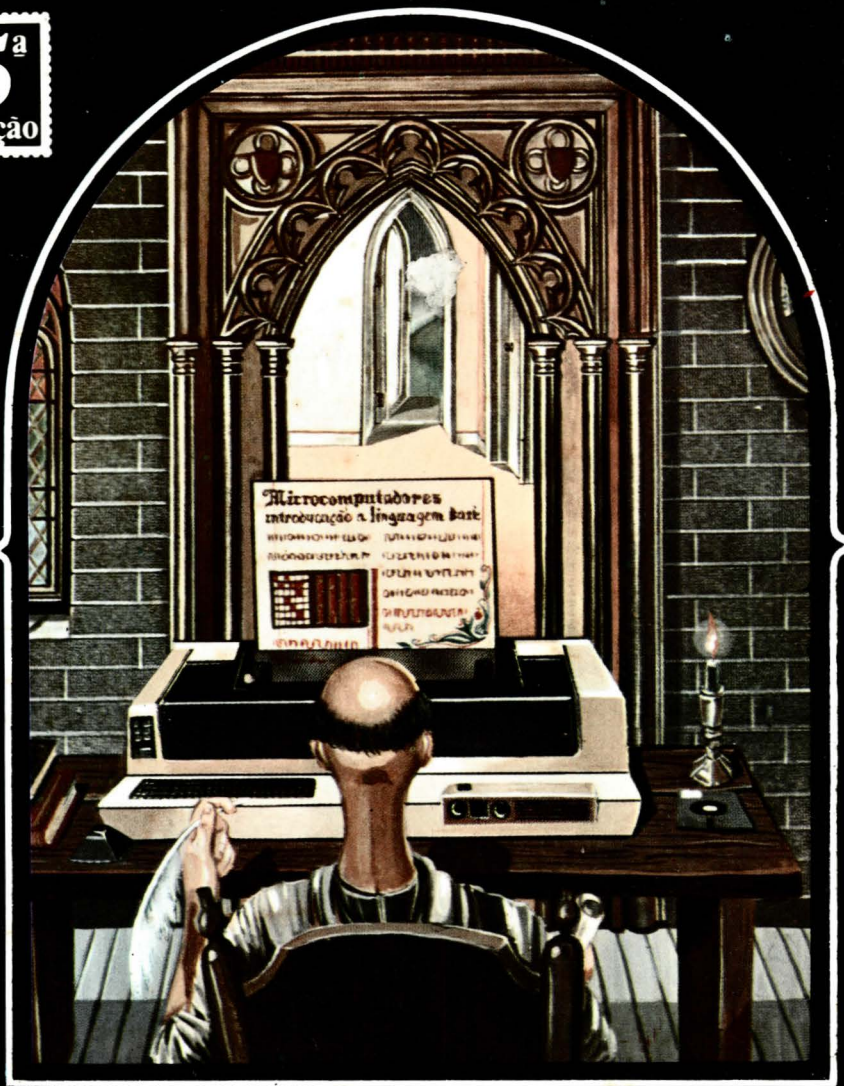
MICROCOMPUTADORES

Introdução à linguagem basic

5ª edição



ROBERTO KRESCH

Editora Rio
ESTACIO DE SÁ

Este livro foi editado em convênio com

FACULDADE POLYTECHNICA ESTÁCIO DE SÁ
E
EMPRESA DIGITAL BRASILEIRA S/A – DIGIBRÁS

ROBERTO KRESCH
Engenheiro de Telecomunicações

MICROCOMPUTADORES

Introdução à linguagem basic

5ª edição

Editora Rio
ESTÁCIO DE SÁ

Capa:
Arte final da Editora Rio
inspirada na ilustração de Hovik
Dilakian, publicada na revista
***Electronic Learning*, de março/abril/82**

EQUIPE DA EDITORA RIO
Revisão:
Francisco de Castro Azevedo
Aleidis de Beltran
Composição:
Bento José Neto
Montagem:
Luiz Epitácio de Oliveira
Arte:
Renato Martins Dias

As meus filhos André, Liana, Renata e Felipe, que participarão ativamente de uma nova era da história humana, assustadora para alguns e maravilhosa para outros mas, acima de tudo, real e vigorosa.

APRESENTAÇÃO

O brasileiro, por sua formação cultural, é um leitor que dispensa o maior apreço, e mesmo uma acentuada preferência pelos textos técnicos redigidos de forma simples e clara, sem prejuízo do valor teórico, conceitual ou didático do trabalho. Poderia até acrescentar que ele exige, comumente, do autor, um estilo escorreito, associado a uma exposição rigorosamente lógica. Partimos, talvez, do princípio de que a exatidão das idéias induz necessariamente à sua apresentação facilmente inteligível.

Por isso, estou certo de que o livro que tenho a grande satisfação de apresentar, obterá livre curso entre todos os que se interessam por conhecer as linguagens de programação, de alto nível, elaboradas para computadores, não importando o grau de conhecimento que possuam sobre o assunto.

Roberto Kresch foi muito feliz na escolha do tema de seu livro, bem como na transmissão de seus princípios fundamentais e do mecanismo de seu uso.

Contribui, ademais, para a difusão do uso de equipamentos eletrônicos de tratamento de dados e da informação, prestando desta forma um grande serviço à Informática no Brasil.

Afirmo, sem constrangimento, que ao cabo da primeira leitura da "Introdução à Linguagem Basic" tornei-me um programador razoável de microcomputadores, e com grande surpresa para mim, que me considerava um leigo na matéria, habilitado a resolver, com rapidez, problemas de cálculos

aritméticos, estatísticos e algébricos, cuja solução sempre me pareceu exigir um trabalho árduo e relativamente longo.

Que essa afirmação sirva de estímulo aos que pretendem empregar a linguagem de programação BASIC na utilização dos mini e dos microcomputadores.

Que desperte também a curiosidade dos que ainda não tiveram nenhum contacto com o místico instrumento da computação para conhecer sua extraordinária e diversificada potencialidade, como ferramenta inusitada de trabalho, utilizável na grande maioria das disciplinas do conhecimento.

Por último, anuncio com prazer que este livro é o primeiro de uma série de publicações técnico-didáticas que a DIGIBRÁS pretende patrocinar.

Brasília, 29 de abril de 1982.

Paulo Augusto Cotrim Rodrigues Pereira
Diretor Presidente
DIGIBRÁS

PREFÁCIO

Este texto foi preparado com a finalidade de suprir uma deficiência observada pelo autor no mercado brasileiro, em relação à literatura em língua portuguesa descrevendo a linguagem de programação BASIC. Considerando a crescente introdução do uso de microcomputadores no País, e o crescente número de usuários "leigos", isto é, pessoas que não tiveram, anteriormente, treinamento formal ou de longa prática no campo de computação, ficou patente a necessidade de se difundir essa linguagem, mesmo no âmbito daqueles que não tenham condições de se utilizarem de textos em língua estrangeira.

A linguagem BASIC ou, simplesmente, o BASIC, talvez seja a linguagem de alto nível de mais fácil apreensão pelos operadores de computadores e, também, aquela que pode ser mais imediatamente usada; em outras palavras, à proporção que o "vocabulário" do BASIC vai sendo aprendido, ele pode ser, imediatamente, usado em um computador, permitindo, desde níveis elementares de conhecimento, a preparação de pequenos programas, os quais serão, paulatinamente, sofisticados, à medida que o vocabulário conhecido for aumentado. É experiência pessoal do autor que essa característica do BASIC é altamente gratificante, em termos de desenvolvimento dos usuários na arte e ciência da programação, em virtude da possibilidade de poder penetrar nos meandros da linguagem aos poucos, sem prejuízo da facilidade de usar imediatamente cada tipo de instrução aprendida.

Este texto é, obviamente, baseado em diversos livros publicados recentemente, no estrangeiro, e também em algumas experiências próprias do autor e seu filho, André; assim, pudemos inserir no texto "dicas" de programação que foram aprendidas no dia-a-dia da preparação de programas e,

também, diversos exemplos, que poderão ser experimentados pelos leitores, desde que feitas as devidas adaptações relativas a peculiaridades do BASIC da máquina a que cada um possa ter acesso.

Não poderia deixar de agradecer à DIGIBRÁS, e aos companheiros que nela trabalham, pela assistência e sugestões que deram em relação ao texto, fazendo valer a sua experiência profissional no ramo de computação e em muito contribuindo para o aperfeiçoamento do trabalho.

Gostaria de agradecer, também, às FACULDADES INTEGRADAS ESTÁCIO DE SÁ, pelo apoio e colaboração na parte didática deste texto, e à EDITORA RIO, pela ajuda na montagem e pela impressão do mesmo.

Finalmente, não poderia deixar de agradecer à tremenda paciência de minha secretária, Maria Amélia, durante todo o trabalho de datilografia dos manuscritos, especialmente as listagens de programas de exemplos.

Brasília, DF, dezembro de 1981.

Roberto Kresch

PREFÁCIO DA SEGUNDA EDIÇÃO

O grande incremento ocorrido nos últimos meses, na fabricação e venda de microcomputadores no País, contribuiu, sem dúvida, para a necessidade de se reeditar este livro, pouco depois da primeira impressão.

Como não podia deixar de ser, notei, após o lançamento da primeira edição, que havia diversos aspectos merecedores de melhor explicação, algumas expansões de conceitos e, por que não dizer, correções no texto daquela edição.

Procurei, até onde me foi possível, contando com o auxílio de amigos e colegas, melhorar o texto anterior, dando-lhe maior generalidade, em termos de máquinas e, mesmo, de variações existentes na linguagem BASIC.

A própria existência, hoje, no mercado brasileiro, de microcomputadores nacionais, o que não ocorria quando iniciei a preparação do texto original, muito contribuiu para a inclusão de alguns conceitos e adaptação de outros, tornando a obra mais atualizada, dentro da dinâmica própria deste setor da atividade humana.

Grande parte dos conceitos adicionais foi concentrada em um APÊNDICE, para o qual gostaria de chamar a atenção do leitor.

Reitero meus agradecimentos aos que colaboraram, direta ou indiretamente na preparação deste trabalho.

Brasília, setembro de 1982.

Roberto Kresch

ÍNDICE GERAL

Capítulo I
INTRODUÇÃO

A linguagem de programação BASIC, cujo nome é derivado das iniciais de Beginners All-purpose Symbolic Instruction Code (Código de Instrução Simbólico, de Uso Geral, para Principiantes), foi desenvolvida por John Kemeny e Thomas Kurtz, no Dartmouth College, no início da década de 60, sendo introduzida no mercado em 1964.

A princípio procurou-se implementar uma linguagem que não fosse especializada para determinadas aplicações e, portanto, permitisse um uso mais generalizado, ao mesmo tempo que fosse adequada para a iniciação de pessoas de formação em áreas as mais diversas, na técnica da programação de computadores. Com o passar do tempo as máquinas foram sendo aperfeiçoadas, seu preço foi caindo rapidamente e, hoje, já se pode encontrar microcomputadores usados para fins residenciais, especialmente na área do lazer. Ao mesmo tempo, o "vocabulário" e a "gramática" do BASIC foram sendo aperfeiçoados e ampliados, de modo que, atualmente, essa linguagem pode ser considerada bastante poderosa, compatível com as aplicações mais variadas, sejam comerciais ou científicas, sem prejuízo de sua facilidade de manipulação e rapidez do aprendizado.

O BASIC pode ser encontrado, nos computadores, como programa residente, isto é, já estar gravado em memórias permanentes dentro do próprio computador, ou estar gravado em um meio de armazenagem externo, tal como disco magnético ou fita cassete. Neste caso, antes de se poder usar o computador com a linguagem BASIC, é necessário carregá-lo com o programa de linguagem, ou seja, colocar nas memórias do computador o conteúdo do disco magnético, cassete ou outro tipo de armazenador externo de dados.

No texto que se segue suporemos sempre que o programa de linguagem BASIC já se encontra armazenado nas memórias do computador e que está pronto para uso.

O programa que "traduz" para o computador as instruções que serão introduzidas no mesmo em BASIC pode assumir três formas principais: interpretador, tradutor e compilador. Para poder explicar melhor essas expressões devemos fazer uma curta digressão, a respeito da maneira de funcionar de um computador típico, no que tange à sua capacidade de entender e executar instruções.

As diversas partes que compõem o computador se comunicam entre si usando um código especial, binário (ou seja, formado apenas por zeros e um's); esse código, que contém as instruções de operações e os dados a serem computados, recebe a denominação genérica de "linguagem de máquina". Se um programa for introduzido no computador usando a sua linguagem de máquina (que é diferente para cada tipo de computador), ele executará esse programa com extrema rapidez, pois não haverá necessidade de se fazer nenhuma tradução de linguagem.

Quando se usa uma linguagem cujas instruções são palavras do vocabulário comum (vocabulário inglês, bem entendido), essa linguagem é denominada de alto nível e necessita que haja, no computador, um programa especial, capaz de passar as instruções tipo palavra ou frase para instruções em linguagem de máquina, que o computador será capaz de entender e executar.

O programa especial para cada tipo de linguagem de alto nível e cada linguagem de máquina pode ter, entre várias outras, três formas principais: interpretador, tradutor e compilador.

No interpretador as instruções são convertidas para linguagem de máquina no momento em que a instrução é lida, ou seja, cada vez que o computador começa a execução do programa, as instruções são convertidas e executadas uma a uma (por conjunto de instruções, denominado linha, como explicaremos adiante) em linguagem de máquina.

No caso do compilador, o programa introduzido em linguagem de alto nível é convertido de uma vez só em linguagem de máquina, resultando dessa conversão um novo programa, que é denominado compilação do programa original

ou programa objeto (para significar que é o programa que o computador é capaz de entender e executar). Uma vez que o programa foi compilado pela primeira vez, as próximas execuções do mesmo serão feitas diretamente por leitura do programa objeto, ou seja, muito mais rapidamente (por não ser necessário "traduzir" novamente o programa em linguagem de alto nível). No entanto, cada vez que se introduz uma alteração em um programa já compilado, é necessário fazer nova compilação do programa inteiro, o que diminui um pouco a facilidade de operação para o usuário, na fase de preparação dos programas.

O tradutor se comporta de maneira semelhante ao interpretador. Apenas, ao invés de armazenar as instruções e comandos tal como os mesmos são entrados pelo teclado, o tradutor vai armazenando uma versão codificada das instruções, diminuindo o espaço necessário na memória para guardar o programa. No entanto, cada linha (já codificada) será convertida em linguagem própria de máquina, uma a uma, não sendo produzido um programa completo compilado. O tradutor é mais rápido que o interpretador, na execução, mas ainda é bem mais lento que o compilador.

Guarda a vantagem da flexibilidade de operação, da mesma forma que o interpretador.

No que se segue, deixaremos de lado as questões relativas aos méritos de interpretadores, tradutores e compiladores e nos dedicaremos à descrição da linguagem em si. No entanto, diversos exemplos que serão mostrados no texto pressupõem que o computador possui um interpretador e não um compilador (embora todas as instruções e mecanismos de programação sejam as mesmas para ambos os casos).

A linguagem BASIC possui um "vocabulário" especial para o trabalho com unidades de gravação em disco magnético. Esse vocabulário, que permite criar arquivos para gravação em disco e ler arquivos já gravados, além de uma série de outras funções relacionadas com o armazenamento e recuperação de dados nesses dispositivos periféricos, não será descrito neste livro introdutório.

O BASIC aqui descrito é apenas o necessário para a operação de um microcomputador ligado a uma tela de vídeo, um gravador de fita cassete e, eventualmente, uma impressora.

Também não estão incluídas as funções de edição que permitem alterar e rearranjar programas e textos já armazenados no computador, através de comandos especiais.

Esperamos em um próximo volume incluir o vocabulário BASIC para operação com discos magnéticos e para edição de textos.

Finalmente, gostaríamos de enfatizar que o tipo de interpretador BASIC exposto neste livro é o denominado "MICROSOFT", devido ao nome da empresa de "software" que o desenvolveu e que é, talvez, o mais difundido atualmente. Outros tipos de BASIC podem diferir do que está descrito a seguir, dependendo de sua origem.

Capítulo II
ORGANIZAÇÃO DO BASIC

A linguagem BASIC obedece a certas regras de organização que descreveremos a seguir. Como toda linguagem, ela tem sua gramática própria, que deverá ser obedecida, sob pena de o computador não poder entender a mensagem.

As informações que são contidas no programa em BASIC podem ser de várias naturezas:

— COMANDO é um tipo de instrução que determina ao computador que execute determinada tarefa, diretamente e sem condições;

— STATEMENT é uma frase escrita em BASIC, contendo instruções para a execução de uma certa tarefa e dados a serem manipulados pelo computador; cada STATEMENT instrui o computador para executar um tipo de operação (que pode ser direta, do tipo COMANDO, ou condicional, como veremos mais adiante);

— DADOS são valores numéricos ou alfanuméricos (incluindo letras do alfabeto) que serão usados na execução das instruções, pelo computador;

— VARIÁVEIS são letras ou combinações de letras e números, aos quais, na execução do programa, serão atribuídos valores numéricos ou alfanuméricos;

— STRING é um conjunto de letras e números que o computador manipulará tal como foram escritos no programa; as letras e os números não têm nenhum significado especial para o computador.

Os "statements", que incluem comandos, instruções de operação que não são comandos, variáveis e "strings", devem, no programa, ser sempre precedidos de um NÚMERO DE LINHA. Assim, o programa é constituído de informações colocadas em linhas numeradas seqüencialmente. Mais adiante daremos maiores detalhes sobre as linhas e seus números.

Quando o programa é introduzido no computador, seu aspecto geral é o seguinte:

XX (STATEMENT)
XY (STATEMENT)
XZ (STATEMENT)

em que XX, XY- - - -XZ são os números das linhas, em ordem crescente (esses números podem variar de Ø até um máximo que é definido pela estrutura do computador, sendo, em alguns casos, igual a 9999 e, em outros, até 65000).

Neste ponto, várias dúvidas podem estar surgindo na mente do leitor; vamos tentar esclarecer algumas.

— Como o programa é preparado? O programa pode já existir e estar gravado em fita cassete, fita aberta, disco magnético rígido, fita de papel perfurado, disco magnético flexível ou algum outro meio de armazenamento; se o programa está sendo feito pelo próprio usuário, ele será, normalmente, escrito, analisado e verificado antes de ser introduzido no computador; a introdução será feita através de um teclado, semelhante ao de uma máquina de escrever.

