem outra forma, além de **REM**, para comentários? Quais são elas?

PROBLEMAS:

---

1) Escreva o programa:

```
10 PRINT  0.00001, 0.000001
20 PRINT  0.00001 - 0.000001, 0.00001 + 0.000001
30 PRINT  1E38 - 1E37, 1E38 + 1E37
40 PRINT  1.3E19 * 1.3E19 / 1.3E19, 1.3E19 / 1.3E19 * 1.3E19
50 PRINT  -1.3E19 * (-1.3E19), 1.3E19 * (1.3E19)
60 PRINT  -1.3E-19 * (-1.3E-19), 1.3E-19 * (1.3E-19)
```

Execute. Observe bem os resultados. Altere os valores, acima e abaixo dos limites de representação interna e externa dos números e verifique o que ocorre. Reexecute.

2) Escreva o programa:

```
10 PRINT  "ABCDEFGHIJKLM";
20 PRINT  "NOPQRSTUVWXYZ"
```

Execute. Observe os resultados. Escreva um programa que apresente o mesmo resultado, com quatro instruções **PRINT**. Execute.

3) Escreva o programa:

```
10 PRINT  "CALCULO DO QUADRADO DE UM NUMERO"
20 PRINT  "ESCREVA UM NUMERO POSITIVO"
30 INPUT  X
40 PRINT  "VALOR="; X, "QUADRADO="; X ↑ 2
```

Execute. Altere o programa para fornecer outras potências. Reexecute.

4) Escreva o programa:

```
10 PRINT  "QUAL O SEU NOME?"
20 INPUT  X$
30 PRINT  "SEU NOME EH"; " "; X$
```


Execute. Altere o programa para que leia e exiba outros dados, como o endereço, telefone etc.

5) Diariamente uma pessoa compra, pela manhã, pães, leite e carne. Um programa que permite calcular o valor total a pagar é:

```
10 PRINT  "QUANTOS PAES?"
20 INPUT  P
30 PRINT  "PRECO DO PAO=?"
40 INPUT  P1
50 PRINT  "LITROS DE LEITE?"
60 INPUT  L
70 PRINT  "PRECO DO LEITE=?"
80 INPUT  L1
90 PRINT  "QUILOS DE CARNE?"
100 INPUT  C
110 PRINT  "PRECO DA CARNE=?"
120 INPUT  C1
130 PRINT  "VOCE VAI PAGAR CR$"; P * P1 + L * L1 + C * C1
```

Execute. Altere o programa para incluir um maior número de compras. Altere a exibição do resultado para incluir a quantidade e o valor de cada item comprado, antes do total a pagar. Execute.

6) Escreva um programa para calcular as despesas fixas do mês, incluindo as contas de água, luz, telefone, impostos, condomínio etc.

7) Uma pessoa toma um empréstimo a juros. Recebeu como empréstimo uma certa quantia e deverá pagar, depois de um certo tempo, a quantia acrescida do valor dos juros. Um programa que calcula os juros e o montante a pagar, em Cr$ (*cruzeiros*), a partir do valor do principal (quantia emprestada), também em Cr$ (*cruzeiros*), do tempo, expresso como uma fração decimal do período básico, e da taxa (%) de juros do período básico é:

```
10 PRINT  "CALCULO DE JUROS SIMPLES"
20 PRINT  "QUANTIA EMPRESTADA=?"
30 INPUT  P
40 PRINT  "PERIODO=?"
50 INPUT  T
60 PRINT  "TAXA DE JUROS=?"
70 INPUT  J
80 LET    V=P*J*T
90 PRINT  "EMPRESTIMO=CR$"; P
100 PRINT  "JUROS=CR$"; V
110 PRINT  "MONTANTE=CR$"; P + V
```

Sabendo que a fórmula para cálculo de juros compostos é dada por:

$$j = p \cdot (1 + i)^t$$

onde:

j = juros compostos em cruzeiros (Cr$)
p = valor do principal (empréstimo), em cruzeiros (Cr$)
i = taxa de juros por período em forma de fração decimal
t = número de períodos, expresso como fração decimal

escrever um programa para cálculo de juros compostos, imprimindo o valor do principal (empréstimo), o valor dos juros e o valor do montante a pagar, todos esses valores em cruzeiros (Cr$).

QUARTO CAPÍTULO
INSTRUÇÕES BÁSICAS

expressões – instrução de atribuição – instruções de controle – instruções de repetição controlada

4.1 Expressões

Em BASIC, uma fórmula matemática, como

$$((a-b):(c+d)\times e)^3$$

pode ser representada por uma *expressão aritmética*, como

```
((A-B) / (C+D) * E)↑3
```

onde se pode perceber a presença dos *operadores aritméticos*

```
-   /   +   *   ↑
```

Muitos operadores da matemática não estão disponíveis em BASIC, mas existem funções predefinidas que os substituem. Por exemplo, a expressão

$$\sqrt{b^2-4.a.c}$$

é representada em BASIC por

```
SQR ( B ↑ 2 – 4 * A * C )
```

Além dos *operadores aritméticos* as *expressões aritméticas* têm, como componentes, *constantes numéricas, variáveis numéricas* e *funções numéricas*.

O resultado da avaliação de uma expressão aritmética é um valor numérico.

Em BASIC, é possível escrever também *expressões alfanuméricas*, como

```
A$ , " CONCATENADO"
```

Além do *operador de concatenação*, uma *expressão alfanumérica* tem, como componentes, *constantes alfanuméricas, variáveis alfanuméricas* e *funções alfanuméricas*.

O resultado da avaliação de uma expressão alfanumérica é um valor alfanumérico.

Existe ainda, em BASIC, um terceiro tipo de expressões, como

```
( A + B ) ↑ 2 >= 3
```

onde aparece o operador >= (*é maior ou igual*), que é um *operador relacional*, chamada *expressão relacional*. As expressões relacionais fazem parte do grupo das *expressões lógicas*. Nestas podem aparecer *operadores lógicos* como: NOT, AND e OR.

O resultado da avaliação de expressões lógicas é um valor *falso* ou *verdadeiro*, isto é, um valor lógico.

Os três tipos básicos de expressões disponíveis na linguagem BASIC são sumariados no quadro 4-1.

QUADRO 4-1: *EXPRESSÕES*

expressão	aritmética
	alfanumérica
	lógica

As *expressões aritméticas* estão implementadas em todos os sistemas BASIC.

As *expressões aritméticas* que são formadas *sem* o uso de *operadores aritméticos* (*binários*) são as *expressões aritméticas simples*, enquanto as *expressões aritméticas* que são formadas com a presença de *operadores aritméticos* (*binários*) são as expressões *aritméticas compostas*.

Exemplos de *expressões aritméticas simples* são apresentados na figura 4-1.

FIGURA 4-1: *EXEMPLOS DE EXPRESSÕES NUMÉRICAS SIMPLES*

EXPRESSÃO (E COMPONENTE)	SIGNIFICADO
–485	constante numérica (inteira)
372.746294	constante numérica (de ponto fixo)
347E–6	constante numérica (de ponto flutuante)
B3	variável numérica simples
J (A – 7)	referência a elemento de conjunto (variável numérica subscrita)
SQR (A / C1)	referência a função numérica
(A ↑ 4 + B)	expressão aritmética entre parênteses

A forma geral de uma *expressão aritmética simples* é dada por

```
{< constante numérica > | < variável numérica simples > |
 < variável numérica subscrita > | < referência a função numérica > |
 < expressão numérica dentro de parênteses >}
```

As *expressões aritméticas simples* são os *componentes básicos* das *expressões aritméticas compostas*.

Exemplos de *expressões aritméticas compostas* são os que se apresentam na figura 4-2.

FIGURA 4-2: *EXEMPLOS DE EXPRESSÕES NUMÉRICAS COMPOSTAS*

EXPRESSÃO	COMPONENTES	SIGNIFICADO
A (J – 3) / EXP (4.7)	A (J – 3)	variável numérica subscrita
	/	operador aritmético (divisão)
	EXP (4.7)	referência a função numérica
C7 – 3 + K (B2 – 4) ↑ H6	C7	variável numérica simples
	–	operador aritmético (subtração)
	3	constante numérica (inteira)
	+	operador aritmético (adição)
	K (B2 – 4)	variável numérica subscrita
	↑	operador aritmético (potenciação)
	H6	variável numérica simples

A forma geral de uma *expressão aritmética composta* é dada por

> < *expressão aritmética simples* > < *operador aritmético* >
> < *expressão aritmética simples* > [< *operador aritmético* >
> < *expressão aritmética simples* >] . . .

Em outras palavras, uma *expressão aritmética composta* é constituída por *expressões aritméticas simples* ligadas por *operadores aritméticos* (*binários*).

As *regras* para a formação de *expressões aritméticas compostas* são as constantes do quadro 4-2.

QUADRO 4-2: *REGRAS PARA A FORMAÇÃO DE EXPRESSÕES ARITMÉTICAS COMPOSTAS*

1) Não pode haver dois operadores juntos (um ao lado do outro).

2) Todas as operações devem ser declaradas explicitamente, isto é, serão realizadas de acordo com os operadores presentes na expressão.

3) Da avaliação de uma expressão resulta um valor segundo as regras de avaliação de expressões.

A avaliação de *expressões aritméticas compostas* também obedece a *regras* que estão traduzidas no quadro 4-3.

QUADRO 4-3: *REGRAS DE AVALIAÇÃO DE EXPRESSÕES ARITMÉTICAS COMPOSTAS*

1) Regra geral: Avaliar a expressão no sentido

esquerda → direita ,

isto é, no sentido geral da esquerda para a direita.

2) Parênteses embutidos, em vários níveis, devem ser avaliados no sentido

interno → externo ,

isto é, do mais interno para o mais externo.

3) Funções serão avaliadas antes de realizar operações com o seu resultado.

4) Entre dois operadores aritméticos consecutivos existe uma precedência, que será estabelecida segundo a escala de prioridade seguinte:

NÍVEL DE PRECEDÊNCIA	OPERADOR	SÍMBOLO
1	potenciação	↑ (opcionalmente: **)
2	multiplicação/divisão	* /
3	adição/subtração	+ –

5) Operadores com a mesma precedência (nível 2 ou 3 acima) serão avaliados pela regra geral (item 1 acima).

Exemplos da aplicação das regras de avaliação de *expressões aritméticas* são apresentados na figura 4-3.

FIGURA 4-3: *EXEMPLOS DE AVALIAÇÃO DE EXPRESSÕES ARITMÉTICAS COMPOSTAS*

EXPRESSÃO	ETAPA	OPERAÇÃO
SIN (K – 3) + M2 * V5 ↑ 2	1. SIN (K – 3) 2. V5 ↑ 2 3. M2 * V5 ↑ 2 4. SIN (K – 3) + M2 * V5 ↑ 2	função potenciação multiplicação adição
5 – (X1 + 7) ↑ COS (L9)	1. (X1 + 7) 2. COS (L9) 3. (X1 – 7) ↑ COS (L9) 4. 5 – (X1 – 7) ↑ COS (L9)	parênteses função potenciação subtração

O segundo tipo de expressões são as *alfanuméricas*, que serão desenvolvidas em detalhe no sétimo capítulo, que trata de operações sobre cadeias de caracteres. O operador alfanumérico é o de concatenação, representado pelo símbolo , (*vírgula*) ou + (*mais*) ou & (*e comercial*), dependendo da implementação.

Um terceiro tipo de expressões é a chamada *expressão lógica*, de cuja avaliação resulta um valor *verdadeiro* ou *falso*. Basicamente, as *expressões lógicas* são constituídas por *expressões relacionais*.

Exemplos de *expressões relacionais* são apresentados na figura 4-4.

FIGURA 4-4: *EXEMPLOS DE EXPRESSÕES RELACIONAIS (LÓGICAS)*

EXPRESSÃO	COMPONENTE	SIGNIFICADO
A > = 725	A	variável numérica simples
	> =	operador relacional
	725	constante numérica inteira

A forma geral de uma *expressão relacional* é dada por

< *expressão* > < *operador relacional* > < *expressão* >

Uma *expressão relacional* é constituída por duas *expressões* quaisquer (*aritméticas* ou *alfanuméricas*) ligadas por um *operador de relação*. Deve-se sublinhar que são duas e apenas duas as *expressões*, quando se tratar de *expressão lógica*, e que *ambas* as *expressões* têm que ser *simultaneamente aritméticas* ou *simultaneamente alfanuméricas*. *Não* se permite que, das *expressões componentes*, uma seja *aritmética* e outra seja *alfanumérica*.

Algumas implementações ampliadas da linguagem BASIC permitem a construção de *expressões lógicas* usando *operadores lógicos*. Neste caso, as *expressões relacionais*, anteriormente vistas, são *componentes* de *expressões lógicas*. Cabe, neste caso, distinguir entre *expressões simples* e *compostas*.

A forma geral de uma *expressão lógica simples* é dada por

{ (<*expressão lógica*>) | <*expressão relacional*> | <*expressão aritmética*> }

Uma *expressão lógica simples* é, portanto, constituída ou de uma *expressão relacional*, de uma *expressão aritmética* ou *expressão lógica entre parênteses*. Se uma *expressão relacional* é *verdadeira*, o valor é *1* e, se *falsa*, ∅ (*zero*). Por isso as *expressões aritméticas* são incluídas em *expressões lógicas*, isto é, quando têm um valor diferente de ∅ (*zero*) são consideradas com valor lógico *verdadeiro* (*verdade*) e, quando têm valor ∅ (*zero*), são consideradas com valor lógico *falso* (*falsidade*).

< *expressão lógica simples* > < *operador lógico* > < *expressão lógica simples* >

Em geral, uma *expressão lógica composta* é constituída por duas *expressões relacionais* ligadas por um *operador lógico*. Os *operadores* são *três*: NOT, AND e OR. Alguns sistemas permitem outros *operadores*, como XOR. O *operador* NOT é *unário*, isto é, aplica-se a um *único operando*. Neste *único* caso a expressão possuirá *um só operando* (*o segundo operando*). Em todos os demais casos, é obrigatória a presença de *dois operandos*, pois os *operadores* são *binários*.

QUADRO 4-4: *TABELA DE VERDADES DOS OPERADORES LÓGICOS*

A	B	NOT B	A AND B	A OR E	A XOR B
falso	falso	verdadeiro	falso	falso	falso
falso	verdadeiro	falso	falso	verdadeiro	verdadeiro
verdadeiro	falso		falso	verdadeiro	verdadeiro
verdadeiro	verdadeiro		verdadeiro	verdadeiro	falso

Somente para aquelas implementações que permitem o uso de *expressões lógicas compostas*, com *operadores lógicos*, é estabelecida uma *hierarquia* de *prioridades* entre todos os *operadores*, sejam *aritméticos, relacionais* ou *lógicos*. Essa precedência consta do quadro 4-5.

QUADRO 4-5: *PRECEDÊNCIA ENTRE OPERADORES*

NÍVEL	OPERADOR	SÍMBOLO
1	potenciação	↑ (opcionalmente: **)
2	multiplicação/divisão	* /
3	adição/subtração	+ −
4	concatenação	, (opcionalmente: + e &)
5	operadores relacionais	< <= = <> >= >
6	*não* lógico	NOT
7	*e* lógico	AND
8	*ou* e *ou exclusivo* lógico	OR XOR

São válidas, para as *expressões lógicas*, todas as *regras* já definidas para *avaliação* de *expressões aritméticas*. No caso de *expressões lógicas* estarem compondo *expressões aritméticas*, também são válidas as mesmas *regras*. Algumas implementações colocam o *operador* NOT em um *nível de precedência superior* ao das operações de *multiplicação* e *divisão* (*nível 2*). Para evitar que ocorra uma divergência de resultados de uma implementação para outra, é conveniente usar os *parênteses* para se ter uma forma de *avaliação* definida.

4.2 Instrução de atribuição

Quando é *iniciada* a execução de um programa BASIC, *antes* da execução das instruções, é feita uma *pré-execução*, que é uma fase *preparatória*. Durante a *pré-execução*, são *inicializadas* as *variáveis* e *conjuntos*. As *variáveis* e *conjuntos numéricos* são *inicializados* com o valor Ø (*zero*), enquanto as *variáveis* e *conjuntos alfanuméricos* são *inicializados* com o valor *vazio*.

Portanto, para aquelas *variáveis* ou *conjuntos* que devam ter um valor diferente do adquirido *implicitamente* na fase de *pré-execução*, é preciso atribuir-lhes *explicitamente* o valor que será usado em cálculo subseqüente.

Isso deve ser feito com uma instrução apropriada, de *atribuição de valor*. Essa instrução é **LET**, que quer dizer *seja*. A *instrução de atribuição* pode atribuir valores a *variáveis numéricas* ou *alfanuméricas*.

Por exemplo, uma *instrução de atribuição*, como

```
LET X = 5
```

,

faz a atribuição à *variável numérica* X do valor *numérico inteiro* 5, isto é, após a execução da instrução, o valor de X é 5.

Num outro exemplo de *instrução de atribuição*, como

```
LET X$ = "SIM"
```

,

é feita a *atribuição* à *variável alfanumérica* X$ do valor *alfanumérico* SIM, isto é, após a execução da instrução, o valor de X$ é SIM.

Podem-se considerar, portanto, dois *tipos* de *instruções de atribuição*, como mostra a divisão apresentada no quadro 4-6.

QUADRO 4-6: *INSTRUÇÕES DE ATRIBUIÇÃO*

instrução de atribuição	aritmética
	alfanumérica

Duas *funções* podem ser distinguidas na execução de uma *instrução de atribuição*, **LET**, as quais constam do quadro 4-7.

QUADRO 4-7: *FUNÇÕES DA INSTRUÇÃO DE ATRIBUIÇÃO*

1) Avaliar a *expressão à direita* do *sinal de atribuição*.

2) Atribuir à *variável à esquerda do sinal de atribuição* o valor calculado da *expressão à direita*.

A *instrução de atribuição* pode ser mais complexa do que as dos exemplos até aqui apresentados. Exemplos diversos são apresentados na figura 4-5.

FIGURA 4-5: *EXEMPLOS DE INSTRUÇÕES DE ATRIBUIÇÃO*

EXEMPLO	EXPLICAÇÃO
LET Y = Y − 1	O valor da *variável* Y é diminuído de uma unidade, 1.
LET X (J , 2) = Y + 4	Ao elemento na posição (J , 2) da *matriz* X é atribuído o valor calculado de Y + 4.
LET A (X ↑ B + Y * C) = Q / (R + S)	Ao elemento na posição (X ↑ B + Y * C) do *vetor* A é atribuído o valor calculado de Q / (R + S).

A forma geral de uma *instrução de atribuição aritmética* é

LET < *variável numérica* > = < *expressão aritmética* >

A forma geral da *instrução de atribuição* (*numérica* e *alfanumérica*) é

LET < *variável* > = < *expressão* >

observando-se que *variável* e *expressão* têm que ser ambas *numéricas* ou ambas *alfanuméricas*.

Em sistemas BASIC que distinguem *variáveis numéricas inteiras* e *fracionárias*, antes de atribuir à *variável numérica* o valor da *expressão aritmética*, se os *tipos* são diferentes, é feita uma conversão interna. Essas *regras* são simples e se encontram no quadro 4-8.

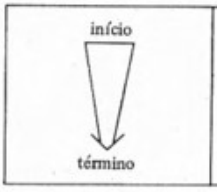
QUADRO 4-8: *REGRAS DE CONVERSÃO NA ATRIBUIÇÃO DE VALORES NUMÉRICOS*

tipo de variável \ tipo de expressão	inteira	fracionária
inteira	atribui	converte a valor inteiro e atribui
fracionária	atribui	atribui

4.3 Instruções de controle

Um programa é executado, instrução por instrução, em seqüência, da linha de mais baixo *número de linha* até a de mais alto *número de linha*. A figura 4-6 apresenta um exemplo de programa tipicamente seqüencial.

FIGURA 4-6: *EXEMPLO DE PROGRAMA TIPICAMENTE SEQÜENCIAL*




```
10 INPUT "A=", A
20 INPUT "B=", B
30 LET C = A * B
40 PRINT "C="; C
50 END
```

É *praticamente impossível* resolver a maioria dos problemas com programas de execução puramente *seqüencial*. Por isso foram criadas *instruções de desvio*, que permitem mudar a ordem normal de execução.

Existem *dois tipos* de *instruções de desvio*, divisão esta que é apresentada no quadro 4-9.

QUADRO 4-9: *INSTRUÇÕES DE DESVIO*

instrução de desvio	incondicional
	condicional

Exemplos de *instrução de desvio incondicional* encontram-se na figura 4-7.

FIGURA 4-7: *EXEMPLOS DE INSTRUÇÃO DE DESVIO INCONDICIONAL SIMPLES*

EXEMPLO	EXPLICAÇÃO
GOTO 9000	a execução do programa é desviada desta instrução para a de número **9000**

A forma geral da *instrução de desvio incondicional* é dada por

GOTO < *número de linha* >

O *desvio* pode ser feito para qualquer instrução do programa, seja *executável* ou *não-executável*, seja um *desvio* para *trás* ou para *frente*. No caso de *desvio* para uma instrução *executável*, a execução do programa continua com esta instrução, *inclusive*. No caso de *desvio* para uma instrução *não-executável*, a execução prosseguirá com a próxima instrução *executável*. O *número da linha para desvio* tem necessariamente que ser um *número de linha* do *mesmo* programa. Normalmente, a execução do programa continuará com a instrução para a qual for feito o *desvio*.

Nos menores microcomputadores pessoais (primeiro grupo), o *número de linha* pode ser obtido como o valor inteiro de uma *expressão aritmética*, o que confere à instrução **GOTO**, uma grande flexibilidade. Quer dizer que com uma única instrução de desvio, é possível fazer o desvio não apenas para uma, mas para várias instruções, de acordo com o resultado da avaliação da expressão aritmética. Neste caso, a instrução **GOTO** se comporta como uma instrução de desvio múltiplo, **ON . . . GOTO**, que será apresentada a seguir.

Além da *instrução de desvio incondicional simples*, **GOTO**, existem duas outras instruções de *desvio*, também *incondicional*. A primeira delas é a instrução de *desvio incondicional para sub-rotina*, **GOSUB**, à qual está associada uma outra instrução, para indicar *término de sub-rotina*, **RETURN**. A diferença entre o *desvio incondicional simples*, da *instrução* **GOTO**, e o *desvio para sub-rotina*, da *instrução* **GOSUB**, é que esta última permite executar trechos de programa e depois, *automaticamente*, retornar para executar a instrução *seguinte* à *instrução* **GOSUB**. A segunda forma de *desvio incondicional* mencionada é a *instrução de desvio incondicional múltiplo*, **ON . . . GOTO**.

A *instrução de desvio para sub-rotina*, **GOSUB**, e a *instrução* **RETURN** serão tratadas posteriormente, juntamente com *funções* e *subprogramas*.

Quanto à *instrução de desvio incondicional múltiplo*, apresenta-se na figura 4-8 um exemplo.

FIGURA 4-8: *EXEMPLOS DE INSTRUÇÃO DE DESVIO INCONDICIONAL MÚLTIPLO*

EXEMPLO	EXPLICAÇÃO
ON U ↑ 3 − B **GOTO** 5, 10, 15, 10	com base no valor (inteiro) da expressão numérica U ↑ 3 − B, será feito um desvio incondicional, respectivamente: • se *expressão* = 1, para linha 5 • se *expressão* = 2, para linha 10 • se *expressão* = 3, para linha 15 • se *expressão* = 4, para linha 10

A forma geral de uma *instrução de desvio incondicional múltiplo* é dada por

ON < *expressão aritmética* > **GOTO** < *número de linha* > [< *número de linha* >] . . .

O valor da *expressão aritmética*, depois de calculado, é *truncado*, isto é, é tomado seu valor *inteiro*. Se este valor é menor do que *1* ou maior do que o número de *números de linhas* da lista à direita, dependendo do sistema BASIC, poderá ocorrer uma mensagem de erro ou simplesmente transferência da execução para a instrução imediatamente seguinte. Normalmente, a execução continuará na linha para a qual foi feito o desvio. O mesmo *número de linha* poderá aparecer diversas vezes na lista à direita. A instrução **ON . . . GOTO** *não* está disponível nos menores microcomputadores (primeiro grupo).

A *instrução de desvio condicional* permite a realização de *testes de condição*, ou seja, permite a decisão sobre *alternativas de execução*. Exemplos de *instrução de desvio condicional* são apresentados na figura 4-9.

FIGURA 4-9: *EXEMPLOS DE INSTRUÇÃO DE DESVIO CONDICIONAL*

EXEMPLO	EXPLICAÇÃO
IF 3 * D > = E − 3 **THEN PRINT** " ALO!"	• se a *condição* 3 * D > = E − 3 for *verdadeira* então a instrução **PRINT** " ALO!" será executada e, após, a instrução seguinte. • se a *condição* é *falsa*, a execução passa para a instrução seguinte e a

	instrução **PRINT** " ALO!" é ignorada.
IF A$ = "SIM" **THE GOTO** 5Ø	• se A$ tem o valor SIM, *desvia* para linha 5Ø. • se A$ tem valor diferente de SIM, executa a linha seguinte.

A forma geral da *instrução de desvio condicional* é dada por

IF < *expressão lógica simples* > THEN < *instrução* >

A *condição* pode ser representada por uma *expressão aritmética* ou uma *expressão relacional*. Na grande maioria dos casos, é uma *expressão relacional*, e a *instrução* é uma *instrução de desvio incondicional*, **GOTO**, se bem que não necessariamente esta instrução tenha que ser usada. Qualquer instrução executável, em princípio, pode ser usada, com *exceção* de **FOR**, **NEXT** e outro **IF . . . THEN**. As *instruções de especificação* (ou *declaração*) como **REM**, **DIM**, **END**, **DEF** FN, **IMAGE** e outras análogas *não* podem ser usadas na instrução **IF . . .** THEN. A execução da *instrução* ocorrerá se o resultado da avaliação da condição for *verdadeira* ou *não-nula* (diferente de *zero*), e *não* ocorrerá se o resultado da avaliação da condição for *falsa* ou *nula* (*zero*). Em geral a execução do programa *continuará* com a instrução seguinte, seja ou *não* executada a *instrução* de **IF . . .** THEN, a não ser que a *instrução* seja um *desvio incondicional*, **GOTO** ou **ON . . . GOTO**. O leitor pode avaliar a grande flexibilidade de programação propiciada por esta instrução.

Entre as *instruções de controle* se incluem também as *instruções de término de execução*, **END** e **STOP**. A *instrução* **END** já é conhecida do leitor, da primeira lição, pois é a última instrução de um programa, isto é, constitui também o *fim físico do programa*. Por isso tem que ter sempre o *mais alto* (*maior*) *número de linha*. A diferença entre as *instruções* **END** e **STOP** é apenas esta: a posição dentro do programa. Um programa tem que ter e só pode ter uma *única instrução* **END**, a *última*, mas pode ter *várias instruções* **STOP**, em *diversos pontos* do programa. *Não* pode haver nenhuma instrução *após* a instrução **END**.

A forma geral da *instrução de término de execução* e *fim físico de programa* é dada por

END

A forma geral da *instrução de término de execução* (simplesmente) é dada por

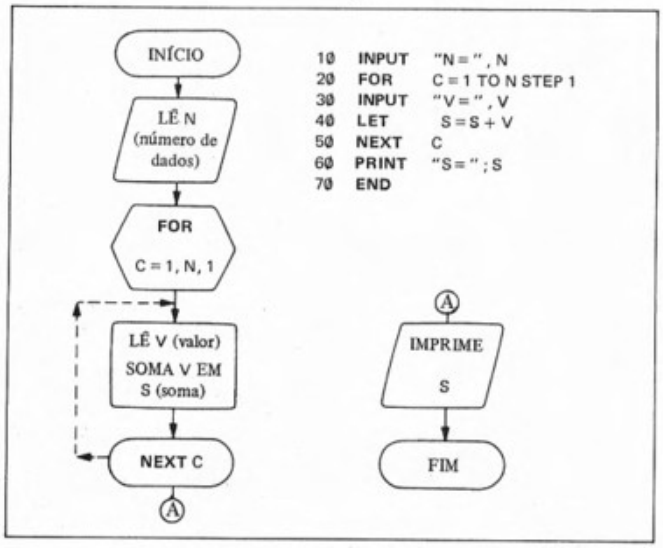
STOP .

4.4 Instruções de repetição controlada

Embora sejam também instruções de *controle*, as instruções **FOR** e **NEXT** merecem uma seção especial, dada sua importância. Permitem a execução de *malhas finitas, automaticamente*, sob controle de condições predeterminadas.

Um exemplo de programa mostrando o uso da *instrução de repetição controlada* aparece na figura 4-1Ø.

FIGURA 4-1Ø: *EXEMPLO DE PROGRAMA COM MALHA DE* **FOR** *E* **NEXT**



A instrução **FOR** *automaticamente inicializa* o controle C com o valor que está à direita do sinal de atribuição (=), no caso, 1. O *valor final* ou *máximo* do *índice* ou *controle* C é dado pelo valor que aparece à direita da palavra-chave TO, no caso, N. O *incremento* a ser aplicado ao *controle* C é o valor que aparece após a palavra-chave STEP, no caso, 1. O *limite* ou *alcance da malha* é o conjunto de instruções entre a instrução **FOR** e a instrução **NEXT**, com a mesma *variável de controle* C, no caso, **NEXT** C. Ao atingir a instrução **NEXT**, o valor do *índice* C é *automaticamente* aumentado, com o valor do *incremento*. Se o *valor final* não foi ultrapassado, é feito um desvio para a primeira instrução dentro do *alcance*, isto é, aquela imediatamente seguinte à instrução **FOR**, no caso, **INPUT**. As *instruções dentro do alcance da malha* serão executadas tantas vezes quantas forem necessárias, até que o valor do *índice* ou *controle*, no caso, C, ultrapasse o *valor final*, no caso, N. Quando ultrapassado o *valor final*, será executada a instrução imediatamente seguinte à *malha*, isto é, depois da *instrução* **NEXT**. No exemplo, após a execução desta *instrução*, **PRINT**, o programa *termina*, com a *instrução* **END**.

Alguns exemplos esclarecerão melhor a forma da *instrução de repetição controlada* e são apresentados na figura 4-11.

FIGURA 4-11: *EXEMPLOS DE INSTRUÇÃO DE REPETIÇÃO CONTROLADA*

EXEMPLO	EXPLICAÇÃO
FOR V = −4.678 TO 5.53 STEP Ø.126	A *variável de controle* V é *inicializada* com o valor −4.678 e são guardados os *valores final*, 5.53, e do *incremento*, Ø.126.
FOR B1 = 1ØØ TO −1ØØ STEP − 2.5	São atribuídos à *variável de controle* B1, ao *limite* e ao *incremento*, respectivamente, os valores 1ØØ, −1ØØ e −2.5.
FOR D = A (K , 3) TO R (G + 1) * P	O *controle* D é *inicializado* com o valor da variável subscrita A (K , 3), enquanto o valor final é calculado da expressão R (G + 1) * P e o *incremento*, por falta de expecificação, assume o valor 1.

A forma geral da *instrução de repetição controlada* é dada por

FOR < *variável de controle* > = < *valor inicial* >
TO < *limite* > [**STEP** < *incremento* >]

A presença de uma *instrução* **FOR** determina o *início* de uma *malha de repetição controlada*, obrigando a presença, no *final* da *malha*, de uma instrução **NEXT**. *Não* pode haver instrução **FOR** sem a correspondente instrução **NEXT**, e vice-versa. A *variável de controle* é sempre uma *variável numérica simples* e *não* uma *variável numérica subscrita*. O *valor inicial*, o *valor final* ou *limite* e o *incremento* são representados por *expressões aritméticas* quaisquer. Os valores dessas *expressões* são calculados e guardados internamente. Quer dizer que os valores das *variáveis* que participam na formação das *expressões* podem ser alterados durante a execução da *malha*, pois isto não mudará mais os *valores inicial*, *final* e do *incremento*. O valor da variável de *controle não* pode ser alterado durante a execução da *malha*, pois o *resultado é imprevisível*, por perda do controle.

Os *valores inicial*, *final* e do *incremento* podem ser quaisquer, isto é, *negativos*, *positivos* ou *nulos*. Evidentemente *não* há sentido em ter um *incremento nulo* (*zero*). Se o *valor final* é menor do que o *inicial*, o *incremento* tem que ser *negativo*. Se o valor *final* é *maior* do que o *inicial*, o *incremento* tem que ser *positivo*. Essas condições são verificadas e, se *não* forem obedecidas, resultará em mensagem de erro.

Um exemplo da *instrução de fim de malha de repetição controlada* consta da figura 4-12.

FIGURA 4-12: *EXEMPLO DA INSTRUÇÃO DE FIM DE REPETIÇÃO CONTROLADA*

EXEMPLO	EXPLICAÇÃO
NEXT L	À *variável de controle* L é adicionado o valor (negativo ou positivo) do *incremento* e seu novo valor comparado com o *limite*; • se o *limite* é ultrapassado, a execução continua com a instrução seguinte; • se o *limite* não é ultrapassado, é feito um *desvio* para a primeira instrução dentro do alcance da malha.

A forma geral da *instrução de fim de repetição controlada* é dada por

NEXT < *variável de controle* >.

A *instrução* **NEXT** é, obrigatoriamente, a última da *malha de repetição controlada* e, portanto, tem que ter numeração de linha imediatamente superior à última instrução dentro do *alcance*. A uma *instrução* **NEXT**, corresponde necessariamente uma *instrução* **FOR**, e vice-versa. A *variável de controle* é necessariamente a mesma em ambas as instruções, sem o que o sistema BASIC não poderá identificar a que *instrução* **FOR** está ligada uma *instrução* **NEXT**.

O valor da *variável de controle*, quando se esgota, isto é, quando termina a execução da *malha*, é em geral indeterminado. Caso seja feito um *desvio* para fora da *malha*, antes que se esgote, o valor da *variável de controle* é o último que lhe foi atribuído.