— Como se pode ver o que está sendo introduzido no computador? Na maioria dos casos são utilizados monitores (telas tipo televisão) ou aparelhos de televisão comerciais (nesse caso o computador precisa ter um adaptador para ligar na antena do aparelho); esses "vídeos" mostram continuamente o que está sendo colocado na memória do computador através do teclado ou, em caso de gravação em baixa velocidade, quando conteúdo de uma fita cassete é colocado dentro do computador (quando são usados outros meios de armazenamento, mais rápidos, tais como discos flexíveis ou rígidos, não é possível se ver conteúdo do programa, no vídeo, durante seu "carregamento" ou "loading" na memória).

— O que é necessário fazer para iniciar a "teclagem" do programa para dentro do computador? É necessário saber se o computador está pronto para receber instruções, o que é indicado em geral pela palavra OK (ou, em algumas versões de BASIC, READY), que aparece escrita no canto inferior

(ou superior) esquerdo do vídeo; nesse momento pode ser iniciado o procedimento para a carga, a partir de um meio externo (p. ex.: fita cassete), ou iniciada a teclagem manual das instruções; é necessário, também, apagar o que está na memória do computador (a menos que se esteja adicionando instruções a um programa já armazenado); quando for necessário apagar o que está na memória, basta teclar o comando NEW (ou, em algumas versões do BASIC, os comandos CLEAR ou SCR).

— Como dar a partida na execução do programa? Logo após a introdução da última linha do programa pode-se teclar o comando RUN, o que fará com que o programa seja executado.

Esclarecidos esses pontos fundamentais, vamos prosseguir com a descrição da organização e introdução de uma programa em BASIC no computador.

Cada vez que se quer entrar com uma nova linha (que terá um novo número), é necessário bater na tecla RETURN (proveniente de CARRIAGE RETURN, que significa retorno do carro da máquina de escrever e mudança de linha), ou, em alguns teclados, C.R. ou, ainda, ENTER. É importante entender bem a seguinte mecânica:

Uma linha numerada só pode ter um certo número máximo de caracteres (letras, números, espaços ou sinais diversos); esse número, em geral, é menor que 256, podendo variar com a estrutura do computador usado. Assim, é conveniente mudar de linha antes de chegar a esse valor, para não ter que "estourar" a capacidade da linha, provocando uma indicação de erro, por parte do computador.

Acontece que os vídeos normalmente usados não podem colocar 256 caracteres em uma linha. Nesses casos o tamanho máximo de uma linha no vídeo varia de 22 a 80 (aparelhos comerciais de TV usados como monitor). Por essa razão, o computador subdivide a linha numerada em linhas menores, que cabem na tela do vídeo.

Assim, se introduzirmos uma linha numerada de 72 caracteres e o computador estiver preparado para um vídeo de 24 caracteres, a linha numerada aparecerá na tela como três linhas (porém as duas últimas não começam com um núme-

ro, pois pertencem à mesma linha numerada de 72 caracteres).

Quando se está introduzindo uma linha de programa (numerada) no computador e se termina a teclagem da mesma, é necessário bater a tecla RETURN, o que permitirá a introdução de nova linha.

Os COMANDOS têm a característica de atuação direta, isto é, eles não precisam fazer parte de um programa e podem ser introduzidos no computador sem a necessidade de usar uma linha numerada.

Desse modo, ao se pressionar a tecla RETURN, o comando será executado imediatamente, não requerendo a instrução RUN. Os comandos também podem fazer parte de STATEMENTS, em uma linha numerada; nesse caso, eles só atuarão quando o programa for executado, após o comando RUN. Como exemplo de comando podemos mencionar:

```
PRINT XXXX
```

Quando esse comando for colocado na máquina e pressionada a tecla RETURN, aparecerá "impresso" na tela o número XXXX (ou em uma impressora, se estiver ligada ao computador). Mais adiante mostraremos os comandos utilizados no BASIC.

Quanto à numeração das linhas, deve ser notado o seguinte:

- não é necessário usar números consecutivos;
- a boa prática recomenda que, ao ser escrito um programa, a numeração deixe espaços em branco; por exemplo, poder-se-á numerar as linhas de 10 em 10 (primeira linha, 1Ø — segunda linha, 2Ø, e assim por diante); desse modo, se for necessário introduzir instruções no meio do programa, essas instruções poderão ser numeradas nos espaços vagos (por exemplo, 11-12-13, etc.);
- na hora de escrever o programa não é necessário manter rigorosamente a ordem numérica crescente; o computador colocará as instruções na ordem correta, ao executar o programa; assim, se tivermos introduzido as linhas 1Ø, 2Ø, 3Ø e 4Ø e quisermos colocar uma instrução anterior, podemos teclar, em seguida à

linha 4Ø, por exemplo a linha 25; na hora da execução a linha 25 será executada logo após a linha 2Ø;
- após ser acionada a tecla RETURN, qualquer número colocado no computador será automaticamente considerado como o número de uma nova linha;
- toda a vez que uma linha for introduzida no computador com um número de linha que já existia na memória, a nova linha substituirá automaticamente a anterior de mesmo número; por isso, é conveniente tomar cuidado ao teclar os números das linhas, pois corre-se o risco de apagar ou alterar uma linha já armazenada;
- se for introduzido o número de uma linha, sem nenhum "statement", pressionada a tecla RETURN, essa linha ficará em branco até que seja repetido o seu número seguido de instruções; se o número teclado corresponder a um número de linha já armazenado, a linha anterior será apagada; por exemplo, se já estava armazenada no computador a linha

```
1Ø PRINT XXXX
```

e, posteriormente, for introduzido o número 1Ø e batido o RETURN, a linha anterior deixará de existir no programa, até que uma nova linha com o mesmo número seja introduzida.

Nos capítulos seguintes descreveremos os comandos e instruções existentes no BASIC, assim como a maneira de utilizá-los.

Gostaríamos, neste ponto, de fazer uma observação importante, relativamente aos sinais do alfabeto português, em comparação com o inglês. Na língua inglesa não existem certos sinais, como, por exemplo, cedilha, til e acentos agudos, graves ou circunflexos. Assim, todos os textos que forem escritos em português aparecerão na tela ou na impressora sem esses sinais. No correr deste livro nós usaremos a grafia correta em português, mas o leitor já está prevenido para o não aparecimento desses sinais na tela.

Ex.: quando dizemos que o computador deve imprimir COMPOSIÇÃO, daremos o comando da seguinte forma:

```
10 ? "COMPOSIÇÃO"
```

Mas, na tela, aparecerá escrito:

```
10 ? "COMPOSICAO".
```

Capítulo III
USO DO TECLADO E DOS COMANDOS

3.1 – TECLADO

O teclado normalmente usado para a introdução de programas em BASIC é o denominado ASCII de 53 ou mais teclas, do qual um exemplo é mostrado na página seguinte; deve-se notar que há variantes do tipo de teclado mostrado, principalmente no que se refere às teclas de funções especiais. Mencionaremos algumas das diferenças que podem ser encontradas, mas o leitor deverá verificar, no teclado que tiver à sua disposição, quais as teclas que efetuam as funções que serão descritas. Em alguns teclados, a segunda função, que aparece desenhada em cima das letras, não está explicitada nas teclas, ou seja, só aparece a letra propriamente dita; no entanto a segunda função existe, e o leitor deverá verificar qual a sua posição no teclado (provavelmente é a mesma que será descrita a seguir).

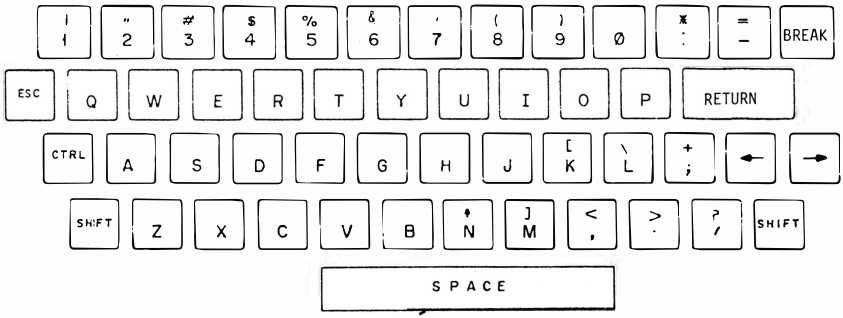
3.1.1 – Teclas numéricas: as teclas indicadas 1 a Ø (zero) representam esses algarismos quando não está pressionada a tecla SHIFT (falaremos mais sobre esta tecla mais adiante); quando se pressiona a tecla SHIFT e, ao mesmo tempo, uma das teclas numeradas, será introduzido no computador o símbolo que está em cima do número pressionado: ! " ≠ $ % & ' ( ).

3.1.2 – Teclas de letras: quando não está pressionada a tecla SHIFT, as letras maiúsculas serão introduzidas; quando é apertado o SHIFT e, ao mesmo tempo uma tecla de letra, será introduzido um outro símbolo ou segunda função, que depende do teclado usado.

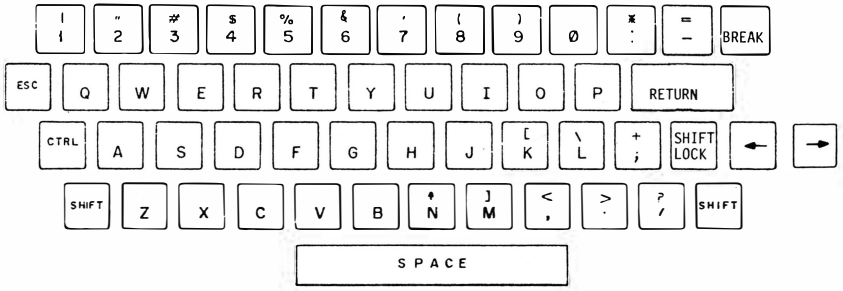
Geralmente os segundos símbolos ou funções são:

– em cima do O: ←, flecha que indica o apagamento do último caracter introduzido (letra, número, símbo-

ASPECTO DO TECLADO
ASC II-53(*)
VERSÃO NORMAL



ASPECTO DO TECLADO
ASC II-53(*)
VERSÃO COM MINÚSCULAS



*** ASCII: AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE**
ESTE TECLADO, O MAIS USUAL, TEM 53 TECLAS – EXISTEM MAIORES E MENORES.

lo ou espaço); isso tem a finalidade de se efetuar correções na hora de introduzir um programa no computador, quando se verifica ter sido cometido um erro da datilografia; esse mesmo efeito é obtido com a tecla ←;

Exemplos:

a) foi teclado **1Ø PRINN**
verificado o erro, tecla-se o O com SHIFT ou ←
aparece **1Ø PRIN** _ (o cursor voltou uma casa)
tecla-se a letra desejada
1Ø PRINT
na memória do computador foi introduzida a palavra certa PRINT.

b) pode-se apagar mais de uma letra:
foi teclado **1Ø PRRIN T**
verificado o erro, tecla-se O com SHIFT tantas vezes quantas letras se quer corrigir (segundo R, I, N, espaço, T, ou seja, 5 vezes). Aparece na tela:

1Ø PR_
correção: **1Ø PRINT**

—em cima do P: @ (arroba), cuja finalidade, em alguns tipos de BASIC, é apagar a linha que está sendo escrita, ***sem alterar*** uma linha anterior que já tenha o mesmo número; ao ser teclado P com SHIFT, ou seja, @, não é necessário teclar RETURN, pois a tecla @ produz o efeito de mudança de linha e retorno do carro.

Exemplo: Programa já existente
1Ø PRINT XXXX
2Ø PRINT YYYY
3Ø END

Queremos corrigir a linha 2Ø, mudando para ZZZ.
Teclamos **2Ø PRINT Z**
Nesse momento, ***antes de teclar RETURN***, mudamos de idéia e queremos manter a linha 2Ø tal como esta-

va e colocar a nova linha com o número 25; teclamos @

```
20 PRINT Z @
25 PRINT ZZZ
```

Agora o programa estará na memória com o seguinte aspecto:

```
10 PRINT XXXX
20 PRINT YYYY
25 PRINT ZZZ
30 END
```

Nota Importante: se, depois de teclar 20 PRINT Z, teclarmos RETURN, a linha anterior será substituída pela nova; nesse caso, para corrigir o programa teremos que teclar novamente a linha 20 PRINT YYYY e depois teclar a nova linha 25 PRINT ZZZ.

— em cima do N: ↑ ou ∧, cuja finalidade é elevar um número a uma potência; por exemplo, se quisermos escrever 2^3, deveremos teclar 2 ↑ 3 (ou, conforme o teclado, 2 ∧ 3).
— em cima do K: [(segundo colchete, esquerdo); serve para se usar dentro de STRINGS; não tem função específica no BASIC, em geral.
— em cima do L: \ (barra inclinada para a esquerda); dependendo da versão do BASIC utilizada, serve para separar instruções diferentes colocadas na mesma linha (veremos isso mais adiante); além disso é um sinal que pode ser usado dentro de um STRING.
— em cima do M:] (mesma observação já vista acima).

Observação: em diversos tipos de teclado aparece a tecla ← ou a tecla Back Space, que tem a mesma função descrita em relação ao SHIFT 0; da mesma forma pode ser encontrada a tecla ↑, que tem a função descrita para SHIFT N.

3.1.3 — Letras minúsculas: em alguns teclados é possível se teclar letras minúsculas; em geral elas não podem ser usadas para introduzir instruções ou comandos no computador, sendo permitidas apenas para imprimir títulos de

programas ou outras frases, ***exclusivamente*** como parte de STRINGS (veja mais adiante).

Nos teclados que permitem letras minúsculas, as duas teclas SHIFT (esquerda e direita) têm funções diferentes: a tecla SHIFT esquerda apertada faz o teclado funcionar normalmente (teclas maiúsculas, números, etc.); a tecla direita faz aparecer a segunda função ou segundo símbolo; se nenhuma das teclas SHIFT for apertada, as teclas de letras farão aparecer as minúsculas.

É evidente que, se queremos trabalhar a maior parte do tempo com o teclado normal, torna-se inconveniente ficar apertando a tecla SHIFT esquerda; nesse tipo de teclado aparece, então, uma nova tecla, que prende ao ser apertada, denominada SHIFT LOCK e que faz o papel da SHIFT esquerda; quando queremos usar minúsculas, basta pressionar e soltar a SHIFT LOCK; quando a SHIFT direita é apertada, ela passa a prevalecer sobre a SHIFT LOCK, permitindo o uso da segunda função sem necessidade de soltar esta última.

Em diversos tipos de teclado, existe uma tecla especial que faz passar de maiúsculas para minúsculas e vice-versa; basta pressionar uma vez e é feita a mudança; pressionando outra vez volta ao tipo anterior; nesses teclados as duas teclas SHIFT tem a mesma função, já mencionada.

3.1.4 — Teclas especiais: as teclas que têm função específica são as seguintes:

— RETURN (ou C.R.): serve para mudar de linha, quando se está introduzindo um programa; deve ser apertada, também, para introduzir qualquer comando, do tipo PRINT, RUN, etc.

— BREAK: interrompe a execução de qualquer programa e faz o computador retornar à posição inicial (aquela que aparece logo após a máquina ser ligada); dependendo do tipo de computador e da tecla apertada após o BREAK, o programa que estava armazenado poderá ou não ser apagado; o leitor deve consultar as instruções do computador que estiver usando para verificar as conseqüências do uso da tecla BREAK. A título de exemplo descrevemos o que ocorre com o microcomputador do autor.

Ao ser ligado o computador, aparece escrito na tecla do vídeo:

D/ C/ W/ M ?

O operador deverá teclar uma dessas letras para indicar o tipo de operação que vai realizar:

D — operação com disco magnético (que tem algumas instruções especiais no BASIC).

C — operação em BASIC, com inicialização, isto é, certas rotinas internas do programa interpretador são passadas para a memória de trabalho, possibilitando o início da operação; essa tecla é usada quando o computador é ligado e se deseja trabalhar em BASIC (C provém de Cold Start).

W — é usada quando, no meio do trabalho em BASIC, houver necessidade de se apertar a tecla BREAK (ou quando isso acontece acidentalmente); depois do BREAK, o computador volta para a posição D/C/W/M?; ao ser apertado o W, o computador está, novamente, em ação, não tendo sido apagados os programas já gravados (Warm Start).

M — é usada para operar em linguagem de máquina (que não é objeto deste livro).

Observação: em diversos computadores a tecla BREAK tem a função de interromper a execução de um programa, de maneira idêntica ao que é descrito em relação a CTRL C; nesses casos a função descrita acima (retorno à condição inicial) é obtida com uma tecla ou botão denominado RESET.

— CONTROL ou CTRL: essa tecla deverá ser usada simultaneamente com algumas outras teclas, para obter diferentes efeitos:

a) com a tecla C: provoca a interrupção do programa que estiver sendo executado, resultando uma mensagem do computador para o usuário do tipo:

BREAK IN XX

o que indica que a próxima linha que seria executada, antes da interrupção, seria a linha nº XX. O comando CTRL C é muito útil para interromper um programa que, por exemplo, tenha entrado em "loop" (o programa roda de uma instrução para ou-

tra e volta, ficando em operação ininterrupta); com o comando CTRL C o "loop" pode ser interrompido e efetuada a correção de um eventual defeito no programa;

b) existem outras letras que podem ser usadas em conjunto com CONTROL; no entanto, seu uso não é comum no BASIC e as funções que elas têm são preenchidas por outras teclas ou conjunto de teclas, como está sendo explicado neste capítulo. Em teleprocessamento, ou em sistemas mais complexos que usam BASIC, poderá aparecer o uso dessas teclas, que descrevemos para completar a informação:

CTRL G — para acionar sineta de alarme
CTRL H — para dar retrocesso no cursor
CTRL J — provoca o pulo de uma linha (LINE FEED)
CTRL V — avança o cursor
CTRL X — apaga a linha que está sendo introduzida
CTRL A — produz o mesmo efeito indicado acima, para SHIFT P (arroba), em certos microcomputadores

— ESC é um caracter usado para dar significado especial a um conjunto de caracteres contíguos, denominados seqüência de escape.

3.2 — ORGANIZAÇÃO DAS INSTRUÇÕES NO VÍDEO

Neste item abordaremos alguns aspectos que julgamos ser do interesse dos usuários de microcomputadores em BASIC, no que se refere à maneira de aparecer e à localização das instruções e resultados na tela.

a) A tela é dividida em ***n*** casas, ou seja, permite a introdução de ***n*** caracteres por linha: a posição do próximo caracter a entrar é marcada por um cursor, ou seja, um sinal que se move da esquerda para a direita, à proporção que os caracteres vão entrando; suponhamos que o monitor ou aparelho de TV usado permita 24 caracteres em uma linha e permita 24 linhas na tela inteira. As casas são numeradas de Ø a 23, e

tudo o que for teclado pelo operador começará na casa Ø, depois de um RETURN.

Ex. 1: Vamos teclar um programa, após apagar o que já estava na memória (veja mais adiante os comandos NEW, PRINT e LIST):

```
NEW
OK
10 PRINT "ABCDEFGHIJKLMN
OPQRSTUVXYZ"
RUN
ABCDEFGHIJKLMNOPQRSTUVXY
Z

OK
```

(casa Ø)

Verificamos que tudo o que foi teclado e o programa que foi executado começou na casa Ø e foi até a casa 23 (evidentemente o programa corta as palavras sem se importar com a questão da divisão correta de sílabas).

b) Dependendo do tipo de microcomputador usado, as linhas serão preenchidas de cima para baixo (o cursor vai descendo, após cada linha ser completada) ou as linhas são sempre impressas na primeira carreira de baixo da tela e vão subindo, à proporção que vão ficando completas: nesse caso o cursor nunca sai da primeira linha de baixo: fica andando, da posição Ø para a posição 23 e depois volta para posição Ø, enquanto a linha completada sobe um passo.

Ex. 2: de cima para baixo:

```
10 PRINT "ABCDEFGHIJKLMN ← (1ª linha superior da tela)
OPQRSTUVXYZ –←
  (cursor aguardando o
   próximo caracter)
```

Ex. 3: de baixo para cima:

```
10 PRINT "ABCDEFGHIJKLMN ┐(1ª linha inferior da tela)
OPQRSTUVXYZ –←           ┘
  (cursor)↑
```

É claro que, no primeiro caso, quando o cursor chega na última linha inferior da tela, a partir daí, enquanto não for apagada a tela, o cursor trabalhará nessa linha, se comportando como no segundo caso.

c) No BASIC não é necessário deixar espaços entre as instruções, comandos, dados e STRINGS; o próprio programa de BASIC se encarrega de fazer a separação das diversas palavras entre si.

Ex. 4:
```
10PRINT"ABC";"DEF"
20PRINT5+3*12
```

No entanto, quando for dado o comando LIST, a listagem já aparecerá na tela com todos os espaços necessários (aqueles que separam algarismos ou números dos outros caracteres ou, em certos casos, separando os comandos ou instruções).

```
LIST
 10 PRINT"ABC";"DEF"
 20 PRINT 5 + 3 * 12
```

d) Um outro ponto interessante de notar é que, quando introduzimos um programa no computador, todas as entradas começam na casa Ø; no entanto, quando mandamos listar, os números correspondentes às linhas deixam um espaço em branco; todo número, ao ser processado pelo computador e ao ser impresso, deixa uma casa em branco na frente (para um eventual sinal negativo):

Veja o exemplo anterior:

```
LIST
 10 PRINT"ABC";"DEF"
 20 PRINT 5 + 3 * 12
[] ←(casa Ø)

OK
```

3.3 – COMANDOS

Conforme foi mencionado, os comandos do BASIC são instruções que o computador executa no denominado "mo-

do direto'', isto é, tão logo se dá entrada, através da tecla RETURN.

Os comandos podem ser executados a qualquer momento (exceto enquanto o computador está executando um programa), isto é, mesmo que haja um programa em sua memória (já executado ou ainda não).

Ao estudarmos cada um dos comandos veremos que, freqüentemente, é útil, durante a teclagem de um programa, efetuar um comando para introduzir dados, verificar certas situações de estado de memórias, etc. A execução de um comando, durante ou após a introdução de um programa no computador não afeta esse programa, podendo a introdução ou execução do mesmo ser feita normalmente, após a execução do comando (há exceções, já que certos comandos podem alterar o conteúdo das memórias, modificando, naturalmente, o programa que está sendo introduzido).

No que se segue vamos admitir que está entendido que, para acionar um comando, ou para entrar com nova linha numerada, sempre será batida a tecla RETURN (ou C.R.); por isso, não mais faremos referência à mesma, a não ser em casos bem especiais. Uma outra observação importante é que, toda a vez que for introduzido um comando e teclado o RETURN, o computador executará o comando e mostrará a mensagem OK (em geral pulando uma linha após a execução do comando). O mesmo ocorre após à execução de programas introduzidos em modo indireto (com linhas numeradas).

Os principais comandos do BASIC são:

3.3.1 — NEW (ou CLEAR ou SCR): diz ao computador que todos os programas anteriormente armazenados, em BASIC, devem ser apagados, para a introdução de um novo programa.

Ex.: para apagar os programas armazenados teclaremos

NEW

e, após, a tecla RETURN; o computador responderá com OK (ou READY), e as memórias estão prontas para receber novo programa.

3.3.2 — RUN: é o comando que faz o computador executar o programa que está armazenado; após a entrada de todo o programa, com as linhas numeradas, e após teclar o

último RETURN, entra-se com o comando RUN e, depois do RETURN, o programa será executado; em algumas versões de BASIC é possível colocar um número de linha após a palavra RUN, o que fará com que o computador execute o programa *a partir* da linha cujo número foi indicado.

Ex. 5: **RUN 3Ø**

o programa começará a ser executado a partir da linha 3Ø; esse tipo de possibilidade é bastante comum nas versões de BASIC mais encontradas no mercado.

Essa facilidade é importante durante o teste de programas, quando os mesmos estão sendo preparados.

Ex. 6: **1Ø (statement)**
2Ø (statement)
3Ø (statement)
4Ø (statement)
5Ø (statement)

Queremos testar se as linhas 3Ø, 4Ø e 5Ø estão funcionando:

Teclamos RUN 3Ø e o computador executará o programa da linha 3Ø em diante, até o fim.

3.3.3 – LIST é o comando que instrui o computador para apresentar, na tela do vídeo ou em uma impressora, a listagem do programa que está armazenado em sua memória. É importante notar que tudo o que aparecer após um número será interpretado pelo computador como sendo uma linha de instrução numerada; como já mencionado atrás, qualquer número teclado após um RETURN será interpretado como número de uma nova linha.

Uma característica importante da listagem é que as linhas aparecerão na ordem crescente de numeração, mesmo que tenham sido introduzidas fora de ordem; isto é importante para se poder verificar como está armazenado o programa na memória. É uma boa prática listar o programa, para uma verificação visual, antes de mandar executá-lo (ou seja, antes de dar o comando RUN).

Ex. 7: introduz-se o comando LIST
aperta-se a tecla RETURN

o computador listará o que se encontra na memória:

```
10 (statement)
20 (statement)
--------
190 (statement)

OK (comando ou programa concluído)
```

O comando LIST também pode ser endereçado, isto é, colocando-se o número de uma linha após a palavra LIST, o computador listará a linha que tem aquele número:

Ex. 8: **LIST 20**

o computador apresentará a linha 20

```
20 (statement)

OK
```

Também podem ser dados dois números, e o computador listará o programa que se encontra entre as linhas que têm aqueles números:

LIST 20 – 50

o computador apresentará:

```
20 (statement)
30 (statement)
--------
50 (statement)

OK
```

Se, após o primeiro número, for colocado o hífen e não for colocado o segundo número, o computador listará todo o programa ***a partir*** do primeiro número indicado:

LIST 30 –

o computador apresentará:

```
30 (statement)
40 (statement)
--------
100 (statement)

OK
```

(supondo que 1ØØ seja o número da última linha do programa que estava armazenado na memória).

A contrário, se for colocado um hifen e, depois, um número, o computador listará o programa desde a primeira linha armazenada até a linha cujo número foi indicado:

```
LIST – 6Ø
```

o computador apresentará:

```
1Ø (statement)
2Ø (statement)
6Ø (statement)

OK
```

O comando LIST, associado às indicações de número de linha, é muito útil na fase de preparação e correção de programas, pois dá a possibilidade de se verificar como está armazenada cada instrução, linha a linha ou por conjunto de linhas.

3.3.4 – PRINT é o comando que instrui o computador para apresentar, na tela ou em uma impressora, uma determinada mensagem, numérica ou alfanumérica.

Esse comando pode ser usado de diversas maneiras:

A) Impressão de STRINGS

Uso para impressão de palavras ou conjunto de palavras, selecionadas pelo programador; esse conjunto de palavras ou números deverá ser colocado entre aspas (''), para que o computador ''imprima'' exatamente tudo o que está dentro das aspas, sejam letras, algarismos, sinais, espaços, etc. (obviamente não se pode colocar aspas nesse conjunto de símbolos entre aspas; ver nota mais adiante).

Ex. 9:
```
1Ø PRINT "ESTÁ TUDO BEM"
2Ø PRINT "ISTO É UM PROGRAMA"
3Ø END

RUN
ESTÁ TUDO BEM
ISTO É UM PROGRAMA

OK
```

A instrução 1Ø manda o computador imprimir a frase ESTÁ TUDO BEM; a instrução 2Ø manda o computador imprimir ISTO É UM PROGRAMA; a instrução 3Ø indica o fim do programa; as letras OK indicam que o programa foi executado e o computador aguarda novas instruções; nota-se que as frases foram impressas, ou colocadas no vídeo, sem aspas, pois a instrução manda imprimir o que está ***entre*** as aspas.

Denomina-se STRING o conjunto de letras, números, símbolos, espaços, etc. que aparecem entre aspas e que são tratados pelo computador como um bloco sem nenhum significado especial; mesmo que se coloque dentro do STRING um comando ou instrução, o computador não interpretará esse comando, pois considera o conjunto entre aspas como uma "caixa preta", a ser tratada apenas como um conjunto de caracteres.

Ex.10:
```
1Ø PRINT "LIST 2Ø"
2Ø PRINT "NEW"
3Ø END

RUN
LIST 2Ø
NEW      ← isto é o que aparece na tela

OK
```

É muito comum encontrar-se versões de BASIC nas quais a palavra PRINT pode ser substituída pelo caracter ? (interrogação); assim, em vez de se escrever PRINT, usa-se o sinal ?, o que diminui o esforço datilográfico, entre outras vantagens.

Ex.11:
```
1Ø ? "COMO VAI ?"

RUN
COMO VAI?

OK
```

Note-se que o sinal de interrogação que está dentro do STRING não é interpretado pelo computador como instrução de PRINT.

Quando for pedida uma listagem do programa armazenado, a interrogação será substituída pela palavra PRINT (que é o significado da interrogação, para o BASIC).

Ex.12:
```
10 ? "HOJE É DIA"
20 ? "15 DE JULHO"
30 END
RUN
HOJE É DIA
15 DE JULHO

OK

LIST
 10 PRINT "HOJE É DIA"
 20 PRINT "15 DE JULHO"
 30 END

OK
```

Nota-se, nos exemplos acima, que cada instrução PRINT (ou ?) faz com que o STRING correspondente seja impresso em uma linha diferente na tela ou na impressora. Existem, no entanto, maneiras de fazer com que a frase toda seja impressa na mesma linha; a maneira óbvia é colocar tudo dentro do mesmo STRING.

```
10 PRINT "HOJE É DIA 15 DE JULHO"
```

Porém, existem casos em que não é possível fazer-se isso; por exemplo, quando a frase e muito longa (mais de 256 caracteres na linha) ou quando os pedaços da frase precisam, por questões de programa, aparecer em linhas numeradas diferentes.

Nesse caso pode ser usado o seguinte artifício:

Após cada instrução PRINT e depois do STRING, após as aspas, é colocado o sinal de ponto-e-vírgula (;), o qual é interpretado como uma instrução para que o próximo

STRING a ser impresso seja colocado logo em seguida ao primeiro.

```
NEW  (apagamos os programas anteriores)

OK   (o computador está pronto)
```

Ex. 13:
```
10 PRINT  "HOJE É DIA"; (note o ponto-e-vírgula)
20 ? " 15 DE JULHO" ←  (não precisa de ponto-e-vírgula se for o fim da frase)
30 END

RUN
HOJE É DIA 15 DE JULHO

OK
```

Ex. 14:
```
10 ? "HOJE ";
10 ? "É DIA ";   (note o ponto-e-vírgula)
30 ? "15 DE ";
40 ? "JULHO"     (sem ponto-e-vírgula)
50 END

RUN
HOJE É DIA 15 DE JULHO

OK
```

É importante notar que não se deve colocar ponto-e-vírgula após a última parte da frase que deve ficar na mesma linha; se for colocado o ponto-e-vírgula, a próxima instrução de PRINT será executada na mesma linha da frase anterior, o que, provavelmente, não é desejado.

Ex. 15:
```
10 ? "HOJE É ";
20 ? "DIA 15 DE JULHO"
30 ? "AMANHÃ SERÁ ";
40 ? "16 DE JULHO"
50 END

RUN
```

```
HOJE É DIA 15 DE JULHO
AMANHÃ SERÁ 16 DE JULHO

OK
```

Observe que as linhas 2Ø e 4Ø não terminam em ponto-e-vírgula; dessa forma a segunda frase ficou em linha separada e uma nova frase ficará em outra linha na tela.

Uma outra observação importante se relaciona com os espaços entre palavras: todos os espaços necessários devem ser colocados dentro do STRING.

Vejamos os seguintes exemplos:

Ex.16: **NEW**

```
OK
10 PRINT "COMO VAI"
20 PRINT "VOCÊ ?"
30 END
RUN
COMO VAIVOCÊ?

OK
```

(verifique que as aspas depois do VAI e antes do VOCÊ estão colocadas sem espaço; assim, o segundo STRING será emendado no primeiro).

Ex. 17: **NEW**

```
OK
10 ? "COMO VAI "; (espaço depois do VAI)
20 ? "VOCÊ ?"     (sem espaços)
30 END
RUN
COMO VAI VOCÊ?

OK
```

(o espaço colocado dentro do primeiro STRING aparece na frase após a palavra VAI).

Obviamente, o mesmo efeito seria conseguido se fosse colocado um espaço no segundo STRING, antes da palavra VOCÊ.

Uma outra característica importante relacionada com a instrução PRINT é o fato de se poder colocar mais de um STRING na mesma linha de programa; nesse caso os STRINGS diferentes devem ser separados por ponto-e-vírgula.

Ex. 18:
```
10 PRINT "HOJE É"; " DIA 15 DE JULHO"
20 END

RUN
HOJE É DIA 15 DE JULHO

OK
```

Mais adiante veremos qual a utilidade de se colocar mais de um STRING na mesma linha, embora, em princípio, se pudesse colocar tudo dentro das mesmas aspas.

Até agora vimos que o uso de ponto-e-vírgula permite justapor expressões colocadas em STRINGS diferentes. É possível, no entranto, fazer a mesma coisa com vírgula ao invés de ponto-e-vírgula. Apenas, e isto é muito importante, a vírgula tem um tipo de operação diferente:

— já vimos que uma linha de programa tem espaço para N caracteres; a vírgula separa esses N caracteres em zonas, ou seja, são marcadas posições bem definidas na linha, nas quais começará a ser impresso o próximo STRING indicando após a vírgula; no caso de um computador cuja zona tenha 14 caracteres as posições correspondentes aos caracteres nº Ø, 14, 28, 42, 56 e 7Ø são fixas, marcando o lugar onde começa a impressão do próximo STRING (considerando um vídeo com 24 caracteres por linha).

Ex. 19:
```
10 PRINT "A", "B", "CD"
```
(vírgula)
(sem vírgula)
```
20 END
RUN
A               B
    CD
OK
```

A está na posição Ø, B na posição 14 e CD começa na posição 28 (que é na outra linha de tela).

Ex. 20:
```
10 ? "AAAAAAAAAAAAAAA",     (15 A's)
                    ↑
          (vírgula)─┘
20 ? "BB"
30 END
RUN
```
(uma linha de vídeo com 24 posições)

```
AAAAAAAAAAAAAAA
↑              ↑
(pos. Ø)   (pos. 14)
    BB         (próxima linha de vídeo)
    ↑
    (pos. 28)

OK
```

No exemplo 20 verificamos duas coisas: a vírgula está no fim da linha de número 1Ø; isso funciona da mesma forma como se ela estivesse entre os STRINGS (como no exemplo 19); a posição 14 não foi usada pelo segundo STRING, pois o primeiro tinha mais de 14 caracteres; assim as letras começaram a ser impressas na próxima posição disponível (posição 28).

Nota: o tamanho das zonas varia conforme o computador usado; normalmente uma zona não tem mais de 16 caracteres.

A instrução PRINT, dentro de um programa, pode ser usada para pular linhas (isto é, imprimir linhas em branco); para isso basta colocar PRINT, sem nenhum STRING após essa instrução.

Ex. 21:
```
10 PRINT "AA"
20 PRINT
30 PRINT
40 PRINT "BB"
50 END
RUN
```

```
AA
   ] (duas linhas em branco)
BB

OK
```

Ex. 22:
```
10 PRINT
20 ?
30 PRINT
40 PRINT "A"
50 PRINT
60 PRINT "C"
70 ?
80 ?
90 PRINT
100 END
RUN
   ] (três linhas em branco, conforme instruções
      10, 20 e 30)
A     (instrução da linha 40)
   ]  (instrução da linha 50)
C     (instrução da linha 60)
   ]  (instruções das linhas 70, 80, 90)
   ]  (espaço normalmente existente)

OK
```

Vamos aproveitar o programa do exemplo 22 para fazer algumas manipulações que permitirão a assimilação dos conceitos já expostos (inclusive outros comandos):

a) Após a execução do programa do exemplo 22 acima, *sem teclar* NEW, vamos LISTar o programa, para ver como o mesmo se encontra na memória do computador:

```
LIST
 10 PRINT
 20 PRINT
 30 PRINT
 40 PRINT "A"
 50 PRINT
 60 PRINT "C"
 70 PRINT
 80 PRINT
 90 PRINT
 100 END

OK  (indica que a instrução, ou seja, o comando
       LIST, foi executado)
```

Nota-se que os sinais ? foram substituídos, na listagem, pelas palavras PRINT.

b) Vamos diminuir alguns espaços, refazendo algumas linhas:

tecle — **20** (não coloque nenhuma instrução)
(aperte RETURN)
tecle — **80** (nenhuma instrução)
(aperte RETURN)
tecle **LIST** (para ver como ficou o programa)
(aperte RETURN)

```
 10 PRINT
 30 PRINT
 40 PRINT "A"
 50 PRINT
 60 PRINT "C"
 70 PRINT
 90 PRINT
 100 END

OK
```

as linhas 20 e 80 desapareceram do programa; agora vamos ver o que acontece, na execução:

```
RUN

          (duas linhas em branco)
A
          (linha em branco)
C
          (duas linhas em branco)

          (linha em branco normal)
OK
```

c) Agora, vamos acrescentar a letra B, que havíamos, quem sabe, esquecido de programar:

tecle — **50 ? "B"**

Verifique:

```
LIST
 10 PRINT
 30 PRINT
 40 PRINT "A"
 50 PRINT "B"
 60 PRINT "C"
 70 PRINT
 90 PRINT
 100 END

OK
```

A nova linha 50 substitui a anterior:
Agora execute:

```
RUN


A
B
C


OK
```

d) Se, por acaso, acharmos que as letras estão muito juntas, podemos separá-las:

tecle — **45 PRINT**
55 PRINT

vejamos como ficou:

```
LIST
 10 PRINT
 30 PRINT
 40 PRINT "A"
 45 PRINT
 50 PRINT "B"
 55 PRINT
 60 PRINT "C"
 70 PRINT
 90 PRINT
 100 END

OK
```

as novas linhas (45 e 55) foram devidamente encaixadas no lugar certo.