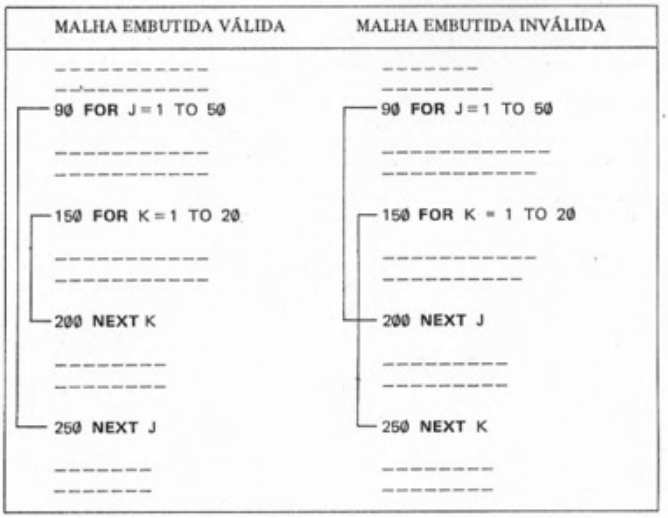
Saindo-se de uma *malha de repetição controlada* por meio de um *desvio*, antes que esta se esgote, é permitido voltar novamente para dentro da *malha*, por um novo *desvio*, de fora para dentro. Todavia, em qualquer outra hipótese, é inválido fazer um *desvio* para dentro de uma *malha de repetição controlada*, resultando em *erro*. O conjunto de instruções executadas fora da *malha*, quando se sai para novamente retornar através de instruções de *desvio*, é o *alcance estendido*. Além do *retorno* a partir de um *alcance estendido*, só existe uma única possibilidade de voltar a executar uma *malha de repetição controlada*: reiniciá-la pela *instrução* **FOR**, caso em que todas as condições iniciais serão restabelecidas.

O ponto em que a *variável de controle* é comparada com o *limite* para a decisão de terminar ou não uma *malha* varia de posição, de uma implementação para outra. Naquelas em que esse teste ou verificação é feito antes de executar as instruções dentro do *alcance da malha*, se o *controle* ultrapassa o *limite* antes da primeira execução, *não* haverá nenhuma execução. Se o teste ou verificação for colocado ao *final* do *alcance* da *malha*, isto é, na *instrução* **NEXT**, as instruções do *alcance* da *malha* serão executadas pelo menos uma vez, se o *controle* ultrapassa o *limite*.

Dentro do alcance de uma *instrução de repetição controlada* é permitido que haja outra, e assim sucessivamente, em geral até um número de *oito níveis*, no total. As *malhas mais internas* são chamadas *embutidas* ou *aninhadas*.

Uma *malha embutida* dentro de outra tem que estar inteiramente contida dentro do *alcance* da *malha mais externa*. A figura 4-13 mostra exemplos de *malhas embutidas válidas* e *inválidas*.

FIGURA 4-13: *EXEMPLOS DE MALHAS EMBUTIDAS VÁLIDAS E INVÁLIDAS*



Exercícios:

ESTUDO DIRIGIDO:

1) Quais são os principais tipos básicos de expressões? Que é uma expressão aritmética? Que é uma expressão aritmética simples? Que é uma expressão aritmética composta? Quais são os operadores aritméticos? Quais são as regras para a formação de expressões aritméticas compostas? Quais são as regras para a avaliação de expressões aritméticas compostas? Que é uma expressão alfanumérica? Que é uma expressão alfanumérica simples? Que é uma expressão alfanumérica composta? Qual o operador alfanumérico? Como é formada uma expressão alfanumérica composta? Como é avaliada uma expressão alfanumérica composta? Que é uma expressão lógica? Que é uma expressão relacional? Quais os operadores relacionais? Que é uma expressão relacional simples? Que é uma expressão relacional composta? Como é formada a expressão relacional composta? Que é uma expressão lógica simples? Quais os operadores lógicos mais usuais? Que é uma expressão lógica composta? Como são formadas as expressões lógicas compostas? Como são avaliadas as expressões lógicas compostas? Qual a prioridade dos operadores, na avaliação de expressões lógicas compostas?

2) Que é uma instrução de atribuição? Quais os tipos de instrução de atribuição? Qual o sinal de atribuição? Que funções executa a instrução de atribuição? Qual a forma geral da instrução de atribuição? Que ocorre quando a variável e a expressão numéricas são de tipos diferentes?

3) Quais as principais instruções de controle? Quais os tipos de instruções de desvio? Que é desvio incondicional? Qual a instrução de desvio incondicional? Qual a forma geral da instrução de desvio incondicional? Que é desvio incondicional múltiplo? Qual a instrução de desvio incondicional múltiplo? Qual a forma geral da instrução de desvio incondicional múltiplo? Que é desvio condicional? Qual a instrução de desvio condicional? Qual a forma geral da instrução de desvio condicional? Que outras instruções de controle são comuns? Que é instrução de fim físico? Que é instrução de fim lógico? Qual é a instrução de fim físico? Qual é a instrução de fim lógico? Que é fim de execução?

4) Que é repetição controlada? Quais as instruções usadas para realizar uma malha de repetição? Que é variável de controle? Que é valor inicial? Que é valor final ou limite? Que é incremento? Quando deve ser negativo o incremento? Quando é encerrada a execução da malha de repetição? Que ocorre se as condições iniciais já fazem com que o limite esteja atingido? Qual a forma geral da instrução de repetição controlada? Qual a forma geral da instrução de fim da repetição controlada? A variável de controle pode ter o seu valor alterado dentro do alcance? As variáveis que entram na formação das expressões do valor inicial, valor final e incremento podem ter seu valor alterado dentro do alcance? Que acontece se a variável de controle tiver seu valor alterado dentro do alcance? As malhas de repetição podem ser embutidas ou aninhadas, umas dentro de outras? Malhas mais internas podem ultrapassar o alcance de malhas mais externas?

PROBLEMAS:

---

1) Através de um programa do tipo

```
10 PRINT   expressão
```

,

mandar exibir o valor das seguintes *expressões* que são dadas em notação matemática:

a) 4,739 x 3,72 (Resp.: 17.62908)
b) 456 ÷ 3,5 (Resp.: 130.28571)
c) 7 − 1,478 (Resp.: 5.522)
d) 4 + 7 (Resp.: 11)
e) $(4{,}786)^3$ (Resp.: 109.62714)

2) Com o auxílio de um programa do tipo

```
10 LET     A$ = constante alfanumérica-1
20 LET     B$ = constante alfanumérica-2
30 LET     C$ = A$ , B$
40 PRINT   C$
```

,

fazer exibir as seguintes *cadeias de caracteres* (partindo-as):

a) BOM DIA
b) ALO AMIGO
c) CADEIA DE CARACTERES
d) ALFANUMERICO
e) CONCATENADO

3) Com o auxílio de um programa com a forma

```
10 LET     A = constante numérica-1
20 LET     B = constante numérica-2
-----------
-----------
--- PRINT   expressão com A, B, C etc.
```

,

efetuar, com valores da sua escolha, o *cálculo* das *fórmulas e imprimir:*

a) $a - b \times c^2$
b) $(a - 3) : c + e$
c) $x^2 - y : z$
d) $p \times q^r - s : t + 7$
e) $(x\,(y : (z\,(a - b + c))))$

4) Escreva um programa com a forma

```
10 DIM     A$ (expressão numérica)
20 DIM     B$ (expressão numérica)
30 INPUT   " A$ = " , A$
40 INPUT   "B$ = " , B$
50 PRINT   A$ ; B$
```

,

e o execute, fornecendo dados, diversas vezes. Altere o programa para que tenha uma única instrução **DIM** e uma única instrução **INPUT**. Altere a ordem das variáveis na instrução **PRINT** e, depois, o formato implícito de saída. Experimente diversas dimensões para A$ e B$. Observe os efeitos.

5) Um programa tem a forma:

```
10 PRINT   "A = "
20 INPUT   A
30 IF      A > = 99999 THEN GOTO 90
40 PRINT   " B = "
50 INPUT   B
60 LET     C = A * B
70 PRINT   "C = " ; C
80 GOTO    10
90 STOP
```

Executar. Para determinar, dê um valor de A igual ou maior que 99999. Substitua as variáveis numéricas por alfanuméricas, a expressão numérica por alfanumérica e o teste de fim de execução, feito com o valor 99999, por outro, com a cadeia de caracteres SIM. Executar com vários dados. Altere o programa para exibir cada dado, assim que seja lido, na mesma linha que pede o dado. Reexecutar.

6) Um programa tem a forma:

```
10  LET    S=0
20  PRINT  "N="
30  INPUT  N
40  LET    C=1
50  IF     C>N THEN GOTO 100
60  PRINT  "V="
70  INPUT  V
80  LET    S=S+V
90  LET    C=C+1
100 GOTO   40
110 PRINT  "S=";S
```

Execute o programa com vários dados e obtenha a soma de uma série de números. Inclua, no final do programa, as instruções:

```
120 PRINT  "REINICIO= SIM, TERMINO=NAO"
130 PRINT  "SIM OU NAO?"
140 INPUT  X$
150 IF     X$="SIM" THEN GOTO 10
160 PRINT  "TERMINO DO PROGRAMA"
```

Execute novamente e verifique se ficou mais fácil. Altere o programa para que os dados sejam exibidos assim que forem lidos, na mesma linha que os pede. Reexecutar.

7) Reescreva o programa do exercício 6 de tal maneira que o teste da condição seja feito no final da malha de repetição. O teste deverá ser feito com a condição complementar, isto é, C < = N. Execute com os mesmos casos do exercício 6 e verifique se os resultados estão corretos.

8) Com a utilização das instruções **FOR** e **NEXT**, substitua a malha de repetição do programa do exercício anterior por outra, que faça a mesma coisa, com o uso das instruções de início e de fim de repetição controlada. Execute com os mesmos casos do exercício anterior e verifique se os resultados são os mesmos.

9) Escreva um programa que leia três valores numéricos e os exiba na tela, bem como informe qual deles é o maior (ou menor, como alternativa), precedido da mensagem MAIOR= (ou MENOR=). Deve ser usada a instrução **IF . . . THEN** para a comparação dos valores. Execute.

10) Escreva um programa que leia um número de valores numéricos previamente informado e calcule e exiba a sua soma. Execute.

11) Escreva um programa que leia um número qualquer de valores numéricos e calcule e exiba a sua média aritmética. Execute.

12) Escreva um programa que leia uma série de valores numéricos, com qualquer número de valores, e determine qual deles é o maior (ou menor, como alternativa), exibindo-o precedido da mensagem MAIOR= (ou MENOR=).

QUINTO CAPÍTULO
A TELA E SEU USO

características da tela – caracteres gráficos – instruções para uso da tela – títulos e mensagens – gráficos – o uso do tempo e do movimento

5.1 Características da tela

O tamanho da tela de microcomputadores pessoais varia bastante de uma marca para outra. Será tomada por base a tela gerada pelos micros pessoais de menor porte (primeiro grupo). Os demais micros geralmente têm telas maiores, com maior número de posições, tanto na horizontal como na vertical.

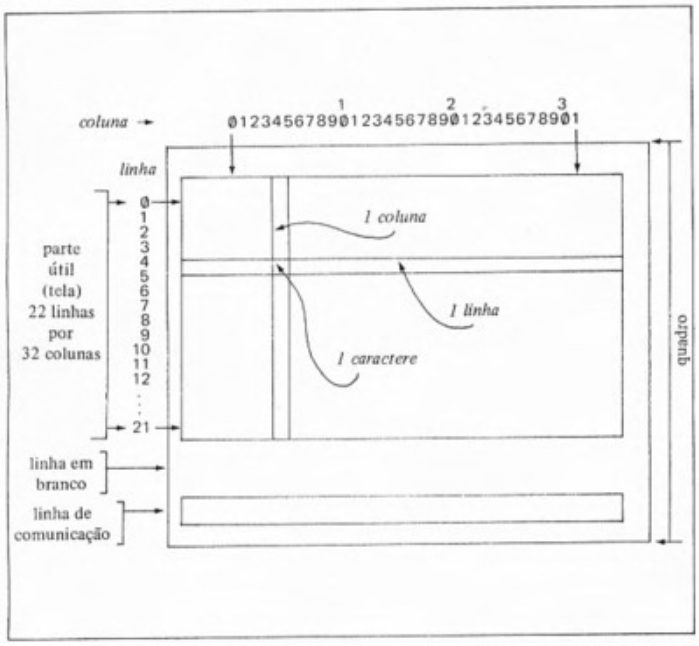
A parte útil do vídeo de microcomputadores pessoais de menor porte tem *22* linhas por *32* colunas, numeradas, respectivamente, de Ø a 21 (de cima para baixo) e de Ø a 31, (da esquerda para a direita), para a exibição de dados. Além disso, possui uma linha de comunicação na parte inferior do quadro, separada da parte útil (tela) por uma linha em *branco*. A linha de comunicação serve para a edição de comandos e instruções BASIC e para mensagens do sistema BASIC, reportando o resultado da execução de comandos e programas.

Na figura 5-1 é apresentado um diagrama que mostra o quadro do vídeo de um pequeno micro.

Quando um programa exibe dados no vídeo, a parte útil é a área do vídeo ocupada pela tela. Se o número de linhas exibido for maior do que *22*, o programa pára, por falta de espaço. Para continuar, deve-se dar o comando **CONT**. Ao se esgotar cada nova tela, novo comando **CONT** permitirá a exibição das seguintes, até que todos os dados tenham sido exibidos.

A linha de comunicação, se for totalmente esgotada, é estendida, deslocando-se para cima os dados que ocupam a linha corrente, deixando de novo a linha de comunicação limpa para a entrada de novos dados. Todos os dados da parte útil, se houver, neste caso serão deslocados também para cima, de uma linha. Esse processo se repete quantas vezes forem necessárias para

FIGURA 5-1: *QUADRO DO VÍDEO DE UM MICROCOMPUTADOR PESSOAL DE MENOR PORTE*



permitir a escrita na linha de comunicação. As linhas anteriores, que ocupavam a linha de comunicação e que se deslocam para cima, vão entrando na parte útil, após passar da linha em *branco*.

Cada posição da parte útil do quadro do vídeo é ocupada por um único caractere da tela, seja ele numérico, alfabético, caractere especial ou gráfico.

Um programa para marcação do número de colunas da tela de vídeo de um desses micros pessoais tem o aspecto da figura 5-2.

FIGURA 5-2: *EXEMPLO DE PROGRAMA PARA MARCAÇÃO DO NÚMERO DE COLUNAS DA TELA*

```
                  10 espaços    9 espaços    9 espaços
10  PRINT  "          1         2         3"
20  PRINT  "01234567890123456789012345678901"
```

Um programa para marcação dos números de linhas da tela de vídeo de um micro pessoal aparece na figura 5-3.

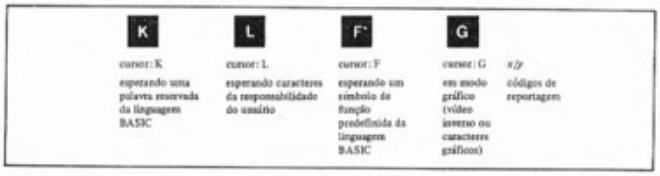
FIGURA 5-3: *EXEMPLO DE PROGRAMA PARA MARCAÇÃO DO NÚMERO DE LINHAS DA TELA*

```
10  FOR   I=0 TO 21
20  PRINT I
30  NEXT  I
```

Na linha de comunicação aparece um *cursor*, que é um quadradinho preto com uma letra em branco dentro (vídeo invertido). As letras que aparecem são K, L, F e G. A letra K aparece sempre que o micro é ligado e quando está esperando uma palavra reservada da linguagem BASIC. Quando o que vai ser escrito é da responsabilidade do usuário, a letra que aparece é a L. No modo gráfico, isto é, vídeo invertido ou caracteres gráficos, a letra que aparece é G. Para obter o símbolo de uma função é preciso que o cursor tenha a letra F.

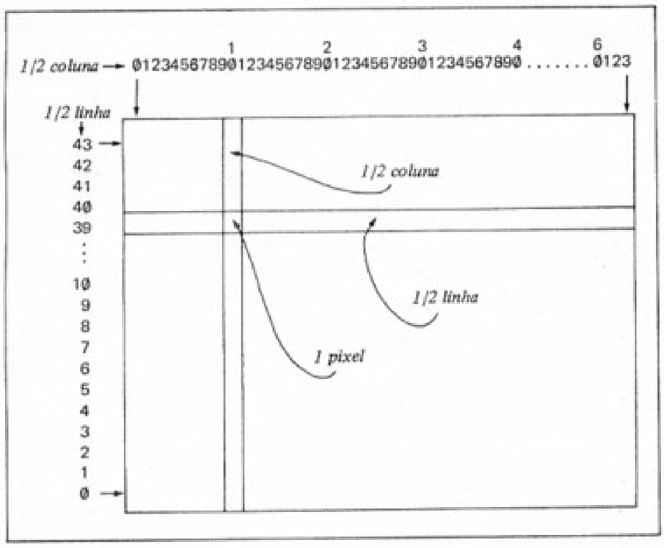
Além disso, a linha de comunicação é usada pelo sistema BASIC para enviar mensagens ao usuário, os códigos de reportagem, que têm a forma x/y, onde x é um algarismo ou letra (hexadecimal), indicativo de uma ocorrência, e y é o número da linha de programa onde ocorreu. Este código é exibido sempre que é feita a execução de um comando ou programa. A figura 5-3 mostra diversas situações da linha de comunicação com os cursores e os códigos de reportagem.

FIGURA 5-3: *CURSORES E CÓDIGOS DE REPORTAGEM NA LINHA DE COMUNICAÇÃO*



Para permitir uma melhor definição de figuras na tela, a parte útil foi ainda mais dividida, cada caractere em quatro partes, cada uma das quais é um *pixel*. Esses pequenos quadradinhos são colocados e retirados da tela, respectivamente, pelas instruções **PLOT** e **UNPLOT**, que serão apresentadas posteriormente neste capítulo. A divisão da tela em *pixéis* produz uma matriz de *44* linhas por *64* colunas, numeradas, respectivamente, de Ø a 43 (de baixo para cima) e de Ø a 63 (da esquerda para a direita). Cada *pixel* ocupa o espaço de *1/2* linha por *1/2* coluna. A divisão da tela em *pixéis* é mostrada na figura 5-4.

FIGURA 5-4: *DIVISÃO DA TELA EM PIXÉIS*



5.2 Caracteres gráficos

Os caracteres gráficos são usados, como qualquer outro caractere, para a formação de cadeias de caracteres, em constantes ou variáveis.

Nos menores micros pessoais, o conjunto de caracteres gráficos é constituído por *2Ø* símbolos que estão relacionados no quadro 5-1.

QUADRO 5-1: *SÍMBOLOS GRÁFICOS USUAIS NOS MENORES MICROS PESSOAIS*

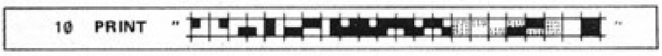
CARACTERE	CÓDIGO DEC	CÓDIGO HEX	TECLA (em modo gráfico e com SHIFT)	CARACTERE	CÓDIGO DEC	CÓDIGO HEX	TECLA (em modo gráfico e com SHIFT)
	1	01	1		7	07	E
	2	02	2		132	84	R
	135	87	3		6	06	T
	4	04	4		134	86	Y
	5	05	5		8	08	A
	131	83	6		10	0A	S
	3	03	7		9	09	D
	133	85	8		138	8A	F
	129	81	Q		137	89	G
	130	82	W		136	88	H

Para obter um caractere gráfico em *branco*, a tecla SPACE deve ser calcada, em modo normal de operação do teclado. Para obter um caractere gráfico totalmente *preto*, deve-se entrar em modo gráfico e então calcar a tecla SPACE, que dará o vídeo inverso de *branco*. Incluindo-se esses dois símbolos, o conjunto de símbolos gráficos chega a ter *22* símbolos.

Recorda-se que para entrar em ou sair de modo gráfico a tecla SHIFT deve ser calcada simultaneamente com a tecla 9 (GRAPHICS).

Um programa simples que mostrará os símbolos gráficos na tela é o apresentado na figura 5-5.

FIGURA 5-5: *EXEMPLO DE PROGRAMA PARA EXIBIR OS SÍMBOLOS GRÁFICOS*



No exemplo da figura 5-5, a cadeia de caracteres constituída por símbolos gráficos a ser exibida é uma constante alfanumérica (literal). A mesma cadeia de caracteres pode ser exibida através de uma variável alfanumérica, como no programa da figura 5-6.

FIGURA 5-6: *EXEMPLO DE PROGRAMA PARA EXIBIR SÍMBOLOS GRÁFICOS A PARTIR DE UMA VARIÁVEL*

```
10 LET   A$ = "
20 PRINT A$
```

Além disso, símbolos gráficos poderão ser lidos durante a execução do programa, para uma variável alfanumérica, a partir do teclado. Basta que o usuário digite os símbolos gráficos ao teclado, como o faz para qualquer outra cadeia de caracteres. Um programa para ler qualquer cadeia de caracteres e que pode ser usado para ler também caracteres gráficos é o da figura 5-7.

FIGURA 5-7: *EXEMPLO DE PROGRAMA PARA EXIBIR SÍMBOLOS GRÁFICOS LIDOS A PARTIR DO TECLADO PARA UMA VARIÁVEL*

```
10 PRINT "DIGITE OS SIMBOLOS GRÁFICOS"
20 PRINT "
30 INPUT A$
40 PRINT "VOCE DIGITOU OS SIMBOLOS"
50 PRINT A$
```

5.3 Instruções para o uso da tela

A principal instrução para o uso da tela é **PRINT**. O leitor já conhece esta instrução, na sua forma mais usual, bem como o efeito do uso de ; (*ponto e vírgula*) e , (*vírgula*) para a formatação implícita da saída de dados.

Além desses recursos, para posicionar os dados de saída na tela, existem alguns outros. O mais comum de todos é a função TAB, que é incluída na lista de saída para tabular os dados. O efeito da tabulação é análogo ao que existe nas máquinas de escrever. A função TAB provoca o salto do *cursor* na linha de impressão para uma posição especificada, na mesma linha, onde será iniciada a colocação dos dados. O cursor nunca pode saltar para trás. Se uma referência à função TAB implicar um salto para trás, primeiro será feito um salto de linha e o cursor irá para a posição especificada, na nova linha.

Um exemplo de *instrução de saída com tabulação* é

```
PRINT A, B; TAB (30); X, TAB (50), Y
```

Nesta instrução será dada saída ao valor de A, seguido do valor de B, na próxima *zona*, isto é, com *formato aberto*. A *pontuação após* a *variável* B é *irrelevante* porque a *função* TAB provocará um *salto* para a *posição* 30. A *pontuação* após a *função* TAB é levada em consideração para a colocação do valor seguinte. Assim, o *ponto e vírgula* (;) determina um *formato fechado* para a colocação do valor da *variável* seguinte, X. Novamente a *pontuação, após* a *variável* X, é *irrelevante*, posto que vem a *função* TAB, provocando agora um *posicionamento* na *coluna* 50. Outra vez a *pontuação após* a *função* TAB é levada em consideração e o próximo valor, da *variável* Y, é colocado na zona seguinte. Exemplos de formas possíveis para a *função* TAB estão na figura 5-8.

FIGURA 5-8: *EXEMPLOS DE FUNÇÃO DE TABULAÇÃO* TAB

EXEMPLO	EXPLICAÇÃO
TAB (A + B – 4)	Será posicionada a saída na coluna cujo número é o valor inteiro da expressão entre parênteses, A + B – 4.
TAB (25)	A saída será posicionada na coluna número 25, que é o valor entre parênteses.

A forma geral da *função de tabulação da saída* é dada por

```
TAB ( < expressão aritmética > )
```

Deve-se observar qual a *pontuação* ou *delimitador* que se *segue* a uma *função* TAB, pois determinará a colocação do valor que se *segue, sem espaço* (;) ou na *próxima zona* (,).

Em um pequeno micro pessoal, um programa que enche a tela de asteriscos, como demonstração do uso da função TAB, é apresentado na figura 5-9.

FIGURA 5-9: *EXEMPLO DE PROGRAMA COM O USO DA FUNÇÃO* TAB

```
10  FOR    I = 0 TO 21
20  FOR    J = 0 TO 31
30  PRINT  TAB (J); "*";
40  NEXT   J
50  NEXT   I
```

Outra possibilidade para controlar a posição de exibição de um dado é o uso da palavra-chave AT (em alguns casos: @), na lista de saída da instrução **PRINT**. A palavra-chave AT, requer em geral, dois parâmetros. O primeiro parâmetro indica o número da linha e o segundo, o número da coluna para onde será movido o cursor, antes da exibição do dado seguinte. Em uma ou outra implementação, AT tem apenas um parâmetro, isto é, tem a mesma função que TAB, com a diferença apenas de o valor do parâmetro poder ser maior do que o número de caracteres de uma linha (exemplo: de 0 a 1023). Neste caso, linhas serão saltadas até que o número de caracteres saltados corresponda à posição especificada.

Um exemplo de *instrução de saída com posicionamento do cursor* é

```
PRINT X, AT 7, 5; Y, Z, AT 5, 10, W; T
```

.

Nesta instrução será dada saída ao valor de X, numa posição que depende da instrução **PRINT** anterior. Depois, na coluna 5 da linha 7, o valor de Y e, na mesma linha, no início da próxima zona, o valor de Z. Essa colocação é provocada por AT 7,5, seguido de ; (*ponto e vírgula*) e pela , (*vírgula*) entre Y e Z. O posicionamento na tela sofrerá um retrocesso, para a colocação, para a zona que se segue imediatamente à posição 10 da linha 5, do valor de W, seguido do valor de T, sem intervalo. Esse posicionamento é provocado por AT 5, 10 seguido de , (*vírgula*) e pelo ; (*ponto e vírgula*) entre W e T.

Exemplos de formas possíveis do uso da palavra reservada AT encontram-se na figura 5-10.

FIGURA 5-10: *EXEMPLOS DA PALAVRA RESERVADA* AT *PARA POSICIONAMENTO DO CURSOR*

EXEMPLO	EXPLICAÇÃO
AT J + 3, I – 2	O cursor será posicionado na coluna cujo número é o valor inteiro da expressão I – 2 da linha cujo número é o valor inteiro da expressão J + 3.
AT 3, J	O cursor será posicionado na coluna cujo número é o valor inteiro da variável J da linha cujo número é 3.

A forma geral da *palavra reservada da linguagem para posicionamento do cursor* é dada por

```
AT <expressão aritmética>, <expressão aritmética>
```

.

O *delimitador* que seguir os *parâmetros* da palavra reservada AT é relevante para a colocação do valor que se *segue, sem espaço* (;) ou *na próxima zona* (,).

Um exemplo do uso da palavra reservada AT em instrução **PRINT**, para produzir na tela o mesmo desenho (encher de asteriscos) que o programa da figura 5-9, é apresentado no programa da figura 5-11.

FIGURA 5-11: *EXEMPLO DE PROGRAMA COM USO DA PALAVRA RESERVADA* AT

```
10  FOR    I = 0 TO 21
20  FOR    J = 0 TO 31
30  PRINT  AT I, J; "*"
40  NEXT   J
50  NEXT   I
```

Uma modificação de ambos os programas para utilizar simultaneamente a função TAB e a palavra reservada AT, produzindo o mesmo desenho (encher a tela de asteriscos), é apresentada na figura 5-12.

FIGURA 5-12: *EXEMPLO DE PROGRAMA COM O USO DE* TAB *E* AT

```
10  FOR    I = 0 TO 21
20  PRINT  AT I, 0; "*";
30  FOR    J = 1 TO 31
40  PRINT  TAB (J); "*";
50  NEXT   J
60  NEXT   I
```

A exibição de gráficos na tela muitas vezes requer uma melhor definição da imagem através de um maior detalhamento. Em vez de usar a divisão da tela em caracteres, é melhor usar a divisão da tela em pixéis (*1/4* de caractere). Para tal, as instruções **PLOT** e **UNPLOT** são utilizadas. A instrução **PLOT** coloca uma marca (um quadradinho preto) em um pixel, enquanto a instrução **UNPLOT** retira a marca que tiver sido colocada, isto é, apaga a marca. A colocação ou retirada de marcas é feita com base no sistema de coordenadas cartesianas correspondente à divisão da tela em pixéis, conforme mostrado na figura 5-4. O eixo *X* é o das *abscissas*, tem a origem, isto é, a posição **Ø**, à esquerda da tela e vai até a posição 63, à direita, com um total de *64* posições. O eixo *Y* é o das *ordenadas*, tem a origem, isto é, a posição **Ø**, na parte inferior da tela e vai até a posição 43, na parte superior, com um total de *44* posições.

Além da função acima mencionada, a instrução **PLOT** (mas *não* **UNPLOT**) provoca sobre uma instrução **PRINT** que se seguir imediatamente um efeito que é o de alinhar os dados a partir da posição (caractere) imediatamente seguinte, na tela, àquela onde foi colocada a marca.

Exemplos da instrução **PLOT** são apresentados na figura 5-13.

FIGURA 5-13: *EXEMPLOS DA INSTRUÇÃO PARA COLOCAR MARCA EM UM PIXEL*

EXEMPLO	EXPLICAÇÃO
PLOT X + 2, Z – 1	É colocada uma marca (quadradinho preto) no pixel cuja abscissa é o valor inteiro da expressão X + 2 e cuja ordenada é o valor inteiro da expressão Z – 1.
PLOT 21, J	É colocada uma marca no pixel cuja abscissa é 21 e cuja ordenada é o valor inteiro da variável J.

A forma geral da *instrução para colocar marca em um pixel* é dada por

PLOT < *expressão aritmética* > , < *expressão aritmética* > .

Exemplos da instrução **UNPLOT** são apresentados na figura 5-14.

FIGURA 5-14: *EXEMPLOS DA INSTRUÇÃO PARA RETIRAR MARCA EM UM PIXEL*

EXEMPLO	EXPLICAÇÃO
UNPLOT X ** 2, X	É retirada (apagada) uma marca (quadradinho preto) no pixel cuja abscissa é o valor inteiro da expressão X **2 e cuja ordenada é o valor inteiro da variável X.
UNPLOT 5,7	É retirada uma marca no pixel cuja abscissa é 5 e cuja ordenada é 7.

A forma geral da *instrução para retirar marca em um pixel* é dada por

UNPLOT < *expressão aritmética* > , < *expressão aritmética* > .

Em alguns sistemas BASIC, as instruções correspondentes a **PLOT** e **UNPLOT** são, respectivamente, **SET** e **RESET**, devendo as coordenadas ser colocadas entre parênteses.

Um exemplo de programa com o uso da instrução **PLOT** para colocar uma constante no centro da tela, entre marcas, aparece na figura 5-15.

FIGURA 5-15: *EXEMPLO DE PROGRAMA COM O USO DA INSTRUÇÃO* **PLOT**

```
1Ø  PLOT   21,21
2Ø  PRINT  "ALO AMIGO"
3Ø  PLOT   4Ø,21
```

Normalmente, quando os dados são exibidos através da instrução **PRINT**, o preenchimento da tela é feito de *cima* para *baixo*. É possível alterar essa ordem, introduzindo novos dados pela parte *inferior* da tela (linha 21) e provocando o rolamento dos dados para cima, jogando fora da tela a linha que anteriormente ocupava a primeira linha (linha **Ø**), isto é, removendo a linha na posição *superior* da tela. Este efeito é provocado sobre a instrução **PRINT** pela execução anterior de uma instrução **SCROLL**.

Um exemplo da instrução **SCROLL** é apresentado na figura 5-16.

FIGURA 5-16: *EXEMPLO DA INSTRUÇÃO DE ROLAR TELA*

EXEMPLO	EXPLICAÇÃO
SCROLL	A próxima instrução **PRINT** exibirá os dados na linha inferior da tela, provocando o rolamento de todas as linhas para cima e a saída (remoção) da linha da parte superior da tela.

A forma geral da *instrução de rolamento de tela* é dada por

```
SCROLL
```

Um exemplo de programa que faz a exibição de uma mensagem na linha inferior da tela é apresentado na figura 5-17.

FIGURA 5-17: *EXEMPLO DE PROGRAMA COM O USO DA INSTRUÇÃO* **SCROLL**

```
10  SCROLL
20  PRINT   "DIGITE SEU NOME"
30  INPUT   X$
40  SCROLL
50  PRINT   "SEU NOME EH "; X$
```

Além das instruções apresentadas, todas destinadas à exibição de dados na tela, uma instrução que provoca a limpeza da tela é bastante útil. Essa instrução é **CLS**, mas também pode ser usada como comando. *Limpar* aqui é um termo usado como sinônimo de *apagar* todos os dados que se encontrem na tela.

Um exemplo da instrução **CLS**, para limpar a tela, é apresentado na figura 5-18.

FIGURA 5-18: *EXEMPLO DE INSTRUÇÃO PARA LIMPAR A TELA*

EXEMPLO	EXPLICAÇÃO
CLS	limpa (apaga todos os caracteres de) a tela

A forma geral da *instrução para limpar tela* é dada por

```
CLS
```

Um exemplo de um programa que usa a instrução **CLS** para limpar a tela é apresentado na figura 5-19.

FIGURA 5-19: *EXEMPLO DE PROGRAMA COM O USO DA INSTRUÇÃO* **CLS**

```
10  PRINT   "EM QUE ANO NASCEU?"
20  INPUT   A$
30  PRINT   "EM QUE MES NASCEU?"
40  INPUT   M$
50  PRINT   "EM QUE DIA NASCEU?"
60  INPUT   D$
70  CLS
80  PRINT   "VOCE NASCEU EM ";
90  PRINT   D$; "/"; M$; "/"; A$
```

5.4 Títulos e mensagens

Usando as instruções que foram apresentadas até aqui, é possível exibir na tela títulos e mensagens bastante variadas.

Um programa que coloca um título na parte superior da tela, entre duas linhas de asteriscos, é apresentado na figura 5-20.

FIGURA 5-20: *EXEMPLO DE PROGRAMA QUE EXIBE CABEÇALHO NA TELA*

```
10  FOR     I = 0 TO 31
20  PRINT   TAB (I); "*";
30  NEXT    I
40  PRINT   TAB (11); "CABECALHO"
50  FOR     J = 0 TO 31
60  PRINT   TAB (J); "*";
70  NEXT    J
```

Um programa que coloca uma mensagem no meio da tela, entre duas barras escuras, é apresentado na figura 5-21.

FIGURA 5-21: *EXEMPLO DE PROGRAMA QUE EXIBE MENSAGEM NA TELA*

```
10 FOR   I=24 TO 39
20 PLOT  I,23
30 NEXT  I
40 PRINT TAB(12);"MENSAGEM"
50 FOR   J=24 TO 39
60 PLOT  J,18
70 NEXT  J
```

Um programa que exibe um rodapé, na parte inferior da tela, entre duas linhas de barras duplas, é apresentado na figura 5-22.

FIGURA 5-22: *EXEMPLO DE PROGRAMA QUE EXIBE RODAPÉ NA TELA*

```
10 SCROLL          32 caracteres
20 PRINT   "================================"
30 SCROLL
40 PRINT   TAB(13);"RODAPÉ"
50 SCROLL          32 caracteres
60 PRINT   "================================"
```

5.5 Gráficos

A construção de gráficos de barras e histogramas fica facilitada pelo uso dos símbolos gráficos. A título de demonstração, um programa que exibe em linhas horizontais alguns símbolos gráficos é apresentado na figura 5-23.

FIGURA 5-23: *EXEMPLO DE PROGRAMA QUE EXIBE SÍMBOLOS GRÁFICOS EM LINHAS HORIZONTAIS*

```
10 FOR   J=0 TO 31
20 PRINT AT  1,J;"▗"
30 PRINT AT  6,J;"▚"
40 PRINT AT 11,J;"▓"
50 PRINT AT 16,J;"▒"
60 PRINT AT 21,J;"▀"
70 NEXT  J
```

Um programa que exibe alguns símbolos gráficos, em linhas verticais, é apresentado na figura 5-24.

FIGURA 5-24: *EXEMPLO DE PROGRAMA QUE EXIBE SÍMBOLOS GRÁFICOS EM LINHAS VERTICAIS*

```
10 FOR   I=0 TO 21
20 PRINT AT I,1;  "▙"
30 PRINT AT I,7;  "▚"
40 PRINT AT I,13; "▜"
50 PRINT AT I,19; "▒"
60 PRINT AT I,25  "▒"
70 PRINT AT I,31; "▀"
80 NEXT  I
```

O desenho de figuras e curvas pode ser mais bem realizado na tela com a instrução **PLOT**, que coloca uma marca (quadradinho preto) na tela, em um pixel (*1/4* de caractere).

Um exemplo de programa que desenha um quadrado na tela é apresentado na figura 5-25.

FIGURA 5-25: *EXEMPLO DE PROGRAMA PARA DESENHAR UM QUADRADO*

```
10 FOR   K=0 TO 43
20 PLOT  K+10,0
30 PLOT  K+10,43
40 PLOT  10,K
50 PLOT  53,K
60 NEXT  K
```

Podem-se acrescentar as diagonais do quadrado no desenho, incluindo, no programa da figura 5-25, o trecho de programa que aparece na figura 5-26.

FIGURA 5-26: *ACRÉSCIMO AO EXEMPLO DE PROGRAMA PARA TRAÇAR DIAGONAIS DE QUADRADO*

```
60 PLOT  K+10,K
70 PLOT  K+10,43-K
80 NEXT  K
```

O desenho de outras curvas requer muitas vezes o uso de funções predefinidas da linguagem BASIC, que serão apresentadas posteriormente, no próximo capítulo. Todavia, será antecipado o uso da função que calcula a raiz

quadrada de um número, SQR, em um exemplo de programa que desenha uma circunferência na tela, apresentado na figura 5-27.

FIGURA 5-27: *EXEMPLO DE PROGRAMA PARA DESENHAR UMA CIRCUNFERÊNCIA*

```
10  FOR   K=0 TO 42
20  LET   H=SQR (42*K+0.1-K**2)
30  PLOT  K+10,21-H
40  PLOT  K+10,21+H
50  NEXT  K
```

5.6 O uso do tempo e do movimento

Existe uma instrução que permite paralisar temporariamente a execução do programa. Com isso, é possível retardar a apresentação de dados na tela do vídeo. Essa instrução é **PAUSE**. Exemplos da instrução **PAUSE** são apresentados na figura 5-28.

FIGURA 5-28: *EXEMPLOS DA INSTRUÇÃO DE PAUSA PROGRAMADA*

EXEMPLO	EXPLICAÇÃO
PAUSE K ** 2 – 327	A execução do programa é paralisada por um tempo correspondente a (K ** 2 – 327)/60 seg.
PAUSE 60	A execução do programa é paralisada por um tempo correspondente a 60/60 seg, isto é, por 1 segundo.

A forma geral da *instrução de pausa programada* é dada por

PAUSE < *expressão aritmética* >

O valor da *expressão aritmética* tem que estar dentro do intervalo de 0 a 32767 (9 minutos). É possível interromper qualquer pausa, isto é, fazer o programa continuar a execução, tocando-se qualquer tecla. O tempo de paralisação do programa é uma fração de segundos em que o numerador é o valor da *expressão aritmética* e o denominador é sempre a constante 60.

A pausa programada é usada para provocar a exibição de dados na tela com um certo intervalo de tempo entre eles. Um exemplo simples é apresentado na figura 5-29.

FIGURA 5-29: *EXEMPLO DE PROGRAMA COM PAUSAS PROGRAMADAS DE EXIBIÇÃO*

```
10  PRINT  AT 11,11;"ALO AMIGO"
20  PAUSE  180
30  CLS
40  PRINT  AT 11,11;"TUDO BEM?"
50  PAUSE  180
60  CLS
70  PRINT  AT 11,13;"TCHAU"
80  PAUSE  180
90  CLS
```

A pausa programada pode servir também para dar um certo ritmo à apresentação de dados na tela, como mostra o exemplo de programa da figura 5-30, que repete um nome em cada linha da tela, com um intervalo de tempo de 3 segundos.

FIGURA 5-30: *EXEMPLO DE PROGRAMA COM EXIBIÇÃO PAUSADA DE DADOS*

```
10  PRINT  "DIGITE SEU NOME"
20  INPUT  A$
30  CLS
40  FOR    I=0 TO 21
50  PRINT  "SEU NOME EH ";A$
60  PAUSE  180
70  NEXT   I
```

A impressão de movimento pode ser causada pela apresentação de um dado em sucessivas posições da tela. Uma pequena modificação no programa da figura 5-30 e resulta o programa da figura 5-31, que faz o deslocamento do nome verticalmente na tela, a cada 3 segundos.

FIGURA 5-31: *EXEMPLO DE PROGRAMA QUE MOVIMENTA DADO NA TELA*

```
10  PRINT  "DIGITE SEU NOME"
20  INPUT  A$
```

```
30  CLS
40  FOR   I = 0 TO 21
50  PRINT AT I, 0 ; "SEU NOME EH "; A$
60  PAUSE 180
70  CLS
80  NEXT  I
```

O leitor observará, ao executar os exemplos apresentados até aqui, que os gráficos ou desenhos, nos menores micros, são apresentados na tela já completamente desenvolvidos, isto é, de uma forma estática. Pode ser desejável acompanhar o desenvolvimento de um desenho ou gráfico dinamicamente.

Pode-se acompanhar o desenvolvimento do traçado gráfico da circunferência, por exemplo, introduzindo-se uma instrução **PAUSE 60** no programa da figura 5-27, que ficará como mostra a figura 5-32.

FIGURA 5-32: *EXEMPLO (MODIFICADO) DE PROGRAMA PARA DESENHAR CIRCUNFERÊNCIA NA TELA*

```
10  FOR   K = 0 TO 42
20  LET   H = SQR (42 * K + 0.1 - K ** 2)
30  PLOT  K + 10, 21 - H
40  PLOT  K + 10, 21 + H
50  PAUSE 60
60  NEXT  K
```

A impressão de movimento, em gráficos, pode ser provocada apagando-se da tela marcas anteriormente exibidas em um pixel antes de exibir as seguintes. O programa da figura 5-32 pode ser alterado para exibir o movimento simétrico de duas marcas que obedecem ao traçado das semicircunferências inferior e superior, como na figura 5-33.

FIGURA 5-33: *EXEMPLO DE PROGRAMA PARA MOVIMENTAR MARCAS (PIXEL) NA TELA*

```
10  FOR   K = 0 TO 42
20  LET   H = SQR (42 * K + 0.1 - K ** 2)
30  LET   I = K + 10
40  PLOT  I, 21 - H
50  PLOT  I, 21 + H
60  PAUSE 60
70  UNPLOT I, 21 - H
```



```
80  UNPLOT I, 21 + H
90  NEXT  K
```

Exercícios:

ESTUDO DIRIGIDO:

1) Como se divide a tela (quadro) de vídeo de um pequeno microcomputador? Para que serve a linha de comunicação? Para que serve a parte útil da tela? Quantas linhas e quantas colunas tem a parte útil da tela? Qual o número total de caracteres da tela (parte útil)? Como são numeradas as linhas? Como são numeradas as colunas? Que é cursor? Que é código de reportagem? Como pode ser dividida a tela, em *pixéis*? Qual o tamanho de um *pixel*? Como são numeradas as 1/2s linhas? Como são numeradas as 1/2s colunas? Quais as instruções que permitem ter acesso ao *pixel*?

2) Quantos são os caracteres gráficos? Quais os símbolos gráficos adicionais que podem ser considerados? Que é modo gráfico? Como se entra e sai do modo gráfico? Como obter os símbolos gráficos no teclado?

3) Qual a instrução padrão para o uso da tela? Quais os símbolos correntemente usados para formatar dados? Qual a função da tabulação? Para que serve a função TAB? Qual a forma geral de TAB? Como posicionar o cursor da tela em uma linha e coluna determinadas? Qual o efeito da palavra reservada AT? Qual a forma geral de AT? Pode-se usar simultaneamente AT e TAB? Qual a instrução que coloca uma marca em um *pixel*, na tela? Qual a instrução que retira (apaga) uma marca em um *pixel*, na tela? Qual a forma geral de **PLOT**? Qual a forma geral de **UNPLOT**? Como são referenciadas as coordenadas das instruções **PLOT** e **UNPLOT**, em relação à tela? Qual o efeito de **PLOT** sobre a instrução **PRINT** seguinte? Para que serve a instrução **SCROLL**? Para que serve a instrução **CLS**? **CLS** pode ser usada como comando?

4) Quais as instruções que podem ser usadas para colocar cabeçalhos na tela? Quais as instruções que você usaria para colocar mensagens no centro ou outra qualquer parte da tela? Quais as instruções que você usaria para colocar um rodapé na tela?

5) Que símbolos podem ser usados para construir gráficos de barras e histogramas? Quais as instruções que você usaria para traçar gráficos? Como fazer desenhos e traçar curvas na tela? Qual a instrução mais adequada para fazer desenhos e mostrar a forma de funções na tela?

6) Qual a instrução que permite a parada do programa temporariamente? Qual o significado do parâmetro (expressão aritmética) da instrução **PAUSE**? De que maneiras pode ser usada a instrução **PAUSE** para retardar a apresentação de dados na tela? Como obter o efeito de movimento de um dado na tela? Como mostrar o desenvolvimento de um gráfico, dinamicamente, na tela? Como mostrar o movimento de marcas (*pixéis*) na tela?

PROBLEMAS:

1) Uma linha de *32* caracteres pode ser mostrada no vídeo pelo programa

```
10 PRINT "01234567890123456789012345678901"
```

Execute o programa e observe. Reformule o programa para repetir a mesma marcação em todas as linhas da tela. Execute novamente.

2) Uma linha que repete um único tipo de caractere gráfico é exibida pelo programa

```
                    32 caracteres gráficos
10 PRINT "■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
```

Execute o programa e observe. Reformule o programa para preencher cada uma das 22 linhas da tela com um caractere gráfico diferente. Reexecute.

3) Um programa que usa o posicionamento do cursor para exibição de uma linha com um único caractere gráfico é

```
10 FOR  J=0 TO 31
20 PRINT AT 1,J;"■"
30 NEXT J
```

Execute-o e observe. Inclua no programa instruções para exibir um diferente caractere gráfico em cada uma das 22 linhas. Reexecute.

4) Um programa que exibe um caractere gráfico em uma coluna no centro da tela é

```
10 FOR  I=0 TO 21
20 PRINT TAB (21);"■"
30 NEXT I
```

Execute-o e observe. Altere a instrução número 20 para imprimir outros caracteres gráficos, por exemplo, nas posições (colunas) 1, 5, 9, 13, 17, 21, 25, 29 ou, ainda, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31. Reexecute.

5) Escreva um programa que, utilizando as instruções **PLOT** e **PRINT**, coloque no centro da tela a constante literal MENSAGEM, entre marcas de um *pixel*. Execute-o. Observe a posição das marcas em relação à constante literal.

6) Escreva um programa que solicite, respectivamente, nesta ordem, o ano, o mês e o dia do nascimento de uma pessoa, através de mensagens na parte *inferior* da tela. Após ter lido todas as informações, limpe (apague) a tela e coloque, também na parte *inferior* da tela, a mensagem A DATA DO SEU NASCIMENTO EH e a data, na forma *dia/mês/ano*, entre linhas de asteriscos.

7) Escreva um programa que leia seis valores numéricos e, tomando por base o maior, reduza-os a relações que possam ser representadas na tela, em gráficos de barras horizontais espaçadas convenientemente. À maior das relações, que é *1*, deverá corresponder um número de colunas que não pode ser superior ao número máximo (*32* colunas nos menores micros).

8) A instrução de desvio condicional **IF** . . . THEN permite a execução alternativa de instruções **PRINT**. Escreva um programa que alterne dois símbolos gráficos, por exemplo, *branco* e *preto*, para formar um tabuleiro na tela. A decisão sobre que instrução **PRINT** executar será tomada na instrução **IF**, com base, por exemplo, no valor de uma variável K que é inicializada com o valor 1 e alterna este valor com −1 (troca de sinal), dentro de uma malha de repetição. O problema sugere o uso de duas malhas de repetição embutidas (uma para as linhas e outra para as colunas).

9) Escreva um programa que desenhe um retângulo na tela, usando como símbolo gráfico a marca de um *pixel*. O retângulo deverá ter as dimensões da tela.

10) Escreva um programa que, usando o símbolo gráfico da marca de um *pixel*, desenhe uma elipse cujo eixo maior seja a 1/2 linha central e cujo eixo menor seja a 1/2 coluna central. A equação da elipse, quando tem centro sobre o eixo das abscissas e passa pela origem, é:

$$a^2y^2 + b^2(x - a)^2 = a^2b^2$$

11) Com o uso da instrução **PAUSE**, escreva um programa que leia o nome de uma pessoa e movimente este nome na vertical, linha por linha, *de baixo para cima*, na tela.

12) Com o uso da instrução **PAUSE**, altere um dos programas para desenho de gráficos (exercício 9 ou 10) de tal maneira que o desenvolvimento da figura possa ser acompanhado na tela.

SEXTO CAPÍTULO
CÁLCULOS MATEMÁTICOS

cálculos com funções matemáticas – variáveis coletivas (coleções) – operações sobre vetores e matrizes – instruções de declaração e de aquisição de dados

6.1 Cálculos com funções matemáticas

Para simplificar a programação, evitando que o usuário tenha que repetir as mesmas *rotinas*, isto é, textos de programa de uso muito freqüente, foi criado o conceito ou facilidade para uso de *função*. Algumas implementações de BASIC permitem ao usuário escrever suas próprias *funções*, como será visto em capítulo posterior. Todavia, para poupar tempo e trabalho de o usuário escrever *funções* já bastante conhecidas, o sistema possui e oferece diversas *funções predefinidas*. Dentre estas, algumas são básicas, estando disponíveis praticamente em todos os sistemas BASIC e são as constantes do quadro 6-1.

QUADRO 6-1: *FUNÇÕES PREDEFINIDAS BÁSICAS DA LINGUAGEM BASIC*

TIPO	NOME	ARGUMENTO	RESULTADO
ARITMÉTICAS	INT	numérico	maior inteiro igual ou menor que o argumento, com sinal
	ABS	numérico	valor absoluto
	SGN	numérico	retoma: -1 se argumento < 0 (negativo) 0 se argumento $= 0$ (nulo) $+1$ se argumento > 0 (positivo)
	RND	numérico	valor pseudo-aleatório entre 0 e 1

	FRA	numérico	parte fracionária, com sinal
	PI	— — —	valor de π
TRANSCENDENTAIS	SQR	numérico positivo	raiz quadrada
	EXP	numérico	potência de base neperiana (e) e expoente igual ao argumento
	LOG	numérico	logaritmo natural (base neperiana: e)
TRIGONOMÉTRICAS	SIN	numérico em radianos	seno
	COS	numérico em radianos	co-seno
	TAN	numérico em radianos	tangente
	ATN	numérico	arcotangente

O *parâmetro* ou *argumento* das *funções* (que possuem argumento) é uma *expressão numérica* qualquer, cujo valor deve ser coerente com a função. A função PI *não* tem argumento, em geral.

As *funções* podem ser *referenciadas* em *expressões numéricas* da mesma forma que as *constantes* e *variáveis numéricas*, bastando figurar apenas o seu *nome* acompanhado do *argumento*.

A forma geral de uma *referência a uma função numérica predefinida* é a seguinte:

< *nome de função* > [(< *expressão* >)]

Uma demonstração do uso de *funções*, em particular as *trigonométricas*, é feita no programa apresentado na figura 6-1.

FIGURA 6-1: *PROGRAMA EXEMPLO DO USO DE FUNÇÕES*

```
10 REM   DEMONSTRACAO DE FUNCOES (TRIGONOMETRICAS)
20 PRINT "CALCULO DE FUNCOES TRIGONOMETRICAS"
30 PRINT
40 INPUT "ANGULO EM GRAUS", A
50 PRINT "ANGULO" , "SIN" , "COS" , "TAN" , "ATN"
60 PRINT A , SIN (A * PI/180) , COS (A * PI/180) ,
70 PRINT TAN (A * PI/180) , ATN (TAN (A * PI/180)) * 180/PI
80 PRINT "TERMINO DO PROGRAMA"
```

O programa apresentado na figura 6-1 imprime um cabeçalho, solicita o fornecimento de um ângulo em graus, lê o ângulo e imprime os valores das funções trigonométricas relativas a este ângulo, sob títulos. Nos menores microcomputadores será necessário desdobrar a instrução na linha 40 em uma instrução para imprimir a mensagem e outra para ler o ângulo.

O efeito do uso da função INT pode ser avaliado em um exemplo simples, que é apresentado na figura 6-2.

FIGURA 6-2: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* INT

```
10 FOR   L = -36.543 TO 17.946 STEP 2.723
20 PRINT "VALOR="; L, "INT="; INT (L)
30 NEXT  L
```

A função INT deve ser usada sempre que se deseja o máximo valor inteiro que seja menor do que o valor de uma expressão aritmética. O uso de INT é útil, por exemplo, na construção de gráficos, como mostra o programa da figura 6-3.

FIGURA 6-3: *EXEMPLO DE PROGRAMA DE GRÁFICO COM APLICAÇÃO DA FUNÇÃO* INT

```
10 PRINT "GRAFICO COMPARATIVO"
20 PRINT "TRES VALORES DE 0 A 1000"
30 INPUT A
40 INPUT B
50 INPUT C
60 FOR   I = INT ( 22 - ((A/1000) * 22)) TO 21
70 PRINT AT I, 7; "■"
80 NEXT  I
```

```
90  FOR   J = INT (22 – (( B/1000) * 22)) TO 21
100 PRINT AT J, 15; "▯"
110 NEXT  J
120 FOR   K = INT (22 – ((C/1000) * 22)) TO 21
130 PRINT AT K, 23; "▯"
140 NEXT  K
```

O efeito do uso da função ABS pode ser percebido com o exemplo de programa da figura 6-4.

FIGURA 6-4: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* ABS

```
10 FOR   I = –17.6358 TO 13.5487 STEP 1.4785
20 PRINT "VALOR="; I , "ABS="; ABS (I)
30 NEXT  I
```

Uma aplicação da função ABS é, por exemplo, a comparação entre dois valores, para dizer se são iguais, a menos de uma diferença. Como não é possível saber que sinal terá a diferença, qualquer comparação deve ser feita em valor absoluto. Um exemplo se encontra no programa da figura 6-5.

FIGURA 6-5: *EXEMPLO DE APLICAÇÃO DA FUNÇÃO* ABS

```
10 PRINT "DIGITE DOIS VALORES"
20 INPUT A
30 INPUT B
40 IF    (ABS (A – B) > 0.0001) THEN GOTO 70
50 PRINT "VALORES IGUAIS"
60 GOTO  80
70 PRINT "VALORES DESIGUAIS"
80 PRINT A ; " E " ; B
```

A função SGN é usada para extrair o sinal de um número para que possa ser aplicado a outro. Um programa que mostra o que ocorre com o uso da função SGN é apresentado na figura 6-6.

FIGURA 6-6: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* SGN

```
10 FOR   J = –7.127464 TO 5.38217 STEP 0.635629
20 PRINT "VALOR="; J , "SGN="; SGN ( J )
30 NEXT  J
```


A função RND é usada para gerar números pseudo-aleatórios entre 0 e 1. Pode ter parâmetro ou não, dependendo da implementação. Nos menores micros, não tem parâmetro. Um programa que mostra uma série de valores pseudo-aleatórios gerados pela função RND encontra-se na figura 6-7.

FIGURA 6-7: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* RND

```
10 FOR   M = 1 TO 22
20 PRINT "VALOR ("; M; ")=" ; RND
30 NEXT  M
```

Se o leitor anotar a série de valores gerados pela função RND, verificará, em cada nova seção em que o programa for executado (uma ou mais vezes), que eles se repetem. Pelo fato de se repetirem, a cada seção, são *pseudo*-aleatórios e não aleatórios. Pode ser desejável evitar a repetição da mesma série de números, em cada seção em que o programa for executado. Em geral, a repetição da série ocorre porque a função RND parte sempre da mesma *raiz*. Na fase de *depuração*, quando erros de programa estão sendo eliminados, é interessante repetir sempre a mesma série. Para evitar a repetição existe uma *instrução de alteração da raiz* que é

```
RANDOMIZE
```

,

que significa *randomizar* ou mudar *aleatoriamente* o valor da *raiz*. A *instrução* **RANDOMIZE** deverá ser executada *antes* que a *função* RND seja executada pela *primeira vez*. Alguns sistemas BASIC *usam* a *palavra-chave* **RANDOM** ou **RAND**, enquanto outros as *permitem* como alternativa, em vez de **RANDOMIZE**.

A forma geral da instrução de alteração randômica da raiz da função **RND** é dada por

```
{RANDOMIZE | RANDOM | RAND}
```

.

Um exemplo do uso da *instrução de randomização* é dado na figura 6-8.

FIGURA 6-8: *EXEMPLO DE USO DA INSTRUÇÃO DE RANDOMIZAÇÃO*

```
10  RAND
20  FOR   I=1 TO 5
30  PRINT (RND * 5) + 5
40  NEXT  I
```

O programa da figura 6-8, em cada seção em que é executado, produz séries diferentes de *5* valores no intervalo *5* a *10*. Em geral, para produzir valores e em um determinado *intervalo* de valores entre A e B, pode-se usar uma expressão da forma

```
( RND * (B – A ) ) + A
```

.

Se o seu microcomputador dispuser da função FRA, experimente o programa da figura 6-9, para obter a parte fracionária de números, com sinal.

FIGURA 6-9: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* FRA

```
10  FOR   N=3.128467 TO –3.93728 STEP –0.30367
20  PRINT "VALOR="; N, "FRA="; FRA (N)
30  NEXT  N
```

Uma forma de aferir a precisão da função PI do seu microcomputador é executar um programa que calcula a tangente e o arcotangente de vários arcos de círculo congruentes (múltiplos) com um ângulo inicial. Um exemplo encontra-se na figura 6-10.

FIGURA 6-10: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* PI

```
10  PRINT "ANGULO EM GRAUS"
20  INPUT A
30  CLS
40  FOR   N=0 TO 21
50  LET   A1 = (2 * PI * N) + ( A * PI / 180 )
60  LET   A2 = ATN ( TAN ( A1 ))
70  PRINT "ANG="; A1 , "ATN = " ; A2
80  NEXT  N
```


A função SQR, raiz quadrada, é uma das mais utilizadas. Não é possível calcular a raiz quadrada de números negativos. Um programa que mostra qual a precisão com que o seu microcomputador faz esse cálculo é dado na figura 6-11.

FIGURA 6-11: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* SQR

```
10  FOR   P=0 TO 763.65494 STEP 34.7259
20  PRINT "NUMERO="; P, "SQR ** 2"; SQR (P) * SQR (P)
30  NEXT  P
```

Um exemplo da aplicação da função de cálculo da raiz quadrada está no problema do cálculo das raízes do trinômio do 2º grau, resolvido no programa da figura 6-12.

FIGURA 6-12: *EXEMPLO DE PROGRAMA APLICANDO A FUNÇÃO* SQR

```
10  PRINT "CALCULO DAS RAIZES DO"
20  PRINT "TRINOMIO DO SEGUNDO GRAU"
30  PRINT "A=";
40  INPUT A
50  PRINT A
60  IF    A=0 THEN GOTO 190
70  PRINT "B=";
80  INPUT B
90  PRINT B
100 PRINT "C=";
110 INPUT C
120 PRINT C
130 LET   D=B ** 2 – 4 * A * C
140 IF    D<0 THEN GOTO 210
150 LET   P1 = –B / (2 * A)
160 LET   P2 = SQR ( D ) / ( 2 * A )
170 PRINT "X1="; P1 + P2 , 'X2='; P1 – P2
180 GOTO  220
190 PRINT "NAO HA TRINOMIO"
200 GOTO  220
210 PRINT "RAIZES IMAGINARIAS"
220 PRINT "TERMINO DO PROGRAMA"
```

Uma forma de verificar a precisão do seu microcomputador, com referência ao cálculo das funções EXP e LOG (nos pequenos micros: LN) é executar o programa da figura 6-13.

FIGURA 6-13: *PROGRAMA DE DEMONSTRAÇÃO DAS FUNÇÕES* EXP *E* LOG *(OU* LN*)*

```
10  FOR     Q = 36.92748 TO 0.082387 STEP -1.78936
20  PRINT   "VALOR="; Q, "EXP (LN) ="; EXP (LN (Q))
30  NEXT    Q
```

As funções trigonométricas são úteis em problemas geométricos, como é o caso de resolução de triângulos. Um programa que mostra com que precisão o seu microcomputador calcula as funções trigonométricas é apresentado na figura 6-14.

FIGURA 6-14: *PROGRAMA DE DEMONSTRAÇÃO DE FUNÇÕES TRIGONOMÉTRICAS*

```
10  FOR     R = 0 TO 2 * PI STEP PI/4 + 1.0000001
20  PRINT   "ANG="; R, "SIN="; SIN (R)
30  PRINT   "COS="; COS (R), "TAN="; TAN (R)
40  NEXT    R
```

6.2 Variáveis coletivas (coleções)

As *variáveis* podem ser *simples* ou *coletivas*. Enquanto uma *variável simples* só pode representar a cada instante um *único* valor, uma *variável coletiva* ou *conjunto* pode representar uma *coleção* de valores todos do *mesmo tipo*. Essa coleção é *ordenada*, isto é, é possível atribuir a cada elemento um *número de ordem* ou *posição relativa* dentro do conjunto. Atribuindo-se à *variável conjunto* um *nome*, é possível *referenciar* cada *elemento* do *conjunto* em particular através de um *índice* ou *subscrito* que representa seu *número de ordem*. Este *subscrito* vem *entre parênteses* após o *nome do conjunto*. O *nome* de um *coletivo* se aplica tanto ao *conjunto* como a cada *elemento*. Se o *nome* está desacompanhado do *subscrito*, refere-se ao *conjunto* como um *todo*.

Exemplos de *nomes* válidos *como nomes de conjuntos* são:

```
D          F3          T1$          X$
```

.

Pode-se observar que estes *nomes* poderiam ser atribuídos também a *variáveis simples, numéricas* ou *alfanuméricas*. Conclui-se que as *regras* para atribuir *nomes* a *conjuntos* são as *mesmas* que para as *variáveis simples*, sejam *numéricas* ou *alfanuméricas*. Desta maneira, são *nomes de conjuntos numéricos* os *nomes*

```
D          F3
```

,

enquanto são *nomes de conjuntos alfanuméricos* os *nomes*

```
                                T1$          X$
```

.

São exemplos de *referências* a *elementos de conjuntos* as seguintes *variáveis subscritas*

```
D (4 , 3)    F (J + 7)    T1$ (X ↑ 3 , Y)    X$ (A2 – 1.73)
```

.

Da mesma forma que os *conjuntos* correspondentes, são valores *numéricos* os *elementos* representados pelas *variáveis subscritas*

```
D (4 , 3)    F (J + 7)
```

,

enquanto são valores *alfanuméricos* os *elementos* representados pelas *variáveis subscritas*

```
                                T1$ (X ↑ 3 , Y)    X$ (A2 – 1.73)
```

.

Toda *referência* a *elemento de conjunto* é uma *variável subscrita* por ser constituída do *nome do conjunto* acompanhado do *subscrito* ou *índice*. Por oposição, as *variáveis simples* são chamadas de *variáveis não-subscritas*.

O *índice* ou *subscrito* é constituído por qualquer *expressão numérica positiva* de cujo cálculo resulta um valor que, truncado, fornece o *número de ordem* ou *posição relativa* do *elemento* dentro do *conjunto*.

Existem sistemas BASIC que admitem *conjuntos* com até *seis dimensões*. Todavia, o *usual* é que um sistema BASIC admita *conjuntos* de até *duas dimensões*. Um *conjunto* de *uma dimensão* é chamado de *vetor* e um *conjunto* de *duas dimensões* é chamado de *matriz*, por analogia com a matemática.

São exemplos de *referências* a *conjuntos* de *uma dimensão, unidimensionais* ou *vetores* as *variáveis subscritas*

```
F (J + 7)                    X$ (A2 – 1.73)
```

,

porque possuem um *único subscrito* ou *índice*, enquanto são exemplos de *referências* a *conjuntos* de *duas dimensões, bidimensionais* ou *matrizes* as *variáveis subscritas*

```
D (4 , 3)                    T1$ (X ↑ 3 , Y)
```

,

porque possuem *dois índices* ou *subscritos*.

A forma geral de uma *referência a elemento de conjunto*, isto é, de uma *variável subscrita é*:

```
< nome > ( < expressão aritmética > [ , < expressão aritmética > ] )
```

As *variáveis coletivas* ou *conjuntos* devem ser *declaradas* em uma *instrução* **DIM**, onde são especificados o *número de dimensões* e o *número máximo* de *elementos* por *dimensão*. Caso uma *variável coletiva não* seja *declarada*, o sistema BASIC adotará como *número de dimensões* o *número de subscritos* e como *número máximo de elementos* por *dimensão* um valor *padrão* que, na falta de especificação, é *10*, praticamente na totalidade das implementações existentes. Portanto, se houver um *único subscrito*, o *conjunto* será um *vetor* com *10 elementos*. Se o número de *subscritos* for de *2*, haverá *10 elementos* por *dimensão*, o que dá um total de *100 elementos* para toda a *matriz*.

A *instrução* **DIM** serve para *declarar variáveis simples*, sejam *numéricas* ou *alfanuméricas*, e *conjuntos*, sejam *numéricos* ou *alfanuméricos*.

A *palavra-chave* **DIM** da linguagem BASIC é uma forma abreviada de **DIMENSION**, ou seja, *dimensão*.

Deve-se observar que o uso da *instrução* **DIM** para *dimensionar conjuntos alfanuméricos* é mais *raro*, isto é, mesmo em implementações da linguagem estendida ou ampliada, esta forma é pouco usual. Todavia, existem alguns poucos exemplos. Para obter maior generalidade e compatibilidade deve-se *evitar* o uso de *variáveis dimensionadas* do tipo *alfanumérico*.

Exemplo do uso da instrução de *dimensionar* ou *declarar variáveis* é

```
DIM U3 , P$ (40) , C (50) , F6 (2 , 20)
```

.

No exemplo apresentado são *declaradas* a *variável numérica simples* U3, a *variável alfanumérica simples* **P$**, com um *tamanho máximo* de **40** caracteres, o *vetor* **C**, com *uma* dimensão e o *máximo* de **50** elementos, e a *matriz* **F6**, com *duas* dimensões, a primeira com o *máximo* de **2** elementos e a segunda com o *máximo* de **20** elementos, ou seja, com um total *máximo* de **40** elementos.

A forma geral da *instrução de declarar ou dimensionar variáveis* é:

```
DIM < nome > [ ( < expressão aritmética > [ , < expressão aritmética > ])]
  [ < nome > [ ( < expressão aritmética > [ , < expressão aritmética > ] ) ] ] ...
```

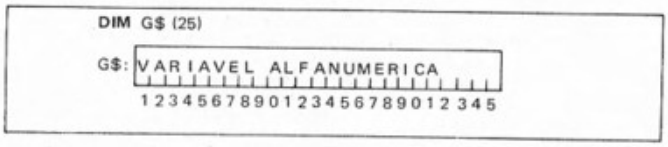
Esta forma só permite *declarar conjuntos numéricos* e *variáveis alfanuméricas simples*.

Tanto o *tamanho máximo* de cada *dimensão* de *vetores* e *matrizes* como o *número máximo de caracteres* de *variáveis alfanuméricas* são *expressões numéricas positivas* quaisquer. A *instrução* **DIM** pode estar colocada em qualquer parte do programa, pois é uma *instrução de especificação*, isto é, *não-executável*. Cada *variável* que for *declarada* só pode aparecer uma *única* vez em uma *instrução* **DIM**, em todo o programa. Pode haver um número qualquer de *instruções* **DIM** no programa, isto é, a *declaração* de *variáveis é completamente arbitrária*.

Ao ser iniciada a execução do programa, o sistema BASIC *automaticamente* inicializa as *variáveis* e *conjuntos*, isto é, coloca *zero* (**0**) em *variáveis simples* e *elementos de conjuntos numéricos* e coloca *vazio* (*limpa*) em *variáveis alfanuméricas simples* e *elementos* de *conjuntos alfanuméricos*.

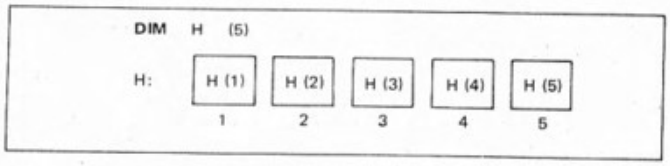
Uma *variável alfanumérica* pode ser vista como uma *cadeia de caracteres*, isto é, uma seqüência de *posições* ordenadas, cada qual contendo a representação de um *caractere*. Essas *posições*, se o *número máximo de caracteres* é *n*, são numeradas de *1* a *n*. Uma *variável alfanumérica* **G$** que tenha sido *declarada* como tendo no *máximo* **25** caracteres e que contenha o valor *alfanumérico* VARIAVEL ALFANUMERICA é o exemplo mostrado na figura 6-15.

FIGURA 6-15: *EXEMPLO DE VARIÁVEL ALFANUMÉRICA SIMPLES*



Um *vetor* pode ser visto como uma sucessão de *posições* de memória ordenada, *cada* uma das quais corresponde a um *elemento* do *conjunto*. Essas *posições* de memória correspondem à ordem crescente dos valores dos *índices* ou *subscritos*. Essas *posições*, se o *número máximo de elementos* ou *tamanho* é *n*, são numeradas de *1* a *n*. Por exemplo, uma *variável* H que tenha sido *declarada* como um *conjunto unidimensional* ou *vetor*, com o *número máximo de elementos* de 5 é mostrada na figura 6-16.

FIGURA 6-16: *EXEMPLO DE VETOR*



O *número máximo de elementos* de um *vetor* varia, de sistema para sistema, entre *255* e *32.767*.