Execução:

```
RUN


A

B

C


OK
```

Um outro exemplo interessante de aplicação das vírgulas e ponto-e-vírgulas em conjunto com a instrução PRINT é

a colocação desses sinais após a instrução, sem ser seguida de argumentos (variáveis, strings e números), na forma:

Ex. 23:
```
10 PRINT; (note o ponto-e-vírgula)
20 PRINT "AA"
30 END
   ou
10 PRINT, (note a vírgula)
20 PRINT "AA"
30 END
```

Vejamos o que ocorre no primeiro caso; como já vimos, o ponto-e-vírgula faz com que o próximo comando PRINT comece a impressão logo após o que tem ponto-e-vírgula, na mesma linha. Ora, o comando da linha 10 não mandou imprimir nada, mas mandou continuar na mesma linha; assim, a impressão de AA será feita como se a linha 10 não existisse;

```
RUN
AA

OK
```

No outro caso, a vírgula manda começar a próxima impressão na segunda posição (casa 14); desse modo teremos:

```
RUN
          AA  (o primeiro A foi impresso na casa 14)

OK
```

Se colocarmos duas vírgulas, a impressão começará na posição 28, e assim por diante.

Nota: mencionamos, no início, que não é possível colocar aspas (") dentro de STRING, pois esses STRINGS começam e terminam por aspas.

Em alguns tipos de BASIC a colocação das aspas dentro do STRING pode ser feita usando duplas aspas (""), de modo que esse sinal seja interpretado como ***aspas dentro de STRING***:

Ex. 24:
```
10 ? "EU ME CHAMO""ROBERTO"""
20 END
RUN
EU ME CHAMO "ROBERTO"

OK
```

Ex. 25:
```
10 ? """DIGA SEU NOME"""
20 END
RUN
"DIGA SEU NOME"

OK
```

Veja que, onde aparecem as aspas principais (início e fim do STRING), somos obrigados a colocar três aspas, sendo uma a do STRING e as outras duas significando ***aspas dentro do STRING***.

Nos casos em que não é possível fazer o artifício das duplas aspas, será necessário usar-se outro método para imprimir STRINGS que contenham aspas:

Ex. 26:
```
10 ? "MEU NOME É " CHR$(34); "ROBERTO";
CHR$ (34)

20 END
RUN
MEU NOME É "ROBERTO"

OK
```

A função CHR$(34), que será explicada mais adiante, instrui o computador para imprimir o sinal de aspas (cujo código é 34).

B) Impressão de números

Até aqui falamos apenas sobre a impressão de STRINGS; no entanto, a instrução PRINT pode ser usada para "imprimir" valores numéricos, tanto no modo direto (comando) como no indireto (statement em linha numerada); nesse caso

não se usam aspas, pois o número a ser impresso tem um significado bem definido, ao contrário do que aparece dentro de um STRING (é claro que pode haver números dentro do STRING, mas, nesse caso, eles não são tratados como números e, sim, como meros símbolos):

Ex. 27:
```
PRINT 5        (sem aspas)
<RETURN>       (usaremos esse símbolo para indicar que a tecla RETURN deve ser pressionada)
 5             (aparece na tela ou na impressora)

OK
```

Ex. 28:
```
10 ? 18    (sem aspas)
20 END
RUN
 18

OK
```

C) Impressão de variáveis

Denomina-se variável uma letra, ou conjunto de letras que, no decorrer do programa, poderá(ão) ter um valor numérico a ele alocado ou que poderá representar um STRING. Na maioria dos casos encontrados na prática do BASIC para microcomputadores, as variáveis só podem ser constituídas de duas letras ou uma letra e um número (sendo o primeiro caracter sempre a letra).

Exemplos de variáveis: A, AB, X, TX, A1, B3, etc.

No início do programa as variáveis têm valor zero (a menos que tenha ficado alguma indicação na memória do computador).

Quando mandamos imprimir uma variável, será impresso o seu ***valor numérico*** ou o STRING que foi associado a essa variável.

Mais adiante veremos diversas maneiras de associar valores ou STRINGS a variáveis. Por enquanto, vamos apenas usar o método mais imediato, que utiliza o comando LET.

O comando LET (ou instrução dentro de uma linha) permite associar diretamente um valor a uma variável:

Ex. 29: **LET A = 2**
(a variável passou a ter o valor 2)

Vejamos o seguinte programa:

Ex. 30: **NEW**

```
OK
10 PRINT A
20 PRINT
30 LET A=2
40 PRINT A
50 END
RUN
 0    (no início do programa o valor de A é
       zero)
 2    (novo valor de A)

OK
```

É importante notar que toda a vez que um novo valor for associado à mesma variável, prevalecerá o último, isto é a variável vai mudando de valor, de acordo com as instruções recebidas.

Ex. 31: **NEW**

```
OK
10 PRINT A1
20 PRINT A2
30 LET A1=2
40 LET A2=4
50 LET A1=3
```

```
60 PRINT A1; A2 (veja o uso do ponto-e-vírgula)
70 END
RUN
 0  0   (1º valor de A1 e A2)
 3  4   (último valor de A1 e último valor de
         A2)
OK
```

Um detalhe deve ser mencionado, no que respeita à impressão de números: todos os números são armazenados na memória do computador com um espaço vago na frente, se forem positivos; se forem negativos esse espaço é preenchido pelo sinal (-).

Ex. 32:
```
10 PRINT 15
20 PRINT -4
30 END
RUN
 15  (veja o espaço na frente)
-4

OK
```

Além disso, os programas BASIC, em geral, deixam vago um espaço após cada número, quando se usa separação por ponto-e-vírgula.

Ex. 33:
```
10 PRINT 15;-4; 2
20 END
RUN
 15  -4  22

OK
```

Isso não acontece com STRINGS, que não têm nenhum valor numérico.

Ex. 34:
```
PRINT "AA"; "BB"
      <RETURN>
AABB  (sem espaços)

OK
```

Quando os números são separados por vírgula, tudo se passa como já foi mostrado no caso dos STRINGS (acrescentando-se os espaços antes e após os números):

Ex. 35:
```
PRINT 5, 18
<RETURN>
 5            18

OK
```

O número dezoito foi impresso na 15ª casa, pois a casa 14 contém o espaço para o seu sinal.

Quando queremos associar um STRING a uma variável, é necessário colocar o sinal $ (cifrão ou dólar) após o nome da variável e colocar o STRING entre aspas (para mostrar que é um STRING).

Ex. 36:
```
10 LET A$ ="HOJE"
20 LET B1$ = " É DIA "
30 LET CC$ = "15 DE JULHO"
40 ? A$; B1$; CC$
50 END
RUN
HOJE É DIA 15 DE JULHO
OK
```

Há limites para o tamanho dos números e dos STRINGS que podem ser alocados a variáveis; esse tamanho depende da estrutura do computador e da versão do BASIC usado.

Daremos alguns exemplos, relativos ao microcomputador usado pelo autor.

Quando o computador é chamado a processar um programa cujo resultado é maior do que 6 algarismos, esse resultado será mostrado em notação científica, isto é, uma mantissa e um fator multiplicador em base de potência de 10 (outras máquinas podem dar resultados com maior número de algarismos; no entanto, os exemplos abaixo são elucidativos do processo):

Ex. 37: **NEW**

```
OK
10 LET A= 37256
20 PRINT A
30 END
RUN
 37256  (o número só tem cinco algarismos)

OK
```

Ex. 38: **10 LET A = 3725687** (7 algarismos)

```
20 ?A
30 END
RUN
 3.72569E+6

OK
```

O computador recebe o número, na hora da programação, com mais de 6 algarismos; porém, na hora de processar (isto é, nesse caso visto acima, imprimir) o resultado, este será limitado a 6 algarismos; a letra E indica "vezes 10 elevado a" e o número que se segue é o expoente (o sinal + indica que o expoente é positivo).

3.72569E+6 equivale a
3.72569×10^6 ou seja
3725690 (comparando com o número original — 3725687 — verifica-se que houve um arredondamento para cima)

Ex. 39: **10 LET C= 578432439** (nove algarismos)

```
20 ? C
30 END
RUN
 5.78432E+8

OK
```

(5.78432E+8 = 5.78432×10^8 = 578432000, ou seja, houve uma aproximação para baixo, em

virtude de os próximos algarismos serem menores que 5XX)

Ex. 40:
```
10 LET B=-387532839
20 PRINT B
30 END
RUN
-3.87533E+8  (veja a posição do sinal)

OK
```

Ex. 41:
```
10 LET X= 0.03284783
20 ? X
30 END
RUN
 3.28478E -2

OK
```

No caso de números fracionários menores que 1, o expoente de 10 será negativo:

3.28478E -2 = 0,0328478

Ex. 42:
```
10 LET A= -0.0003478257
20 ? A
30 END
RUN
-3.47825E - 4

OK
```

Nota sobre a escrita de números fracionários. É importante lembrar que, em inglês, a nossa vírgula corresponde a um ponto:

3,4832 (em português) equivale a 3.4832 (em inglês).

Além disso, os números fracionários menores que 1, que nós escrevemos Ø,XXX, não têm o zero inicial, para os americanos:

Ø,3456 (em português) equivale a .3456 (em inglês).

É importante notar esses aspectos, pois os computadores que, na maioria dos casos, possuem linguagens produzi-

das nos Estados Unidos, só entendem a notação americana, que é a que usaremos daqui para a frente, neste texto.

Quando se trata de STRINGS, a possibilidade é bem maior; a rigor, o tamanho de um STRING que pode ser alocado a uma variavel é limitado, em geral, pelo tamanho da linha, podendo atingir 256 caracteres; devem ser descontados desses totais os espaços e a palavra LET, assim como a variável, o cifrão, o sinal de igual e as aspas.

Ex. 43: (no caso de limite de 72 caracteres)

```
10 LET AB$ ="ABCDE .....XYZ..."
20 ? AB$
30 END
RUN
ABCDE ..... XYZ .....
(61 caracteres, no máximo)

OK
```

Foram descontados 2 espaços, 3 letras (LET), a variável (AB), o cifrão, o sinal de igual e as duas aspas, no total de 11 caracteres.

Pode-se mandar imprimir várias coisas com o mesmo comando.

Para isso, como já vimos, é necessário que os STRINGS fiquem entre aspas e que as diversas coisas a serem impressas (números, variáveis, STRINGS) sejam separadas por ponto-e-vírgula ou vírgula; usa-se mais normalmente o ponto-e-vírgula, que não deixa espaços entre os diversos itens a serem impressos.

Ex. 44:

```
10 PRINT "EU TENHO";N;"ANOS"
20 END
RUN
EU TENHO 0 ANOS

OK
```

O número zero apareceu porque a variável N não tinha nenhum valor a ela atribuído.

Ex. 45:
```
10 PRINT "EU TENHO";15;"ANOS"
20 M= 15 + 2
30 PRINT "E VOCÊ TEM";M;"ANOS"
40 END
RUN
EU TENHO 15 ANOS
E VOCÊ TEM 17 ANOS

OK
```

3.3.5 – LET é o comando que associa um valor a uma variável: já vimos diversos exemplos mais acima.

Cabe acrescentar que, em grande parte das versões de BASIC usadas hoje em dia, esse comando é opcional, isto é, não é necessário usá-lo, para associar um valor a uma variável.

LET A=3 pode ser simplesmente escrito A=3 e o computador entenderá que estamos associando o valor 3 à variável A.

Ex. 46:
```
10 A =18
20 ? A
30 END
RUN
 18

OK
```

Ex. 47:
```
10 A$ = "COMPUTADOR"
20 ? A$
30 END
RUN
COMPUTADOR

OK
```

Todos os exemplos vistos na descrição do comando PRINT, com o uso de LET, funcionam igualmente sem o LET.

3.3.6 – CLEAR, quando não tem o significado NEW, é usado para "zerar" todas as variáveis, isto é, apagar todos os valores que foram atribuídos anteriormente às variáveis.

Ex.48:
```
10 A=15
20 B=18
30 ? A;B
40 END
RUN
 15    18

OK
```

Esse programa continua armazenado no computador, de modo que podemos mandar executá-lo novamente:

```
RUN
 15    18

OK
```

Agora, antes de mandar executar novamente o programa, vamos introduzir um alteração:

Tecle: **25 CLEAR**
E, agora, execute:
```
RUN
 0  0

OK
```

Note que, apesar de termos dado os valores 15 e 18 para as variáveis A e B, a instrução CLEAR apagou esses valores e, assim, o comando PRINT A;B só conseguiu imprimir os zeros.

Às vezes é importante usar os valores atribuídos a uma variável em uma parte do programa e, depois, em outra seção do programa, apagar esses valores (para recomeçar uma contagem, por exemplo):

Ex. 49:
```
10 A=1
20 ? A
30 A=A+1
40 ? A
50 A=A+1
60 ? A
70 CLEAR
80 A=A+1
90 ? A
100 A=A+1
110? A
120 END
RUN
 1
 2
 3
 1
 2

OK
```

Analisemos o que aconteceu: no início do programa declaramos que A = 1; a próxima instrução (linha 2Ø) mandou imprimir esse valor de A.

Na linha 3Ø declaramos que A = A + 1, ou seja, o valor anterior de A (1) foi acrescido de 1, passando para 2; assim, o novo valor de A, após a execução da linha 3Ø, é 2; a linha 4Ø manda imprimir o valor de A, de modo que será impresso o número 2; a mesma mecânica ocorre com as linhas 5Ø e 6Ø.

Na linha 7Ø damos o comando CLEAR, que "apaga" o valor anterior de A; assim, nesse momento, A = Ø; na linha 8Ø mandamos somar 1 ao valor de A, de modo que o novo valor passa a ser 1 (Ø + 1); esse valor será impresso pelo comando da linha 9Ø; as linhas 1ØØ e 11Ø se comportam do modo já visto.

3.3.7 — CONT é um comando que manda o programa continuar a ser executado, após uma interrupção causada por CTRL C (Control C), pela instrução STOP, ou, em alguns casos, pela tecla BREAK.

Como já vimos anteriormente, durante a execução de um programa, podemos interrompê-lo apertando simultaneamente as teclas CTRL e C, o que provocará o aparecimento da mensagem:

BREAK IN XX

sendo XX o número da próxima linha que seria executada, se não houvesse a interrupção.

Se quisermos continuar a execução do programa, a partir do ponto em que foi interrompido, basta teclar CONT e bater RETURN; o programa prosseguirá normalmente.

Essa interrupção também pode ser programada, colocando-se, em um certo ponto do programa a instrução STOP; quando o programa chegar àquele ponto será automaticamente interrompido, aparecendo a mensagem relativa ao número da linha do STOP.

Também nesse caso o programa pode ser continuado pela teclagem de CONT e RETURN.

A técnica de uso do STOP e CONT é muito útil para se descobrir erros no programa:

Ex. 50: **NEW**

```
OK
10 A=1
20 B=A+2
30 PRINT B
40 A=A+1
50 PRINT B
60 A=A+1
70 PRINT B
80 END
```

Nesse programa queremos que seja impresso o valor de B, que é igual ao valor de A+2 e, após aumentarmos o valor de A de uma unidade, queremos imprimir o novo valor de B (essa operação será repetida nas linhas 60 e 70).

Vejamos o que acontece:

```
RUN
 3  (primeiro valor de B)
 3  (segundo valor de B)
 3  (terceiro valor de B)

OK
```

Parece que alguma coisa está errada no programa, pois esperávamos ter os valores 3, 4, 5, impressos, respectivamente, como primeiro, segundo e terceiro valores de B.

Será que a variável A não foi incrementada, na linha 4Ø?

Façamos o seguinte: introduzamos duas novas linhas no programa:

```
45 PRINT A
46 STOP
```

Agora, executamos o programa:

```
RUN
  3    (primeiro valor de B)
  2    (valor de A)

BREAK IN 46
```

Aparentemente tudo está correto; o valor de A, que era 1, passou para 2.

Vamos continuar o programa:

```
CONT
<RETURN>
  3    (segundo valor de B)
  3    (terceiro valor de B)

OK
```

Continua errado! Mas já sabemos que o erro não é no acréscimo de A. Se, após a execução da linha 4Ø, o valor de A passou a ser 2 e, depois (linha 5Ø), o valor de B continuou sendo 3, isto significa que não foi feita a soma de A + 2 (= B).

Vamos tentar o seguinte: acrescentar:

```
46 B=A+2
(a nova linha 46 apagou a instrução STOP)
55 STOP
```

Vejamos, agora, como está este programa:

```
LIST
 10 A=1
 20 B=A+2
 30 PRINT B
 40 A=A+1
 45 PRINT A
 46 B=A+2
 50 PRINT B
 55 STOP
 60 A=A+1
 70 PRINT B
 80 END

OK
```

E, agora, vamos rodar o programa:

```
RUN
  3    (primeiro valor de B)
  2    (valor de A)
  4    (segundo valor de B)
BREAK IN 55
```

Aí está. A conta B=A+2 só estava sendo executada uma vez (linha 20) e, assim, a variável B não mudava mais de valor.

Vamos continuar:

```
CONT
  4    (terceiro valor de B)

OK
```

Chegamos à conclusão de que estaria faltando uma nova linha, após a linha 60 e antes da 70, para mandar fazer novamente a conta B=A+2.

Mais adiante veremos outras maneiras de se fazer a mesma coisa. Neste momento quisemos apenas ilustrar o uso de STOP e CONT, como meio de corrigir programas.

3.3.8 – LOAD é um comando que instrui o computador para armazenar em sua memória o conteúdo de um meio externo, tal como fita cassete ou disco magnético.