Uma *matriz* corresponde a uma *tabela de dupla entrada* onde cada *elemento* pode ser localizado pelo *número da linha* combinado com o *número da coluna*. O *número de elementos* de uma *matriz* é dado pelo produto

número de linhas x número de colunas .

O *número da linha* corresponde ao *primeiro subscrito*, enquanto o *número da coluna* corresponde ao *segundo subscrito*, da *matriz*. Por exemplo, uma *variável* I7 que tenha sido *declarada* como um *conjunto bidimensional* ou *matriz*, com no *máximo* 3 *elementos* na *primeira dimensão* e 4 *elementos* na *segunda dimensão*, é mostrada na figura 6-17.

FIGURA 6-17: *EXEMPLO DE MATRIZ*

DIM I7 (3 , 4)

I7:	I7 (1,1)	I7 (1,2)	I7 (1,3)	I7 (1,4)
	I7 (2,1)	I7 (2,2)	I7 (2,3)	I7 (2,4)
	I7 (3,1)	I7 (3,2)	I7 (3,3)	I7 (3,4)

As *matrizes*, em geral, são armazenadas na memória *por linhas* da *matriz*, isto é, crescendo *mais rapidamente* o *índice* ou *subscrito mais à direita*. O *número máximo de elementos* de uma *matriz* está limitado, em princípio, pela memória disponível. O *limite máximo de elementos* fixado varia de *255* a *32.767 por subscrito*, isto é, em *cada dimensão*, o que resulta em um *número total de elementos* bem elevado.

6.3 Operações sobre vetores e matrizes

A principal maneira de *manipular conjuntos* é com o uso de *instruções de repetição controlada*, **FOR** e **NEXT**.

No caso de *vetores*, isto é, *conjuntos* de uma *única dimensão*, basta *um único nível* de *malha de repetição controlada*, isto é, apenas *um par* de instruções **FOR** e **NEXT**, como no exemplo da figura 6-18.

FIGURA 6-18: *EXEMPLO DE PROGRAMA COM OPERAÇÕES SOBRE VETORES*

```
10 DIM   G (10)
20 FOR   L = 1 TO 10
30 INPUT G (L)
40 NEXT  L
50 FOR   L = 1 TO 10
60 PRINT G (L)
70 NEXT  L
```

No programa da figura 6-18, dentro de uma *malha de repetição controlada*, é feita a *leitura de dados* para cada um dos elementos do *vetor* G, em número de 10. Da mesma forma, em outra *malha*, é feita a *saída*, em formato fechado, de cada um dos elementos do *vetor* G, em número de 10. Em seguida, o programa *termina*.

Se se pretendem realizar operações sobre *matrizes*, isto é, *conjuntos de duas dimensões*, são necessários *dois níveis de malhas de repetição controlada*, isto é, uma *malha embutida* dentro de outra, como exemplificado na figura 6-19.

FIGURA 6-19: *EXEMPLO DE PROGRAMA COM OPERAÇÕES SOBRE MATRIZES*

```
10  DIM    H (3,5)
20  FOR    J = 1 TO 3
30  FOR    K = 1 TO 5
40  INPUT  H (J, K)
50  NEXT   K
60  NEXT   J
70  INPUT  " ACRESCIMO", X
80  FOR    M = 1 TO 3
90  FOR    N = 1 TO 5
100 LET    H (M, N) = H (M, N) + X
110 NEXT   N
120 NEXT   M
130 FOR    P = 1 TO 3
140 FOR    Q = 1 TO 5
150 PRINT  H (P, Q)
160 NEXT   Q
170 NEXT   P
```

No programa da figura 6-19, dentro da *primeira malha (dupla) de repetição controlada* é feita a entrada de *15* valores para os elementos da *matriz* H, *linha por linha*, isto é, variando *mais rapidamente* o *segundo subscrito*, controlado pela *malha mais interna*. Depois, é lido um valor que servirá de acréscimo, para ser adicionado a todos os elementos da *matriz*, o que é feito na *malha (dupla) seguinte*. Após a adição do valor, na *terceira malha (dupla)*, é feita a saída dos *15* novos valores dos elementos da *matriz* H, um por linha. A seguir, o programa *termina*. Nos menores micros a instrução na linha 70 deverá ser desdobrada em duas: uma de exibição da mensagem e outra de leitura do acréscimo.

6.4 Instruções de declaração e de aquisição de dados

Em muitos programas, alguns dados são conhecidos previamente, isto é, são constantes do programa. Neste caso, é mais confortável para o programador fazer a declaração desses valores uma única vez, através da instrução

```
DATA
```

,

que significa *dados*, que é uma instrução de *especificação*, isto é, *não-executável*.

Se é possível *declarar* os *valores de dados* para serem usados em cálculos no programa, nada mais natural do que dispor também de uma instrução para fazer a *aquisição* desses dados, nos pontos do programa onde se fazem necessários, a *instrução*

```
READ
```

,

que *não* é uma instrução de *especificação*, mas uma instrução *executável*. É apresentada aqui por razões didáticas e significa *ler*.

O uso dessas instruções é exemplificado com o programa da figura 6-20.

FIGURA 6-20: *EXEMPLO DE USO DE INSTRUÇÕES* **DATA** *E* **READ**

```
10  DATA   2, 3.5, 0.678, "ALO", " AMIGO"
20  READ   X, Y, Z
30  PRINT  X
40  PRINT  Y ; Z
50  READ   A$, B$
60  PRINT  A$; B$
70  END
```

O *resultado* da execução do programa da figura 6-20 é a *saída* dos valores

```
 2
 3.5      0.678
 ALO      AMIGO
```

.

No programa da figura 6-2Ø, a *declaração dos dados* é feita na primeira instrução do programa, na linha 1Ø. A instrução seguinte, na linha 2Ø, faz a *aquisição* simultânea de *três* valores *numéricos*, para as *variáveis* X, Y e Z, na ordem em que se encontram na instrução **DATA**, isto é, da esquerda para a direita. Desta forma, X conterá 2, Y conterá 3.5 e Z conterá Ø.678. A seguir, nas linhas 3Ø e 4Ø são impressos o valor de X e os valores de Y e Z, respectivamente. Novamente, na instrução na linha 5Ø, é feita a *aquisição* de dados, *alfanuméricos*, para as *variáveis alfanuméricas* A$ e B$, ou seja, respectivamente os valores das *constantes alfanuméricas* ALO e AMIGO. A instrução da linha número 6Ø imprimirá o conteúdo de A$ e B$ na mesma linha, em *formato fechado*. Com a linha 7Ø o programa termina.

A posição da instrução **DATA** é totalmente irrelevante, isto é, ela pode estar em *qualquer posição* dentro do programa. Além disso, os dados podem ser declarados em *mais de uma* instrução **DATA**. A *ordem* de *aquisição* dos dados obedecerá à *ordem numérica crescente* das instruções **DATA**. O mesmo programa da figura 6-2Ø pode ser reescrito como na figura 6-21, produzindo a mesma saída.

FIGURA 6-21: *EXEMPLO (MODIFICADO) DE USO DE INSTRUÇÕES* **READ** *E* **DATA**

```
1Ø  READ  X, Y
2Ø  READ  Z
3Ø  PRINT X
4Ø  PRINT Y; Z
5Ø  READ  A$
6Ø  READ  B$
7Ø  PRINT A$; B$
8Ø  DATA  2, 3.5, Ø.678
9Ø  DATA  "ALO", " AMIGO"
1ØØ END
```

Recomenda-se colocar as instruções de *declaração de dados*, **DATA**, *sempre* no *início* ou sempre no *final* do programa, embora elas possam vir em *qualquer posição*, apenas por uma questão de organização. O *mais natural* é colocá-las sempre no *início* do programa.

É totalmente indiferente *declarar dados* com *uma* instrução como

```
1Ø DATA 2.56837, "BOM DIA", 4, "SIM", 5.294, "NAO", 5, "FIM"
```

,

ou através de um *grupo* de instruções como

```
1Ø DATA 2.56837, "BOM DIA", 4
2Ø DATA "SIM", 5.294, "NAO"
3Ø DATA 5, "FIM"
```

,

ou na forma

```
1Ø DATA 2.56837, "BOM DIA"
2Ø DATA 4, "SIM", 5.294
3Ø DATA "NAO", 5
4Ø DATA "FIM"
```

,

ou ainda em muitas outras formas válidas análogas a estas.

Da mesma forma, é possível fazer a *aquisição* dos dados acima com, por exemplo, uma instrução como

```
1Ø READ A, A$, B, B$, C, C$, D, D$
```

,

ou com o conjunto de instruções

```
1Ø READ A, A$, B, B$, C
2Ø READ C$, D, D$
```

,

ou ainda

```
1Ø READ A, A$, B
2Ø READ B$, C
3Ø READ C$, D
4Ø READ D$
```

,

ou ainda com qualquer conjunto válido de instruções análogas.

Os exemplos acima servem apenas para esclarecer o funcionamento conjunto das instruções **DATA** e **READ**. Todavia, não foi esclarecido o seu uso, isto é, em que circunstâncias são usadas. Com relação às circunstâncias em que o seu uso é oportuno, deve-se dizer que é quando os *dados são conhecidos previamente* pelo programador.

A forma geral da *instrução de declaração de dados* é

DATA <*constante*> [, <*constante*>] . . .

Em um programa pode haver diversas instruções de *declaração de dados*. A *lista de dados* é constituída pela *seqüência de constantes* que se inicia com o *primeiro* dado da instrução **DATA** com *menor* número de linha até o *último* dado da instrução **DATA** de *maior* número de linha. As *constantes* da *lista* podem ser *numéricas ou alfanuméricas*. Neste último caso devem vir sempre entre aspas.

A forma geral da *instrução de aquisição de dados* é

READ <*variável*> [, <*variável*>] . . .

As *variáveis* da *lista* de uma instrução **READ** devem ser do *mesmo tipo* que os *dados* correspondentes da *lista* das instruções **DATA**.

Os dados serão adquiridos na *seqüência*, isto é, do *primeiro* até o *último*. Se houver *maior* número de *aquisições* de dados, através de instruções **READ**, do que *dados disponíveis na lista*, isto é, no conjunto de instruções **DATA**, a execução do programa será *suspensa* e o sistema enviará uma *mensagem de erro*.

Só se pode usar a instrução de *aquisição de dados*, **READ**, se houver uma *lista de dados* estabelecida por uma ou mais instruções de *declaração de dados*, **DATA**.

As *variáveis* da *lista*, na instrução **READ**, podem ser *numéricas simples* ou *subscritas* ou *alfanuméricas*.

Esgotada a *lista de valores* de instruções **DATA**, é possível reiniciar a aquisição de dados, a partir do primeiro valor da lista, em muitos sistemas BASIC, por meio da *instrução*

RESTORE ,

que significa *restaurar* e se refere ao *apontador de dados* da instrução **DATA**. Este *apontador, antes* de qualquer leitura, se encontra com valor *1*, isto é, apontando para o *primeiro* elemento da *lista*. A cada *execução* da instrução **READ**, seu valor será aumentado do *número de elementos lidos* da *lista*. Estará sempre apontando para o *próximo* elemento da *lista* a ser adquirido.

Um exemplo simples, que mostra o uso da instrução **READ** para *adquirir dados*, combinado com o uso da instrução **RESTORE**, é apresentado na figura 6-22.

FIGURA 6-22: *EXEMPLO DE USO DE INSTRUÇÃO* **RESTORE**

```
10  DATA    1, 10, 1
20  READ    X, Y, Z
30  FOR     I = X TO Y STEP Z
40  PRINT   I;
50  NEXT    I
60  RESTORE
70  READ    X, Y, Z
80  FOR     J = X TO Y STEP Z
90  PRINT   J,
100 NEXT    J
110 RESTORE
120 READ    X, Y, Z
130 FOR     K = X TO Y STEP Z
140 PRINT   K
150 NEXT    K
160 END
```

Uma *mesma* e *única* instrução **READ** pode ser *usada diversas vezes*, para fazer a *aquisição de dados*, como é mostrado no exemplo da figura 6-23.

FIGURA 6-23: *EXEMPLO DE USO DE INSTRUÇÃO* **READ**

```
10  DATA    1, 2, 3, 4, 5, 6, 7, 8, 9, 10
20  FOR     I = 9 TO 100 STEP 10
30  READ    A
40  PRINT   A;
50  NEXT    I
60  END
```

Em geral, os menores microcomputadores *não* dispõem das instruções **DATA**, **READ** e **RESTORE**.

Exercícios:

ESTUDO DIRIGIDO:

1) Que tipos de funções pode o programador dispor na linguagem BASIC? Que são funções numéricas predefinidas? Quais são os grupos de funções numéricas predefinidas? Quais são as principais funções aritméticas? Quais são as principais funções transcendentais? Quais são as principais funções trigonométricas? Que é parâmetro ou argumento de uma função? Todas as funções têm argumento? Como invocar ou referenciar uma função numérica predefinida? Para que serve a instrução **RANDOMIZE** (ou **RANDOM** ou **RAND**)?

2) Que são variáveis coletivas? Qual a diferença entre uma variável coletiva e uma variável simples? Como são atribuídos nomes a variáveis coletivas? Como é feita uma referência a um elemento de uma coleção? Que é índice ou subscrito? Que são vetores? Que são matrizes? Qual a instrução para declaração de variáveis coletivas? É obrigatória a declaração de variáveis coletivas?

3) Quais as instruções mais usadas para manipulação de vetores e matrizes? Quantas malhas de repetição embutidas são necessárias para manipular um vetor? Quantas malhas de repetição embutidas são necessárias para manipular uma matriz?

4) Qual a instrução de declaração de dados? Qual a instrução de aquisição (interna) de dados? O que é instrução de especificação? A instrução **DATA** é de especificação? A instrução **READ** é de especificação? Quantas instruções **DATA** pode ter um programa? Quantas instruções **READ** pode ter um programa? Um programa pode ter instruções **READ** sem instruções **DATA**? Em que posição do programa devem ser colocadas as instruções **DATA**? Uma instrução **DATA** pode misturar dados numéricos e alfanuméricos? Que correspondência deve existir entre as variáveis da instrução **READ** e os dados da instrução **DATA**? Uma única instrução **READ** pode adquirir dados declarados em várias instruções **DATA**? Várias instruções **READ** podem adquirir dados de uma única instrução **DATA**? Várias instruções **READ** podem adquirir dados de várias instruções **DATA**? Qual a forma geral da instrução **DATA**? Qual a forma geral da instrução **READ**? Que é o apontador de dados? Como restaurar o apontador de dados? Em que casos é usada a instrução de restauração do apontador de dados? Qual a forma geral da instrução **RESTORE**? Os menores microcomputadores dispõem das instruções **READ**, **DATA** e **RESTORE**?

PROBLEMAS:

1) Desejam-se ler valores numéricos quaisquer e exibi-los de forma que tenham sempre duas casas fracionárias decimais. Um programa para realizar esta operação é.

```
10 PRINT "INFORME NUMERO"
20 INPUT X
30 PRINT "VALOR LIDO="; X
40 PRINT "VALOR CALCULADO="; SGN (X) * INT (ABS (X) * 100) / 100
```

Execute, com vários valores positivos e negativos. Modifique o programa para fazer o arredondamento na segunda casa decimal (sugestão: adicione .005 ao valor absoluto de X). Execute novamente, com a mesma série de valores anteriormente usada e compare os resultados.

2) Modificar o programa anterior, para que exiba números quaisquer, com um número de casas fracionárias decimais, a escolher. O número de casas decimais deve ser, portanto, um dado a ser lido (sugestão: usar como expoente de uma potência de base 10, para multiplicar e dividir o valor que será exibido).

3) Escreva um programa simples que gere palpites para a loto. Leia o número de palpites (entre 5 e 10). Se for menor do que 5, tome 5. Se for maior do que 10, tome 10. Usando a instrução **RAND** e a função **RND**, dentro de uma malha de instruções **FOR** e **NEXT**, exiba os valores gerados (no intervalo entre 0 e 99). Faça o jogo e boa sorte.

4) Deseja-se obter a soma de uma série de valores numéricos que serão lidos pelo teclado. Esses valores deverão ser exibidos com duas casas decimais e também a sua soma. Todavia, devido ao truncamento, a soma apresentada não estará de acordo com os valores exibidos se for feita com os valores originais. Deseja-se que a soma corresponda aos valores exibidos (e não aos originais) mas que a diferença entre este resultado e a soma real seja conhecida. Uma solução é adicionar as partes fracionárias abandonadas com o truncamento, que podem ser obtidas com o uso da função FRA. Se o seu microcomputador possui esta função, escreva um programa que resolva este problema.

5) Um programa que converte um ângulo dado em graus, minutos e segundos para graus é

```
10 PRINT "CONVERSAO DE ANGULO"
20 PRINT "PARA GRAUS"
30 PRINT "GRAUS="
40 INPUT G
50 IF G = 0 THEN LET G = 1E-38
60 PRINT "MINUTOS="
70 INPUT M
80 PRINT "SEGUNDOS="
90 INPUT S
100 PRINT "ANG (GRAUS) = "; SIGN (G) * (ABS (G) + (ABS (M)
      + (ABS (S) / 60)) / 60)
```

Execute. O sinal do ângulo deve ser fornecido com os graus. Altere o programa para que exiba também o valor do ângulo em radianos, precedido da mensagem ANG (RAD) =. Altere também a saída do cabeçalho na tela.

6) Reescreva o programa da figura 6-12 para que as raízes imaginárias também possam ser calculadas e exibidas, na forma complexa, $a \pm jb$. Execute com coeficientes que produzam a ausência de trinômio, raízes reais e desiguais, raízes reais e iguais e raízes imaginárias.

7) Sabe-se que é possível a mudança de base logarítmica de um sistema de base a para um de base b multiplicando-se o logaritmo na base b pelo módulo do sistema de base b em relação ao de base a. O módulo é dado por: $1 / \log_a b$. Escrever um programa que faça a conversão da base neperiana para a base decimal, exclusivamente para valores positivos (não existem logaritmos de números negativos).

8) Em um triângulo retângulo são conhecidos a hipotenusa, a, e um ângulo B (oposto ao

lado *b*). Pedem-se o ângulo adjacente, C, os catetos *b* e *c* e a área do triângulo, S. As fórmulas que permitem o cálculo são:

$$C = 90° - B$$
$$b = a \text{ sen } B$$
$$c = a \cos B$$
$$S = 1/2\ a^2 \text{ sen } B \cos B$$

Escreva um programa que faça os cálculos. Lembre-se de que o ângulo B deve ser menor do que 90°. Mesmo valores de B muito próximos de 90° poderão conduzir a valores muito grandes de *b*, que não possam ser processados pelo computador. Um limite menor deve ser estabelecido para o ângulo B.

9) É possível avaliar a altura *h* de um objeto à distância *d* observando-se o ângulo C sob o qual é visto o objeto, segundo a fórmula:

$$h = d \text{ tg } C$$

Escrever um programa para fazer o cálculo. Limitar os valores muito grandes de C (próximos a 90°), para evitar valores muito grandes de *h*.

SÉTIMO CAPÍTULO
OPERAÇÕES COM CADEIAS DE CARACTERES

expressões alfanuméricas – instrução de atribuição alfanumérica – subcadeias de caracteres – variáveis coletivas alfanuméricas – funções para manipulação de cadeias de caracteres

7.1 Expressões alfanuméricas

As *expressões alfanuméricas* estão implementadas na maioria dos sistemas, diferindo apenas em detalhes menores, como no símbolo usado para *operador de concatenação*. Da mesma maneira que ocorreu com as *expressões aritméticas*, existem *expressões alfanuméricas simples* e *expressões alfanuméricas compostas*.

Exemplos de *expressões alfanuméricas simples* são apresentados na figura 7-1.

FIGURA 7-1: *EXEMPLO DE EXPRESSÕES ALFANUMÉRICAS SIMPLES*

EXPRESSÃO (COMPONENTE)	SIGNIFICADO
"CONSTANTE"	constante alfanumérica
V4$	variável alfanumérica simples

A forma geral de uma *expressão alfanumérica simples* é dada por

$$\{< \textit{constante alfanumérica} > | < \textit{variável alfanumérica simples} >\}$$

As *expressões alfanuméricas simples* são os *componentes básicos* das *expressões alfanuméricas compostas*.

Exemplos de *expressões alfanuméricas compostas* são os que estão apresentados na figura 7-2.

FIGURA 7-2: *EXEMPLOS DE EXPRESSÕES ALFANUMÉRICAS COMPOSTAS*

EXPRESSÃO	COMPONENTE	SIGNIFICADO
X$, "VARIAVEL"	X$	variável alfanumérica simples
	,	operador alfanumérico (concatenação)
	"VARIAVEL"	constante alfanumérica (literal)
"BOM", W3$	"BOM"	constante alfanumérica (literal)
	,	operador alfanumérico (concatenação)
	"W3$"	variável alfanumérica simples
"A" , "B" , "C"	"A"	constante alfanumérica (literal)
	,	operador alfanumérico (concatenação)
	"B"	constante alfanumérica (literal)
	,	operador alfanumérico (concatenação)
	"C"	constante alfanumérica (literal)

A forma geral de uma *expressão alfanumérica composta* é dada por

< *expressão alfanumérica simples* > < *operador alfanumérico* >
< *expressão alfanumérica simples* > [< *operador alfanumérico* >
< *expressão alfanumérica simples* >] . . .

Portanto, uma *expressão alfanumérica composta* é constituída por *expressões alfanuméricas simples* ligadas por *operadores alfanuméricos* (*concatenação*).

O *operador de concatenação* mais utilizado é a *vírgula* (,), porém existem implementações utilizando o *sinal de adição* (+) ou o *sinal e comercial* (&).

As *regras* para formação de *expressões alfanuméricas compostas* são análogas às das *expressões aritméticas*.

A *regra* de avaliação de *expressões alfanuméricas* diz que a *expressão* será *concatenada* na *mesma ordem* em que as *expressões alfanuméricas simples* figuram como *componentes* da *expressão alfanumérica composta* da esquerda para a direita. O *resultado* é uma *cadeia de caracteres*, ou seja, um valor *alfanumérico* em que todos os valores alfanuméricos dos componentes estão justapostos, segundo a regra acima.

Exemplos da aplicação de *regras* de avaliação de *expressões alfanuméricas* são apresentados na figura 7-3.

FIGURA 7-3: *EXEMPLOS DE AVALIAÇÃO DE EXPRESSÕES ALFANUMÉRICAS COMPOSTAS*

EXPRESSÃO	COMPONENTE	SIGNIFICADO
"ALFA" , "NUMERICO"	1. "ALFA" , "NUMERICO"	concatenação
A$, B3$, "XPTO"	1. A$, B3$ 2. A$, B3$, "XPTO"	concatenação concatenação

7.2 Instrução de atribuição alfanumérica

Uma *variável* pode ser usada para substituir um valor que aparece muitas vezes em várias *expressões* dentro do programa, com economia, sendo seu valor *atribuído* uma única vez pela instrução **LET**.

A figura 7-4 mostra duas formas possíveis para um mesmo programa, a primeira *sem* o uso da *instrução* **LET** e a segunda *com* o uso da *instrução* **LET**, especificamente para *atribuição aritmética*.

FIGURA 7-4: *EXEMPLO DE USO DA INSTRUÇÃO DE ATRIBUIÇÃO ARITMÉTICA*

SEM O USO DE **LET**	COM O USO DE **LET**
10 **PRINT** 3.765 – 2 20 **PRINT** 435.65 ↑ 2 30 **PRINT** 4.6 * 2 40 **END**	10 **LET** X = 2 20 **PRINT** 3.765 – X 30 **PRINT** 435.65 ↑ X 40 **PRINT** 4.6 * X 50 **END**

A economia é mais evidente quando se trata de *atribuição alfanumérica*, dando-se um exemplo de programa sob duas formas, respectivamente *sem* uso e *com* uso da instrução **LET**, na figura 7-5.

FIGURA 7-5: *EXEMPLO DE USO DA INSTRUÇÃO DE ATRIBUIÇÃO ALFANUMÉRICA*

SEM O USO DE **LET**	COM O USO DE **LET**
10 **PRINT** "DIARIA" ; "MENTE" 20 **PRINT** "MENSAL" ; "MENTE" 30 **PRINT** "ANUAL" ; "MENTE" 40 **END**	10 **LET** X$ = "MENTE" 20 **PRINT** "DIARIA" ; X$ 30 **PRINT** "MENSAL" ; X$ 40 **PRINT** "ANUAL" ; X$ 50 **END**

Alguns exemplos da instrução de *atribuição alfanumérica* encontram-se na figura 7-6.

FIGURA 7-6: *EXEMPLOS DE INSTRUÇÃO DE ATRIBUIÇÃO ALFANUMÉRICA*

LET D$ = E$	O valor *alfanumérico de* E$ é atribuído a D$.
LET F$ = "AO", K$, "PT", L1$	À *variável alfanumérica* F$ é atribuído o valor da *expressão alfanumérica* "AO" , K$, "PT" , L1$.

A forma geral de uma instrução de *atribuição alfanumérica* é

LET < *variável alfanumérica simples* > = < *expressão alfanumérica* >

É possível, na maioria das versões *ampliadas* ou *estendidas* da linguagem BASIC, atribuir valor a *pedaços* da *cadeia* representada por uma *variável alfanumérica*, com o uso da *instrução* **LET**.

Na figura 7-7 encontram-se alguns exemplos de *atribuição* de valor a *subcadeias* de *variável alfanumérica*.

FIGURA 7-7: *EXEMPLOS DE ATRIBUIÇÃO DE VALOR A SUBCADEIA DE VARIÁVEL ALFANUMÉRICA*

EXEMPLO	EXPLICAÇÃO
LET A1$ (1 , 3) = "ALF"	*Atribui às três* primeiras posições da *cadeia* representada pela *variável alfanumérica* A1$, isto é, posições *1* a *3*, o valor da *expressão alfanumérica* à direita do sinal de atribuição, ALF.
LET A1$ (4) = "ANUMERICO"	*Atribui* à parte da *cadeia* representando a *variável alfanumérica* A1$, que se inicia na posição *4* e vai para a direita, o valor da *expressão alfanumérica* à direita do sinal de atribuição, ANUMERICO.

O conteúdo da variável A1$, após a execução das duas instruções exemplos da figura 7-7, é ALFANUMERICO.

A forma geral da *instrução de atribuição de valor alfanumérico a subcadeia de variável alfanumérica* é dada por

LET < *variável alfanumérica* > (< *expressão aritmética* > [, < *expressão aritmética* >]) = < *expressão alfanumérica* >

Evidentemente, os *índices* ou *subscritos* de *subcadeia* têm que estar dentro das *dimensões* atribuídas implícita ou explicitamente à *variável alfanumérica*. Toda *cadeia* resultante da avaliação da *expressão alfanumérica à direita do sinal de atribuição* (=) será *truncada* e *perdida* caso ultrapasse o *limite final* da *variável alfanumérica*. *Se* houver um *único índice*, refere-se à *posição inicial da subcadeia*. Se houver dois *índices* referem-se, o *primeiro* à *posição inicial da subcadeia* e o *segundo* à *posição final da subcadeia*, dentro da *cadeia* representada pela *variável alfanumérica*.

Caso a versão *estendida* ou *ampliada* da linguagem permita a declaração e uso de *conjuntos alfanuméricos*, onde cada *elemento* é um valor *alfanumérico*, isto é, uma *cadeia de caracteres*, é usado, em geral, o *colchete à direita* (]) para substituir o *parêntese à direita* e o *colchete à esquerda* ([) para substituir o *parêntese à esquerda*, evitando assim possível confusão com referência a *elemento* de *vetor* ou *matriz alfanumérico*.

7.3 Subcadeias de caracteres

O leitor já sabe o que é cadeia de caracteres. Subcadeia de caracteres é também uma cadeia de caracteres que é *parte* de outra cadeia de caracteres.

Não apenas as versões estendidas ou ampliadas da linguagem BASIC possuem facilidades de manipulação de subcadeias. Os menores microcomputadores dispõem também dessas facilidades, se bem que formalmente diferentes. As facilidades de manipulação de subcadeia aqui apresentadas se referem à implementação de BASIC encontrada nos menores microcomputadores pessoais.

Exemplos de referências a subcadeias de caracteres se encontram na figura 7-8.

FIGURA 7-8: *EXEMPLOS DE REFERÊNCIAS A SUBCADEIAS DE CARACTERES*

EXEMPLO	EXPLICAÇÃO
"ABCDEF" (2 TO 4)	A subcadeia é formada com 3 caracteres retirados da constante alfanumérica ABCDEF, da posição 2 até a posição 4, isto é, a cadeia BCD.
X$ (5)	A subcadeia é formada de um caractere, o quinto, a partir da cadeia de caracteres contida na variável alfanumérica X$.
("ABC" + X$) (TO 6)	A subcadeia é formada com 6 caracteres, retirados da primeira até a sexta posição da cadeia obtida com a avaliação da expressão alfanumérica entre parênteses, isto é, com a concatenação da constante ABC e o conteúdo da variável X$.
(A$ + B$ + "XYZ") (3 TO)	A subcadeia é formada com os caracteres que se encontram na terceira posição até a última da cadeia de caracteres resultante da avaliação da expressão entre parênteses, isto é, com a concatenação das variáveis A$ e B$ e a constante XYZ.
Z$ (7 TO 7)	A subcadeia é formada com um único caractere, o sétimo, da cadeia de caracteres contida na variável Z$.
X$ (2 * A TO B + 4)	A subcadeia é formada com os caracteres retirados da cadeia contida na variável X$, a partir da posição correspondente ao valor inteiro da expressão aritmética 2 * A até a posição correspondente ao valor inteiro da expressão B + 4.

A forma geral da *referência a subcadeia de caracteres* ou *subcadeia alfanumérica* é dada por

< *expressão alfanumérica* > ([< *expressão aritmética* >] TO [< *expressão aritmética* >]) .

Se a cadeia a partir da qual for extraída a subcadeia terminar antes que todos os caracteres da subcadeia sejam obtidos, as posições à direita desta serão preenchidas com caracteres *branco*.

A referência a subcadeia pode ser usada na formação de expressões alfanuméricas ou como variável à esquerda de um sinal de atribuição de uma instrução **LET**. No primeiro caso, a subcadeia é extraída a partir de uma outra cadeia. No segundo caso, a subcadeia é atribuída à variável alfanumérica, nas posições especificadas.

Um exemplo de programa simples que mostra como opera uma referência a subcadeia aparece na figura 7-9.

FIGURA 7-9: *EXEMPLO DE PROGRAMA COM REFERÊNCIA A SUBCADEIA DE CARACTERES*

```
10  FOR    I=0 TO 21
20  PRINT  "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (1 TO I + 5)
30  PAUSE  60
40  NEXT   I
```

O programa da figura 7-9 pode ser alterado para que a subcadeia seja obtida a partir de uma variável alfanumérica, em vez de uma constante alfanumérica, como anteriormente, resultando no programa da figura 7-10.

FIGURA 7-10: *EXEMPLO DE PROGRAMA (MODIFICADO) COM REFERÊNCIA A SUBCADEIA*

```
10  LET    X$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20  FOR    I = 0 TO 21
30  PRINT  X$ (1 TO I + 5)
40  PAUSE  60
50  NEXT   I
```

Uma variante do programa da figura 7-10, que demonstra a obtenção de subcadeia a partir de uma expressão alfanumérica, encontra-se na figura 7-11.

FIGURA 7-11: *EXEMPLO DE PROGRAMA (ALTERNATIVA) QUE EXTRAI SUBCADEIA A PARTIR DE EXPRESSÃO ALFANUMÉRICA*

```
10  LET    X$ = " ABCDE"
20  LET    Y$ = " FGHIJKLM"
30  FOR    I = 0 TO 21
40  PRINT  TAB (I); (X$ + Y$ + "NOPQRSTUVWXYZ") (I + 1 TO I + 5)
50  PAUSE  60
60  NEXT   I
```

Um programa mais simples e que produz o mesmo efeito que o da figura 7-11 se encontra na figura 7-12.

FIGURA 7-12: *EXEMPLO DE PROGRAMA (SIMPLIFICADO) PARA EXTRAIR SUBCADEIA*

```
10  FOR    I = 0 TO 21
20  LET    X$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (I + 1 TO I + 5)
30  PRINT  TAB (I); X$
40  PAUSE  60
50  NEXT   I
```

A referência a subcadeia também pode ser usada para atribuição de valor, devendo aparecer à esquerda de um sinal de atribuição de uma instrução **LET**, como foi mencionado na seção anterior.

Um exemplo simples que demonstre a forma de atribuir valor a uma subcadeia é apresentado na figura 7-13.

FIGURA 7-13: *EXEMPLO DE PROGRAMA PARA ATRIBUIÇÃO DE VALOR A SUBCADEIA*

```
10  DIM    X$ (9)
20  LET    X$ (1 TO 3) = "SUB"
30  LET    X$ (4 TO 9) = "CADEIA"
40  PRINT  X$
```


No exemplo da figura 7-13, a variável **X$** não continha inicialmente nenhum valor e apenas foi dimensionada, com o tamanho de 9 caracteres. Mesmo que uma variável alfanumérica já contenha um valor inicial, este poderá ser alterado pela atribuição de valor a subcadeias. Um exemplo simples que demonstra o que ocorre é apresentado na figura 7-14.

FIGURA 7-14: *EXEMPLO DE PROGRAMA (MODIFICADO) PARA ATRIBUIR VALOR A SUBCADEIA*

```
10  LET    X$ = "*********"
20  PRINT  X$
30  LET    X$ (1 TO 3) = "SUB"
40  PRINT  X$
50  LET    X$ (4 TO 9) = "CADEIA"
60  PRINT  X$
```

7.4 Variáveis coletivas alfanuméricas

Uma *cadeia* de caracteres pode ser vista como um *vetor* de caracteres, como se pode observar no exemplo apresentado na figura 6-15 da seção 6-2. Este conceito é coerente com o de subcadeia e existe compatibilidade inclusive na notação. Um exemplo disso é o programa da figura 7-15, que atribui um caractere a uma série de posições de uma variável alfanumérica.

FIGURA 7-15: *EXEMPLO DE PROGRAMA DE ATRIBUIÇÃO DE CARACTERE A SUCESSIVAS POSIÇÕES DE UMA VARIÁVEL ALFANUMÉRICA*

```
10  LET    X$ = "SUBCADEIA"
20  PRINT  X$
30  FOR    I = 1 TO 9
40  LET    X$ (I) = "*"
50  PRINT  X$
60  NEXT   I
```

Da mesma forma, uma matriz de caracteres pode ser vista como um vetor de cadeias de caracteres.

A forma de dimensionar uma *matriz de caracteres* **X$**, com 3 linhas de caracteres, cada uma com **10** colunas, é

```
DIM X$ (3, 10)
```

,

que corresponde a um *vetor* de 3 posições, cada uma ocupada por uma *cadeia de caracteres* com 10 posições. Observa-se que o número de caracteres de cada cadeia de caracteres é dado pelo índice mais à *direita* da matriz. O índice mais à *esquerda* aponta para o elemento do vetor de cadeias, isto é, para uma cadeia de caracteres. Alguns exemplos de referências a cadeias ou caracteres de cadeias de uma *matriz de caracteres* (ou *vetor de cadeias de caracteres*) são apresentados na figura 7-16.

FIGURA 7-16: *EXEMPLOS DE REFERÊNCIA A ELEMENTOS DA MATRIZ DE CARACTERES*

EXEMPLO	EXPLICAÇÃO
X$ (J – 3)	referência à cadeia de caracteres que se encontra na linha da matriz de caracteres X$ correspondente ao valor inteiro da expressão J – 3, que deve ser um valor dentro dos limites do índice
X$ (J, 3)	referência ao terceiro caractere da cadeia de caracteres que se encontra na linha correspondente ao valor inteiro de J, que deve estar dentro dos limites do índice, da matriz X$
X$ (3, 2 TO 5)	referência à subcadeia nas posições de 2 a 5 da terceira cadeia de caracteres da matriz de caracteres X$
X$ (K – 2) (J TO L + 1)	referência à subcadeia que começa na posição correspondente ao valor inteiro de J e vai até à posição correspondente ao valor inteiro da expressão L + 1 da cadeia de caracteres que se encontra na linha correspondente ao valor inteiro da expressão K – 2, respeitados os limites da matriz X$

O leitor terá observado nos exemplos da figura 7-16 que é possível fazer referência a uma cadeia de caracteres, a um caractere ou a uma subcadeia de caracteres. Esta notação, contudo, é usual apenas nos menores microcomputadores.

A forma geral de uma *referência a subcadeia de caracteres* de uma *matriz de caracteres* é dada por

< *nome de matriz de caracteres* > (< *expressão aritmética* > [,
[< *expressão aritmética* >] [TO [*expressão aritmética*]]]) |
< *nome de matriz de caracteres* > (< *expressão aritmética* >)
[([< *expressão aritmética* >][TO [< *expressão aritmética* >]])]

Um exemplo de programa que utiliza a matriz de caracteres encontra-se na figura 7-17.

FIGURA 7-17: *EXEMPLO DE PROGRAMA QUE USA MATRIZ DE CARACTERES*

```
10  DIM    A$ (10 , 3)
20  FOR    K = 1 TO 10
30  LET    A$ (K) = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCD"
           (3 * K – 2 TO 3 * K)
40  PRINT  A$ (K)
50  NEXT   K
```

Os dados a serem armazenados em uma matriz de caracteres poderão ser lidos a partir do teclado, o que é demonstrado pelo programa da figura 7-18.

FIGURA 7-18: *EXEMPLO DE PROGRAMA QUE LÊ DADOS PARA UMA MATRIZ DE CARACTERES*

```
10  DIM    N$ (5, 25)
20  PRINT  "ESCREVA 5 NOMES"
30  PRINT  "COM ATE 25 CARACTERES"
40  FOR    M = 1 TO 5
50  INPUT  N$ (M)
60  NEXT   M
70  PRINT  " OS NOMES SAO "
80  FOR    L = 1 TO 5
90  PRINT  "N("; L; ")="; N$ (L)
100 NEXT   L
```

Em uma cadeia de caracteres pode haver dados de naturezas diversas. Por exemplo, em uma matriz de caracteres B$ cada cadeia de caracteres tem o comprimento de 60 caracteres, dos quais os 20 primeiros contêm o nome

de uma pessoa, os 30 seguintes o endereço da mesma pessoa e os 10 restantes, o telefone. Tome-se a matriz B$ com 5 dessas cadeias de caracteres e pode-se escrever, por exemplo, o programa da figura 7-19.

FIGURA 7-19: *EXEMPLO DE PROGRAMA QUE MANIPULA SUBCADEIA DE MATRIZ DE CARACTERES*

```
10  PRINT   " INFORME 5 NOMES ATE 20 CARACT."
20  PRINT   "COM ENDERECO ATE 30 CARACT. E"
30  PRINT   "TELEFONE COM ATE 10 CARACT."
40  DIM     B$ (5, 60)
50  FOR     P = 1 TO 5
60  INPUT   B$ (P) (1 TO 20)
70  INPUT   B$ (P) (21 TO 50)
80  INPUT   B$ (P) (51 TO 60)
90  NEXT    P
100 PRINT   "NOMES, ENDERECOS E TELEFONES"
110 FOR     Q = 1 TO 5
120 PRINT   "NOME ("; Q; ") = "; B$ (Q) (1 TO 20)
130 PRINT   "E:"; B$ (Q) (21 TO 50)
140 PRINT   "TEL:"; B$ (Q) (51 TO 60)
150 NEXT    Q
```

7.5 Funções para manipulação de cadeias de caracteres

Assim como existem funções matemáticas, que auxiliam a realização de cálculos numéricos, a maioria das implementações de BASIC também dispõem de funções numéricas ou alfanuméricas para a manipulação de caracteres e cadeias de caracteres.

As funções mais comuns nas diversas versões da linguagem BASIC são as que se encontram no quadro 7-1.

QUADRO 7-1: *FUNÇÕES PARA MANIPULAÇÃO DE CADEIAS DE CARACTERES ALFANUMÉRICOS*

FUNÇÃO	DESCRIÇÃO
LEN < *expressão alfanumérica* > ou LEN (< *expressão alfanumérica* >)	retorna o *tamanho da cadeia* resultante da avaliação da expressão alfanumérica
VAL (< *expressão alfanumérica* >)	retorna o *valor numérico* de uma *cadeia* de caracteres *numéricos*

VAL$ (< *expressão aritmética* >) ou STR$ (< *expressão aritmética* >)	retorna uma *cadeia* de caracteres numéricos correspondente ao valor da expressão aritmética
POS (< *expressão alfanumérica-1* > , < *expressão alfanumérica-2* >)	retorna a *posição* dentro da cadeia de caracteres da primeira expressão alfanumérica a partir da qual se inicia a cadeia de caracteres da segunda expressão alfanumérica
CHR$ (< *expressão aritmética* >)	retorna um *caractere* cujo código corresponde ao valor inteiro da expressão aritmética
CODE (< *expressão alfanumérica* >)	retorna o *código numérico* do primeiro caractere da cadeia de caracteres resultante da avaliação da expressão alfanumérica
INKEY$	retorna um *caractere* correspondente a um símbolo do *teclado* de acordo com a combinação de teclas que estiver sendo digitada (calcada) senão, nada (vazio)

Das funções do quadro 7-1, as quatro primeiras são comuns à maioria das implementações de BASIC, inclusive as *estendidas* ou *ampliadas*. Dessas quatro primeiras, as três primeiras estão presentes também nos microcomputadores pessoais de menor porte, observando-se que STR$ é a forma utilizada neste caso, em substituição a VAL$. A função POS não está disponível nos menores micros. Em compensação, as funções CHR$, CODE e INKEY$ são típicas dos menores microcomputadores.

Um programa que demonstre como opera a função LEN encontra-se na figura 7-20.

FIGURA 7-20: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* LEN

```
10  PRINT   "ESCREVA UMA CADEIA"
20  INPUT   X$
30  PRINT   "CADEIA="; X$
40  PRINT   "TAMANHO="; LEN (X$)
```

Um programa que utiliza a função LEN para permitir a correta repetição de uma cadeia na tela é apresentado na figura 7-21.

FIGURA 7-21: *EXEMPLO DE PROGRAMA COM APLICAÇÃO DA FUNÇÃO* LEN

```
10  SCROLL
20  PRINT   "INFORME SEU NOME"
30  INPUT   A$
40  CLS
50  PRINT   "SEU NOME EH "; A$
60  PAUSE   600
70  CLS
80  LET     L = LEN (A$) + 1
90  LET     K = 32 - L
100 FOR     J = 0 TO K STEP L
110 FOR     I = 0 TO 21
120 PRINT   AT I, J; A$
130 PAUSE   30
140 NEXT    I
150 NEXT    J
```

A função VAL serve para converter uma cadeia numérica em um número. Uma demonstração de como isto ocorre pode ser acompanhada com a execução do programa da figura 7-22.

FIGURA 7-22: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* VAL

```
10  PRINT   "ESCREVA UMA CADEIA NUMERICA"
20  PRINT   "INTEIRA OU FRACIONARIA"
30  PRINT   "CADEIA=";
40  INPUT   X$
50  PRINT   X$
60  PRINT   "NUMERO="; VAL (X$)
```

Nos menores microcomputadores, a função VAL é um pouco mais potente que nos maiores e permite a avaliação de uma cadeia de caracteres na forma de uma expressão numérica. Um exemplo que ilustra esta potencialidade se encontra no programa da figura 7-23.

FIGURA 7-23: *EXEMPLO DE AVALIAÇÃO DE CADEIA DE CARACTERES COMO EXPRESSÃO NUMÉRICA*

```
10  LET     X = 3
20  LET     X$ = "X + 3"
30  PRINT   VAL (X$) + 3
```


Nos menores microcomputadores, algumas restrições se aplicam ao uso da função VAL, como ter que ser o primeiro operando em expressões aritméticas e também a primeira coordenada em instruções **PRINT** AT, **PLOT** e **UNPLOT**.

Essa potencialidade da função VAL permite, por exemplo, a avaliação de expressões quaisquer, lidas como cadeias de caracteres, a partir do teclado, como mostra o programa da figura 7-24.

FIGURA 7-24: *DEMONSTRAÇÃO DE AVALIAÇÃO DE CADEIA LIDA DO TECLADO, COMO EXPRESSÃO NUMÉRICA*

```
10  PRINT   "DIGITE EXPRESSAO ARITMETICA"
20  PRINT   "EM FUNCAO DA VARIAVEL X"
30  INPUT   X$
40  PRINT   "EXPRESSAO="; X$
50  PRINT   "DIGITE VALOR DE X"
60  INPUT   X
70  PRINT   "X="; X
80  PRINT   "VALOR DA EXPRESSAO="; VAL (X$)
```

A função STR$ (ou VAL$ em algumas implementações) executa a operação inversa da função VAL, isto é, converte um número em uma cadeia de caracteres. Um programa que demonstra como é feita a conversão encontra-se na figura 7-25.

FIGURA 7-25: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* STR$

```
10  PRINT   "ESCREVA UM NUMERO"
20  PRINT   "INTEIRO OU FRACIONARIO"
30  PRINT   "NUMERO=";
40  INPUT   X
50  PRINT   X
60  PRINT   "CADEIA="; STR$ (X)
```

A função POS localiza onde se encontra uma subcadeia, dentro de uma outra cadeia, e fornece a posição em que a subcadeia começa. Pode ser útil, por exemplo, para pesquisar o início e o fim de uma subcadeia, antes de extraí-la. Não será apresentado exemplo por não estar disponível nos menores microcomputadores, escapando ao interesse imediato desta seção.

A função CHR$ retorna um caractere a partir de um valor que é o código do caractere. O código tem que estar no intervalo de 0 a 255. Um progra-

ma que exibe na tela o conjunto de caracteres de um pequeno microcomputador pessoal é o da figura 7-26.

FIGURA 7-26: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* CHR$

```
10  FOR     H = 0 TO 255
20  PRINT   CHR$ (H);
30  NEXT    H
```

A função CODE executa a operação inversa da função CHR$, isto é, dada uma cadeia de caracteres, retorna o código numérico do *primeiro* caractere da cadeia. Um programa de demonstração da função CODE é encontrado na figura 7-27.

FIGURA 7-27: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* CODE

```
10  FOR     V = 0 TO 21
20  LET     X$ = "ABCDEFGHIJKLMNOPQRSTUV" (V + 1 )
30  PRINT   X$, CODE (X$)
40  NEXT    V
```

A função INKEY$ permite ler caracteres do teclado, caso alguma tecla esteja calcada, no exato momento em que a função é executada. O leitor perguntará para que existe uma função de ler teclado se já existe a instrução de leitura **INPUT**. Todavia, existe uma diferença fundamental entre as duas operações. Enquanto a instrução **INPUT**, ao ser executada, mantém a execução do programa suspensa até que um dado tenha sido fornecido pelo teclado (dado seguido da tecla NEW LINE), a função INKEY$, quando é executada, não espera o fornecimento de dados. A função INKEY$ lê a tecla que estiver calcada, se houver alguma, e, se não houver, gera um caractere vazio. A função INKEY$ varre o teclado, independente de haver ou não alguma tecla sendo calcada.

Um programa de demonstração da função INKEY$, que poderá capturar e exibir até 704 caracteres (correspondentes à tela cheia: 22 linhas x 32 colunas), encontra-se na figura 7-28.

FIGURA 7-28: *PROGRAMA DE DEMONSTRAÇÃO DA FUNÇÃO* INKEY$

```
10  PRINT   "DIGITE S PARA INICIAR"
20  INPUT   S$
30  IF      S$ < > "S" THEN GOTO 20
40  CLS
50  FOR     W = 1 TO 704
60  PRINT   INKEY$ ;
70  NEXT    W
```

Uma forma de utilizar a função INKEY$ é para capturar os símbolos que estejam sendo enviados pelo teclado. Um exemplo é dado no programa que se encontra na figura 7-29.

FIGURA 7-29: *EXEMPLO DE PROGRAMA COM USO DA FUNÇÃO* INKEY$

```
10  DIM     A$ (10, 1)
20  PRINT   "DIGITE 10 CARACTERES"
30  PRINT   "PARA INICIAR TECLE S"
40  INPUT   S$
50  IF      S$ < > "S" THEN GOTO 40
60  FOR     G = 1 TO 10
70  IF      INKEY$ < > "" THEN GOTO 70
80  IF      INKEY$ =  "" THEN GOTO 80
90  LET     A$ (G) = INKEY$
100 NEXT    G
110 PRINT   "CARACTERES DIGITADOS"
120 FOR     F = 1 TO 10
130 PRINT   A$ (F);
140 NEXT    F
```

sem espaço (vazio)

O leitor terá observado no programa da figura 7-29 que a linha 70 serve para reter a execução do programa se estiver calcada alguma tecla. Se nenhuma tecla estiver calcada, a linha 80 é executada. A linha 80 por sua vez retém a execução do programa até que seja calcada uma tecla. Essa tecla será capturada pela função INKEY$ da instrução **LET**, na linha 90, e armazenada na matriz de caracteres A$. Este ciclo se repete 10 vezes, até que 10 caracteres tenham sido capturados. O caractere *branco* não pode ser capturado por INKEY$ porque a correspondente tecla, SPACE, durante a execução do programa, gera um comando **BREAK**, que fará parar a execução. Caso se queira que *branco* se inclua entre os caracteres exibíveis, é necessário relacioná-lo com outra tecla ou conjunto de teclas, através de programação.

Exercícios:

ESTUDO DIRIGIDO:

1) Que são expressões alfanuméricas? O que é uma expressão alfanumérica simples? E uma composta? Qual o operador alfanumérico? Quais os símbolos utilizados nas diversas implementações de BASIC como operador de concatenação? Quais as regras de formação de expressões alfanuméricas? Quais as regras de avaliação de expressões alfanuméricas compostas?

2) Que é atribuição alfanumérica? Qual a instrução para atribuição alfanumérica? Qual a forma geral da instrução de atribuição alfanumérica? É possível fazer a atribuição de valor a subcadeias? Qual a instrução para atribuição de valor a subcadeia? Qual a forma geral da instrução para atribuição de valor a subcadeia?

3) Que é uma subcadeia de caracteres? Que é uma referência a subcadeia de caracteres? Qual a forma geral de uma referência a subcadeia? Uma referência a subcadeia pode ser usada em uma expressão alfanumérica? Pode-se atribuir valor a uma subcadeia em uma instrução de atribuição alfanumérica (à esquerda do sinal =)? Uma subcadeia pode ser extraída (obtida) a partir de qualquer expressão alfanumérica? E de uma constante alfanumérica? E de uma variável alfanumérica? Pode-se atribuir valor a diversas subcadeias de uma mesma variável alfanumérica?

4) Que são variáveis coletivas alfanuméricas? Que é um vetor de caracteres? Que é uma matriz de caracteres? Como são dimensionadas as variáveis coletivas alfanuméricas? Que é um vetor de cadeias de caracteres? Numa matriz de caracteres o que as linhas representam? Qual o índice da variável coletiva que representa o número da linha? Numa matriz de caracteres, que representam as colunas? Qual o índice da variável coletiva que representa o número da coluna? Que é uma referência a subcadeia de uma matriz de caracteres? Qual a forma geral de uma referência a subcadeia de uma matriz de caracteres? Como obter uma subcadeia de uma matriz de caracteres? É possível atribuir valor a subcadeia de uma matriz de caracteres? Como?

5) É possível dispor de facilidades para a manipulação de caracteres? Quais são essas facilidades? Quais as funções que estão mais freqüentemente disponíveis nas implementações de BASIC? Quais destas são as mais comuns nos menores micros? Que faz a função LEN? Que faz a função VAL? Que faz a função STR$ (ou VAL$)? Para que serve a função POS? O que retorna com a função CHR$? Qual a finalidade da função CODE? A função INKEY$ substitui a instrução **INPUT**? Como opera a função INKEY$? Qual a diferença entre a forma de operar da função INKEY$ e da instrução **INPUT**?

PROBLEMAS:

1) Um programa simples que lê duas cadeias de caracteres e as exibe na tela, justapostas, é

```
10  PRINT   "ESCREVA DOIS NOMES"
20  INPUT   A$
30  INPUT   B$
40  PRINT   A$;" E "; B$
```


Execute. Altere o programa para concatenar a cadeia de caracteres a ser exibida, através de uma instrução **LET**, em uma variável **C$**, antes da exibição (de **C$**). Reexecute e confira.

2) Um programa simples que lê uma cadeia de até 22 caracteres e a exibe progressivamente na tela é

```
10  DIM     N$ (22)
20  PRINT   "INFORME SEU NOME"
30  INPUT   N$
40  CLS
50  FOR     Z = 0 TO 21
60  PRINT   AT Z, 31 – Z; N$ (1 TO Z + 1)
70  NEXT    Z
```

Execute. Altere o programa para que o nome seja exibido de forma crescente, não mais de cima para baixo mas sim crescente de baixo para cima. Reexecute.

3) Um programa que desloca um nome de até 22 caracteres na tela, em diagonais transversais e sentidos opostos, é

```
10  DIM     N$ (22)
20  PRINT   "INFORME SEU NOME"
30  INPUT   N$
40  FOR     L = 0 TO 21
50  CLS
60  PRINT   AT L, 31 – L; N$ (1 TO L + 1 )
70  PRINT   AT 21 – L, 31 – L; N$ (1 TO L + 1 )
80  PAUSE   60
90  NEXT    L
```

Execute. Modifique o programa para que o nome seja exibido, deslocando-se, nas diagonais opostas, também em sentidos opostos alternadamente. Reexecute.

4) Um programa que exibe a contagem de 1 até 10 no centro da tela é

```
10  FOR     K = 1 TO 10
20  CLS
30  PRINT   AT 11, 15 ; K
40  PAUSE   60
50  NEXT    K
```

Execute. Altere o programa para que a contagem se torne regressiva e seja exibida por extenso: DEZ, NOVE, . . . , UM. Sugestão: Colocar o extenso em uma constante ou variável e exibir a subcadeia. Reexecute.

5) Um programa que lê um número inteiro positivo e escreve por extenso cada algarismo é

```
10  DIM     V$ (10, 6)
20  FOR     J = 1 TO 10          60 caracteres (6 para cada extenso)
30  LET     V$ (J) = "ZERO UM DOIS TRES QUATRO CINCO SEIS SETE
            OITO NOVE" (6 * J - 5 TO 6 * J)
40  NEXT    J
50  PRINT   "ESCREVA NUMERO INTEIRO POSITIVO"
60  INPUT   N
70  LET     M = INT (ABS (N))
80  PRINT   "NUMERO = "; M
90  LET     M$ = STR$ (M)
100 LET     I = 1
110 IF      I > LEN (M$) THEN GOTO 150
120 PRINT   V$ (VAL (M$ (I)) + 1); " ";
130 LET     I = I + 1
140 GOTO    110
150 PRINT
160 PRINT   "TERMINO"
```

Execute. Modifique o programa para que escreva o extenso dos algarismos de valores numéricos fracionários (incluir o ponto fracionário). Reexecute. Altere de novo o programa para que escreva o extenso dos algarismos de números fracionários negativos. Reexecute.

6) Escreva um programa que leia o extenso de algarismos de um número inteiro positivo e gere o número, exibindo-os na tela. Execute-o.

7) Escreva um programa para ler palavras isoladas, armazená-las em um vetor ou matriz de caracteres de onde serão extraídas, separadamente, e exibidas na tela. Execute-o.

8) Escreva um programa que, aleatoriamente, gere símbolos gráficos e os armazene em uma matriz de 22 linhas por 32 colunas e depois exiba a matriz na tela. Execute-o.

9) Escreva um programa que leia caracteres com a função INKEY$ e os armazene para depois, a partir deste conjunto de caracteres, gerar e exibir uma matriz na tela, como no programa anterior. Execute-o.

OITAVO CAPÍTULO
FUNÇÃO DO USUÁRIO E SUB-ROTINA

função declaração e seu uso – sub-rotina e instruções relacionadas

8.1 Função declaração e seu uso

É muito comum que em um programa a *mesma fórmula* ou *expressão* seja utilizada diversas vezes. É o caso das *funções predefinidas* da linguagem BASIC, já apresentadas ao leitor na seção 6.1. Como essas *funções* são *predefinidas, não* há possibilidade de modificá-las, nem é necessário. Pode ocorrer, entretanto, que o leitor queira definir as suas próprias *funções*, por *não* haver nenhuma *função predefinida* que se ajuste à sua *fórmula*.

Uma das facilidades mais interessantes da linguagem BASIC é a possibilidade que tem o *usuário* de *definir* sua própria *função*. Há algumas condições para isto. Considere-se a *função linear*, da *forma*

$$a.x + b$$

Na figura 8-1 são apresentados dois textos de programa, um à esquerda, onde a *função linear* aparece várias vezes diretamente no texto, e outro à direita, onde a *função linear* é definida previamente através de uma *declaração de função* e, posteriormente, *referenciada* diversas vezes, da mesma maneira que qualquer *função predefinida*.

FIGURA 8-1: *EXEMPLO DE USO DA DECLARAÇÃO DE FUNÇÃO*

PROGRAMA SEM DECLARAÇÃO DE FUNÇÃO	PROGRAMA COM DECLARAÇÃO DE FUNÇÃO
10 ----------	10 DEF FNA (X) = 0.5 * X + 3
120 LET Y = (0.5 * X + 3)/2	120 LET Y = FNA (X)/2

240 LET Z = (0.5 * T + 3)/3	240 LET Z = FNA (T)/3
330 LET W = (0.5 * U + 3)/5	330 LET W = FNA (U)/5

Pode-se observar que o *argumento da* função FNA, X, é uma *variável formal* ou postiça, isto é, *não* tem nada a ver com a *variável* X do programa. No texto do programa o *argumento postiço* ou *formal* X é *substituído* por outras *variáveis*, respectivamente X (variável do programa) na *instrução* da *linha* 120, T na *instrução* da *linha* 240 e U na *instrução* da *linha* 330, que são os *argumentos verdadeiros* da *função* FNA. É muito importante distinguir entre o *argumento* ou *parâmetro postiço* ou *formal* da *declaração de função* e o *argumento* ou *parâmetro verdadeiro* da *referência* à *função*. Aí, precisamente, reside toda a *flexibilidade* da *função*. Mais alguns exemplos da *declaração de função* são apresentados na figura 8-2.

FIGURA 8-2: *EXEMPLOS DE DECLARAÇÃO DE FUNÇÃO (DECLARAÇÃO)*

EXEMPLOS	EXPLICAÇÃO
DEF FNX (M) = 5 * M ↑ 2 + 3 * M – 7	*definição da função* de *nome* FNX, com *argumento formal* M, como uma função do 2º grau em M.
DEF FNZ (X) = Y ↑ 2	*definição da função* FNZ, com *parâmetro postiço* X, como uma *função* que é o quadrado da *variável* Y, sem nenhuma relação com o *argumento* X, isto é, Y é uma *variável verdadeira* do programa.
DEF FNA (Q) = Q + R	*definição da função* FNA, com *argumento postiço* Q, como uma *função* cujo valor é o valor da soma entre o *argumento formal* Q e a *variável verdadeira* (do programa) R.

A forma geral da *instrução de declaração de função do usuário* é dada por

DEF FN < *letra* > (< *argumento formal* >) = < *expressão aritmética* > ,

onde as *palavras-chave*, **DEF FN**, representam **DEFINE FUNCTION**, isto é, *definir função*. O *argumento formal* ou *postiço* da *função declaração não* é uma *variável* do programa. Serve apenas para permitir, eventualmente, incluir o *argumento formal* na *expressão aritmética à direita do sinal de atribuição* (=). Todavia, como foi visto nos exemplos da figura 8-2, a *expressão aritmética* de *definição da função declaração* pode *não* incluir o *argumento* e conter apenas *constantes* ou *variáveis verdadeiras* do programa.

Dada a *forma* de *definição de função declaração*, o usuário tem a possibilidade de definir *até 26 funções*, ou seja, declarar *26 nomes de funções*, começando por FNA e indo até FNZ, isto é, *três letras* das quais as *duas primeiras* são, *obrigatoriamente*, FN e *uma letra qualquer* dentre as *26* do alfabeto.

A *declaração de função* é uma *instrução de especificação* e, portanto, *não-executável*, isto é, *nenhum* valor resulta da sua interpretação. Pode, portanto, em princípio ser colocada em *qualquer ponto* do programa. Todavia, para *compatibilidade* com diversas implementações, é conveniente colocá-la logo no *início* do programa, onde, *antes* de ser *referenciada*, será *avaliada*. Significa que muitas das principais implementações permitem a *reavaliação* da *mesma função*, ou seja, a *função* pode ser *declarada mais de uma vez* e, portanto, ser *definida* com *expressões* ou *fórmulas diferentes*. Em algumas, entretanto, *declarar* a *mesma função mais de uma vez* constituirá um *erro*. Para essas implementações que *não* permitem *reavaliação* e *não* exigem *avaliação antes* de *referência à função*, a *declaração* pode ser colocada, sem conseqüências, em *qualquer lugar* do programa, pois durante a execução do programa a *declaração* será *saltada*, sendo *executada somente* quando for *referenciada*. Considerar-se-á a *função declaração* como passível de *reavaliação*, para as explicações e exemplos deste livro.

A *instrução de declaração de função* (*declaração*) pode conter *qualquer expressão aritmética*, desde que caiba em uma *única linha* de programa.

Uma *função declaração* pode ser definida com base em *outras funções* declarações, desde que estas tenham sido *previamente declaradas* (e *avaliadas*). Chama-se a isto o *embutimento* ou *aninhamento* de *funções declaração*. Uma *função declaração* pode ser *redefinida* dentro do mesmo programa, durante sua execução, quando será *reavaliada*. Uma *função declaração não* pode ser *definida* em termos *dela mesma*, isto é, *não* pode haver *recursividade* (*definição de uma coisa* com base na *própria coisa* que se quer *definir*).

Um exemplo de *definição de uma função declaração* com *outra* é apresentado na figura 8-3, onde se encontra uma modificação da *definição da função declaração* do exemplo da figura 8-1.

FIGURA 8-3: *EXEMPLO DE DEFINIÇÃO DE FUNÇÃO DECLARAÇÃO COM OUTRA FUNÇÃO*

```
10  DEF   FNB (X) = Ø.5 * X
20  DEF   FNA (X) = FNB (X) + 3
------------------
------------------
```

Pode haver muitos *níveis* de *aninhamento* ou *embutimento* de *funções declaração*. O *número de níveis* varia de um sistema BASIC para outro, entre *5* e *8 níveis*. É aconselhável, portanto, *não* usar mais de *5* níveis, isto é, *não definir uma função declaração* com outras *funções declaração* que já estejam no *nível 6* de *embutimento* ou *aninhamento*.

A forma geral de uma *referência a uma função declaração* é dada por

FN <*letra*> (<*argumento verdadeiro*>) ,

onde a *palavra* FN representa FUNCTION, isto é, *função*. A *referência* à *função declaração* pode ser feita em *qualquer expressão aritmética*, da *mesma forma* que a *funções predefinidas*. O *argumento verdadeiro* deve ser do *mesmo tipo* que o *argumento formal*.

Os menores microcomputadores (primeiro grupo) não possuem a função declaração. Todavia, a falta da função declaração não retira dos menores micros a possibilidade de simplificar programas. Apenas, deve-se procurar substituí-la, por exemplo, pelo uso de sub-rotina. Embora a sub-rotina não tenha toda a flexibilidade da função, resolve satisfatoriamente o problema de evitar cálculos repetitivos. O uso de sub-rotina é objeto da próxima seção.

Com relação à *função declaração*, deve-se mencionar ainda que há alguns sistemas BASIC que permitem o uso de *não apenas um*, mas *vários argumentos*, embora *não* seja uma *facilidade* comumente disponível.

Um exemplo de *definição de função declaração* com *vários parâmetros* é apresentado na figura 8-4, o qual é também um exemplo de *aninhamento de funções declaração*.

FIGURA 8-4: *EXEMPLO DE USO DE FUNÇÃO DECLARAÇÃO COM VÁRIOS PARÂMETROS*

```
10  DEF   FNE (T) = 2 *T
20  DEF   FND (U, T) = U ↑ 2 + FNE (T)
30  DEF   FNC (V, U, T) = V/3 – FND (U, T)
40  DEF   FNB (X, V, U, T) = SIN (X)/FNC (V, U, T)
50  DEF   FNA (Y, X, V, U, T) = FNB (X, V, U, T)/Ø.75 * Y
------
------
240 LET   Z = Z ↑ 3/FNA (E, D, C, B, A)
------
------
```


Também se deve mencionar que *algumas implementações* da linguagem BASIC permitem a *definição de funções declaração não* apenas *numéricas*, mas também *alfanuméricas*. Neste caso, o valor que a *função retorna* é *alfanumérico* e *não numérico*. O *nome da função* deve incluir, *no final*, o caractere *cifrão* ($) e a *expressão à direita do sinal de atribuição* (=) deve ser, *necessariamente*, uma *expressão alfanumérica*.

Um exemplo de *definição de função declaração alfanumérica* é apresentado na figura 8-5.

FIGURA 8-5: *EXEMPLO DE FUNÇÃO DECLARAÇÃO ALFANUMÉRICA*

```
10  DEF     FNA$ (A$) = "ALO ", A$, "!!!"
20  INPUT   "NOME = ", A$
30  PRINT   FNA$ (A$)
```

No exemplo da figura 8-5, a *função declaração alfanumérica* FNA$, fornecido um *parâmetro alfanumérico* A$, produz a *concatenação* da *constante alfanumérica* ALO com o valor da *variável alfanumérica* A$. A *instrução* na *linha* 20 admite um *nome*, que será lido para a *variável* A$. A *instrução* na *linha* 30 *imprime* o valor da *função* FNA$, com base no valor de A$. Se, por exemplo, o *nome lido* para a *variável* A$ for constituído pelo valor JORGE, a *linha impressa* conterá

ALO JORGE !!! .

Evidentemente, se o sistema BASIC em questão admitir a *função declaração numérica* com mais de um *parâmetro*, deverá também admitir a *função declaração alfanumérica* com mais de um *parâmetro*.

8.2 Sub-rotina e instruções relacionadas

Muitas vezes, em lugar de uma única *fórmula* que aparece repetida em um programa, pode ocorrer que se repita uma *seqüência* constituída de *diver-*

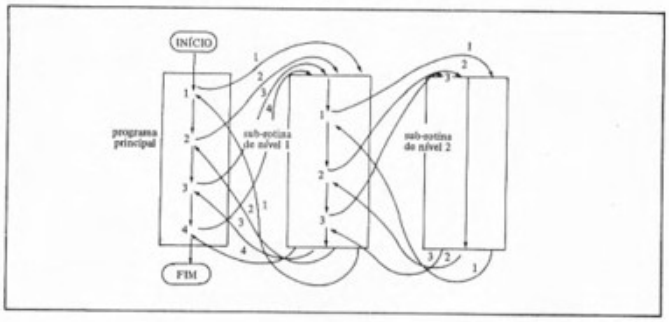
sas instruções. Aliás, de fato essa *repetição* de *instruções* no texto de programa deve ser evitada, por motivos óbvios. Para tal é necessário ter feito um bom *fluxograma* de programa. Na fase de *desenvolvimento da lógica*, ao se constatar que uma *mesma lógica*, correspondente a um *conjunto de instruções*, está *repetida* em *diversos pontos* do programa, deve ser feito um esforço para *evitar a repetição de instruções*. Isto é possível escrevendo-se um *conjunto de instruções isolado, fora da seqüência* do programa, que será *executado a partir de diversos pontos* onde a *lógica correspondente* ocorre no texto do *programa principal*.

Esse texto de programa isolado, que pode ser executado várias vezes a partir de diversos pontos de um programa, é a sub-rotina.

A grande vantagem de se usar a função é que o nome da função retorna um valor, quando a função é referenciada. Portanto, a função é muito útil quando de um cálculo retorna um só valor. Em alguns casos, os cálculos a serem realizados são mais complexos e de sua execução resultam diversos valores. Neste caso, a sub-rotina é a solução. Na linguagem BASIC, a sub-rotina não tem nome. A execução de uma sub-rotina é obtida por meio de uma instrução de desvio, **GOSUB**, que a inicia, e de uma instrução de retorno, **RETURN**, que encerra sua execução. O retorno de uma sub-rotina é feito sempre para a instrução seguinte à instrução **GOSUB** que iniciou sua execução.

Na figura 8-6, um *programa principal* é executado e, a partir de *diversos pontos* do *programa*, uma *sub-rotina* de *nível 1* é executada, *retornando-se sempre* à instrução *seguinte* à de desvio, para *prosseguimento* da *execução* seqüencial do programa. De *diversos pontos* desta *sub-rotina de nível 1*, outra *sub-rotina*, de *nível 2*, é *executada*.

FIGURA 8-6: *DIAGRAMAS DE EXECUÇÃO DE SUB-ROTINAS*



Um exemplo simples, com *dois níveis de sub-rotina*, é apresentado na figura 8-7.