Dependendo do tipo de BASIC usado e do tipo de meio externo, o comando LOAD poderá ser seguido de um rótulo, que identificará o programa a ser lido.

A operação, em geral, funciona da seguinte maneira:

Suponhamos um programa que esteja gravado em fita cassete; vamos passar o conteúdo da fita (ou de parte da fita) para a memória do computador:

```
NEW      (apagamos os programas anteriormente
          armazenados)
OK       (o computador está pronto)
LOAD     (antes de teclar RETURN, ligamos o gra-
          vador, em modo de reprodução)
<RETURN>
 10 (statement)  (as linhas vão sendo armazenadas
 20 (statement)  no computador tal como se esti-
                 vessem sendo tecladas
 100 END
(nesse momento desligamos o gravador)
```

Agora, o programa que foi lido da fita se encontra gravado na memória e está pronto para execução.

Não devemos esquecer de desligar o gravador após a armazenagem do programa, pois, se não, o computador continuará lendo e armazenando o que estiver no resto da fita.

Uma outra observação importante é que, em algumas versões de BASIC (não todas), o comando LOAD pode ser colocado como instrução dentro de um programa; ao chegar nessa linha, o computador começará a armazenar o que estiver sendo lido de uma fita ou disco (se os mesmos estiverem ligados).

Ex. 51:
```
10 A=2
20 PRINT A
30 LOAD
40 END
RUN
 2    (impressão do valor de A)
```

```
1Ø   (statement)  ┐ programa lido da fita
2Ø   (statement)  │
        .         │
        .         │
        .         │
        .         ┘
```

(ao ser desligado o gravador, o programa será terminado)

OK

As linhas lidas da fita passam a estar armazenadas na memória do computador e substituirão as de mesmo número que estavam armazenadas anteriormente.

3.3.9 — SAVE é um comando que instrui o computador para efetuar a gravação dos programas contidos em sua memória, em um meio externo, tal como fita ou disco magnético; esse comando permite armazenar nesses suportes externos o conteúdo dos dados ou programas elaborados ou processados no computador.

De maneira geral, o comando SAVE permite gravar as listagens de programas tais como as mesmas se encontram nas memórias; para isso é necessário teclar o comando LIST após o SAVE; com isso o computador começará a listar as linhas, uma a uma (caracter a caracter) em velocidade compatível com o meio de gravação; essa velocidade é sensivelmente mais lenta do que a que é usada quando se manda listar o programa no vídeo (sem gravação).

Ex. 52: Programa armazenado no computador:

```
1Ø A = 15
2Ø B = 12
3Ø PRINT A
4Ø PRINT B
5Ø END
```

Para gravar esse programa (p. ex., em fita cassete), devemos seguir as seguintes etapas:

a) teclar SAVE
b) teclar <RETURN>

OK (resposta do computador)

c) teclar LIST
d) ligar o gravador, em modo de gravação
e) teclar <RETURN>

as linhas vão sendo listadas lentamente e, ao mesmo tempo, gravadas na fita

```
10 A=15
20 B=12
30 PRINT A
40 PRINT B
50 END

OK  (fim de listagem)
```

f) desligar o gravador
g) teclar LOAD (para desligar o comando SAVE)
h) teclar <RETURN>

OK (está tudo pronto)

Agora, o programa que vai das linhas 10 a 50 está gravado na fita e pode ser reproduzido mais tarde (isto é, recolocado na memória do computador).

Em diversos tipos de BASIC o comando SAVE permite a gravação de tudo o que o computador vier a executar em seguida (em velocidade mais lenta); tudo o que aparecer no vídeo, após o comando que for dado depois do SAVE, será gravado da mesma forma que for mostrado na tela.

Ex. 53: Suponhamos o mesmo programa mostrado no exemplo 52:

a) **SAVE**
b) <RETURN>

OK

c) **RUN** (mandamos executar o programa que estava na memória)
d) (gravador ligado)
e) <RETURN>

```
 15
 12
OK
```

esses números aparecem mais lentamente na tela e, ao mesmo tempo, são gravados.
(fim de execução)

f) desligar o gravador

g) **LOAD**
h) <RETURN>

OK

Agora estão gravados na fita, em seqüência, os números 15 e 12. Isso é muito interessante, porque, às vezes, necessitamos gravar em fita apenas uma seqüência de dados, sem que os mesmos estejam embutidos em uma listagem de programa:

Ex. 54: Programa armazenado

```
10 PRINT "MEU NOME É";
20 PRINT "ROBERTO"
30 PRINT "E O SEU";
40 PRINT "É ANDRÉ "
50 END
```

vamos gravá-lo:

```
SAVE        (<RETURN>)

OK
RUN         (<RETURN>)
MEU NOME É ROBERTO  ┐ essa parte está sen-
E O SEU É ANDRÉ     ┘ do gravada na fita

OK

LOAD        (desligar o SAVE)

OK
```

Agora, para mostrar o que aconteceu, vamos passar o que foi gravado da fita para o computador:

```
NEW   (<RETURN>)

OK
LOAD  (<RETURN>)

MEU NOME É ROBERTO

E O SEU É ANDRÉ
```

```
OK  (esse OK estava na fita)
(desliga-se o gravador)
```

Até aqui tudo bem. Acontece que o que se passa, na realidade, não é exatamente o que foi mostrado acima.

Já mencionamos que, ao ler frases da fita, muitos tipos de BASIC interpretam e executam essas frases imediatamente, a menos que as mesmas estejam precedidas de um número de linha (como se fossem comandos teclados em modo direito).

No exemplo mostrado acima, não há números de linha antes dos STRINGS que foram gravados na fita; assim, o computador, ao ler o que está na fita, acusará erro de sintaxe (isto é, comando ininteligível para o Interpretador BASIC).

Vejamos o que realmente aconteceria:

```
LOAD
MEU NOME É ROBERTO
SN ERROR          (o computador acusa erro de
                   sintaxe e diz que está pronto
OK                 para a próxima frase)
E O SEU É ANDRÉ
SN ERROR

OK
(desligar)
```

Nesse momento, se quisermos listar o que está na memória, não encontraremos nada, pois as frases que foram gravadas não correspondem a comandos diretos nem a instruções numeradas.

```
LIST          (<RETURN>)

OK            (nada a listar)
```

Se, no entanto, quisermos gravar um conjunto de dados, o computador os aceitará e guardará corretamente na memória, pois, como já vimos, LET é um comando direto.

Ex. 55: programa existente

```
10 PRINT "A = 10"     (note as aspas)
20 PRINT "B = 18"
30 PRINT "C = 8"
40 PRINT "D = 11"
50 END
```

Vamos gravá-lo:

```
SAVE

OK
RUN
A = 10   ┐
B = 18   ┆ foi gravado na fita
C = 8    ┆
D = 11   ┆
         ┆
OK       ┘
(desligar o gravador)

LOAD     (desfazer o SAVE)

OK
```

Agora, se reproduzirmos essa fita, os valores de A, B, C e D serão lidos e aceitos pelo computador.

3.3.10 — COPY é um comando que permite transcrever para uma impressora exatamente o que está na tela do vídeo, naquele momento. É natural que, se houver movimento, isto é, símbolos e caracteres mudando de lugar, é necessário, primeiro, parar o processamento do programa, por exemplo, através do comando CTRL C.

Após a tela estar "congelada", ela pode ser copiada em uma impressora, fazendo o que se costuma chamar "hard copy" do vídeo.

É claro, também, que as características da impressora devem ser compatíveis com as da tela, caso contrário a impressão não reproduzirá fielmente o que está mostrado no vídeo. Como exemplo de diferenças podemos citar o número

de caracteres por linha (que deve ser compatível, caso contrário as linhas não ficarão com a mesma disposição que apareceu na tela), o tipo de caracteres (há caracteres, especialmente gráficos, no computador, que não existem nas impressoras, o que significa que será impresso um símbolo diferente do que se encontra na tela).

3.3.11 — Observação sobre a instrução END: até agora mostramos todos os programas terminados com a instrução END; no entanto, na maioria das versões de BASIC atualmente usadas esta instrução não é necessária.

O programa interpretador ou compilador reconhece quando não há nenhuma linha de instrução e considera o programa terminado.

Ex. 56:
```
10 PRINT "BONITO DIA"
20 PRINT "MAS ESTÁ FICANDO ";
30 PRINT "NUBLADO"
RUN
BONITO DIA
MAS ESTÁ FICANDO NUBLADO

OK
```

O computador sabe que não há nenhuma instrução com número acima de 30 e, portanto, terminou o programa.

É preciso tomar cuidado porque, se em um programa anterior sobrou alguma linha numerada que não foi apagada, o programa prosseguirá até a linha de maior número. Para evitar esse tipo de surpresa é conveniente mandar listar os programas antes de executá-los; assim, todas as linhas numeradas que estiverem armazenadas no computador aparecerão na lista.

Ex. 57: Suponhamos que havia no computador um programa que começava no número 100:
```
100 PRINT "A SUA IDADE É";15;
110 PRINT "ANOS"
120 PRINT "E A MINHA É";
130 PRINT 17;"ANOS"
```

Se estivermos preparando um programa começando no número 1Ø e esse programa for até o número 5Ø, normalmente não será necessário colocar a instrução 6Ø END.

Acontece que, ao mandarmos executar o programa ele começará na linha 1Ø e prosseguirá até a linha 13Ø, executando o programa novo e o anterior; se quisermos evitar que isso aconteça, podemos usar as seguintes alternativas:

a) apagar o programa anterior a partir da linha 1ØØ; assim, ao ser executado, o programa só irá até a linha 5Ø;

b) colocar a instrução 6Ø END; nesse caso, o programa começará da linha 1Ø e parará na linha 6Ø, ao ler a instrução END; veremos mais adiante que a colocação de END no meio de um programa é, muitas vezes, um artifício bastante útil de programação.

Se usarmos a alternativa *b*, o programa anterior não será apagado e poderemos executá-lo, bastando dar o comando RUN 1ØØ; assim, o programa começará a ser executado a partir da linha 1ØØ.

Com a colocação do END e do RUN 1ØØ nós temos a possibilidade de executar independentemente dois programas diferentes, armazenados na memória do computador.

Capítulo IV
INSTRUÇÕES MÚLTIPLAS POR LINHA

Até agora tudo o que foi mostrado apresentou apenas uma instrução por linha numerada.

Ex. 1:
```
10 A = 5
20 B = 3
30 C = A + B
40 PRINT C
```

Na maioria das versões de BASIC usadas nos microcomputadores (exceto os bem reduzidos), é possível colocar mais de uma instrução por linha numerada. Essas instruções devem ser separadas por dois pontos (:) ou barra invertida (\). Nos exemplos seguintes usaremos dois pontos, cabendo ao leitor verificar como isso funciona no seu computador.

O programa acima pode ser reescrito da seguinte maneira:

```
10 A = 5 : B = 3 : C= A + B : ? C
```

Quando for executado, tudo se passará da mesma forma como se tivesse sido escrito da maneira anterior. O uso de instruções múltiplas, por linha, provoca economia de memórias, na armazenagem do programa.

No entanto, é necessário não exagerar:

- uma linha numerada só comporta um certo número de caracteres, o que limita a quantidade de instruções e comandos por linha numerada;
- existem certas instruções (como veremos mais adiante) que influenciam todas as outras instruções colocadas na mesma linha; por isso, em muitos casos, é necessário mudar de linha para colocar certas instruções;

— certos comandos ou instruções só podem ficar no fim da linha (última instrução), para não perturbarem as instruções seguintes:

Ex. 2:
```
10 A=5 : B=6 : END: C=B+A:?C
RUN

 OK
```

não aconteceu nada, pois o programa terminou no END; esse comando só poderia ficar no fim da linha;

— é recomendável não colocar mais do que um número suficiente de instruções que preencha até duas linhas de vídeo, caso contrário, se tivermos que corrigir alguma coisa no meio da linha, teremos que teclar novamente a linha inteira, o que poderá ser trabalhoso.

Caberá ao leitor se habituar com a maneira mais prática de utilizar esses comandos múltiplos por linha.

Uma observação importante sobre os comandos múltiplos é que os mesmos podem ser introduzidos de modo direto e executados imediatamente:

Ex. 3:
```
A= 5: B = 3: PRINT A + B  (<RETURN>)
 8

 OK
```

Às vezes, no meio de um programa, é importante fazer um pequeno programa de verificação, em modo direto, para efetuar correções e saber o estado das diversas variáveis; esse programa em modo direto não interfere com o programa já gravado, a menos que haja instruções nele que alterem dados ou variáveis do anterior; depois da verificação, podemos prosseguir com a execução do programa:

Ex. 4: programa existente
```
10 A=5 : B=8 : C=3*B
20 A= A+2
30 C =C+A
40 ? C
RUN
```

```
 31

OK
```

Se, por acaso, houvesse um engano no programa e o resultado não fosse o esperado, poderíamos usar um programa em modo direto para saber o que está acontecendo.

Vamos supor que, por engano, na linha 20, havíamos escrito **20 A=A-2**
em vez de **20 A=A+2**

Ao ser executado o programa, o resultado seria 27 e não 31.

Suponhamos que não percebemos esse erro, na listagem do programa.

Façamos o seguinte:

```
25 STOP
```

Agora

```
RUN

BREAK IN 25
OK
```

Nesse ponto, podemos introduzir um programa em modo direto, para saber qual o valor das variáveis, na hora da interrupção:

```
? A; B; C          (<RETURN>)
  3  8  24

OK
```

Já ficamos sabendo que A = 3, quando deveria ser 7; logo, houve um erro na linha 20.

Se quisermos saber se o erro foi só esse, façamos o seguinte:

Mantenha o
```
25 STOP
RUN
BREAK IN 25
OK
```

Agora, em modo direto, façamos o seguinte:

```
A = 7:CONT
  31

OK
```

Quando o programa for novamente executado, o comando direto A=7 não será considerado (mas não esqueça de retirar o 25 STOP).

Capítulo V
COMPUTAÇÕES MATEMÁTICAS

Na maioria dos programas interpretadores ou compiladores BASIC já está embutida a capacidade de executar uma série de operações aritméticas e algébricas, que podem variar desde simples somas e subtrações até computações bem mais complexas, dependendo do tamanho do interpretador ou compilador.

Vamos dar uma visão geral do que é, normalmente, encontrado nos programas de tamanho médio para bom, ficando a critério do leitor verificar quais as possibilidades de seu microcomputador.

5.1 — Operações elementares: são as de Soma, Subtração, Multiplicação e Divisão.

Para usar o computador na execução de uma dessas operações, basta usar o sinal correspondente no teclado.

Por exemplo: na tecla onde aparece o ponto-e-vírgula, aparece também o sinal "+"; para teclar +, é necessário bater simultaneamente a tecla SHIFT (como já vimos anteriormente) com aquela tecla.

O sinal "-" aparece abaixo do igual (=), ou seja, não é necessário teclar SHIFT.

O sinal de multiplicação é um asterisco (*), que aparece em cima do sinal de dois pontos; é necessário teclar simultaneamente o SHIFT; finalmente, o sinal de divisão é a barra (/), que aparece abaixo da interrogação, não sendo necessário teclar SHIFT.

A execução das operações é imediata, por comando direto ou por programa:

Ex. 1: **PRINT 15 + 12** **(<RETURN>)**

27

OK

(é necessário dar a instrução PRINT quando queremos ver o resultado; em certos programas não queremos ver resultados intermediários e só mandamos executar a operação, sem PRINT).

Ex. 2:
```
10 A= 13
20 B= 15
30 C= A-B
40 PRINT C
RUN
-2

OK
```

Ex. 3:
```
PRINT 2*15
 30

OK
```

Ex. 4:
```
PRINT 50/2
 25

OK
```

Os exemplos acima mostram as operações executadas uma a uma; no entanto, é perfeitamente possível executar de uma só vez um conjunto de operações, como mostraremos nos exemplos a seguir:

Ex. 5:
```
PRINT 20*3/2+18/2
 39

OK
```

Ex. 6:
```
10 C= 12
20 D = C*3
```

```
30 F = D+C
40 G = F/4+12
50 ?G
RUN
 24

OK
```

Também é possível usar parênteses, para efetuar as operações, seguindo as regras usuais de aritmética:

Ex. 7:
```
PRINT(12+13)*4
 100

OK
```

Ex. 8:
```
10 A=3
20 B=3*6
30 C=(A+B)/2
40 PRINT C
50 D=C* (A+B+5)/B
60 PRINT D
RUN
 10.5
 15.1667

OK
```

Verifica-se que foi impresso um resultado parcial (C = = 10.5) e que o resultado final foi impresso arredondado na 6ª casa.

5.2 – Exponenciação permite elevar um número a determinada potência, usando o sinal ↑ ou ∧.

Ex. 9:
```
PRINT 2↑3
 8

OK
```

Ex. 10:
```
10 A=3
20 B=6
30 ? (A+B)↑2
RUN
 81

OK
```

Ex. 11:
```
PRINT 3↑2↑2    (potenciação repetida)
 81

OK
```

5.3 — Extração de Raiz: essa operação não existe, normalmente, como instrução específica, na linguagem BASIC (a não ser em versões científicas especiais); o que se faz é elevar o número a uma potência fracionária, do tipo X= A↑(1/N), em que A é o número cuja raiz de grau N se deseja. A exceção é a raiz quadrada, que possui uma instrução especial, como será mencionado mais adiante, ao tratarmos de funções matemáticas.

5.4 — Regras gerais de organização das operações: as operações aritméticas são executadas pelo computador em uma certa ordem, que deve ser entendida (da mesma forma que cada calculadora de bolso tem a sua própria ordem de execução das operações).

As regras são as seguintes:

a) as operações começam da esquerda para a direita;
b) as operações de exponenciação são efetuadas primeiro (todas as que existirem na expressão);
c) depois da exponenciação o programa volta ao início da expressão e executa todas as multiplicações e divisões, até o fim da expressão;
d) começando novamente no início da expressão, serão executadas todas as operações de adição e subtração;
e) se houver parênteses, as operações dentro dos parênteses serão executadas em primeiro lugar;
f) se houver parênteses dentro de parênteses, as operações serão começadas pelos parênteses interiores, seguindo-se os parênteses mais exteriores, e assim por diante.

Vejamos essas regras, seguidas passo a passo:

Ex. 12: **PRINT (5 * 3 + (4 - 2) ↑ 3 - (13 + 3)/4)**

primeiras operações: execução dos parênteses internos:
(4 - 2) = 2
(13 + 3) = 16
em seguida, potência:
2 ↑ 3 = 8
em seguida, multiplicações e divisões:
5 * 3 = 15
16/4 = 4
finalmente, somas e subtrações:
15 + 8 - 4 = 19
o computador mostrará:

```
 19

OK
```

Ex. 13:

```
10 A=1
20 B=3
30 C=5
40 D=(A*B+C)/C+B↑3*(B+C)
50 PRINT "O VALOR DE D É"; D
RUN
O VALOR DE D É 217.6

OK
```

5.5 – Os programas de BASIC mais completos possuem um repertório grande de funções matemáticas, das quais mostraremos as mais usualmente encontradas. São funções que já estão programadas no interpretador ou compilador e podem ser obtidas por simples teclagem do nome da função e do argumento:

– RND(X) é uma função que dá um número aleatório *entre* 0 e 1; normalmente o valor de X deve ser positivo e maior que zero: nessas condições, cada vez que o computador executar a função RND(X) escolherá um número diferente, aleatoriamente; se o valor de X for zero, será esco-

lhido um número aleatório mas, no mesmo programa, será o mesmo; se o número (X) for negativo, para cada valor de X será selecionado um número aleatório, o que será repetido enquanto não se fizer CLEAR no programa.

Daremos um exemplo simples e mostraremos, em outros capítulos, novos exemplos da função RND (RANDOM = ALEATÓRIO).

Ex. 14:
```
10 PRINT RND (1)
20 PRINT RND (1)
30 PRINT RND (1)
RUN
 .478563
 .954328
 .763166
OK
```
cada operação dá um número diferente (maior que Ø e menor que 1)

Ex. 15:
```
10 PRINT RND (Ø)
20 PRINT RND (Ø)
30 PRINT RND (Ø)
RUN
 .543287
 .543287
 .543287
OK
```
o número é aleatório mas é sempre o mesmo, dentro do mesmo programa

Ex. 16:
```
10 PRINT RND (-4)
20 PRINT RND (-0.3)
30 PRINT RND (-4)
RUN
 .332854
 .473462
 .332854
OK
```
(igual ao primeiro)

Há muitas outras observações a fazer sobre esta importante função, como se verá adiante.

— LEN (A$): essa função dá o número de caracteres do STRING A$, inclusive sinais e espaços.

Ex. 17:
```
10 A$ = "ABCDE"
20 PRINT LEN (A$)
RUN
 5            (cinco caracteres)

OK
```

Ex. 18:
```
10 A$ = "JOÃO R. SOUZA"
20 ?LEN(A$)
RUN
 13       (inclusive o ponto e os espaços)

OK
```

— ABS(X), dá o valor absoluto da expressão X:

Ex. 19:
```
10 X=13 - 18
20 ?ABS(X)
RUN
 5          (valor absoluto de -5)

OK
```

— INT(X), dá o maior valor inteiro contido em X:

Ex. 20:
```
10 X=13.478
20 ? INT (X)
RUN
 13

OK
```

Essa função é bastante importante, pois, em muitos casos, necessitamos apenas de valores inteiros de números, como, por exemplo, no caso de efetuar contagens de eventos, a partir de equações, cujos resultados não são sempre inteiros.

— TAB(X) é uma função de tabulação que determina a posição do cursor, para impressão de um número ou STRING, seja na tela de vídeo, seja em uma impressora; o argumento X representa a posição, considerando como 1ª

posição o lado mais à esquerda de cada linha do vídeo ou da impressora. É necessário notar que cada linha só contém N caracteres, em geral, o que limita a posição que podemos atribuir a "X".

Ex. 21:
```
PRINT TAB (3); "POSIÇÃO 3"
```
```
 POSIÇÃO 3      (P está na posição 3)

OK
```
(observe o ponto-e-vírgula entre TAB(3) e o STRING)

Ex. 22:
```
10 ? TAB (18); "POSIÇÃO"
20 ? TAB (32); "NÚMERO 32"
RUN
                    POSIÇÃO
          NÚMERO 32

OK
```

O TAB(32) ficou em outra linha, pois estamos considerando um vídeo com 24 posições por linha.

— SGN(X) é uma função que indica se X é maior, igual ou menor que zero. Quando X é maior que zero, SGN (X) = 1; se X = Ø, SGN (X) = Ø e se X é menor que zero, SGN(X) = -1. É uma função muito útil, pois serve para produzir artifícios de programação em que desejamos saber quando um certo valor é maior, menor ou igual a zero. Basta saber se SGN(X) é igual a 1, -1 ou Ø.

Ex. 23:
```
10 A= SGN(X)
20 X= -4.18 *.56
30 ? A
RUN
 Ø

OK
```

O valor zero apareceu porque a função A = SGN(X) ainda não sabia quem era X, ao passar pela linha 1Ø e, assim, considerou X = Ø.

Agora, vamos inverter:

```
10 X= -4.18 *.56
20 A = SGN(X)
30 ? A
RUN
-1          (X tem valor negativo)

OK
```

Ex. 24: modo direto

```
PRINT SGN(4.8)
 1

OK
```

Ex. 25:

```
10 A=3*-.15 + 18/3 *.09
20 ? SGN(A)
RUN
 1

OK
```

Mesmo sem saber o valor da expressão, sabemos que ela é positiva.

– SIN(X) dá o seno (função trigonométrica) de X, sendo X expresso em radianos (convém lembrar que 360º correspondem a 2π radianos).

Ex. 26:

```
PRINT SIN (.523)      (equiv. a 30º)
 .499481              (aproximação de 0.5)

OK
```

– COS(X) dará o cosseno de X, expresso em radianos.

Ex. 27: (cálculo de um dos lados de um triângulo retângulo cuja hipotenusa é igual a H e o ângulo adjacente ao lado desejado é 60º, ou 1.046 radianos).

```
10 H = 5          (hipotenusa)
20 ? COS (1.046)
30 C = COS (1.046)
```

```
40 D = C*H    (para confirmar o valor do cosse-
50 ? D/5      no)
RUN
 .50
 .50

OK
```

(no exemplo acima usamos a propriedade dos triângulos retângulos de que o lado adjacente ao ângulo X dividido pela hipotenusa é igual ao COS(X))

— TAN(X) dá o valor da tangente de um ângulo expresso em radianos.

— ATN(X) dá o valor do ângulo em radianos, cuja tangente é X, ou seja, dá o arcotangente (X):

Ex. 28:
```
10 X = .785398      (ou seja, 45º)
20 ? TAN(X)
30 C = TAN(X)
40 ? ATN(C)
RUN
 1          (tangente de 45º)
 .785398    (ângulo em radianos cuja tangente
            é 1)

OK
```

— SQR(X) dá a raiz quadrada de X; como já mencionamos atrás, não é uma operação que pode ser feita através da teclagem de um símbolo (tal como multiplicação ou exponenciação): é uma função matemática que é produzida por um programa especial que já faz parte do interpretador ou compilador BASIC (nos de tipo mais completo).

Ex. 29:
```
PRINT SQR(25)
 5

OK
```

É claro que X deve ser positivo, pois a linguagem BASIC não trabalha com números complexos, a não ser que haja programa especialmente preparado para isto.

— EXP(X) dá o valor da constante "e" elevada a X (e = 2,71828...)

Ex. 30: **PRINT EXP (3)**
20.0855 ("e" elevado ao cubo)

OK

— FRE(X) dá a quantidade de Bytes ainda livres na memória onde estamos programando; é muito útil para saber se, ao preparar um programa muito longo, não vamos ultrapassar a capacidade da memória do computador.

Ex. 31: **PRINT FRE(Ø)**
5437

OK

(isto significa que, se o total de Bytes da memória com que estamos trabalhando é, p. ex., 8.192, já usamos, até agora, 1.755 Bytes nos programas já armazenados).

O valor atribuído a X significa, apenas, que X, nessa função é um argumento falso ("dummy"), ou seja, pode assumir qualquer valor e só existe para que a função tenha algum argumento.

Em alguns tipos de BASIC (os mais completos), se X = = Ø, o número de Bytes livres não conta os Bytes ocupados por STRINGS; quando queremos incluir os STRINGS, X deve ser substituído por A$. Esse arranjo é muito útil para se determinar quais os tipos de programas ou arranjos que podemos fazer utilizando o mínimo possível de memórias.

— LOG(X) dá o valor do logaritmo neperiano (de base "e"), de X.

Ex. 32: **PRINT LOG(2.718)**
.999896 (log. nep. de "e" é igual a 1)

OK

Para se ter o logaritmo de base 10, basta dividir LOG (X) por LOG(10) (logaritmo neperiano de 10).

Ex. 33:
```
10 X = 100
20 C = LOG(X)
30 ? C
40 D = C/LOG (10)
50 ? D
RUN
 4.60517      (log. nep. de 100)
 2            (log. 10 de 100)

OK
```

— POS(X) dá a posição, naquele momento, do cursor do vídeo (posição onde está sendo impresso um caracter) ou da cabeça de impressão, no caso de uma impressora serial. X é uma variável falsa; é necessário atribuir-lhe um valor.

Ex. 34:
```
10 ? "ABCD"; POS(1)
RUN
ABCD 4   (próxima posição do cursor; a primeira posição é 0)

OK
```

Note o seguinte: a função POS(1) foi colocada na mesma linha que a instrução PRINT e está separada por ponto-e-vírgula. Isto significa: imprimir o STRING "ABCD" e, logo depois, a posição do cursor.

Se colocarmos PRINT POS(1) em outra linha, a posição será sempre zero, pois, quando mudou de linha, o cursor voltou para a posição zero.

Ex. 35:
```
10 ? "ABC"
20 ? POS(1)
RUN
ABC
 0

OK
```

— SPC(X) é uma instrução de espaço, que manda colocar na linha X espaços em branco; essa função só funciona associada ao comando PRINT.

Ex. 36:
```
PRINT SPC(4); "ABC"
    ABC     (começa no 5º espaço)

OK
```

Ex. 37:
```
10 A= 15
20 B= RND(1)
30 ? SPC(A);B
40 C=INT (10*B)
50 ? SPC(C);A
RUN
 .473685
 15
OK
```

(deixa 15 espaços + o espaço do sinal) → .473685

→ 15

(deixa 4 espaços + o espaço sinal) → OK

Explicação: na primeira linha o número B foi impresso após 15 espaços; o número B é aleatório e, no caso, correspondeu a .473685; na segunda linha o número de espaços foi igual ao maior valor inteiro (INT) de B multiplicado por 10, ou seja, inteiro de 4.73685, o que dá 4 espaços.

— DEF FNA(X) essa função DEF, em conjunto com FNA(X), serve para definir, em um programa, a equação de uma função A, da variável X. Depois de feita essa definição, no restante do programa podemos fazer referência apenas a FNA(X), e o computador já saberá qual a equação correspondente.

O uso dessa DEFinição de função é muito importante, especialmente quando há várias funções diferentes, de variáveis diferentes, no mesmo programa.

Ex. 38:
```
10 DEF FNA(X) = 15 * X ↑ 2
20 DEF FNB(Y) = Y * 2 + SQR (Y)
30 X = 3
40 Y = 2
```

```
50 PRINT FNA(X)
60 PRINT FNB(X)
RUN
 135
 5.41421

OK
```

Durante o programa podemos fazer variar o valor de X e Y, e o programa sempre dará o valor da função definida previamente FNA(X) e FNB(Y).

Ex. 39:
```
10 DEF FNT(W) = 3* W↑3
20 DEF FNQ(Z) = Z + 18/Z
30 W = 1
40 Z = 2
50 ? FNT(W); FNQ(Z)
60 W = W + 1
70 Z = Z + 2
80 ? FNT(W); FNQ(Z)
90 W = W + 3
100 Z = Z + 4
110 ? FNT(W); FNQ(Z)
RUN
 3    11          (primeira vez)
 24    8.5        (segunda vez)
 375    10.25     (terceira vez)

OK
```

Observe que os valores de W e Z foram alterados, mas as funções já estavam definidas e, assim, seus valores puderam ser impressos para cada valor das variáveis (W e Z); na linha 30 W era igual a 1, depois passou para 2 (W = W + 1) e depois, para 5 (W = W + 3); o mesmo aconteceu com Y.

As funções matemáticas ou de posicionamento que acabamos de descrever são as mais usuais nas versões de BASIC do tipo denominado ALTAIR. Existe, no entanto, o que se denomina BASIC PLUS, mais completo, que contém outras funções e instruções que não descreveremos neste livro introdutório.

— Existe uma observação importante a fazer ao final deste capítulo: em certos tipos de BASIC não podemos mandar imprimir A + B, sem dizer antes que C = A + B. Depois, mandamos imprimir C.

Ex. 40:
```
10 A=2; B=3: ? A+B
```

Em certas versões de BASIC isso não funciona; é preciso escrever:

```
10 A=2:B=3:C=A+B:?C
```

Capítulo VI
INSTRUÇÕES PARA ENTRADA DE DADOS

Neste capítulo cobriremos a definição e a explicação das instruções mais usualmente encontradas nos programas de BASIC de certo porte para atribuir valores a variáveis. Fica claro que existem outras instruções não mencionadas neste livro, que fazem parte de versões mais sofisticadas ou são instruções especiais para certos tipos de sistema operacional com comando de periféricos, tipo disco magnético. No entanto, acreditamos que estão descritas as instruções que o programador poderá encontrar e usar nos microcomputadores mais usuais do mercado.

Já vimos que uma variável pode ter valores a ela atribuídos e que esses valores podem ser alterados, ao longo do programa.

Podemos dar um valor a uma variável de várias formas, usando instruções de BASIC. Vejamos as principais:

6.1 – INPUT X

Essa instrução faz com que o programa pare e fique aguardando a entrada, pelo teclado, do valor que queremos dar a X (a instrução é INPUT e X ou qualquer outra variável é aquela que tomará o valor que for teclado). Ao interromper o programa, aparece na tela do vídeo uma interrogação, que indica ao operador que o programa está aguardando a entrada de um valor.

Ex. 1:
```
10 INPUT B
20 C = B*3
30 A = B+C
40 ? A
RUN
?
```

(o programa parou na linha 20 e a interrogação pergunta ao operador qual o valor de B; após ser teclado um valor, B passará a ter esse valor durante todo o resto do programa, e a execução continua imediatamente após ser apertada a tecla RETURN)

Vamos continuar:

Tecle o número 3

```
? 3
```

(o número 3 aparece logo depois da interrogação).

Tecle RETURN e o programa prosseguirá:

```
 12

OK
```

(B=3, C=3 *3, A = 3 + 9 = 12)

Ex. 2:

```
10 PRINT "ESCOLHA UM VALOR"
20 INPUT Q
30 PRINT "O VALOR"; Q; "FOI ESCOLHIDO"
```

(suponhamos que vamos escolher o valor 18)

```
RUN
ESCOLHA UM VALOR
? 18
O VALOR 18 FOI ESCOLHIDO

OK
```

É sempre aconselhável, antes de pedir o valor de uma variável, imprimir alguma explicação sobre qual a variável que está sendo pedida, para que seja facilitado o trabalho do operador. Caso contrário nós teríamos que lembrar a cada momento qual a variável que está sendo pedida pelo programa (lembre-se que a instrução INPUT pode aparecer várias vezes no mesmo programa, pedindo variáveis diferentes).

Ex. 3:

```
10 ? "DEMONSTRAÇÃO DE SOMA E MULTIPLICAÇÃO"
```

```
15 ? "QUAL O VALOR DE A?"
20 INPUT A
25 ? "QUAL O VALOR DE B?"
30 INPUT B
35 C = A + B
40 PRINT C
45 ? "QUAL O VALOR DE D?"
50 INPUT D
60 E = D * C
70 ? "O VALOR FINAL DE E SERÁ"; E
```

(Vamos supor escolhidos A = 2, B = 3 e D = 4)

```
RUN
DEMONSTRAÇÃO DE SOMA E MULTIPLICAÇÃO
QUAL O VALOR DE A?
? 2

QUAL O VALOR DE B?
? 3
5      (valor de C)

QUAL O VALOR DE D?
? 4

O VALOR FINAL DE E SERÁ 20

OK
```

Essa instrução INPUT serve para promover um meio de comunicação entre o operador e o computador. Veremos, mais adiante, como essa instrução pode ser utilizada em toda a sorte de programas, inclusive jogos para vídeo.

A mesma instrução pode ser usada para STRINGS, isto é, variáveis de STRINGS.

Se chamarmos A$ uma variável de STRING, podemos dar a ela um significado qualquer, que ficará armazenado na memória do computador.

Ex. 4:
```
10 INPUT Q$
20 PRINT Q$;" É O MEU NOME"
RUN
```

```
? ROBERTO          (esse nome foi teclado pelo
                    operador)
ROBERTO É O MEU NOME

OK
```

Agora a variável Q$ passou a significar o STRING "ROBERTO".

Enquanto não for mudado o valor de Q$, dado o comando CLEAR (zerando as variáveis) ou o comando NEW (apagando o programa), essa variável permanecerá com esse significado.

É preciso não esquecer que um novo comando RUN apagará os valores já atribuídos às variáveis.

Façamos um novo programa, com numeração mais alta (para não perturbar o primeiro programa).

```
50 ? "MEU NOME É "; Q$
60 INPUT B$
70 ? "E O TEU É "; B$
```

Ao rodar este programa (a partir da linha 50), veremos que o valor anterior de Q$ foi apagado:

```
RUN 50
MEU NOME É          (falta o valor de Q$)
? ANDRÉ             (entramos com ANDRÉ)
E O TEU É ANDRÉ

OK
```

Observe que, no caso de uma variável numérica, se não dermos nenhum valor à mesma, o programa considerará que a variável tem valor Ø (zero). No caso de uma variável de STRING, no entanto, o programa considerará que o STRING está vazio (não imprimirá nada).

Agora vamos rodar o programa do início e mudar os nomes:

```
RUN
? ANDRÉ
ANDRÉ É O MEU NOME
```

```
MEU NOME É ANDRÉ
? ROBERTO
E O TEU É ROBERTO

OK
```

Mudamos o significado das variáveis Q$ e B$.

A instrução INPUT pode ser precedida, na mesma linha, de um STRING, que será impresso antes do ponto de interrogação. Isso é importante para saber que informação o programa deseja naquele momento.

Ex. 5:
```
10 INPUT "QUAL É O TEU NOME"; N$
          (ponto-e-vírgula)
20 PRINT N$; " É O TEU NOME"
RUN
QUAL É O TEU NOME? JOÃO
(interrogação do
        INPUT)
(nome teclado)
JOÃO É O TEU NOME

OK
```

Se não fosse colocado um espaço no STRING " É O TEU NOME", as palavras impressas ficariam sem espaço: JOÃOÉ O TEU NOME.