FIGURA 8-7: *EXEMPLO DE USO DE SUB-ROTINAS*

```
10  LET   A$= "ESTE PROGRAMA TEM "
20  LET   B$= "DOIS NIVEIS DE SUB-ROTINA "
30  GOSUB 50
40  GOTO  100
50  PRINT A$;
60  GOSUB 80
70  RETURN
80  PRINT B$
90  RETURN
100 STOP
```

Depois de passar pelas *instruções* das *linhas* 10 a 20, de *atribuição de valores*, o *desvio* para a primeira *sub-rotina* (*nível 1*) ocorre na *linha* 30. A *instrução* **GOSUB** da *linha* 30 provoca um *desvio* para a *sub-rotina* na *linha* 50. Na *linha* 50 é feita a *impressão* com uma nova *linha*, do *valor* da *variável alfanumérica* A$, ou seja, ESTE PROGRAMA TEM. Dentro da *sub-rotina* de *nível 1*, na *instrução* 60, é feito novo *desvio* para a *sub-rotina* de *nível 2*, com a *instrução* **GOSUB**, que se inicia na *linha* 80. A execução da *linha* 80 provoca a *impressão* do conteúdo da *variável alfanumérica* B$, na *mesma linha* e *sem espaço*, ou seja, do valor DOIS NIVEIS DE SUB-ROTINA. Agora, a *linha de impressão* contém

```
ESTE PROGRAMA TEM DOIS NIVEIS DE SUB-ROTINA
```

Com a *instrução* **RETURN** da *linha* 90, é feito o *retorno da sub-rotina* de *nível 2* para a *sub-rotina* de *nível 1*, na *instrução* seguinte à de desvio. Esta *instrução* é a *instrução* de *número de linha* 70, que é um **RETURN** também, só que agora significando um *retorno da sub-rotina de nível 1* para o *programa principal*. O *retorno ao programa principal* se faz na *instrução* seguinte à que provocou o *desvio* para a *sub-rotina*, ou seja, a da *linha* 40, que é um **GOTO**, que provoca um desvio para a *linha* 100. A linha 100 contém a *instrução* **STOP**, que é o *término lógico* do programa. O leitor deve observar que as *sub-rotinas* constituem *trechos isolados* de programa, isto é, é *impossível*, a partir do pro-

grama principal, fazer um *desvio para dentro* ou *entrar na sub-rotina*, a *não* ser pela execução de uma *instrução* **GOSUB**.

Um exemplo de *instrução de desvio para sub-rotina* é apresentado na figura 8-8.

FIGURA 8-8: *EXEMPLO DE INSTRUÇÃO DE DESVIO PARA SUB-ROTINA*

EXEMPLO	EXPLICAÇÃO
GOSUB 1000	*Desvio para a sub-rotina* que se inicia na linha de número **1000**.

A forma geral da *instrução de desvio para sub-rotina* é dada por

GOSUB < *número de linha* > ,

onde a *palavra-chave*, **GOSUB**, representa **GO SUBROUTINE**, isto é, *vá para sub-rotina.*

Nos menores microcomputadores (primeiro grupo), o *número de linha* pode ser obtido como o valor inteiro de uma *expressão aritmética*, o que confere à instrução **GOSUB** neste caso, uma grande flexibilidade. Torna-se possível desviar não apenas para uma única linha mas para diversas, com uma única instrução **GOSUB**, dependendo do resultado da avaliação da expressão aritmética.

A *instrução* **GOSUB** é uma *instrução de controle*, da mesma maneira que a *instrução* **GOTO** o é. Em outras palavras, a *instrução* **GOSUB** é uma *instrução de desvio incondicional para uma sub-rotina.* O *número da linha* que é especificado na *instrução* tem que ser o da *linha* onde se encontra a *primeira instrução da sub-rotina. Após* o *desvio*, a execução prossegue *seqüencialmente*, com a *instrução especificada* e as *instruções seguintes*, até encontrar uma *instrução* **RETURN.** A instrução **RETURN** provoca o *retorno da sub-rotina*, ou seja, *termina a execução da sub-rotina.* Dentro de uma *sub-rotina* pode haver *desvios* para *outras sub-rotinas em níveis mais baixos* e assim *sucessivamente*, chamando-se a essas *sub-rotinas* de *aninhadas* ou *embutidas*. O *aninhamento* ou *embutimento* de *sub-rotinas* pode ser feito, tipicamente, *até 8 níveis.*

Um exemplo da *instrução de retorno* ou *volta de sub-rotina* é apresentado na figura 8-9.

FIGURA 8-9: *EXEMPLO DE INSTRUÇÃO DE RETORNO DE SUB-ROTINA*

EXEMPLO	EXPLICAÇÃO
RETURN	Retornar de sub-rotina para a instrução seguinte à que provocou o desvio, no programa principal ou sub-rotina mais externa.

A forma geral da *instrução de retorno de sub-rotina* é dada por

RETURN .

onde a *palavra-chave*, **RETURN**, significa *retornar.*

A *instrução de retorno de sub-rotina* provoca o *encerramento da sub-rotina* e a *volta para a instrução seguinte à de desvio incondicional para sub-rotina* que provocou o *desvio*, na *sub-rotina mais externa* ou no *programa principal*, isto é, no *nível imediatamente superior.*

Em uma *sub-rotina* pode haver *não* apenas uma *instrução* **RETURN**, mas *diversas.* Em *qualquer ponto* onde a lógica indique que deva haver um *retorno* em uma *sub-rotina*, deve ser colocada uma *instrução* **RETURN.** Todavia, qualquer *sub-rotina* tem que ter *pelo menos uma instrução* **RETURN**, sem o que é *impossível retornar da sub-rotina*. Em princípio, *qualquer instrução* pode ser usada numa *sub-rotina*. Todavia, *não* podem ser feitos *desvios* para fora do *alcance* ou *corpo da sub-rotina*, assim como *não se pode desviar* do *programa principal* para dentro do *alcance de uma sub-rotina*. Uma maneira de isolar uma *sub-rotina* do restante do *programa principal*, evitando uma execução inadvertida, é usar uma *instrução* **STOP** ou **GOTO** *imediatamente antes.* Embora a *sub-rotina* possa ficar em qualquer ponto do programa principal, por uma questão de *organização*, é conveniente colocá-la *sempre no final* de programa.

Um problema familiar ao leitor é a solução do trinômio do 2º grau, que será tomado novamente como exemplo, com uma pequena diferença em relação à solução anterior, no sentido de que deverão ser calculadas também as raízes complexas. Uma primeira solução do problema se encontra na figura 8-10.

FIGURA 8-10: *EXEMPLO DE PROGRAMA PARA SOLUÇÃO DO TRINÔMIO DO 2º GRAU, CALCULANDO AS RAÍZES COMPLEXAS*

```
10 PRINT "SOLUCAO DO TRINOMIO"
20 PRINT "DO SEGUNDO GRAU"
30 PRINT "A=";
40 INPUT A
```

```
50  PRINT   A
60  IF      A = 0 THEN GOTO 360
70  PRINT   "B=";
80  INPUT   B
90  PRINT   B
100 PRINT   "C=";
110 INPUT   C
120 PRINT   C
130 IF      B>=0 THEN GOTO 170
140 LET     A = - A
150 LET     B = - B
160 LET     C = - C
170 LET     D = B ** 2 - 4 * A * C
180 IF      D <0 THEN GOTO 300
190 IF      D = 0 THEN GOTO 260
200 PRINT   "RAIZES REAIS E DESIGUAIS"
210 LET     P1 = -(B/ (2 * A ))
220 LET     P2 = SQR (D)/ (2 * A)
230 PRINT   "X1="; P1 + P2
240 PRINT   "X2="; P1 - P2
250 GOTO    370
260 PRINT   "RAIZES REAIS E IGUAIS"
270 LET     P1 = -(B/ (2 * A))
280 PRINT   "X1 = X2 ="; P1
290 GOTO    370
300 PRINT   "RAIZES IMAGINARIAS"
310 LET     P1 = -(B/ (2 * A))
320 LET     P2 = SQR (-D)/ (2 * A)
330 PRINT   "X1=("; P1; ") + ("; P2; ") I"
340 PRINT   "X2=("; P1; ") - ("; P2; ") I"
350 GOTO    370
360 PRINT   "NAO HA TRINOMIO"
370 PRINT   "TERMINO"
```

Pode-se observar, na figura 8-10, que o cálculo da parte 1 da expressão da raíz, P1, está repetido nas linhas 210, 270 e 310. Da mesma forma, pode-se observar a repetição da parte 2, P2, que se repete nas linhas 220 e 320, com a diferença apenas do sinal do determinante, D. O cálculo desses valores pode ser definido por instruções colocadas em um único local do programa, como *sub-rotina*. A execução será feita nos três pontos do programa onde é requerida para o cálculo das raízes, conforme o valor do determinante, D, seja negativo, zero ou positivo, através de uma instrução de desvio para sub-rotina, **GOSUB**. O programa resultante da aplicação dessa técnica é o constante da figura 8-11.

FIGURA 8-11: *EXEMPLO DE PROGRAMA (MODIFICADO) PARA SOLUÇÃO DO TRINÔMIO DO 2º GRAU, COM O USO DE SUB-ROTINA*

```
10  PRINT   "SOLUCAO DO TRINOMIO"
20  PRINT   "DO SEGUNDO GRAU"
30  PRINT   "A=";
40  INPUT   A
50  PRINT   A
60  IF      A = 0 THEN GOTO 360
70  PRINT   "B=";
80  INPUT   B
90  PRINT   B
100 PRINT   "C=";
110 INPUT   C
120 PRINT   C
130 IF      B>=0 THEN GOTO 170
140 LET     A = - A
150 LET     B = - B
160 LET     C = - C
170 LET     D = B ** 2 - 4*A *C
180 IF      D < 0 THEN GOTO 300
190 IF      D = 0 THEN GOTO 260
200 PRINT   "RAIZES REAIS E DESIGUAIS"
210 GOSUB   390
230 PRINT   "X1="; P1 + P2
240 PRINT   "X2="; P1 - P2
250 GOTO    370
260 PRINT   "RAIZES REAIS E IGUAIS"
270 GOSUB   390
280 PRINT   "X1 = X2 ="; P1
290 GOTO    370
300 PRINT   "RAIZES IMAGINARIAS"
310 LET     D = -D
320 GOSUB   390
330 PRINT   "X1=("; P1; ") + ("; P2; ") I"
340 PRINT   "X2=("; P1; ") - ("; P2; ") I"
350 GOTO    370
360 PRINT   "NAO HA TRINOMIO"
370 PRINT   "TERMINO"
380 STOP
390 LET     P1 = -(B/ (2 * A ))
400 LET     P2 = SQR ( D )/ (2 * A)
410 RETURN
```

O leitor pode pensar, com base no exemplo da figura 8-11, que não houve ganho, em termos de números de instruções, com o uso da sub-rotina. Tem razão, porque o número de expressões que foram calculadas é pequeno (duas), no exemplo apresentado. Todavia, pretende-se exemplificar a técnica de uso

da sub-rotina mais do ponto de vista de como fazê-lo. Naturalmente, nos casos concretos que surgirem, haverá ganhos significativos para mais de duas expressões a calcular, isto é, com um maior número de linhas a substituir pelo desvio para sub-rotina, **GOSUB**.

Um programa de demonstração de como utilizar a instrução **GOSUB** nos menores micros (primeiro grupo), como *desvio múltiplo para sub-rotina*, encontra-se na figura 8-12.

FIGURA 8-12: *PROGRAMA DE DEMONSTRAÇÃO DO USO DA INSTRUÇÃO* **GOSUB** *COM DESVIO MÚLTIPLO, NOS MENORES MICROS (1º GRUPO)*

```
10  PRINT   "INFORME UM NUMERO ENTRE 0 E 10"
20  INPUT   X
30  PRINT   "O NUMERO EH ";
40  GOSUB   (50 *X + 100)
50  PRINT   "TERMINO"
60  STOP
100 PRINT   "ZERO"
110 RETURN
150 PRINT   "UM"
160 RETURN
200 PRINT   "DOIS"
210 RETURN
250 PRINT   "TRES"
260 RETURN
300 PRINT   "QUATRO"
310 RETURN
350 PRINT   "CINCO"
360 RETURN
400 PRINT   "SEIS"
410 RETURN
450 PRINT   "SETE"
460 RETURN
500 PRINT   "OITO"
510 RETURN
550 PRINT   "NOVE"
560 RETURN
600 PRINT   "DEZ"
610 RETURN
```

Nos microcomputadores pessoais do segundo grupo e nos maiores, a instrução de desvio múltiplo para sub-rotina é a instrução **ON . . . GOSUB**. Neste caso, o desvio é feito com base em um índice, para uma entre diversas instruções cujos números de linhas estão enumerados, como uma lista, na instrução.

Alguns exemplos da instrução de desvio incondicional múltiplo para sub-rotina são encontrados na figura 8-13.

FIGURA 8-13: *EXEMPLOS DE INSTRUÇÃO DE DESVIO INCONDICIONAL MÚLTIPLO PARA SUB-ROTINA*

EXEMPLO	EXPLICAÇÃO
ON J GOSUB 100, 200, 300	ocorrerá um desvio incondicional para uma das sub-rotinas que se iniciam por um dos números de linhas constantes da lista, de acordo com o valor do índice, isto é, o valor inteiro da variável numérica J: • se J = 1, desvio para a linha 100 • se J = 2, desvio para a linha 200 • se J = 3, desvio para a linha 300
ON 3 * K – 10 GOSUB 500, 700	ocorrerá um desvio incondicional para uma das sub-rotinas que se iniciam por um dos números de linhas constantes da lista, de acordo com o valor do índice, isto é, o valor inteiro da expressão 3 * K – 10, respectivamente: • se expressão=1, desvio para linha 500 • se expressão=2, desvio para linha 700

A forma geral da instrução de *desvio incondicional múltiplo para sub-rotina* é dada por

ON <*expressão aritmética*> **GOSUB** <*número de linha*> [, <*número de linha*>] . . .

O valor da *expressão aritmética*, depois de calculado, é truncado, isto é, é tomado o seu valor *inteiro*. Se este valor é menor do que 1 ou maior do que o número de *números de linhas* da lista à direita, dependendo do sistema BASIC, poderá ocorrer uma mensagem de erro ou simplesmente a transferência da execução para a instrução imediatamente seguinte. Normalmente a execução continuará na linha para a qual foi feito o desvio. O mesmo *número de linha* poderá aparecer diversas vezes na lista à direita. A instrução **ON . . . GOSUB** não está disponível nos menores micros (primeiro grupo).

Exercícios:

ESTUDO DIRIGIDO:

1) Que é função do usuário? Que é função declaração? Que é declaração de função? Qual a forma geral da função declaração? Quais as palavras-chave da declaração de função do usuário? Que é referência a uma função do usuário? Uma referência à função do usuário é feita como? Expressões aritméticas podem incluir referências a funções do usuário? Como atribuir nome à função declaração? Quantas alternativas existem para o nome da função do usuário? Quantos argumentos pode ter uma função do usuário? A variável usada como argumento formal pertence ao programa? Que é argumento verdadeiro? O argumento formal tem que entrar na formação da expressão aritmética à direita do sinal de atribuição (=)? As variáveis que entram na formação da expressão aritmética à direita do sinal de atribuição (=) podem ser variáveis do programa (além do argumento)? Podem ser incluídas constantes na expressão aritmética da função declaração? Uma função declaração pode ser definida com base em outra função declaração previamente declarada? Quantos níveis de embutimento ou aninhamento de funções declaração são usuais? Os menores microcomputadores (primeiro grupo) possuem a função declaração? Existem versões de BASIC que permitem mais de um argumento da função declaração? Pode-se declarar uma função alfanumérica? Qual a forma de declarar uma função alfanumérica? Como referenciá-la?

2) Qual a diferença entre uma função e uma sub-rotina? Qual a utilidade de funções e sub-rotinas? Em que casos devem ser usadas as funções? Em que casos devem ser usadas sub-rotinas? Quais as instruções usadas para desviar para e retornar de uma sub-rotina? Como isolar uma sub-rotina do restante do programa? Quais as instruções usadas para isolar uma sub-rotina de outras partes do programa? Uma sub-rotina em BASIC tem nome? Uma sub-rotina em BASIC tem argumentos? Que tipos de instruções podem compor uma sub-rotina? Dentro de uma sub-rotina pode haver o desvio para outras sub-rotinas? Quantos níveis de embutimento ou aninhamento são usuais? Qual a forma geral da instrução de desvio incondicional para sub-rotina? Nos menores micros (primeiro grupo), qual a forma da instrução **GOSUB**? Como pode ser usada a instrução **GOSUB** dos pequenos micros para execução de desvios múltiplos? Nos demais microcomputadores, qual a instrução de desvio múltiplo para sub-rotina? Qual a forma geral da instrução **ON . . . GOSUB**? Para que serve a instrução de retorno de sub-rotina? Em que instrução do programa continua a execução, após a execução da instrução **RETURN** de uma sub-rotina? A instrução **ON . . . GOSUB** está disponível nos menores micros? Como substituí-la, neste caso?

PROBLEMAS:

1) Uma função y existe apenas no intervalo de 0 a 100 de uma variável x e é definida formalmente de maneira diferente em determinados intervalos de x. Os intervalos e fórmulas são:

INTERVALO	FÓRMULA
0 – 29.999 . . .	3 x + 4
30 – 59.999 . . .	5 x + 7
60 – 89.999 . . .	–5 x + 7
90 – 99.999. . .	–3 x + 4

Um programa que calcula o valor da função, dado um valor de x, é

```
10  DEF   FNA (X) = 3 * X + 4
20  DEF   FNB (X) = 5 * X + 7
30  DEF   FNC (X) = –5 * X + 7
40  DEF   FND (X) = –3 * + X 4
50  PRINT "CALCULO DE FUNCAO"
60  PRINT "DE UMA VARIAVEL"
70  PRINT "INFORME VALOR ENTRE 0 E 100"
80  INPUT X
90  PRINT "X="; X
100 IF    X<0 THEN LET X = 0
110 IF    X>=100 THEN LET X = 99.999999
120 IF    X>= 90 THEN LET Y = FND (X)
130 IF    X>= 60 AND X< 90 THEN LET Y = FNC (X)
140 IF    X>= 30 AND X < 60 THEN LET Y = FNB (X)
150 IF    X< 30 THEN LET Y = FNA (X)
160 PRINT "Y="; Y
170 PRINT "TERMINO"
```

Executar. Alterar o programa para que apenas uma função declaração seja utilizada. A título de sugestão, use como coeficientes variáveis do programa, aos quais seja atribuído valor, dependendo do intervalo em que se encontre o valor de X. Reexecute, com os mesmos dados e compare. Se o seu BASIC permitir, uma alternativa é declarar uma função em que entrem como argumentos formais os coeficientes, além da variável livre, da função. Se o BASIC do seu micro não tem função declaração, transforme o programa para utilização de sub-rotinas.

2) Deseja-se conhecer o valor de várias funções, y, z, t e w de uma variável x, para um mesmo valor de x. As funções têm a forma:

$$y = 3x + 4$$
$$z = -4x - 5$$
$$t = 4x - 3$$
$$w = -5x + 3$$

Um programa que permite este cálculo é

```
10  PRINT   "CALCULO DOS VALORES"
20  PRINT   "DE VARIAS FUNCOES"
30  PRINT   "DE UMA VARIAVEL"
40  PRINT   "INFORME VALOR VARIAVEL"
50  INPUT   X
60  PRINT   "X="; X
70  LET     A = 3
80  LET     B = 4
90  GOSUB   250
100 PRINT   "Y="; V
110 LET     A = -4
120 LET     B = -5
130 GOSUB   250
140 PRINT   "Z="; V
150 LET     A = 4
160 LET     B = -3
170 GOSUB   250
180 PRINT   "T="; V
190 LET     A = -5
200 LET     B = 3
210 GOSUB   250
220 PRINT   "W="; V
230 PRINT   "TERMINO"
240 STOP
250 LET     V = A * X + B
260 RETURN
```

Execute. Altere o programa para que a sub-rotina seja substituída por uma função declaração, se o seu BASIC permitir. Alternativamente, a sub-rotina pode ser alterada de forma a exibir na tela, por meio da instrução **PRINT**, o nome da função e o seu valor, substituindo-se as atuais instruções de exibição por novas, de atribuição alfanumérica, do nome da função a uma variável alfanumérica, antes do desvio para a sub-rotina. Reexecute. Compare os resultados com os obtidos com a forma anterior do programa.

3) Uma função *y* é definida com base em uma variável *x*, no intervalo de 0 a 100 de *x*. A definição dos valores de *y* é

valores de *x*	valor da função
0 a 39.999999	y = 5
40 a 59.999999	y = 7
60 a 99.999999	y = 9

Um programa que permite calcular o valor de *y*, dentro do intervalo especificado para *x* é


```
10  PRINT   "FUNCAO NO INTERVALO 0-100"
20  PRINT   "INFORME VALOR NO INTERVALO"
30  INPUT   X
40  PRINT   "X="; X
50  IF      X < 0 THEN LET X = 0
60  IF      X >= 100 THEN LET X = 99.999999
70  IF      X >= 60 THEN LET Y = 9
80  IF      X >= 40 AND X < 60 THEN LET Y = 7
90  IF      X < 40 THEN LET Y = 5
100 PRINT   "Y="; Y
110 PRINT   "TERMINO"
```

Executar. Alterar o programa para que o cálculo do valor da função, isto é, as atuais linhas 70, 80 e 90, seja substituído por um desvio para sub-rotina, isto é, essas instruções passem para uma sub-rotina, no final do programa. Reexecute e compare os valores. Altere novamente o programa, para que a exibição de Y seja feita diretamente em cada instrução **IF** de teste de determinação do intervalo de X. Reexecute e confira os resultados.

4) Um programa que faz sugestões para um cardápio de verduras com baixo conteúdo de calorias é

```
10  DIM     A (10)
20  PRINT   "SUGESTOES PARA CARDAPIO"
30  PRINT   "ATE 10 VERDURAS"
40  PRINT   "QUANTAS SUGESTOES QUER?"
50  INPUT   N
60  PRINT   "VOCE QUER "; N; " SUGESTOES"
70  PRINT
80  IF      N < 1 THEN LET N = 1
90  IF      N > 10 THEN LET N = 10
100 RAND
110 FOR     I = 1 TO N
120 LET     J = INT (RND * 10) + 1
130 FOR     L = 1 TO I - 1
140 IF      J = A(L) THEN GOTO 120
150 NEXT    L
160 LET     A (I) = J
170 GOSUB   (20 * (J-1)) + 300
180 NEXT    I
190 PRINT
200 PRINT   "TERMINO"
210 STOP
300 PRINT   "ALFACE"
310 RETURN
320 PRINT   "AIPO"
330 RETURN
340 PRINT   "BERTALHA"
350 RETURN
```

```
360 PRINT "BROCOLIS"
370 RETURN
380 PRINT "CHICORIA"
390 RETURN
400 PRINT "COUVE"
410 RETURN
420 PRINT "COUVE-FLOR"
430 RETURN
440 PRINT "ESPINAFRE"
450 RETURN
460 PRINT "REPOLHO"
470 RETURN
480 PRINT "TAIOBA"
490 RETURN
```

Se o BASIC do seu micro não tem a instrução **GOSUB** na forma como aparece na linha 170, substitua-a por **ON . . . GOTO**, usando como índice o valor de J. Execute. Modifique o programa para que, alternativamente, sejam escolhidos também legumes, igualmente em número de 10, que são: abobrinha-d'água, berinjela, cebola, jiló, maxixe, palmito, pepino, pimentão, rabanete e tomate. Reexecute. Se o BASIC do seu micro dispõe de função declaração alfanumérica, coloque os nomes dos vegetais em matrizes de caracteres e declare uma função do usuário alfanumérica que retorne o nome do vegetal com base no valor do argumento, que funcionará como índice das matrizes. Reexecute.

5) Se o leitor deseja complementar as sugestões para o cardápio de vegetais para incluir frutas, uma lista é: abacate, ameixa fresca, tangerina, morango, abacaxi, laranja, melão, melancia, maracujá, maçã. Escreva um programa que sugira um cardápio de frutas (até 10). Os nomes das frutas devem ser colocados em uma matriz de caracteres. O nome de cada fruta selecionada deve ser exibido por uma instrução **PRINT** em uma sub-rotina única, utilizando-se para índice da matriz o valor que é gerado com auxílio da função RND. Execute. Se o BASIC do seu micro dispõe de função declaração alfanumérica, em vez de sub-rotina, use a função declaração alfanumérica para obter o nome de cada fruta. Reexecute. Alternativamente, inclua a escolha opcional de frutas no programa do exercício anterior, junto com os vegetais. Reexecute.

6) Você pode impressionar o seu público falando o *economês*, a linguagem dos *tecnoburocratas*. É muito fácil escrever um discurso com expressões que nada significam mas que são empoladas. Basta dispor de um programa para gerar expressões nessa estranha linguagem. Coloque cada grupo de palavras em uma matriz de caracteres. Os três grupos são

1º grupo		2º grupo		3º grupo	
código	palavra	código	palavra	código	palavra
1	PROGRAMAÇÃO	1	FUNCIONAL	1	SISTEMÁTICA
2	ESTRATÉGIA	2	OPERACIONAL	2	INTEGRADA
3	MOBILIDADE	3	DIMENSIONAL	3	EQUILIBRADA
4	PLANIFICAÇÃO	4	TRANSACIONAL	4	TOTALIZADA
5	DINÂMICA	5	ESTRUTURAL	5	INSUMIDA
6	FLEXIBILIDADE	6	GLOBAL	6	BALANCEADA
7	IMPLEMENTAÇÃO	7	DIRECIONAL	7	COORDENADA
8	INSTRUMENTAÇÃO	8	OPCIONAL	8	COMBINADA
9	RETROAÇÃO	9	CENTRAL	9	ESTABILIZADA
10	PROJEÇÃO	10	LOGÍSTICA	10	PARALELA

Usando a função RND, gere três valores pseudo-aleatórios entre 1 e 10, cada um dos quais será usado como índice de uma matriz de caracteres. Esse índice corresponde ao código de cada palavra. Uma expressão será formada com uma palavra de cada grupo. Por exemplo, se os aleatórios gerados forem 5, 3 e 9, a expressão correspondente é DINÂMICA TRANSACIONAL ESTABILIZADA. Escreva o programa, usando sub-rotina ou função. Execute.

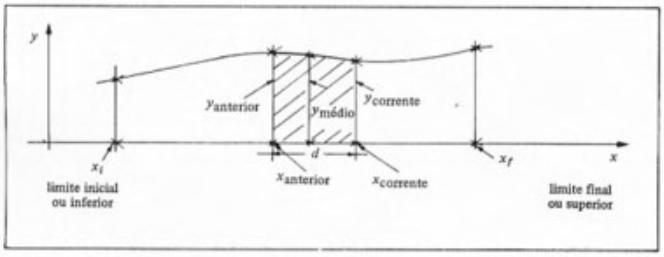
7) O valor de um polinômio é obtido pela soma do valor de todos os seus termos. A forma geral de um polinômio de uma variável, do grau m, é

$$a_0x^m + a_1x^{m-1} + a_2x^{m-2} + \ldots + a_{m-1}x + a_m$$

Um polinômio do grau m tem $m+1$ termos. Escrever um programa que calcule a soma dos termos até um determinado grau. Estabelecer o grau máximo do polinômio. O valor dos coeficientes é fixo. Utilizar sub-rotina para o cálculo de cada termo e a acumulação do valor do polinômio.

8) O cálculo da área sob uma curva, representada por uma função y de uma única variável x, entre dois limites de x, pode ser obtido por diversos métodos. Um deles é conhecido por aproximação trapezoidal. Consiste em acumular o valor de pequenas áreas, com a forma de um pequeno trapézio, que dão uma aproximação da área sob a curva entre duas ordenadas consecutivas. A soma de todas essas pequenas áreas é uma aproximação razoável da área entre a curva, o eixo das abscissas e as ordenadas que passam pelos limites. Escrever um programa que calcule a área sob uma curva, entre dois limites estabelecidos para as abscissas x, utilizando sub-rotina para calcular a área de cada pequeno trapézio e acumular. É preciso estabelecer os limites do intervalo de x dentro do qual a curva está definida. A área de cada pequeno trapézio é calculada pela fórmula: $a = y_{médio} \times d$. O valor de y_{med} é o da ordenada média, isto é, a média das ordenadas que limitam pequeno trapézio, ou seja, $(y_{anterior} + y_{corrente})/2$. A primeira ordenada $y_{anterior}$ é calculada no limite inferior do intervalo, tomando-se $x_{anterior} = limite\ inferior$. O valor de $x_{corrente}$ é calculado sempre por $x_{corrente} = x_{anterior} + d$. O valor de $x_{corrente}$ será

calculado até que seja atingido o *limite superior* do intervalo. O incremento de área, d, deve ser escolhido de forma que o cálculo da área tenha uma razoável precisão. O gráfico abaixo ilustra o significado das variáveis envolvidas.



NONO CAPÍTULO
OPERAÇÕES COM TABELAS

tabelas – classificação de tabelas – pesquisa em tabelas

9.1 Tabelas

Todas as pessoas têm uma noção do que seja uma tabela. As tabelas se nos apresentam no nosso dia-a-dia. O exemplo mais comum é a lista telefônica. Para os técnicos, especialmente engenheiros, a tabela faz parte do cotidiano, não apenas dos que projetam mas também dos que executam. As tabelas seriam inúteis ou muito difíceis de utilizar se não estivessem ordenadas, classificadas, segundo um determinado campo de valor (coluna da tabela). Pode-se encontrar um telefone na lista porque os nomes dos assinantes se encontram ordenados, classificados, por ordem alfabética de nomes (em particular, sobrenomes). São possíveis outras classificações da lista telefônica, como é o caso da lista por endereços. Embora inexistente, é viável a lista telefônica por número de telefone, isto é, classificada na ordem crescente dos telefones. A classificação mais usual é a crescente, embora seja possível também a decrescente. As listas telefônicas, por exemplo, estão classificadas em ordem alfabética crescente, tanto a de assinantes como a de endereços. Para não mencionar as páginas amarelas, que incluem apenas os telefones com interesse comercial. Portanto, obtendo-se uma tabela qualquer, um dos problemas para sua organização é o da classificação ou ordenação. Além disso, as tabelas só são úteis porque podem ser consultadas. Para consultar uma tabela, diversos métodos existem, dependentes da organização da tabela. Alguns métodos de pesquisa são mais simples e outros mais complexos. Alguns métodos de pesquisa são mais eficientes e outros menos eficientes. Isto é, um método de pesquisa pode ser mais rápido do que outro.

Tratando-se de computador, um problema que existe com relação às tabelas é a sua implementação. Isto é, o armazenamento de uma tabela em computador requer a escolha de uma técnica. A escolha dessa técnica está ligada basicamente à necessidade de classificar e pesquisar a tabela. Todavia, o formato da tabela é dependente também das operações que serão realizadas sobre os dados contidos na tabela. Afinal, o que são tabelas? Pode-se dizer que

são estruturas nas quais se podem armazenar e recuperar dados. Em outros termos, são dispositivos onde se podem guardar dados que depois serão consultados. A implementação desses dispositivos pode ser realizada de forma seqüencial ou encadeada. A implementação seqüencial utiliza a contigüidade física para realizar a tabela. A implementação encadeada utiliza um artifício que permite realizar a tabela independentemente de contigüidade física, através da introdução de dados auxiliares, os chamados apontadores. Esta última forma não será abordada neste livro. Na figura 9-1 são apresentados os dados contidos em uma tabela e dois possíveis formatos para implementação da tabela.

FIGURA 9-1: *EXEMPLO DE IMPLEMENTAÇÃO DE UMA TABELA EM DOIS FORMATOS*

Tabela:

NOME (até 30 caracteres)	ENDEREÇO (até 40 caracteres)	TELEFONE (até 10 caracteres)
Pedro Álvares Cabral	Torre do Tombo, 7º andar	267-0000
Cristóvão Colombo	Praça de Touros	237-9999
Américo Vespúcio	Rua da Inquisição, s/n	999-1111

Implementação como três vetores:

DIM X$ (3, 30)

PEDRO ÁLVARES CABRAL
CRISTÓVÃO COLOMBO
AMÉRICO VESPÚCIO

DIM Y$ (3, 40)

TORRE DO TOMBO, 7º ANDAR
PRAÇA DE TOUROS
RUA DA INQUISIÇÃO S/N

DIM Z$(3,10)

267-0000
237-9999
999-1111

Implementação como um único vetor onde cada cadeia de caracteres está subdividida em subcadeias (campos)

DIM W$ (3, 80)

PEDRO ÁLVARES CABRAL	TORRE DO TOMBO, 7º ANDAR 267-0000
CRISTÓVÃO COLOMBO	PRAÇA DE TOUROS 237-9999
AMÉRICO VESPÚCIO	RUA DA INQUISIÇÃO S/N 999-1111

No caso de implementação da tabela do exemplo da figura 9-1 por três matrizes de caracteres distintos, **X$** para o nome, **Y$** para o endereço e **Z$** para o telefone, uma referência ao nome de Pedro Álvares Cabral é feita como **X$(1)**, uma referência ao endereço de Cristóvão Colombo é feita como **Y$ (2)** e uma referência ao telefone de Américo Vespúcio é feita como **Z$ (3)**.

Se a implementação da tabela for feita por uma única matriz de caracteres, **W$**, uma referência ao nome de Pedro Álvares Cabral é feita como **W$(1 ,1 TO 30)** ou **W$ (1) (1 TO 30)**, uma referência ao endereço de Cristóvão Colombo é feita como **W$ (2, 31 TO 70)** ou **W$ (2) (31 TO 70)**, e uma referência ao telefone de Américo Vespúcio é feita como **W$(3, 71 TO 80)** ou **W$ (3) (71 TO 80)**. Essa forma de referência à subcadeia é a usual nos menores micros. Nos demais, a subcadeia é referenciada por uma função de manipulação de cadeias de caracteres.

As principais operações sobre tabelas estão relacionadas com a sua criação e manutenção. A criação consiste em colocar dados na tabela pela primeira vez. O que se chama de manutenção de tabelas são as operações que ocorrem depois da sua criação. Entre as operações de manutenção mais comuns encontram-se a inclusão de novos registros, a remoção de registros (retirada de dados da tabela), o cancelamento de registros (a marcação de registros para posterior remoção) e a alteração (modificação de dados) de registros. O que se chama de *registro* em uma tabela corresponde a uma *linha* da tabela, isto é, um conjunto de dados correlacionados. Além dessas operações, citam-se a cópia de tabelas, a classificação de tabelas e a pesquisa de tabelas.