Na linha 10 é importante notar que a pergunta que antecede a variável, no statement de INPUT, tem que estar entre aspas e deve haver uma separação, entre a pergunta (entre aspas) e a variável pedida, por ponto-e-vírgula; uma separação usando apenas vírgula não funciona, e o computador acusará erro.

Uma variante do INPUT, para a entrada de diversos dados pode ser usada, da seguinte forma:

Ex. 6:
```
10 INPUT A, B, C                (1)
20 ? A;B;C
RUN
? 5,3,-2                        (2)
 5  3  -2

OK
```

(1) as variáveis pedidas devem ser colocadas uma após a outra, separadas por vírgulas, exceto antes da primeira e depois da última;
(2) a entrada, pelo teclado, das variáveis, deve ser feita usando vírgulas, na mesma ordem em que as variáveis foram pedidas; o primeiro número teclado vai corresponder ao A, o segundo ao B e o terceiro ao C.

Se esquecermos de entrar com uma das variáveis pedidas e teclarmos <RETURN>, o computador colocará outra interrogação, pedindo a próxima ou as próximas variáveis:

Ex. 7:
```
10 PRINT "QUAIS OS NOMES DOS SEUS FILHOS";          (1)
20 INPUT A$, B$, C$, D$
30 ? "O PRIMEIRO É "; A$
40 ? "O SEGUNDO É "; B$
50 ? "O TERCEIRO É "; C$
60 ? "O QUARTO É "; D$
RUN
QUAIS OS NOMES DOS SEUS FILHOS? ANDRÉ,
LIANA                                               (2)
?? RENATA, FELIPE
O PRIMEIRO É ANDRÉ
O SEGUNDO É LIANA
O TERCEIRO É RENATA
O QUARTO É FELIPE

OK
```

(1) o ponto-e-vírgula depois do STRING vai fazer com que a próxima impressão se dê na mesma linha; por isso a interrogação do INPUT vai aparecer logo depois da frase QUAIS OS NOMES DOS SEUS FILHOS e não na próxima linha, como em exemplos que vimos anteriormente (quando não havia ponto-e-vírgula depois do STRING — ver exemplos 2 e 3); a entrada dos dados se fará na mesma linha, após a interrogação;

(2) aqui deveríamos colocar quatro nomes, pois o programa pede 4 variáveis (A$, B$, C$ e D$); por engano teclamos <RETURN> depois do segundo nome e assim, o computador colocou dois pontos de interrogação, pedindo os próximos nomes. ***Nota importante***: o computador já atribuiu os dois primeiros nomes às variáveis A$ e B$; agora está aguardando C$ e D$; por isso, a nova pergunta não significa que devemos repetir os quatro nomes, mas apenas os que faltam.

6.2 — READ e DATA

Essas instruções são, também, usadas para atribuir valores a variáveis; apenas, em vez de dar entrada nas variáveis uma a uma, nós construímos uma tabela de dados (DATA), e o programa vai, progressivamente, dando os valores dessa tabela às variáveis que nós indicamos após a instrução READ.

Ex. 8:
```
10 READ A
20 ? A
30 DATA 18
RUN
 18

OK
```

A instrução READ da linha 1Ø leu os dados da linha 3Ø (só havia o dado 18) e associou o valor 18 à variável A. Esse valor foi impresso pela instrução da linha 2Ø.

Se colocarmos mais de uma variável nas instruções READ, o grupo de dados será lido seqüencialmente, isto é, cada READ lerá o próximo dado listado após a palavra DATA.

Ex. 9:
```
10 READ A
20 READ B
30 READ C
40 ? A;B;C
50 DATA 2,18,17
RUN
```

```
 2  18  17

OK
```

O primeiro READ (READ A) leu o primeiro dado (2), o segundo READ leu o segundo dado e, assim por diante; a partir desse momento, no programa (até ser apagado ou alterado) a variável A terá o valor 2, B será igual a 18 e C igual a 17.

Observação importante: o comando RUN faz com que todas as variáveis sejam zeradas (isto é, fiquem com valor Ø). Assim, após rodar (executar) o programa visto acima, as variáveis A, B e C ficam com os valores atribuídos a elas até que se dê nova execução do programa (RUN).

Em geral isto não tem importância, pois, no programa, ao ser executado novamente, as variáveis voltarão a ter os seus valores associados, por meio de READ e DATA.

Para entender melhor este ponto, faça a seguinte experiência:

Escreva um programa com numeração mais alta do que a última linha do anterior:

```
100 ? "O VALOR DE A É"; A
110 ? "O VALOR DE B É"; B
120 ? "O VALOR DE C É"; C
```

Agora, execute o programa, à partir da linha 100

```
RUN 100
O VALOR DE A É Ø
O VALOR DE B É Ø
O VALOR DE C É Ø

OK
```

O comando RUN 100 fez apagar os valores de A, B e C dados pelo primeiro programa. Agora, vamos executar tudo desde o início (incluídos os dois programas).

```
RUN
 2   18   17
O VALOR DE A É 2
O VALOR DE B É 18
O VALOR DE C É 17

OK
```

os valores de A, B e C não foram apagados

Os dados têm que ser separados por vírgulas, exceto o primeiro e depois do último.

```
50 DATA   2,   18,   17
```

(sem vírgula)
(vírgulas)

Da mesma forma, as variáveis podem ser colocadas sob uma mesma instrução READ, separadas por vírgulas, na mesma linha:

```
10 READ A,B,C
```

A mecânica é sempre a mesma: a instrução READ associa à primeira variável o primeiro dado; à segunda variável o segundo dado; e assim por diante.

Quando houver mais variáveis do que dados, o programa acusará erro.

Ex. 10:
```
10 READ A,B,C,D
20 PRINT A;B;C;D
30 DATA 55,2,18
RUN
 55   2   18
? ERROR IN 10
OK
```

(falta um dado para a variável D)

Se houver mais dados do que variáveis somente serão lidas as variáveis correspondentes aos primeiros dados:

Ex. 11:
```
10 READ A,B
20 ? A;B
30 DATA 12,18,17,19
RUN
```

```
 12  18

OK
```

O programa associou o primeiro dado a A, o segundo dado a B e os outros dados foram ignorados.

As instruções READ e DATA podem ser associadas a variáveis de STRING, funcionando da mesma forma que no caso de variáveis numéricas.

Ex. 12:
```
10 READ A$,B$,C$
20 PRINT C$;B$;A$
30 DATA ÓTIMO,É,ISTO
RUN
ISTOÉÓTIMO

OK
```

O que aconteceu? lembremo-nos que o ponto-e-vírgula nãc introduz nenhuma separação entre os STRINGS que vão ser impressos. No caso de números é diferente, pois sempre é colocado um espaço depois do último algarismo (além do espaço na frente, para o sinal + ou -).

Se usarmos vírgula, em vez de ponto-e-vírgula, a impressão será feita nas casas Ø, 14, 28, etc. (como já foi visto).

Vejamos (substituindo a linha 2Ø)
```
20 PRINT C$,B$,A$
```
Vamos listar
```
LIST
 10 READ A$,B$,C$
 20 PRINT C$,B$,A$
 30 DATA ÓTIMO,É,ISTO
RUN
ISTO          É
   ÓTIMO

OK
```

Uma observação importante a fazer é relativamente à ordem de leitura (READ) e de impressão; uma coisa não tem nada a ver com a outra.

As variáveis A$, B$ e C$ foram lidas na mesma ordem em que foram colocadas após o DATA. Desse modo A$ passou a ser ÓTIMO, B$ ficou sendo É e C$ assumiu o valor ISTO.

Na hora de mandar imprimir, nós podemos mandar a impressão ser feita em qualquer ordem; no caso, foi impresso primeiro a variável C$, depois B$ e por último A$. Isto foi feito apenas para exemplificar a atribuição de um valor a uma variável e o uso dessa variável, que são coisas diferentes.

Podemos, também, misturar variáveis numéricas e variáveis de STRING:

Ex. 13:
```
10 READ A$,N,B$
20 PRINT A$;N;B$
30 DATA TENHO,15,ANOS
RUN
TENHO 15 ANOS

OK
```

A variável A$ ficou sendo TENHO, a variável N ficou sendo 15 e a variável B$ passou a ser ANOS.

É importante notar, no entanto, que não podemos misturar a ordem das variáveis numéricas e de STRING. Assim, quando na linha 10 o programa é instruído para ler (READ) A$, o primeiro dado que deve existir depois do DATA deve ser um STRING, pois A$ só pode ser STRING. Se colocarmos um número, como primeiro dado, o programa considerará esse número como sendo um STRING e o número não poderá ser usado em nenhuma operação matemática.

Vamos inverter a ordem dos dados:

```
30 DATA 15, TENHO, ANOS
```

Tentemos executá-lo.

```
RUN
 15   (o primeiro foi lido e impresso)
? ERROR IN 30   (o computador acusou erro)

OK
```

Acontece que A$ pode ser igual a 15 e, nesse caso, 15 passará a ser um STRING; porém, N (que é uma variável numérica) não pode ser igual a TENHO.

Vamos alterar novamente a linha 3Ø:

```
3Ø DATA 15, 15, ANOS
```

Agora, o programa vai aceitar:

```
RUN
 15  15  ANOS

OK
```

Note o seguinte: o primeiro 15 foi transformado em STRING, mas esse STRING *inclui* um espaço para o sinal; assim, sempre que o STRING A$ for impresso deixará um espaço na frente do número (mas não atrás, pois esse espaço é uma questão de estrutura da linguagem e não faz parte do número).

Ex. 14:

```
1Ø READ A$,N,B$
2Ø ? A$;N;B$
3Ø DATA 15,15,ANOS
4Ø ? N↑2
5Ø ? A$*2
RUN
 15  15  ANOS
 225                (instrução da linha 4Ø)
ERROR IN 5Ø

OK
```

Verificamos que N = 15 e pode ser manipulado matematicamente, mas o mesmo não acontece com A$, apesar de A$ = 15. Por isso, o computador acusou erro, quando mandamos multiplicar A$ por 2.

Ainda sobre as instruções READ e DATA:

Enquanto houver dados e instruções de leitura, os dados serão lidos pelas variáveis, mesmo que o conjunto de dados esteja mais adiante no programa. A instrução READ vai procurar um dado onde houver uma instrução DATA:

Ex. 15:
```
10 READ A,B
20 ? A;B
30 READ C,D
40 DATA 12
50 ? C;D
60 DATA 18,15
65 E= B+C
70 ?E
80 DATA 10
RUN
 12   18      (A e B)
 15   10      (C e D)
 33            (E)

OK
```

Verifique o seguinte: a primeira instrução READ A foi pegar o primeiro dado depois do DATA na linha 40; READ B não encontrou mais nada na linha 40 e foi procurar o próximo DATA (linha 60), pegando o primeiro valor; READ C foi pegar o segundo valor na linha 60 e READ C foi procurar um novo DATA, que encontrou na linha 80. O conjunto de dados pode estar no fim do programa (o que é até usual).

É importante guardar duas coisas em mente:

- cada linha que tiver dados tem que começar com a instrução DATA;
- se aparecer novamente a mesma variável, depois de uma instrução READ, essa variável assumirá o valor correspondente à ordem em que estão sendo lidos os dados, mudando, inclusive, de valor, se já tiver assumido algum anteriormente.

Ex. 16:
```
10 READ A$,B$
20 ? A$;B$
30 READ A$,B$
40 ? A$;B$
50 DATA "LINDO ","DIA"
60 DATA "FICOU ","FEIO"
RUN
```

```
LINDO DIA
FICOU FEIO

OK
```

Alguns aspectos são interessantes nesse programa:

Na primeira impressão (linha 2Ø), A$ era igual a LINDO e B$ era igual a DIA; quando mandamos continuar a leitura (linha 3Ø), o A$ tomou o valor do próximo dado disponível (FICOU) e B$ assumiu o valor seguinte (FEIO). Esses últimos valores são os que ficaram armazenados na memória, pois o novo valor de A$ e B$ apagou seu valores anteriores.

Façamos uma experiência: vamos mandar imprimir A$ e B$, em modo direto:

```
PRINT A$; B$
FICOU FEIO

OK
```

(os valores de A$ e B$ continuam atualizados, pois não foi dado nenhum CLEAR ou RUN).

Uma outra observação se refere às aspas utilizadas nos dados:

Toda a vez que for necessário colocar espaços como parte de STRING é necessário usar aspas, caso contrário as palavras ficarão grudadas uma na outra, como já vimos em exemplo anterior; nesse caso o espaço passa a fazer parte do STRING. A mesma coisa ocorre se for necessário colocar vírgulas dentro do STRING (já sabemos que a vírgula separa os dados entre si e, portanto, para colocar vígulas em um STRING se necessita das aspas).

Ex. 17:
```
10 READ A$,B$
20 PRINT A$;B$
30 DATA "O CÉU ESTÁ ","AZUL,
LINDO E CLARO"
RUN
O CÉU ESTÁ AZUL, LINDO E CLARO

OK
```

Os espaços foram colocados dentro dos STRINGS, assim como a vírgula depois da palavra AZUL.

6.3 – RESTORE

Essa instrução complementa o READ/DATA em certos casos e tem a seguinte função:

Quando a instrução READ acabou de ler todos os dados contidos em DATA, o programa prosseguirá, como já vimos. No entanto, se quisermos repetir, dentro do mesmo programa, a leitura dos mesmos dados, podemos usar a instrução RESTORE, que fará com que o próximo READ volte a ler o primeiro DATA que for encontrado.

Isto significa: RESTORE é uma instrução que diz ao programa que os dados podem ser lidos novamente, desde o início.

Ex. 18:
```
10 READ A$,B$
20 PRINT A$;B$
30 RESTORE
40 READ C$, D$
50 PRINT C$;D$
60 DATA "SIM"," NÃO"
RUN
SIM NÃO
SIM NÃO

OK
```

Se não existisse a instrução RESTORE, na linha 30, quando o programa mandasse ler C$ e D$, seria acusado erro, pois os dados já teriam acabado (A$ = SIM e B$ = NÃO). Com o RESTORE, o programa entende que deve voltar para o primeiro dado, como se não tivesse havido leitura nenhuma anterior. Assim, o C$ e D$ puderam ser lidos e impressos (C$ = SIM e D$ = NÃO).

Em várias versões de BASIC é permitida a sintaxe RESTORE XX, em que XX é o número da linha onde queremos ler novamente o conjunto de DATA; nesse caso, a próxima instrução READ não irá buscar o primeiro dado

da primeira linha que contém DATA; a instrução irá ler o primeiro dado da linha de DATA indicada em XX.

Ex. 18:
```
10 READ A$,B$
20 PRINT A$;B$
30 READ C$,D$
40 PRINT C$;D$
50 RESTORE 70
60 DATA "SIM", " NÃO"
70 DATA "TALVEZ", " TALVEZ NÃO"
80 READ E$,F$
90 PRINT E$;F$
RUN
SIM NÃO
TALVEZ TALVEZ NÃO
TALVEZ TALVEZ NÃO

OK

OK
```

A instrução RESTORE 70 fez com que as variáveis da linha 80 (E$ e F$) fossem lidas da linha 70; se a instrução fosse RESTORE, aquelas variáveis seriam lidas da linha 60, que é a primeira linha de DATA.

Capítulo VII
INSTRUÇÕES DE "LOOP" E INSTRUÇÕES CONDICIONAIS

7.1 – INSTRUÇÕES DE "LOOP"

Até agora, em todos os exemplos que foram mostrados, as instruções eram executadas seqüencialmente, das linhas de menor para as de maior número.

Em outras palavras, se existia um programa com as linhas:

10 (statement)
15 (statement)
20 (statement)

a execução era feita linha por linha: primeiro a linha 10, depois a linha 15, e assim por diante. As exceções mencionadas foram:

a) começo endereçado:

RUN 15

em que a linha 10 não seria executada, pois o programa seria rodado a partir da linha 15;

b) END, no meio do programa:

10 (statement)
20 (statement)
25 END
30 (statement)
40 (statement)

nesse caso a execução seria encerrada na linha 25 e as linhas 30 e 40 não seriam executadas;

c) uma variante do caso do END seria o uso de STOP; nesse caso, porém, se fosse dado o comando CONT, o programa continuaria normalmente, na próxima linha;

d) obviamente, se houver erro em alguma linha, o programa também não executará as linhas seguintes.

Em suma, exceto no caso de começar fora de ordem e terminar fora de ordem, e as diversas possibilidades de interrupção forçada, o programa executará as linhas na ordem seqüencial.

Pelo menos, foi o que vimos até agora.

Existem instruções no BASIC que orientam o programa para uma execução fora de ordem das linhas numeradas.

Leia com atenção os próximos parágrafos, pois introduziremos conceitos importantíssimos para a execução de programas em BASIC.

7.1.1 — GOTO é uma instrução que determina que o programa "pule" para uma outra linha, diferente da próxima, e continue a execução a partir dessa nova linha.

Ex. 1:
```
10 A=10
20 B=15
30 C=A+B
40 PRINT A
50 PRINT B
60 PRINT C
```

Esse programa, ao ser executado, passará por todas as linhas e produzirá o seguinte resultado:

```
RUN
 10
 15
 25

OK
```

Agora, vamos fazer uma experiência com a introdução de GOTO. Vamos colocar uma nova linha:

```
35 GOTO 60
```

Vamos listar

```
LIST
 10 A=10
```

```
20 B=15
30 C=A+B
35 GOTO 60
40 PRINT A
50 PRINT B
60 PRINT C
```

Vejamos o que acontece quando o programa é rodado:

```
RUN
 25

OK
```

Analisemos: a instrução GOTO 60 mandou o programa pular da linha 35 para a linha 60. Assim, as linhas 40 e 50 não foram executadas e o único resultado que saiu foi o valor de C, devido a:

60 PRINT C

Com a instrução GOTO, o programa vai prosseguir na linha cujo número for indicado.

O número da nova linha, para onde o programa deve pular, não precisa ser maior do que o da linha onde está o GOTO. Em outras palavras, não é obrigatório pular apenas para a frente. A instrução GOTO pode mandar voltar para uma linha já executada, a qual será executada de novo:

Ex. 2:
```
10 A=15
20 B=A+2
30 C=A+B
40 PRINT C
50 GOTO 10
RUN
 32        (valor de C)
 32
 32
 32
 32
```

O programa não vai parar, porque entrou em "loop"; isto significa que, depois de voltar para a linha 10, o programa imprimiu novamente o valor de C (linha 40) e depois voltou novamente para a linha 10, girando sem parar. Para

interromper esse "loop" devemos teclar CTRL C (apertanto simultaneamente as teclas CRTL e C); o programa será interrompido e surgirá uma mensagem de interrupção:

BREAK IN XX

em que XX é o número da linha que seria executada no instante em que apertamos as teclas CTRL e C.