Nos menores microcomputadores (primeiro grupo) que *não* dispõem do conceito de *arquivo*, isto é, *não* dispõem de *instruções de entrada e saída* para armazenamento e recuperação de dados permanentes, as tabelas constituem um importante recurso. Quando os programas são armazenados em fita cassete, as estruturas de dados (variáveis do programa) que se encontram na memória, junto com o programa, são também armazenadas na fita cassete. Esse artifício permite armazenar os dados, junto com os programas, de forma permanente. A implementação de tabelas, armazenadas em matrizes, supre a falta do conceito de arquivo e de instruções para manipulação de arquivos. Quando o programa for lido a partir da fita cassete para ser carregado na memória, as estruturas de dados que foram armazenadas junto com o programa também serão carregadas na memória. O detalhe importante é que esses programas não podem ser executados através do comando **RUN**, porque senão, com a inicialização das estruturas de dados, os dados armazenados serão perdidos (apagados). De novo, um artifício deve ser usado, iniciando-se a execução do programa por um *comando* **GOTO** (*não* uma instrução). Com o comando **GOTO** o programa será iniciado, na instrução cujo número de linha for dado, *sem* inicialização das estruturas de dados (variáveis). É possível, portanto, retomar o

processamento dos dados no ponto em que este foi encerrado (no estado em que ficaram ao final da execução anterior).

De todas as operações mencionadas, aplicáveis às tabelas, serão descritas algumas das principais técnicas de classificação e de pesquisa de tabelas.

9.2 Classificação de tabelas

Para que a consulta a tabelas seja feita de uma forma útil, é necessário que ela possa ser apresentada classificada de acordo com critérios diversos. Em outras palavras, freqüentemente é preciso apresentar os dados de uma tabela classificados por um dos *campos* do registro. Um exemplo de quatro possíveis classificações para a mesma tabela que consta na figura 9-1 é apresentado na figura 9-2.

FIGURA 9-2: *EXEMPLOS DE CLASSIFICAÇÃO DE UMA TABELA*

Por ordem de chegada (dos registros):

NOME	ENDEREÇO	TELEFONE
Pedro Álvares Cabral	Torre do Tombo, 7º andar	267-0000
Cristóvão Colombo	Praça de Touros	237-9999
Américo Vespúcio	Rua da Inquisição, s/n	999-1111

Por ordem alfabética de *nomes* (campo NOME):

NOME	ENDEREÇO	TELEFONE
Américo Vespúcio	Rua da Inquisição, s/n	999-1111
Cristóvão Colombo	Praça de Touros	237-9999
Pedro Álvares Cabral	Torre do Tombo, 7º andar	267-0000

Por ordem alfabética de endereços (campo ENDEREÇO):

NOME	ENDEREÇO	TELEFONE
Cristóvão Colombo	Praça de Touros	237-9999
Américo Vespúcio	Rua da Inquisição, s/n	999-1111
Pedro Álvares Cabral	Torre do Tombo, 7º andar	267-0000

Por ordem numérica de telefones (campo TELEFONE):

NOME	ENDEREÇO	TELEFONE
Cristóvão Colombo	Praça de Touros	237-9999
Pedro Álvares Cabral	Torre do Tombo, 7º andar	267-0000
Américo Vespúcio	Rua da Inquisição, s/n	999-1111

Deve-se observar que a ordem de chegada é aquela que os registros da tabela têm quando ela é criada. Se a sua classificação for alterada, a não ser que os registros contenham por exemplo um campo indicando a ordem de chegada, não será possível restabelecê-la mais. Isto é, é impossível saber em que ordem os dados chegaram para serem armazenados na tabela se não houver uma memória. Alternativamente, pode ser feita uma cópia, para preservar a ordem de chegada. Neste caso, ou se preserva a tabela original ou se preserva a cópia da tabela, isto é, as operações de classificação devem ser realizadas sobre uma única tabela. O normal é que as classificações alternativas sejam feitas sobre a cópia.

Existem muitos métodos de classificação já consagrados. Alguns são mais simples de entender e de implementar (programar), enquanto sua eficiência é menor. Outros métodos são mais difíceis de entender e de implementar (programar), mas sua eficiência é maior. A eficiência aqui é tomada no sentido de que a classificação se processa mais rapidamente, em um tempo menor. Para tabelas muito pequenas, as considerações de eficiência são irrelevantes. Para pequenas tabelas deve-se considerar a facilidade de entender e programar o método de classificação. O que é uma pequena tabela? Não existe um critério universalmente aceito, mas o senso comum e a experiência dizem que se a tabela tem entre 1 e 100 elementos (registros) pode ser considerada pequena. A partir deste número, pode-se considerar a tabela como grande. Todavia, é um critério subjetivo e pode um programador considerar uma tabela de 200 ou 300 registros como pequena se as técnicas de ordenação mais eficientes, para uso em grandes tabelas, lhe custarem muito caro em termos de programação e esta ordenação for realizada poucas vezes.

Serão apresentadas as três técnicas de ordenação mais simples: método da bolha (*bublle sort*), seleção direta e inserção direta. Um registro em uma tabela tem diversos *campos*, isto é, é constituído por diversos tipos de dados. Por qual dos campos ordenar a tabela? Qualquer dos campos pode servir como *campo de controle* ou de *referência* para a ordenação. A escolha do campo de controle é atribuição do programador, de acordo com o resultado desejado para a ordenação. Todos os campos são candidatos. Mais ainda, vários campos podem ser concatenados para servir de campo de controle, dependendo apenas do resultado pretendido. Portanto, quando se falar em ordenação de uma tabela, está definitivamente implícito que é segundo um campo de controle determinado previamente.

O mais popular de todos os métodos de ordenação é o *método da bolha*. Todos os programadores iniciantes aprendem a programar o método da bolha. Sua popularidade deriva de ser de fácil compreensão, embora não seja dos mais eficientes. O método consiste em, dispondo-se de uma tabela com n posições (ou registros), comparar cada um dos $n-1$ primeiros elementos (campos de controle de registros), de 1 a $n-1$, com os imediatamente seguin-

tes, trocando-os de posição, se o corrente é *maior* do que o seguinte (ou se o seguinte é *menor* do que o corrente). Realizando essa operação uma vez, tem-se um *passo*. Ao final do *primeiro passo*, o *primeiro maior* valor estará colocado na *última* posição. Ao final do *segundo passo*, o *segundo maior* estará colocado na *penúltima* posição. E assim sucessivamente. No *i-ésimo passo*, o *i-ésimo maior* estará colocado na posição $(n - i + 1)$. Até que no $(n-1)$-*ésimo passo*, o $(n-1)$-*ésimo maior* valor estará colocado na 2^a posição. Logo, o *n-ésimo maior* valor estará colocado na 1^a posição. A extensão de cada passo consecutivo pode ser reduzida de *1* posição em relação ao anterior. Isto é, o *primeiro passo* alcança $n-1$ posições, o *segundo passo* alcança $n-2$ posições, o *i-ésimo passo* alcança $n-i$ posições e o $(n-1)$-*ésimo passo* alcança *1* posição. Essa medida evita comparações inúteis com valores que já estão colocados nas suas posições corretas, pelos passos anteriores. Adicionalmente, a tabela pode ficar ordenada antes mesmo que todos os $n-1$ passos sejam dados. Um artifício permite interromper a classificação se todos os valores já estão classificados *antes* do $(n-1)$-*ésimo* passo. Esse artifício consiste em introduzir um *indicador* de troca (uma variável) em um passo. Esse indicador vale normalmente Ø (*zero*), desde o início. Se ocorrer uma troca de posição, o indicador passa a ser diferente de Ø. Ao final de cada passo o indicador é testado. Se for diferente de Ø, significa que ainda ocorreu uma classificação de pelo menos um valor, no passo concluído. Se ao terminar um passo o indicador for Ø, significa que *não* houve nenhuma ordenação no passo anterior, isto é, todos os valores estão classificados, dispensando-se o prosseguimento da classificação.

Um programa simples que demonstra o método da bolha é o da figura 9-3. No exemplo apresentado, a tabela é constituída por registros com um único campo numérico, mas poderia ser alfanumérico, bem como o registro constituído por diversos campos.

FIGURA 9-3: *PROGRAMA DE DEMONSTRAÇÃO DO MÉTODO DA BOLHA*

```
10  DIM   A (10)
20  PRINT "CLASSIFICACAO-METODO DA BOLHA"
30  PRINT "INFORME 10 NUMEROS"
40  FOR   I = 1 TO 10
50  INPUT A (I)
60  PRINT A(I); " ";
70  NEXT  I
80  LET   L = 10
90  LET   K = 0
100 FOR   J = 1 TO L - 1
110 IF    A(J) <= A (J+1) THEN GOTO 160
120 LET   T = A (J+1)
130 LET   A(J+1) = A (J)
140 LET   A (J) = T
150 LET   K = J
160 NEXT  J
170 PRINT
180 PRINT "PASSO "; 10 - J
190 FOR   M = 1 TO 10
200 PRINT A(M); " ";
210 NEXT  M
220 IF    K = 0 THEN GOTO 260
230 LET   L = K
240 IF    L = 1 THEN GOTO 260
250 GOTO  90
260 PRINT
270 PRINT "TERMINO"
```

Observando a saída que o programa da figura 9-3 dá ao ser executado, pode-se constatar por que esse método de classificação recebeu o nome de *bolha*. Observando-se bem a seqüência dos passos, verifica-se que os maiores valores tendem a *subir* para as posições de mais alto índice, isto é, tendem a *flutuar* em direção à superfície (a posição mais alta, de mais alto índice), como se fossem *bolhas* de ar em um líquido.

O tempo de execução da rotina de classificação pelo método da bolha depende tanto do número de comparações quanto do tempo gasto em trocar elementos de posição. O menor número de comparações ocorre no caso de a tabela estar totalmente classificada desde o início, quando um único passo será executado, com um número de $n-1$ comparações. Quando a tabela está invertida, são dados $n-1$ passos, o número máximo de comparações ocorre, sendo calculado pela soma dos termos de uma progressão aritmética de razão *1*, cujo primeiro termo é *1* e cujo último termo é $n-1$, isto é, $(1+(n-1))\cdot(n-1)/2 = n.(n-1)/2$. O número médio de comparações é difícil de calcular e uma fórmula aproximada dá como expressão $n^2/2 - 3/4 \cdot n$. Com relação ao número de trocas de posição, podem ocorrer $\emptyset$ trocas, caso a tabela esteja ordenada desde o início, ou um número máximo, $n\cdot(n-1)/2$, quando a tabela está invertida. O número médio de trocas também tem um valor aproximado, mas pode-se considerar como representativo desse valor a fórmula $n\cdot(n-1)/4$.

Um segundo método de classificação é o de *seleção direta*. Esse método utiliza duas tabelas, na sua versão mais simples. A primeira tabela é a que contém os dados originais, como se encontram inicialmente. A segunda tabela será usada para conter os elementos ordenados da tabela, isto é, conterá a tabela ordenada como se deseja. Os *passos* são dados sobre a tabela original para seleção de cada elemento que constituirá a tabela ordenada. Em cada *passo* será selecionado um elemento, que será colocado na posição cujo índice representa sua posição na tabela ordenada. No *primeiro passo* é selecionado o *primeiro menor* elemento, que será colocado na *primeira* posição da tabela orde-

nada. No *segundo passo* será selecionado o *segundo menor* elemento da tabela e colocado na *segunda* posição da tabela ordenada. No *i-ésimo passo* será selecionado o *i-ésimo menor* elemento, que será colocado na *i-ésima* posição da tabela ordenada. Até que no *n-ésimo* passo será selecionado o *n-ésimo menor* elemento, que será colocado na *n-ésima* posição da tabela ordenada.

Um programa simples que demonstra como opera o método da seleção direta encontra-se na figura 9-4. Por simplificação, o exemplo classifica uma tabela com registros contendo apenas um campo, no caso alfanumérico, com um único caractere.

FIGURA 9-4: *PROGRAMA DE DEMONSTRAÇÃO DO MÉTODO DE SELEÇÃO DIRETA*

```
10  DIM   A$ (10)
20  DIM   B$ (10)
30  FOR   M = 1 TO 10
40  LET   B$ (M) = CHR$ (255)
50  NEXT  M
60  PRINT "CLASSIFICACAO-SELECAO DIRETA"
70  PRINT "INFORME 10 CARACTERES - NAO £"
80  FOR   N = 1 TO 10
90  INPUT A$ (N)
100 PRINT A$ (N); " ";
110 NEXT  N
120 FOR   I = 1 TO 10
130 FOR   J = 1 TO 10
140 IF    B$ (I) <= A$ (J) OR A$ (J) = "£" THEN GOTO 170
150 LET   B$ (I) = A$ (J)
160 LET   K = J
170 NEXT  J
180 LET   A$ (K) = "£"
190 PRINT
200 PRINT "PASSO "; I
210 FOR   L = 1 TO I
220 PRINT B$ (L); " ";
230 NEXT  L
240 NEXT  I
250 PRINT
260 PRINT "TERMINO"
```

Uma alternativa para o método de seleção direta é utilizar uma única tabela e fazer a seleção na própria tabela original que se tornará a tabela ordenada. Neste caso, basta ordenar do elemento *1* ao *n–1*. Isso garante que, terminada a ordenação, o *n-ésimo menor* estará na posição *n*. Para colocar cada *i-ésimo menor* elemento na posição *i* é feita uma troca. A pesquisa do *menor* elemento no *i-ésimo passo* começa no elemento *i+1* e vai até o elemento *n*.

No primeiro passo, começa no 2º elemento. No passo *n–1*, começa no elemento *n*. A tabela ordenada se forma entre o *1º* elemento e o *i-ésimo* elemento.

Um programa para demonstração do método de seleção direta (modificado) encontra-se na figura 9-5.

FIGURA 9-5: *PROGRAMA DE DEMONSTRAÇÃO DO MÉTODO DE SELEÇÃO DIRETA (MODIFICADO)*

```
10  DIM   A$ (10)
20  PRINT "SELEÇÃO DIRETA-MODIFICADO"
30  PRINT "INFORME 10 CARACTERES"
40  FOR   N = 1 TO 10
50  INPUT A$ (N)
60  PRINT A$(N); " ";
70  NEXT  N
80  FOR   I = 1 TO 9
90  LET   M$ = A$ (I)
100 LET   M = I
110 FOR   J = I + 1 TO 10
120 IF    A$ (J) >= M$ THEN GOTO 150
130 LET   M$ = A$ (J)
140 LET   M = J
150 NEXT  J
160 LET   T$ = A$ (M)
170 LET   A$ (M) = A$ (I)
180 LET   A$ (I) = T$
190 PRINT
200 PRINT "PASSO "; I
210 FOR   K = 1 TO 10
220 PRINT A$ (K); " ";
230 NEXT  K
240 NEXT  I
250 PRINT
260 PRINT "TERMINO"
```

No tempo gasto na execução da rotina de classificação pelo método de seleção direta, na forma inicial, *sem* troca de posição, deve-se considerar que são dados *n* passos, cada um com *n* comparações, o que perfaz um número total de n^2 comparações. Além disso, deve ser considerada a movimentação de *n* elementos, da tabela original para a tabela ordenada.

O tempo gasto na execução da rotina de classificação pelo método de seleção direta modificado, *com* troca de posições, considera a execução de *n–1* passos, incluindo *(n–1)/2* comparações e um número variável de trocas. O número de trocas varia de $\emptyset$, se a tabela já está ordenada inicialmente, até um máximo de *n–1*, se a tabela está inicialmente invertida. O número médio de trocas não é uma expressão trivial, pelo que deixa de ser apresentada.

Um terceiro método de classificação de tabelas é conhecido como o método de *inserção direta*. Esse método utiliza uma única tabela, como original e ordenada. Assemelha-se à técnica que um jogador de cartas utiliza para mantê-las ordenadas em sua mão. Sempre que chega uma nova carta, coloca-a em último lugar. Depois procura entre as anteriores a posição correta para inserção. Encontrada a posição de inserção, desloca todas as cartas à direita, de uma posição, para abrir espaço. Aberto o espaço, remove a nova carta da sua posição (última) e insere na posição correta (que foi aberta). No caso, a suposição é de que as cartas estavam inicialmente ordenadas, antes da chegada da nova carta. O processo, todavia, pode ser usado para organizar cartas, mesmo que elas estejam totalmente desordenadas. Basta aplicar o método progressivamente, isto é, ordenando conjuntos parciais cada vez maiores, a partir de uma carta. Uma carta está sempre ordenada. Incluindo-se uma segunda carta, tem-se um conjunto desordenado que será de novo ordenado. Ordenado o conjunto de duas cartas, acrescenta-se uma terceira carta. Nova ordenação e assim sucessivamente, até que todas as cartas na mão estejam ordenadas, isto é, o conjunto completo. O método de classificação por inserção direta é equivalente ao processo descrito para ordenação do conjunto de cartas completo que esteja inicialmente desordenado. É fácil perceber por que o método tem o nome de inserção direta, uma vez que para ordenação do conjunto completo, em cada conjunto parcial ordenado, é feita a inserção de uma nova carta.

Deve-se observar que, feita a inserção de uma nova carta em um conjunto parcial ordenado, se ela estiver na posição correspondente à sua ordem, não é necessário ordenar, porque o conjunto já fica ordenado. Tendo-se uma tabela com n elementos (ou registros), a inserção é realizada a partir do *segundo* elemento, até o *n-ésimo* em $n-1$ passos. Se o elemento inserido está ordenado, dispensa-se o processo de ordenação. Para verificar essa condição, basta comparar o elemento inserido com o imediatamente anterior. Se o novo elemento for maior do que o imediatamente anterior, está ordenado. Caso contrário, procede-se à ordenação. O processo de ordenação para o *j-ésimo* elemento consiste em procurar, do $(j-1)$-*ésimo* até o *primeiro* elemento, qual a posição onde o atual *j-ésimo* elemento será inserido. Essa posição é encontrada comparando-se o *j-ésimo* elemento com os anteriores, enquanto for menor. Quando for maior ou igual a um elemento, encerra-se a pesquisa, guardando-se a posição de inserção com relação a esse *i-ésimo* elemento. A posição de inserção é $i+1$. Todos os elementos, desde $j-1$ até $i+1$, serão deslocados de uma posição à direita, para deixar espaço para inserção do *j-ésimo* elemento, previamente retirado da posição e salvo em uma variável auxiliar. Deslocados os elementos e deixado espaço para o *j-ésimo* elemento, ele será inserido na posição $i+1$. Um programa que demonstra como se processa a ordenação pelo método de inserção direta encontra-se na figura 9-6.

FIGURA 9-6: *PROGRAMA DE DEMONSTRAÇÃO DO MÉTODO DE INSERÇÃO DIRETA*

```
10  DIM    A (10)
20  PRINT  "CLASSIFICACAO – INSERCAO DIRETA"
30  PRINT  "INFORME 10 NUMEROS"
40  FOR    M = 1 TO 10
50  INPUT  A (M)
60  PRINT  A (M); " ";
70  NEXT   M
80  FOR    J = 2 TO 10
90  IF     A(J) >= A(J–1) THEN GOTO 160
100 LET    X = A (J)
110 FOR    I = J–1 TO 1 STEP –1
120 IF     X >= A(I) THEN GOTO 150
130 LET    A(I + 1) = A (I)
140 NEXT   I
150 LET    A(I + 1) = X
160 PRINT
170 PRINT  "PASSO "; J–1
180 FOR    K = 1 TO J
190 PRINT  A(K); " ";
200 NEXT   K
210 NEXT   J
220 PRINT
230 PRINT  "TERMINO"
```

O tempo de execução da rotina de classificação por inserção direta inclui o tempo necessário às comparações e ao deslocamento de dados. Deve-se considerar que serão dados sempre $n-1$ passos, do 2^{o} ao *n-ésimo* elemento. O número mínimo de comparações ocorre quando a tabela está ordenada e é calculado pela fórmula $n-1$. O número máximo de comparações é a soma dos termos de uma progressão aritmética de razão 1, com $n-1$ termos, o primeiro sendo 2 e o último sendo n, isto é, $(n\cdot(n+1)/2)-1$. O número médio de comparações é dado pela expressão $(n\cdot(n+1)/4)-(1/2)$. O número de transferências de dados corresponde ao número de comparações, quando um valor não está ordenado, mais 1, correspondente à colocação do dado na sua posição correta. Portanto, na nossa rotina, que não faz movimentação de dado se ele já está na posição correta, o número mínimo de comparações é $\emptyset$. O número máximo de transferências, se todos os dados serão ordenados, é dado pela expressão $(n\cdot(n+1)/2)+n-2$. O número médio de transferências é dado pela expressão $(n\cdot(n+1)/4)-(1/2)$.

9.3 Pesquisa em tabelas

Os métodos de pesquisa em tabelas nos permitem encontrar um registro dessa tabela segundo um determinado critério. O critério adotado é a igualda-

de de um campo de controle do registro com um valor conhecido. Uma pesquisa pode ser bem ou malsucedida, isto é, o valor procurado pode estar ou não em campo de controle da tabela. Por esse motivo, normalmente as rotinas de pesquisa em tabelas têm uma estrutura geral comum: pesquisa; se encontrou, sucesso; se não encontrou, fracasso.

São dois os métodos mais utilizados: a pesquisa seqüencial e a pesquisa binária. Destes dois métodos, a pesquisa seqüêncial é a fundamental, base para todos os demais métodos de pesquisa.

A *pesquisa seqüencial* pode ser feita em uma tabela ordenada ou *não*-ordenada. Em princípio, a pesquisa seqüencial consiste em começar a procura pelo *primeiro* elemento e continuar até o *último*, enquanto o valor procurado do campo de controle não for encontrado. Se o valor procurado é encontrado, a pesquisa é interrompida. Evidentemente, a tabela não pode conter duplicatas. Um programa que demonstra como funciona a pesquisa seqüencial encontra-se na figura 9-7.

FIGURA 9-7: *PROGRAMA DE DEMONSTRAÇÃO DE PESQUISA SEQÜENCIAL EM TABELA NÃO-ORDENADA*

```
10  DIM    A$ (10)
20  PRINT  "PESQUISA SEQUENCIAL"
30  PRINT  "TABELA NAO ORDENADA"
40  PRINT  "INFORME 10 CARACTERES"
50  FOR    M = 1 TO 10
60  INPUT  A$(M)
70  PRINT  A$(M); " ";
80  NEXT   M
90  PRINT
100 PRINT  "INFORME CARACTERE A PESQUISAR"
110 INPUT  C$
120 PRINT  "CARACTERE="; C$
130 FOR    I = 1 TO 10
140 IF     A$ (I) = C$ THEN GOTO 170
150 NEXT   I
160 LET    I = I - 1
170 IF     A$ (I) <> C$ THEN GOTO 200
180 PRINT  "SUCESSO"
190 GOTO   210
200 PRINT  "FRACASSO"
210 PRINT  "TERMINO"
```

O tempo gasto em uma pesquisa é dependente, essencialmente, do número de comparações, pois não há movimentação de dados. Assim sendo, na pesquisa seqüencial, para pesquisas *sem* sucesso, a tabela terá que ser pesquisada por inteiro e o número de comparações é n, o número de elementos da

tabela em todos os casos. Para as pesquisas bem-sucedidas, o número mínimo de pesquisas é de 1, se o elemento procurado for o *primeiro*, o número de pesquisas é de n, se o elemento procurado se encontrar na última posição, e o número médio de pesquisas é de $(n+1)/2$. Essas expressões partem do pressuposto de que todos os valores têm igual probabilidade de serem pesquisados, o que nem sempre é verdade. Todavia, pode ser aceito, em grandes números ou como uma aproximação razoável.

Se a tabela estiver ordenada, o número de comparações, quando o valor *não* está na tabela, pode ser diminuído. Existe, portanto, uma vantagem, em termos de eficiência, em ordenar a tabela antes da pesquisa seqüencial. Com a tabela ordenada, a comparação do valor procurado com o campo de controle é feita até que se encontre um valor na tabela que seja *maior* do que o procurado, quando a pesquisa é interrompida. O valor imediatamente anterior pode ser ou não o valor procurado, isto é, a pesquisa pode ter tido ou não sucesso. Todavia, mesmo no caso de fracasso, evita-se percorrer a tabela inteira fazendo comparações. Terminada a pesquisa, verifica-se se o valor anterior ao ponto em que a pesquisa foi interrompida é ou não o valor procurado. O programa da figura 9-8 serve para demonstrar a pesquisa seqüencial em tabela ordenada.

FIGURA 9-8: *PROGRAMA DE DEMONSTRAÇÃO DE PESQUISA SEQÜENCIAL EM TABELA ORDENADA*

```
10  DIM    A (10)
20  PRINT  "PESQUISA SEQUENCIAL"
30  PRINT  "TABELA ORDENADA"
40  PRINT  "INFORME 10 NUMEROS"
50  FOR    M = 1 TO 10
60  INPUT  A (M)
70  PRINT  A (M); " ";
80  NEXT   M
90  FOR    J = 2 TO 10
100 IF     A(J) >= A(J-1) THEN GOTO 170
110 LET    X = A (J)
120 FOR    I = J-1 TO 1 STEP -1
130 IF     X>= A(I) THEN GOTO 160
140 LET    A(I+1) = A(I)
150 NEXT   I
160 LET    A(I+1) = X
170 NEXT   J
180 PRINT
190 PRINT  "TABELA ORDENADA"
200 FOR    K = 1 TO 10
210 PRINT  A (K); " ";
220 NEXT   K
230 PRINT
240 PRINT  "INFORME NUMERO A PESQUISAR"
```

```
250  INPUT   N
260  PRINT   "NUMERO="; N
270  FOR     L = 1 TO 10
280  IF      A(L) >= N THEN GOTO 310
290  NEXT    L
300  LET     L = L -1
310  IF      A(L) <> N THEN GOTO 340
320  PRINT   "SUCESSO"
330  GOTO    350
340  PRINT   "FRACASSO"
350  PRINT   "TERMINO"
```

No caso da pesquisa seqüencial em tabela ordenada, não há diferença entre pesquisas bem ou malsucedidas para o cálculo do número de comparações. Supondo-se que todos os valores procurados tenham a mesma probabilidade (sejam pesquisados com a mesma freqüência), o número médio de comparações é dado pela expressão $(n+1)/2$. Todavia, essa hipótese nem sempre é verdadeira, embora forneça um valor razoavelmente próximo, bom para a maioria dos casos.

O segundo método de pesquisa de tabelas, a *pesquisa binária* ou *logarítmica, só* é aplicável a tabelas *ordenadas. Não* se pode fazer pesquisa binária em uma tabela que *não* esteja ordenada. Esse método de pesquisa deriva seu nome do procedimento adotado para fazer a pesquisa, por sucessivas divisões da tabela em *duas* partes (binário = relativo à base 2).

O procedimento tem início tomando-se como *começo* a posição do *primeiro* elemento e como *fim* da tabela a posição do *último* elemento, isto é, a tabela inteira será pesquisada. Calcula-se a posição do elemento do *meio* (média aritmética entre as posições de *começo* e de *fim*). O *meio* divide a tabela em duas partes, aproximadamente iguais (em número de elementos). Três situações podem ocorrer: o valor procurado é *igual* ao valor do elemento na posição *meio*, isto é, o *meio* indica a posição do elemento procurado, caso em que a pesquisa está terminada; o valor procurado é *menor* do que o valor que se encontra no elemento apontado por *meio*, caso em que o valor procurado se encontra à *esquerda* do *meio*; ou o valor procurado é *maior* do que o valor que se encontra no elemento indicado por *meio*, caso em que o valor procurado se encontra à *direita* do *meio*. Se a pesquisa prossegue, isto é, se o valor procurado *não* está no elemento do *meio*, deverá prosseguir apenas com uma das metades da tabela, à *esquerda* ou à *direita* do meio. Se prosseguir com a parte da *esquerda*, o *começo* permanece o mesmo, fazendo-se o *fim* igual ao *meio*. Da mesma forma, se a pesquisa prossegue com a parte da *direita*, o *fim* permanece o mesmo enquanto o *começo* é deslocado para o *meio*. De novo é calculado um valor para o *meio*, na parte pesquisada. E assim sucessivamente. Chegando-se a uma parte da tabela constituída por um

único elemento (*meio*, *começo* e *fim* são iguais), está concluída a pesquisa, *com* ou *sem* sucesso. Como já foi dito anteriormente, neste ponto é preciso verificar se a pesquisa teve sucesso ou não.

Um programa que serve como demonstração do método de pesquisa binária encontra-se na figura 9-9.

FIGURA 9-9: *PROGRAMA DE DEMONSTRAÇÃO DE PESQUISA BINÁRIA (SEMPRE EM TABELA ORDENADA)*

```
10   DIM     A$ (10)
20   PRINT   "PESQUISA BINARIA"
30   PRINT   "INFORME 10 CARACTERES"
40   FOR     H = 1 TO 10
50   INPUT   A$ (H)
60   PRINT   A$ (H); " ";
70   NEXT    H
80   FOR     J = 2 TO 10
90   IF      A$ (J) >=A$(J-1) THEN GOTO 160
100  LET     X$ = A$ (J)
110  FOR     I = J - 1 TO 1 STEP -1
120  IF      X$>= A$ (I) THEN GOTO 150
130  LET     A$ (I+1) = A$ (I)
140  NEXT    I
150  LET     A$ (I+1) = X$
160  NEXT    J
170  PRINT
180  PRINT   "TABELA ORDENADA"
190  FOR     F = 1 TO 10
200  PRINT   A$ (F) ; " ";
210  NEXT    F
220  PRINT
230  PRINT   "INFORME CARACTERE A PESQUISAR"
240  INPUT   C$
250  PRINT   "CARACTERE="; C$
260  LET     C = 1
270  LET     F = 10
280  LET     M = INT ((C + F)/2)
290  IF      A$(M) < C$ THEN GOTO 320
300  LET     F = M - 1
310  GOTO    330
320  LET     C = M + 1
330  IF      A$ (M) <> C$ AND C <= F THEN GOTO 280
340  IF      A$ (M) <> C$ THEN GOTO 370
350  PRINT   "SUCESSO"
360  GOTO    380
370  PRINT   " FRACASSO"
380  PRINT   " TERMINO"
```

A pesquisa binária é o método de pesquisa mais rápido disponível para tabelas alocadas seqüencialmente. Qualquer outro método é menos eficiente do que ele, se a tabela é seqüencial. Há outros métodos mais eficientes, mas requerem uma organização específica para a tabela e não serão tratados neste livro. Todavia, a rapidez do método de pesquisa binária já lhe confere um lugar especial entre os demais. O número de comparações máximo, se o valor procurado só for encontrado na última parte da tabela, é dado pela expressão $log_2\ n + 1$, tomando-se o valor inteiro contido em log_2. Portanto, a pesquisa binária deve ser a preferida para arquivos seqüenciais muito grandes.

Exercícios:

ESTUDO DIRIGIDO:

1) Que são tabelas? Uma lista telefônica é uma tabela? Que tipo de acessos são possíveis numa lista telefônica? Que outros tipos de tabelas conhece? Que é classificação? Que tipos de classificação são possíveis? Qual a classificação usual? Que é pesquisa de uma tabela? De quantas maneiras é possível implementar uma tabela em computador? Que é alocação encadeada? Que é alocação seqüencial? Que é contigüidade física? Que são apontadores? De que maneiras é possível alocar tabelas seqüencialmente com o uso de matrizes? É possível colocar uma tabela em uma única matriz? É possível armazenar uma tabela em mais de uma matriz? Que é um registro de uma tabela? Nos menores micros, as tabelas podem substituir os arquivos? Como armazenar uma tabela em fita cassete? Como recuperar a tabela? Como iniciar a execução de um programa num micro que não tem o conceito de arquivo, para não perder os dados das tabelas?

2) Que é classificação? Que é campo de um registro? É possível ter mais de uma classificação para a mesma tabela? Que é classificação por ordem de chegada? Que é classificação por um campo de controle? Quais são os mais comuns métodos de classificação? Qual a idéia básica do método da bolha? De que parâmetros depende a eficiência do método da bolha? Qual a idéia básica do método de seleção direta? Quantas tabelas usa o método de seleção direta (na forma original)? Que é tabela ordenada? É possível alterar o método de seleção direta para se ter uma única tabela? Como funciona a alternativa do método de seleção direta com uma única tabela? Quais os principais parâmetros de medir a eficiência no método de seleção direta? Qual a idéia básica do método de inserção direta? Quantas tabelas usa o método de inserção direta? Quais os principais parâmetros que medem a eficiência do método de inserção direta?

3) Que é pesquisar uma tabela? Qual o esquema geral dos métodos de pesquisa? Quais os principais métodos de pesquisa? Qual a idéia básica do método de pesquisa seqüencial? É preciso ordenar a tabela para que seja pesquisada? Quais as conseqüências de não se ordenar ou de ordenar a tabela sobre o método de pesquisa seqüencial? A pesquisa seqüencial com tabela ordenada é mais eficiente? Como ocorre a pesquisa seqüencial em tabela ordenada? Quais os parâmetros que medem a eficiência de uma pesquisa seqüencial? A que é proporcional o tempo de pesquisa seqüencial, com tabela ordenada e com tabela não-ordenada? Qual a idéia básica do método de pesquisa binária? Quais os principais parâmetros que medem a eficiência da pesquisa binária? A que é proporcional o tempo de pesquisa binária? Para grandes arquivos qual método é mais eficiente, pesquisa seqüencial ou pesquisa binária? Qual a diferença em desempenho entre esses dois métodos?

PROBLEMAS:

1) Escrever um programa que leia uma tabela contendo registros cujos campos sejam: *sobrenome, nome, logradouro, nº, complemento, CEP, cidade, estado (sigla)*. Utilizar uma única matriz de caracteres para a tabela original. Exibir a tabela lida na ordem de chegada dos registros. Utilizando outra tabela para ordenação, fazer uma cópia da original e classificar pelo método da bolha, por *sobrenome*. Exibir a tabela ordenada. Executar o programa.

2) Alterar o programa anterior para que a leitura e a classificação da tabela se tornem opcionais. Isso permitirá armazenar a tabela em fita, nos menores micros, e recuperá-la sem alterações. Após a execução do programa, armazenar programa e dados em fita cassete. Recuperar programa e dados a partir da fita cassete. Reexecutar o programa.

3) Escrever um programa que opcionalmente leia registros para uma tabela armazenada em várias matrizes. Cada registro contém como campos o *nome* e o *telefone*. Exibir a tabela original por ordem de chegada dos registros. Classificar opcionalmente pelo *nome* ou pelo *telefone*, usando o método de seleção direta com uma única tabela. Usar, para a classificação, uma segunda tabela. Exibir a tabela classificada. Executar o programa.

4) Alterar o programa anterior para que seja possível incluir registros na tabela ou excluí-los, tendo como campo-chave o *nome*. Executar o programa. Se o seu micro é dos menores (primeiro grupo), guarde o programa com os dados em fita cassete. Recupere programa e dados. Reexecute o programa com **GOTO** (*não* com **RUN**). Faça inclusões e exclusões de registros.

5) Escrever um programa que opcionalmente leia uma tabela do imposto de renda contendo três colunas: *renda líquida (limite superior)*, *alíquota (taxa em %)* e *dedução*. Classificar opcionalmente por *renda líquida*, pelo método de inserção direta. Exibir a tabela antes e depois da classificação. Executar o programa.

6) Alterar o programa anterior para que, opcionalmente, seja feita uma pesquisa com base na *renda líquida* (limite) e exibidos a renda líquida, a *alíquota*, o valor da parcela de *imposto* resultante da aplicação da alíquota, a *dedução* e o *imposto devido* (com a dedução). Utilize a pesquisa seqüencial em tabela ordenada. Executar o programa.

7) Escrever um programa que, opcionalmente, leia uma tabela constituída por registros com dois campos, um de nome e outro de telefone. A tabela será opcionalmente ordenada por inserção direta. A partir de um valor fornecido, será feita uma pesquisa binária na tabela, com base no campo, nome. Se a pesquisa não for bem-sucedida, emitir a mensagem **VALOR** <*valor*>**NÃO EXISTE**. Se a pesquisa for bem-sucedida, exibir os valores dos dois campos, precedidos da mensagem **VALORES ENCONTRADOS**. Executar o programa. Se o seu micro é dos menores, armazenar programa e dados em fita cassete. Recuperar e reexecutar com a mesma tabela anterior (no estado em que foi armazenada em fita).

8) Escrever um programa que, opcionalmente, leia uma tabela com dois campos numéri-

cos por registro. O primeiro campo será usado para pesquisa. Também opcionalmente, pode ser feita a classificação dos registros da tabela com base no primeiro campo. A partir de um valor lido, será feita a pesquisa seqüencial em tabela ordenada. A pesquisa só é considerada malsucedida se o valor procurado estiver fora dos limites, isto é, for menor que o primeiro ou maior do que o último. Nos demais casos, será sempre bem-sucedida. Normalmente, o valor lido estará entre dois valores do primeiro campo. Neste caso, deseja-se saber qual o valor correspondente no segundo campo, por meio de interpolação linear. Portanto, o valor correspondente ao segundo campo tem que ser calculado com base nos valores extremos do intervalo, para o mesmo campo. Executar o programa.

DÉCIMO CAPÍTULO
OPERAÇÕES DIVERSAS

formatação explícita de dados de saída – programação em linguagem de máquina

10.1 Formatação explícita de dados de saída

O leitor já conhece, da seção 2.8, as regras para formatar dados de saída, de forma *implícita*. A forma implícita é a que utiliza delimitadores como , (*vírgula*) e ; (*ponto e vírgula*) para estabelecer o espaçamento entre os dados que serão impressos ou exibidos. Correspondem, respectivamente, aos formatos (implícitos) *aberto* e *fechado*.

Se o programa requer uma rigorosa colocação dos dados no espaço de saída, obedecendo a uma formatação rígida, outros recursos devem ser utilizados. Essas facilidades são implementadas com as instruções **PRINT** USING e **IMAGE** (alternativamente: **FORMAT**) e um conjunto de símbolos e regras para a definição dos formatos de saída, *não* estando disponíveis, porém, em todos os sistemas BASIC, particularmente nos menores micros (primeiro grupo). Todavia, podem ser encontradas nos micros maiores. A apresentação dessas facilidades será feita através de exemplos que vêm a seguir.

Exemplos de instruções de saída com formato:

1)
```
10 PRINT USING "VALOR DE X = #### ", X
```

onde:

Componente	Significado
10	número de linha

PRINT USING	palavra-chave (imprimir usando)
"VALOR DE X = ####"	formato (imagem)
X	lista de saída

Explicação:

Será impressa ou exibida uma linha que, supondo-se o valor de X como sendo 375, para exemplo, e obedecendo à especificação de formato conterá

```
VALOR DE X = 375
```

2)
```
20 PRINT USING " ##.### #" , -2.7, 35.68, 45.3782
```

onde:

Componente	Significado
20	número de linha
PRINT USING	palavra-chave (imprimir usando)
" ##.###"	formato (imagem)
-2.7, 35.68, 45.3782	lista de saída

Explicação:

Será representada, na impressora ou vídeo, a série de valores das constantes da lista de saída, segundo o *formato* especificado, em *três linhas* que conterão

```
-2.700
35.680
45.378
```


3)
```
30 PRINT USING " ##.###↑↑↑↑" , -132, 6.74635, 0.0345
```

onde:

Componente	Significado
30	número de linha
PRINT USING	palavras-chave (imprimir usando)
" ##.###↑↑↑↑"	formato (imagem)
-132, 6.74635, 0.0345	lista de saída

Explicação:

Serão representados na impressora ou vídeo *três* valores numéricos, cada um em uma linha, a partir dos valores das constantes da lista de saída e segundo a especificação de *formato*, tendo o aspecto

```
-1.320E+02
67.463E-01
34.500E-03
```

4)
```
40 PRINT USING "###.##↑↑↑↑ ###", 29.653, 895, 0.007263
```

onde:

Componente	Significado
40	número de linha

PRINT USING	palavra-chave (imprimir usando)
"###.##↑↑↑↑ ###"	formato (imagem)
29.653, 895, Ø.ØØ7263	lista de saída

Explicação:

Serão representados na impressora ou vídeo, *três* valores, a partir das constantes da lista, sendo *dois* valores na primeira linha e o terceiro na segunda linha, conforme o formato especificado, com o aspecto

```
296.53E-Ø1 895
726.3ØE-Ø5
```

A forma geral da instrução de *saída com especificação de formato é*

PRINT USING <*formato*> [,<*expressão*>]...

Alguns sistemas, em vez de *vírgula* (,) após *formato*, usam *ponto e vírgula* (;) e outros admitem as duas formas.

Cada vez que uma instrução de *saída com especificação de formato* é executada, o sistema BASIC inicia a impressão em uma *nova linha*. Nisto *difere* da instrução de *saída simples* (*sem especificação de formato*), na qual o salto para *nova linha* depende da *última* instrução de *saída* executada (do *separador* ou *delimitador* que encerra a lista de saída ser *branco*, isto é, NEW LINE). Além disso, a partir do primeiro dado representado, os demais dependerão da relação existente entre o número de *formatos* disponíveis para conversão de dados e o número de *valores* constantes na lista de saída. Em geral, o tamanho da especificação de *formato não* pode exceder o tamanho de uma *linha*, que é quase sempre de *132* caracteres. Se exceder o tamanho da *linha*, o sistema geralmente enviará uma mensagem de erro. Cada *especificação de formato* ou *imagem* é composta de caracteres *literais*, que são impressos exatamente como figuram, e de *símbolos* ou *códigos de conversão de dados* que determinam como os valores da *lista* serão representados. Cada valor da *lista de saída* será representado segundo um correspondente *código de conversão*, até que todos os códigos de conversão da imagem tenham sido esgotados. Se existem *mais* valores na *lista de saída* do que *códigos de conversão*, a *imagem* ou *especificação de formato* será *reutilizada* a partir do início. Também os *literais* serão *novamente* impressos, após um *salto* para *nova linha*. Sempre que a *imagem* ou *especificação de formato* for utilizada de novo, haverá um *salto* para a *linha seguinte*.

Os *símbolos de conversão de dados* apresentados servem para a *conversão de dados numéricos* e são sumariados no quadro 1Ø-1.

QUADRO 1Ø-1: *SÍMBOLOS DE CONVERSÃO DE DADOS NUMÉRICOS*

SÍMBOLO	CONVERSÃO
# (*cancela*)	*Dígitos* (*algarismos*) *numéricos* e o *sinal*, *branco* se positivo e *menos* (–) se negativo, um símbolo para cada *dígito* ou *sinal*. Os *zeros* de mais alta ordem são substituídos por *branco*.
. (*ponto*)	*Ponto decimal* (notação matemática: *vírgula*) ou *ponto fraciondrio* (só pode haver um único).
↑↑↑↑ (*sempre 4 setas para cima*)	*Expoente* da *base 1Ø* sob a forma Espp ou Esp sempre após a *mantissa*, isto é, à direita do *código de conversão*.

O aspecto apresentado pelos formatos de conversão de dados numéricos é

```
######.##  ####  ##.#####↑↑↑↑
```

Além da *especificação de conversão numérica*, existem outros tipos de *conversões*, como a de *edição numérica* e a *alfanumérica*. Ao todo, devem ser consideradas como possibilidades de *conversão de formato* os grupos de

1) especificação de formato numérico;
2) especificação de formato numérico editado; e
3) especificação de formato alfanumérico.

Os *símbolos de edição de formato numérico* constam do quadro 1Ø-2.

QUADRO 1Ø-2: *SÍMBOLOS DE EDIÇÃO DE DADOS NUMÉRICOS*

SÍMBOLO	CONVERSÃO
$ (*cifrão*)	*Cifrão fixo*: um único símbolo à esquerda na primeira ou segunda posição. Os *zeros* de mais alta ordem são substituídos por espaço em *branco*. Se na segunda posição, o símbolo à esquerda só pode ser *mais* (+) ou *menos* (–) *fixo*.
$$ (*cifrões*)	*Cifrão flutuante*: dois ou mais *cifrões* começando na primeira ou segunda posição. O *cifrão* ocupará a posição imediatamente anterior ao dígito de mais alta ordem. O símbolo à esquerda, se começar na segunda posição, só pode ser *mais* (+) ou *menos* (–) *fixo*. *Não* pode coexistir com *sinal flutuante*.
+ (ou –) (*sinal*: *mais* ou *menos*)	*Sinal mais* ou *menos fixo*: um único símbolo à esquerda ou direita. Os *zeros* de mais alta ordem são substituídos por *branco*, excluído o *zero* imediatamente precedendo o *ponto decimal*. *Sinal menos*: *branco* corresponde a valor positivo e *menos* a negativo. *Sinal mais*: *mais* corresponde a valor positivo e *menos* a negativo.
+ + (ou – –) (*sinais*: *mais* ou *menos*)	*Sinal mais* ou *menos flutuante*: dois ou mais *sinais mais* ou *menos* à esquerda. O *sinal* ocupará a posição imediatamente anterior ao *dígito* de mais alta ordem. *Sinal menos*: branco corresponde ao valor positivo e *menos* a negativo. *Sinal mais*: *mais* corresponde a valor positivo e *menos* a negativo. Não pode coexistir com o *cifrão flutuante*.
* (*asterisco*)	*Proteção de cheque*: um ou mais *asteriscos* começando na primeira ou segunda posição. Substituirá todos os *zeros* de mais alta ordem. Se começar na segunda posição, símbolo à esquerda só pode ser *cifrão* ou *sinal fixo*.
, (*vírgula*)	*Vírgula* numérica (notação matemática: *ponto numérico*): uma *vírgula* aparecerá na posição *fixa* onde estiver o símbolo, entre dígitos, ou um *branco* se os *zeros* à esquerda forem suprimidos.

O aspecto apresentado pelos formatos de conversão de dados numéricos editados é


```
$$###,###,###.##      $$,$$$.$$      -$#,###.##
####-     #,###.##      +###.##   -$$,$$$,$$$.##
```

Deve-se observar que existem implementações de BASIC que *não* adotam nenhuma dessas convenções, mas sim convenções similares às da linguagem FORTRAN para conversão de dados. Estas, felizmente, são uma minoria. A regra geral é a adoção das convenções da linguagem BASIC. Todavia, há pequenas diferenças na adoção das convenções, principalmente essas relativas à *edição*. Apesar disso, pode haver algum que *não* permita a *conversão de formato* na *forma exponencial* ou *científica*. Para os sistemas BASIC que oferecem a possibilidade de especificar o *formato* em instruções de saída, as convenções mais comuns são o símbolo de *conversão numérica (#)* e o *ponto fracionário* (.). Neste caso a *conversão numérica* sempre inclui também a *conversão do sinal*. Aqueles que oferecem *conversão de sinal* em separado da *conversão numérica* quase sempre requerem a especificação da *posição do sinal*.

Além da *conversão numérica* e da *numérica editada*, alguns sistemas BASIC, principalmente aqueles que têm facilidades para manipulação de dados *alfanuméricos*, oferecem *símbolos* para *conversão de formato alfanumérico*. O número dos sistemas BASIC que oferecem esse tipo de conversão é, porém, pequeno, comparado com o número dos que oferecem as outras formas de conversão.

As convenções para *conversão de formato alfanumérico*, isto é, os *símbolos* de *conversão alfanumérica* são apresentados no quadro 1Ø-3.

QUADRO 1Ø-3: *SÍMBOLOS DE CONVERSÃO DE DADOS ALFANUMÉRICOS*

SÍMBOLO	CONVERSÃO
C (*center*)	*Centralizar*: um *apóstrofo* (') seguido de um símbolo C para cada posição no campo. A cadeia de caracteres alfanuméricos é *centralizada* no campo. Se não pode ser exatamente centralizada, será deslocada uma posição para a *esquerda*. Se a cadeia é maior que o campo, será alinhada à *esquerda* e truncada à *direita*.
L (*left*)	*Alinhar à esquerda*: um *apóstrofo* (') seguido de um símbolo L para cada posição do campo. A cadeia de caracteres é alinhada à *esquerda*. Se a cadeia é maior que o campo, será alinhada à *esquerda* e truncada à *direita*.

R (*right*)	*Alinhar à direita*: um *apóstrofo* (') seguido de um símbolo R para cada posição do campo. A cadeia de caracteres é alinhada à *direita*. Se a cadeia é maior que o campo, será alinhada à *esquerda* e truncada à *direita*.
E (*extended*)	*Estender*: um *apóstrofo* (') seguido de um símbolo E para cada posição do campo. A cadeia de caracteres é alinhada à *esquerda* e, se for maior do que o campo definido, este será estendido à *direita* para dar saída a todos os caracteres da cadeia.
' (*apóstrofo*)	*Início de campo*: primeiro símbolo de *todos* os formatos *alfanuméricos* para diferenciá-los dos *literais*.

O aspecto apresentado pelos formatos alfanuméricos é

```
'LLLLLLLLL 'CCCCCCCCCC 'RRRRRRRRRRR 'EEEEEEEEEE
```

Deve-se mencionar que há sistemas BASIC que permitem que a *especificação de formato*, além de ser fornecida como *constante literal*, também o seja como o conteúdo de uma *variável alfanumérica*. Outros ainda vão mais longe e permitem a *referência* a uma *instrução de especificação de formato*. Essa *instrução de especificação de formato* pode ter como *palavra-chave*

```
IMAGE
```

,

ou, em outra implementação,

```
FORMAT
```

,

ou, simplesmente, *dois pontos*,

```
:
```

.

Nestes casos o *formato* é especificado como uma *cadeia de símbolos* que, nos dois primeiros casos, vem entre *aspas*, ou seja, uma *constante literal*, formada segundo as mesmas regras já apresentadas para a *instrução* **PRINT** USING.

1Ø.2 Programação em linguagem de máquina

É possível, se bem que não seja simples, executar programas em linguagem de máquina, isto é, linguagem do microprocessador Z8Ø. É necessário conhecer a linguagem de máquina ou a linguagem *assembler* ou simbólica desse microprocessador. Com um programa escrito em *assembler*, é possível transcrevê-lo para um código numérico que será armazenado internamente antes da execução. Esse armazenamento é possível através da instrução **POKE**, dentro de um programa escrito em linguagem BASIC. Pode-se, dentro de um programa BASIC, ler o conteúdo de posições de memória através de uma referência à função PEEK. A execução de uma rotina escrita em linguagem de máquina é iniciada a partir de um programa em linguagem BASIC, pela função USR. Necessariamente, a rotina tem que terminar por uma instrução RET (em *assembler* do Z8Ø). Deve-se observar que os códigos a serem fornecidos à instrução **POKE** são valores *decimais*, correspondentes às instruções do microprocessador Z8Ø. Portanto, é preciso dispor de uma tabela com a correspondência entre as instruções de Z8Ø e os códigos (decimais). Além do mais, os endereços a serem referenciados são endereços de máquina, isto é, endereços absolutos de memória principal e registradores internos da UCP. Portanto, é preciso conhecer a arquitetura do microprocessador e também o uso do espaço de endereçamento da memória principal. A programação em linguagem de máquina ou *assembler* do microprocessador Z8Ø está fora do escopo deste livro. Serão apresentadas tão-somente a instrução e funções de BASIC que permitem armazenar e executar os programas em linguagem de máquina. A seguir serão descritas esta instrução e funções.

A primeira das facilidades para programação em código de máquina é a função PEEK, que permite ler o conteúdo de uma posição de memória. Dado um endereço de memória como parâmetro da função PEEK, ela retorna um valor que é o conteúdo dessa posição de memória. Alguns exemplos de referências à função PEEK encontram-se na figura 1Ø-1.

FIGURA 1Ø-1: *EXEMPLOS DE REFERÊNCIA À FUNÇÃO* PEEK

EXEMPLOS	EXPLICAÇÃO
PEEK (Z – 128)	A função retorna o valor decimal correspondente ao código binário que se encontra armazenado na posição de memória cujo endereço é dado pelo valor inteiro da expressão Z – 128.

PEEK (16396)	A função retorna o valor decimal correspondente ao código binário que se encontra armazenado na posição de memória cujo endereço é 16396.

A forma geral de uma referência à função de *leitura de posição de memória* é dada por

PEEK (<*expressão aritmética*>)

A *expressão aritmética* da referência à função PEEK fornece o valor inteiro que será usado como endereço para a leitura da posição de memória. O endereço deve ser um valor *decimal*.

Um exemplo de programa que lê posições de memória onde estão armazenadas as instruções do próprio programa e exibe o seu endereço e conteúdo na tela do vídeo é apresentado na figura 1Ø-2. O exemplo apresentado se refere a micros com teclado plano, em particular o TK 82-C, onde o endereço inicial dos programas é 16506 e o endereço do início da área seguinte à área de programa está contido nas posições 16396 e 16397 de memória. Os endereços são armazenados na forma de dois *bytes*, sendo a parte menos significativa no primeiro *byte* e a mais significativa no segundo *byte*. Para se conseguir o valor de um endereço, é preciso obter a soma do conteúdo da primeira posição com o produto do conteúdo da segunda multiplicado por 256.

FIGURA 1Ø-2: *EXEMPLO DE PROGRAMA COM USO DA FUNÇÃO* PEEK

```
1Ø  LET    A = 16506
2Ø  LET    B = PEEK (16396) + PEEK (16397) * 256 – 1
3Ø  PRINT  "ENDERECO", "CONTEUDO"
4Ø  FOR    I = A TO B
5Ø  PRINT  I, CHR$ (PEEK (I))
6Ø  NEXT   I
```

A instrução **POKE** permite escrever em uma posição de memória, isto é, colocar um dado em uma posição de memória. Com essa instrução é possível, por exemplo, armazenar na memória uma rotina em código de máquina. Também é possível alterar o conteúdo de posições de memória. Em geral, os microcomputadores utilizam o conceito de *memória mapeada*, que são endereços de memória para comunicação com periféricos. Um exemplo comum

são as áreas de memória destinadas a representar os caracteres que serão exibidos na tela do vídeo. A colocação de caracteres nessa área produz sua exibição automaticamente na tela, o que pode ser feito com a instrução **POKE**. Alguns exemplos da instrução POKE encontram-se na figura 1Ø-3.

FIGURA 1Ø-3: *EXEMPLOS DA INSTRUÇÃO* **POKE**

EXEMPLO	EXPLICAÇÃO
POKE X + 3, 213	Armazena, na posição de memória cujo endereço é o valor inteiro da expressão X + 3, o código binário correspondente ao valor decimal 213.
POKE 16732, CODE("A")	Coloca na posição de memória 16732 o código binário correspondente ao caractere A.

A forma geral da instrução para *armazenar código na memória* é dada por

POKE <*expressão aritmética*> , <*expressão aritmética*>

Na forma geral da instrução **POKE**, a primeira *expressão aritmética* fornece o valor inteiro que será usado como endereço da posição de memória onde vai ser armazenado o dado cujo código é o valor inteiro da segunda *expressão aritmética*. Chama-se a atenção para o fato de que tanto o endereço como o código são valores *decimais*.

Um programa que demonstra o uso da instrução **POKE** e que coloca no centro da tela uma coluna de caracteres é apresentado na figura 1Ø-4. O programa utiliza a área de memória que é mapeada na tela do vídeo, isto é, coloca caracteres nas posições de memória de onde o vídeo recebe os caracteres que serão exibidos na tela. Os endereços de memória utilizados se referem a micros com teclado plano, em particular o TK 82-C. Em cada microcomputador os endereços diferem. No programa apresentado, o endereço onde começa a área mapeada na tela se encontra nas posições de memória 16396 e 16397 (dois *bytes*). Essa área tem 793 posições, correspondentes a 24 linhas com 32 caracteres, separadas por um código sem representação gráfica, NEWLINE (decimal: 118; hexadecimal: 76). O código de NEWLINE é incluído na área 25 vezes e aparece na primeira e última posições da área, além de figurar também entre as linhas. Esses códigos de NEWLINE não podem ser borrados quando se escreve nessa área, senão a tela se apaga (é perdida).

FIGURA 10-4: *EXEMPLO DE PROGRAMA COM USO DA INSTRUÇÃO* **POKE**

```
10  LET   L = PEEK (16396) + PEEK (16397) * 256
20  FOR   I = 0 TO 21
30  POKE  L + I * 33 + 16, CODE ("*")
40  NEXT  I
```

Um exemplo mais completo, que utiliza a tela inteira para colocação de caracteres, é apresentado na figura 10-5.

FIGURA 10-5: *EXEMPLO DE PROGRAMA COM A INSTRUÇÃO* **POKE** *COLOCANDO CARACTERES NA TELA INTEIRA*

```
10  LET   L = PEEK (16396) + PEEK (16397) * 256
20  FOR   I = 0 TO 23
30  FOR   J = 1 TO 32
40  POKE  L + I * 33 + J, CODE ("+")
50  NEXT  J
60  NEXT  I
```

Os códigos que podem ser escritos por uma instrução **POKE** em uma posição de memória (um *byte* = 8 *bits*) são valores decimais entre 0 e 255. Para que se possa armazenar um valor numérico entre −128 e +127, um *byte* é suficiente. Todavia, para o armazenamento de valores numéricos inteiros maiores é necessário utilizar 2 *bytes*. Em dois *bytes* é possível escrever valores positivos até 65536 ou no intervalo −32768 a +32767. O armazenamento de valores que ocupem 2 *bytes* é feito colocando-se no primeiro *byte* a parte de mais baixa ordem do número e no segundo *byte* a parte de mais alta ordem do número. Um exemplo é apresentado na figura 10-6, onde nas posições 16444 e 16445 da memória é armazenado um valor inteiro que se encontra em uma variável N.

FIGURA 10-6: *EXEMPLO DE ARMAZENAMENTO DE VALOR INTEIRO EM 2* BYTES

```
----------------
----------------
150  POKE  16444, N − (N/256) * 256
160  POKE  16445, N/256
----------------
----------------
```


A última das facilidades para programação em código de máquina é a função USR. Essa função não apenas provoca a execução de uma sub-rotina, cujo endereço inicial deve ser fornecido como parâmetro, como também retorna um valor, derivado da execução da própria sub-rotina. Essa sub-rotina tem necessariamente que estar armazenada antes, em linguagem de máquina. Adicionalmente, a sub-rotina tem que terminar por uma instrução de retorno de sub-rotina (no *assembler* do Z80: RET).

Alguns exemplos de referência à função USR encontram-se na figura 10-7.

FIGURA 10-7: *EXEMPLOS DE REFERÊNCIA À FUNÇÃO* USR

EXEMPLOS	EXPLICAÇÃO
USR (3 * X 2)	Provoca um desvio para a sub-rotina em linguagem de máquina cujo endereço é o valor inteiro da expressão 3 * X − 2 e retorna um valor numérico, dependente da sub-rotina.
USR (17738)	Inicia a execução da sub-rotina em linguagem de máquina cuja primeira instrução se encontra no endereço 17738 da memória e retorna um valor numérico, dependente da execução da sub-rotina.

A forma geral de uma referência à função de *execução de sub-rotina em linguagem de máquina* é dada por

USR (<*expressão aritmética*>)

O endereço de início da sub-rotina em linguagem de máquina é o da sua primeira instrução executável e é o valor inteiro da *expressão aritmética*. A última instrução da sub-rotina a ser executada deve ser necessariamente uma instrução de *retorno de sub-rotina*. O valor que retorna tem que ser colocado, pela sub-rotina, em um determinado registrador (no caso do micro TK 82-C: par BC), antes do retorno. Após o retorno, a função terá o valor colocado no registrador.

Um exemplo que esclarece o uso da função USR para a execução de uma sub-rotina em linguagem de máquina é apresentado na figura 10-8.

FIGURA 1Ø-8: *EXEMPLO DE PROGRAMA COM O USO DA FUNÇÃO* USR

```
----------------
----------------
1ØØ  LET    Y = A * USR (B ** 2 – C) + 7
----------------
----------------
```

No exemplo da figura 1Ø-8 é suposta a existência de uma sub-rotina em linguagem de máquina a ser iniciada pela função USR, no endereço que é o valor inteiro da expressão B ** 2 – C. Após a execução da sub-rotina, retorna um valor numérico que é atribuído à função USR. Esse valor de USR é usado no cálculo da expressão cujo valor será atribuído a Y. Pressupõe-se que tenham sido previamente atribuídos valores às variáveis A, B e C. Supondo-se que o valor de A seja 5 e o valor de USR seja 2, o valor que será atribuído a Y é 17.

Exercícios:

ESTUDO DIRIGIDO:

1) Que é formatação implícita de dados? Que é formatação explícita de dados? O BASIC dos menores micros (primeiro grupo) possui formatação explícita de dados? Para que serve a formatação explícita? A formatação implícita não é suficiente para colocar os dados em posições determinadas na saída? Qual a instrução usada para a saída de dados explicitamente formatados? Qual a forma geral da instrução **PRINT** USING? Como é definido o formato de saída na instrução **PRINT** USING? Qual a relação entre as expressões que aparecem na lista de saída e o formato? Que é formato numérico? Que é formato numérico editado? Que é formato alfanumérico? Qual o símbolo de conversão de dígitos e sinais numéricos? Qual o efeito do símbolo de conversão numérico sobre os zeros de mais alta ordem (não-significativos)? Qual o papel do ponto fracionário na edição de números? Quais os símbolos de conversão para o expoente se a formatação é em notação científica? Quais os principais símbolos usados na edição de dados numéricos? Como obter o cifrão fixo ou flutuante? Como obter os sinais fixos ou flutuantes? Qual o símbolo de proteção de cheque e como opera? Qual o papel da vírgula na edição de números? Quais os símbolos de edição de alfanuméricos e como operam? É possível definir o formato de saída fora da instrução **PRINT** USING? Para que serve a instrução **IMAGE**? Que outras formas pode ter a instrução **IMAGE** em algumas versões do BASIC?

2) Que é programação em linguagem de máquina? Que instrução e funções estão disponíveis em BASIC para permitir a execução de programas em linguagem de máquina? Que é preciso conhecer para programar em linguagem de máquina? Que é linguagem simbólica ou *assembler*? Que é preciso fazer para que um programa em *assembler* possa ser executado? A instrução **POKE** e as funções PEEK e USR são executadas em um programa escrito em que linguagem? A sub-rotina escrita em linguagem de máquina é executável a partir de um programa escrito em que linguagem? Qual a função que inicia a execução de uma rotina em linguagem de máquina? Qual a instrução que permite escrever na memória do computador a rotina em linguagem de máquina? Qual a função que permite ler uma posição de memória dada por seu endereço absoluto? Os valores usados como endereço e valor na instrução e funções para programação em linguagem de máquina pertencem a que sistema de numeração (decimal, hexadecimal, binário etc.)? Qual a forma geral da função PEEK? Qual a forma geral da instrução **POKE**? Qual a forma geral da função USR?

PROBLEMAS:

1) Escreva um programa que leia valores numéricos quaisquer (inteiros ou fracionários) e exiba apenas a parte inteira, com até seis dígitos, inclusive o sinal, usando a formatação explícita. Execute.

2) Escreva um programa que leia valores numéricos quaisquer e os exiba, com quatro dígitos (inclusive o sinal) na parte inteira, ponto fracionário e três dígitos na parte fracionária. Execute.

3) Escreva um programa que leia valores numéricos quaisquer e os exiba na forma de notação científica (exponencial), com dois dígitos (inclusive o sinal) na parte inteira, ponto fracionário e cinco dígitos fracionários. Execute.

4) Escreva um programa que leia valores numéricos que representam quantias em dinheiro e os exiba em formato editado, com cifrão flutuante, vírgula de separação de classes e ponto fracionário. Exibir os valores com seis dígitos na parte inteira e dois dígitos fracionários. Executar. Alterar o cifrão para que permaneça fixo, na primeira posição. Reexecutar.

5) Escrever um programa que leia uma cadeia de caracteres e a exiba centralizada na tela. Executar. Alterar para alinhar à esquerda (alternativamente: direita). Reexecutar. Alterar para estender o campo se a cadeia de caracteres for maior que o campo. Reexecutar.

6) Alterar os programas dos exercícios anteriores (1 a 5) para a utilização da instrução **IMAGE** (ou **FORMAT**). Se o BASIC que você dispõe permitir, reexecute os programas com a nova forma.

7) Escrever um programa que leia o endereço de carga de uma sub-rotina em linguagem de máquina e leia e carregue essa sub-rotina a partir do teclado. A leitura e carregamento são feitos *byte* a *byte*. Logo após a carga, o programa deverá exibir na tela o endereço inicial e o tamanho da sub-rotina carregada e iniciar sua execução, exibindo ao final, na tela, o valor que a função USR retornar.

8) As posições de memória que se iniciam no endereço 16444 até 16476 do micro TK 82-C não são utilizadas pelo sistema (estão disponíveis). Escreva um programa que leia uma cadeia de até 32 caracteres e a armazene nessa área. A partir da área, leia a cadeia e a exiba na tela. Execute o programa.

BIBLIOGRAFIA

1) DATA GENERAL CORPORATION,
Basic BASIC, order nº 093-000088-01, rev. 01, february/1976, Southboro, Massachusetts, USA, 200 pp.

2) DATA GENERAL CORPORATION,
Business BASIC reference manual, order nº 093-000137-02, preliminary, march/1978, Southboro, Massachusetts, USA, 325 pp.

3) DATA GENERAL CORPORATION,
Extended BASIC system manager's guide, order nº 093-000119-02, rev. 02, october/1977, Westboro, Massachusetts, USA, 54 pp.

4) DATA GENERAL CORPORATION,
Extended BASIC user's manual, order nº 093-000065-09, rev. 09, august/1979, Westboro, Massachusetts, USA, 170 pp.

5) DIGITAL EQUIPMENT CORPORATION,
BASIC-plus-2 language manual, order nº AA-0153A-TK, july/1977, Maynard, Massachusetts, USA, 190 pp.

6) DIGITAL EQUIPMENT CORPORATION,
BASIC Conversational language manual, order nº DEC-10-LBLMA-A-D, march/1974, Maynard, Massachusetts, USA, 160 pp.

7) DIGITAL EQUIPMENT CORPORATION,
BASIC user's guide, order nº DEC-20-LBMAA-A-D, january/1976, Maynard, Massachusetts, USA, 130 pp.

8) HEWLETT-PACKARD COMPANY,
HP 3000 BASIC for beginners, order nº 03000-90025, november/1972, Cupertino, Califórnia, USA, 66 pp.

9) HEWLETT-PACKARD COMPANY,
HP system 45 Desktop Computer: operating and programming, part. nº 09845-90000, september/1977, Loveland, Colorado, USA, 250 pp.

10) HEWLETT-PACKARD COMPANY,
HP System 45 Desktop Computer: beginner's guide, part nº 09845-90001, Loveland, Colorado, USA, 130 pp.

11) HEWLETT-PACKARD COMPANY,
HP System 45 Desktop Computer: reference guide, part nº 09845-90010, Loveland, Colorado, USA, 60 pp.

12) HIRSCH, Seymour C.,
BASIC programming: self-taught, Reston Pub., Reston, Virginia, USA, 1980, 277 pp.

13) KRESCH, Roberto,
Microcomputadores: introdução à linguagem BASIC, Editora Rio, Rio de Janeiro, 1ª ed., 1982, 225 pp.

14) LABO ELETRÔNICA S. A.,
Sistema LABO 8034: manual de programação BASIC Comercial, rev. 01, 15-01-1979, rel. 3.3, São Paulo, 480 pp.

15) MICRODIGITAL Eletrônica Ltda.,
TK 82-C científico – Manual de operações, Microdigital, São Paulo, 1982, 80 pp.

16) OLANDOSKI, Marcos & SALENBAUCH, Pedro,
Sistema BASIC – IBM-1130: manual de uso, 3ª ed., NCE/UFRJ, Rio de janeiro, 40 pp.

17) STEINBRUCH, Marília,
Introdução à linguagem BASIC, Editora Nobel, 3ª imp., 1981, 75 pp.

18) SZWARCFITER, Jaime Luiz,
Uma sistematização do processamento de dados – aplicação em automação de bibliotecas, tese de mestrado, COPPE/UFRJ, Rio de Janeiro, abril/1971, 206 pp.

19) WIRTH, Niklaus,
Programação sistemática, Editora Campus, Rio de Janeiro, 1978, 192 pp.

APÊNDICE

SUMÁRIO DE INSTRUÇÕES BASIC

1 Instruções básicas

DATA	declaração de dados (internos)
END	fim físico e lógico de programa
FOR	início de malha de repetição controlada
GOTO	desvio incondicional simples
IF... THEN	desvio condicional
INPUT	entrada de dados pelo teclado
LET	atribuição de dados
NEXT	fim de malha de repetição controlada
PRINT	saída de dados na tela do vídeo
READ	aquisição de dados (internos)

2 Instruções avançadas

CLS	limpa a tela do vídeo
DEF FN	definição de função do usuário
DIM	declaração de variáveis e conjuntos
GOSUB	desvio incondicional para sub-rotina
IMAGE	especificação de formato de saída de dados
ON... GOSUB	desvio incondicional múltiplo para sub-rotina
ON... GOTO	desvio incondicional múltiplo
PAUSE	interrompe a execução do programa por um intervalo
PLOT	exibição de um *pixel* na tela
POKE	armazenamento de dado em uma posição de memória
PRINT USING	saída de dados com especificação de formato
RANDOMIZE	reinicialização da raiz da função RND
REM	comentários
RESTORE	reinicialização do apontador de dados internos

RETURN	retorno de sub-rotina e de subprograma
SCROLL	provoca a exibição de dados na parte inferior da tela e rola a tela uma linha para cima ao ocorrer a exibição de cada linha
STOP	fim lógico de programa (parada)
UNPLOT	remoção de um *pixel* da tela

ÍNDICE ANALÍTICO