O programa pode também ser interrompido pela tecla BREAK (veja explicação em capítulo anterior).

Nota importante: é preciso tomar cuidado na hora de programar, para evitar loops infinitos. Em geral, quando é necessário interromper o loop, depois de algumas voltas, nós usamos o que se denomina contador, ou simplesmente um FLAG (sinal). Isso será explicado logo adiante, quando trataremos dos desvios condicionais, isto é, instruções que mandam o programa pular para outra linha apenas em certos casos, que nós definimos no programa. A instrução que estamos descrevendo — GOTO — é denominada incondicional, pois ela será obrigatoriamente executada quando o programa chegar na linha em que ela está.

Vejamos um outro exemplo de loop:

Ex. 3:
```
10 T=1
20 PRINT T;        (note o ponto-e-vírgula)
30 T= T+1
40 GOTO 20
RUN
 1  2  3  4  5  6  7  8  9
BREAK IN 20    (apertamos CTRL C)

OK
```

Vejamos o que aconteceu: depois da impressão do valor de T (que era igual a 1, no início), a linha 30 mandou aumentar sempre T de uma unidade; desse modo, na segunda vez que o programa passou pela linha 20, o valor de T já era 2 (depois 3, 4, etc.).

Por isso não mandamos voltar para a linha 10. Se a instrução da linha 40 fosse GOTO 10, só seria impresso o número 1. Por quê?

Cada vez que a linha 30 mandar aumentar T de uma unidade, logo em seguida o GOTO 10 mandaria o programa

para uma instrução que diz que T = 1 e, assim, T voltaria a ter esse valor; o valor impresso, então, seria sempre 1.

Voltemos ao exemplo 2:

Não era necessário mandar voltar o programa para a linha 1Ø, pois as definições.

```
1Ø A=1
2Ø B=A+2
3Ø C=A+B
```

serão sempre repetidas; como não houve alteração no valor de A, B e C, bastaria voltar para a linha

```
4Ø PRINT C
```

e o programa faria a mesma coisa: imprimiria, seguidamente, o número 32.

Em um programa podem ser usados vários GOTO, mas é preciso tomar muito cuidado pois é fácil o programador se perder nos retornos e o programa entrar em loop.

Ex. 4:
```
1Ø PRINT "VALOR DE A";
2Ø INPUT A
3Ø PRINT A
4Ø GOTO 8Ø
5Ø PRINT "VALOR DE B";
6Ø INPUT B
8Ø PRINT "VALOR DE C";
9Ø INPUT C
1ØØ PRINT C
11Ø GOTO 15Ø
12Ø PRINT "VALOR DE D";
13Ø INPUT D
14Ø PRINT D
15Ø PRINT "FIM"
```

Vamos analisar esse programa por partes:

Primeiro, vamos supor que não existem as linhas 4Ø GOTO 8Ø e 11Ø GOTO 15Ø.

O programa seria rodado da seguinte maneira:

```
RUN
VALOR DE A? 15     (teclamos 15)
 15
VALOR DE B? -4     (teclamos -4)
-4
VALOR DE C? .38    (teclamos .38)
 .38
VALOR DE D? Ø      (teclamos Ø)
 Ø
FIM

OK
```

Note o seguinte: a interrogação correspondente aos INPUTS apareceu depois dos STRINGS "VALOR DE . . ."; isto aconteceu porque colocamos um ponto-e-vírgula depois das instruções

```
PRINT "VALOR DE A";
```

esse ponto-e-vírgula trouxe a instrução INPUT A para a mesma linha onde foi impressa a frase VALOR DE A; consequentemente os números que foram teclados apareceram na mesma linha, depois da instrução.

O leitor se lembra que uma maneira diferente de obter o mesmo resultado é usar INPUT "VALOR DE A"; A.

Agora, vamos manter as linhas que têm GOTO:

```
RUN
VALOR DE A? 13     (teclamos 13)
 13
VALOR DE C? -18    (teclamos -18)
-18
FIM

OK
```

Observe que foram puladas as linhas referentes aos valores de B e D; o programa pulou para a linha 8Ø e da linha 11Ø pulou para a linha 15Ø.

Observe, também, que nós teclamos novos números para A e C. Os valores anteriores não ficaram armazenados, depois que nós demos novo RUN no programa.

7.1.2 — FOR/NEXT: essas instruções vêm sempre juntas, isto é, uma dupla de instruções que executa um loop programado, ou seja, faz o programa voltar para uma certa linha e executar as próximas instruções durante um número certo de vezes, que é definido pelo FOR.

Nada melhor que um exemplo:

Ex. 5:
```
10 FOR A=1 TO 10
20 PRINT A
30 NEXT A
RUN
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10

OK
```

Analisemos esse programa: FOR A=1TO 10 significa que, cada vez que o programa voltar para a linha 10, o valor de A passará a ser o próximo da seqüência entre 1 e 10.

Assim, na primeira passada pela linha 10, A=1; em seguida foi executada a instrução seguinte:

```
20 PRINT A
```

na linha 30 a instrução NEXT A manda o programa voltar para a linha 10 (uma instrução NEXT sempre manda voltar para a linha que tem FOR) e pegar o próximo valor de A.

Então, na segunda passada, o valor de A é 2 e a próxima instrução será novamente executada:

```
20 PRINT A
```

Quando chegamos a 30, essa linha manda pegar o próximo valor de A, e, assim por diante.

O que acontece depois que A atingiu o último valor? Quando a instrução NEXT A manda pegar o próximo valor de A e não existe próximo valor, o programa prossegue para a linha que vem logo depois da instrução NEXT.

Ex. 6:
```
10 FOR A=1 TO 5
20 ?"O VALOR DO QUADRADO DE A É =";A ↑ 2
30 ? "O VALOR DO TRIPLO DE A É =";3*A
40 NEXT A
50 PRINT "ACABOU O PROGRAMA"
RUN
O VALOR DO QUADRADO DE A É = 1
O VALOR DO TRIPLO DE A É = 3
O VALOR DO QUADRADO E A É = 4
O VALOR DO TRIPLO DE A É = 6
O VALOR DO QUADRADO DE A É = 9
O VALOR DO TRIPLO DE A É = 9
O VALOR DO QUADRADO DE A É = 16
O VALOR DO TRIPLO DE A É = 12
O VALOR DO QUADRADO DE A É = 25
O VALOR DO TRIPLO DE A É = 15
ACABOU O PROGRAMA

OK
```

Para cada valor que A foi assumindo, foram executadas as próximas instruções até o NEXT e, depois de terminados os valores de A, o programa passou para a próxima linha: 50 ? "ACABOU O PROGRAMA".

É importante notar que, enquanto não acabarem os valores da variável associada à instrução FOR, o programa não sairá desse loop, isto é, só executará aquilo que está entre as linhas do FOR e do NEXT.

Os valores da variável vão aumentando de 1 em 1, a menos que se dê uma instrução em contrário.

Se não dissermos mais nada ao computador, ele fará variar A, de 1 até 10 (ou o número que quisermos) de um em um.

Não é necessário começar em 1: a variável pode assumir valores dentro de uma faixa que nós estabelecemos:

Ex. 7:
```
10 FOR X = 3 TO 6
20 ? X
30 NEXT X
RUN
 3
 4
 5
 6

OK
```

Para fazer variações que não sejam de 1 em 1, é necessário acrescentar uma outra instrução depois do FOR; é a instrução STEP (passo), que indica o intervalo em que a variável vai pular:

Ex. 8:
```
10 FOR P=2 TO 8 STEP 2
20 ? P
30 NEXT P
RUN
 2
 4
 6
 8

OK
```

A variável P andou de 2 em 2, o que significa: o próximo P (NEXT P) representou um avanço de 2 sobre o anterior. A instrução STEP deve ser colocada na mesma linha que o FOR, sem nenhum outro sinal ou caracter separando as duas instruções.

Outra observação: usando STEP, o passo de avanço da variável não precisa ser inteiro nem positivo.

Ex. 9:
```
10 FORT =.5 TO 1.95 STEP .6
20 ? T
30 NEXT T
RUN
```

```
.5
1.1
1.7

OK
```

Apareceu nesse programa um aspecto bastante importante: o último valor de T (1.95) não foi usado, pois o NEXT T, quando somou .6 ao anterior, que era 1.7, chegou a 2.3; esse valor está fora da faixa estabelecida, porque é maior que 1.95. Assim o loop acabou em 1.7, que é o maior valor possível dentro da faixa (de .5 a 1.95).

Quando o passo é negativo, o primeiro valor da variável deve ser maior que o último e, assim, a variável vai diminuindo de valor, em cada passada.

Ex. 10:
```
10 FOR Q=10 TO 5 STEP -1
20 ? 2 * Q
30 NEXT Q
40 ? "NÃO TEM MAIS Q"
RUN
 20
 18
 16
 14
 12
 10
NÃO TEM MAIS Q

OK
```

Os valores negativos também podem ser fracionários:

Ex. 11:
```
10 FOR A=2 TO -1 STEP -.5
20 ? A
30 NEXT A
RUN
 2
 1.5
 1
 .5
```

```
 Ø
-.5
-1

OK
```

A faixa da variável também pode atingir valores negativos, como vimos neste último exemplo (A = 2 TO -1).

Uma outra observação importante é de que podem ser usadas instruções múltiplas para FOR e NEXT (como, aliás, para qualquer outra instrução).

Ex. 12:
```
10 FOR T =1 TO 6: ? T: NEXTT
RUN
 1
 2
 3
 4
 5
 6

OK
```

O programa passou a ter apenas uma linha.

Essa mesma linha pode ser usada em modo direto, isto é, sem número:

Ex. 13:
```
FOR T= 1 TO 6: ? T: NEXT T
          (<RETURN>)
   1
   2
   3
   4
   5
   6

OK
```

Ex. 14:
```
10 READ A,B
20 FOR N= 1 TO 3: A= A+1: B=B+1
30 PRINT A;B
```

```
40 C= A+B
50 ? "O VALOR DE C É"; C
60 NEXT N
70 DATA 3,-4
RUN
 4  -3
O VALOR DE C É 1
 5  -2
O VALOR DE C É 3
 6  -1
O VALOR DE C É 5

OK
```

Explicação: primeiro o programa leu os valores de A e B (3 e -4); em seguida entrou no loop FOR/NEXT, executando as instruções contidas entre essas instruções:

- aumentou A e B de uma unidade;
- imprimiu o valor de A e B;
- calculou o valor de C;
- imprimiu a frase com o valor de C;
- voltou para a linha 20, isto é, aumentou novamente A e B, e assim por diante;
- esse loop foi executado 3 vezes, pois a instrução era FOR N= 1 TO 3.

Agora, vamos supor que o programa estivesse em uma forma diferente:

Ex. 15:
```
10 FOR N= 1 TO 5: READ A,B
20 PRINT A;B
30 NEXT N
40 DATA 5, 12, -1, .5, 2, -2, -.04, 0, 11
50 DATA -2, -6
RUN
 5  12
-1  .5
 2  -.04
 0  11
-2  -6

OK
```

O que aconteceu? Simplesmente, o loop FOR/NEXT ficou antes do READ, o que fez com que a instrução READ fosse executada 5 vezes. Conforme ja mostramos, cada instrução READ vai buscar o próximo DATA e, assim, os valores de A e B foram sendo lidos e impressos, dentro do conjunto de DATA, cada vez que o loop FOR/NEXT deu uma volta (5 vezes).

Os valores associados à instrução FOR podem ser variáveis numéricas, ***desde que o seu valor seja definido no programa*** antes da instrução FOR.

Ex. 16:
```
10 A=5 : B=10
20 FOR C=A TO B : ? " C= " ; C : NEXT C
RUN
C=5
C=6
C=7
C=8
C=9
C=10

OK
```

Ex. 17: No programa anterior vamos substituir a linha 10 pela seguinte:

```
10 INPUT " A= "; A: INPUT " B= "; B
```

e colocar uma nova linha

```
30 ? : GOTO 10
```

Vamos listar:

```
LIST
 10 INPUT " A= "; A: INPUT " B= "; B
 20 FOR C= ATO B : PRINT "C= "; C: NEXT C
 30 PRINT: GOTO 10

OK
```

e executar

```
RUN
A=? 3
B=? 7
C= 3
C= 4
C= 5
C= 6
C= 7 ┌(essa linha foi pulada por causa do
  ←┘  PRINT da linha 3Ø)
A=?
```

Nesse ponto podemos entrar com novos dados ou simplesmente teclar RETURN e interromper o programa.

Os valores inicial e final da variável do loop FOR/NEXT podem ser definidos de qualquer maneira, usando, inclusive, expressões matemáticas; o importante é que o primeiro e o último valor, assim como o STEP, estejam claramente definidos antes da instrução FOR ser executada.

Ex. 18:

```
1Ø A=2 : B=3 : C=4
2Ø FOR N= A*B TO B+C : ? N : NEXT
RUN
 6
 7

OK
```

Outras observações importantes sobre as instruções FOR/NEXT:

- o uso de CTRL C pode interromper o loop no meio (se percebermos, por exemplo, que está havendo um erro);
- o loop pode ser usado para dar um intervalo de tempo no programa; para isto basta fazer o seguinte:

```
FOR N= 1 TO 5ØØ : NEXT N
```

nada acontecerá enquanto o computador não "contar" de 1 a 5ØØ; o valor dos intervalos de tempo depende da velocidade do computador; cada leitor poderá fazer experiências para verificar quantas unida-

des são necessárias para o computador esperar, p. ex., um segundo;

— não é necessário colocar (na maioria das versões mais completas de BASIC) o nome da variável, depois do NEXT (*exceto quando há mais de um loop sendo executado no programa, podendo trazer confusão*)

Ex.: **FOR P= 1TO 5 : ? P : NEXT**

(não escreveremos NEXT P; o computador já sabe que esse NEXT se refere ao P).

7.1.3 — LOOPS EMBUTIDOS (Nested Loops)

Os loops FOR/NEXT podem ser colocados uns dentro dos outros, formando os "loops embutidos".

Ex. 19:
```
10 FOR A= 1 TO 5
20 ? "SÉRIE NÚMERO"; A
25 B= B+4
30 FOR C= B TO B+3: PRINT C
40 NEXT C
50 NEXT A
60 ? "ACABOU"
RUN
SÉRIE NÚMERO 1
 4
 5
 6
 7
SÉRIE NÚMERO 2
 8
 9
 10
 11
SÉRIE NÚMERO 3
 12
 13
 14
 15
```

O que aconteceu? Simplesmente, o loop FOR/NEXT ficou antes do READ, o que fez com que a instrução READ fosse executada 5 vezes. Conforme ja mostramos, cada instrução READ vai buscar o próximo DATA e, assim, os valores de A e B foram sendo lidos e impressos, dentro do conjunto de DATA, cada vez que o loop FOR/NEXT deu uma volta (5 vezes).

Os valores associados à instrução FOR podem ser variáveis numéricas, ***desde que o seu valor seja definido no programa*** antes da instrução FOR.

Ex. 16:
```
10 A=5 : B=10
20 FOR C=A TO B : ? " C= " ; C : NEXT C
RUN
C=5
C=6
C=7
C=8
C=9
C=10

OK
```

Ex. 17: No programa anterior vamos substituir a linha 10 pela seguinte:

```
10 INPUT " A= "; A: INPUT " B= "; B
```

e colocar uma nova linha

```
30 ? : GOTO 10
```

Vamos listar:

```
LIST
 10 INPUT " A= "; A: INPUT " B= "; B
 20 FOR C= ATO B : PRINT "C= "; C: NEXT C
 30 PRINT: GOTO 10

OK
```

e executar

```
RUN
A=? 3
B=? 7
C= 3
C= 4
C= 5
C= 6
C= 7
A=?
```

(essa linha foi pulada por causa do PRINT da linha 3Ø)

Nesse ponto podemos entrar com novos dados ou simplesmente teclar RETURN e interromper o programa.

Os valores inicial e final da variável do loop FOR/NEXT podem ser definidos de qualquer maneira, usando, inclusive, expressões matemáticas; o importante é que o primeiro e o último valor, assim como o STEP, estejam claramente definidos antes da instrução FOR ser executada.

Ex. 18:

```
1Ø A=2 : B=3 : C=4
2Ø FOR N= A*B TO B+C : ? N : NEXT
RUN
 6
 7

OK
```

Outras observações importantes sobre as instruções FOR/NEXT:

- o uso de CTRL C pode interromper o loop no meio (se percebermos, por exemplo, que está havendo um erro);
- o loop pode ser usado para dar um intervalo de tempo no programa; para isto basta fazer o seguinte:

```
FOR N= 1 TO 5ØØ : NEXT N
```

nada acontecerá enquanto o computador não "contar" de 1 a 5ØØ; o valor dos intervalos de tempo depende da velocidade do computador; cada leitor poderá fazer experiências para verificar quantas unida-

des são necessárias para o computador esperar, p. ex., um segundo;
- não é necessário colocar (na maioria das versões mais completas de BASIC) o nome da variável, depois do NEXT (*exceto quando há mais de um loop sendo executado no programa, podendo trazer confusão*)

Ex.: **FOR P= 1TO 5 : ? P : NEXT**

(não escreveremos NEXT P; o computador já sabe que esse NEXT se refere ao P).

7.1.3 – LOOPS EMBUTIDOS (Nested Loops)

Os loops FOR/NEXT podem ser colocados uns dentro dos outros, formando os "loops embutidos".

Ex. 19:
```
10 FOR A= 1 TO 5
20 ? "SÉRIE NÚMERO"; A
25 B= B+4
30 FOR C= B TO B+3: PRINT C
40 NEXT C
50 NEXT A
60 ? "ACABOU"
RUN
SÉRIE NÚMERO 1
 4
 5
 6
 7
SÉRIE NÚMERO 2
 8
 9
 10
 11
SÉRIE NÚMERO 3
 12
 13
 14
 15
```

```
SÉRIE NÚMERO 4
 16
 17
 18
 19
SÉRIE NÚMERO 5
 20
 21
 22
 23
ACABOU

OK
```

Explicação: Vamos acompanhar a execução do programa, passo a passo:

a) o programa começou fazendo A = 1;
b) em seguida imprimiu o número da série (A);
c) na linha 25 ficou estabelecido que B = 4 (na primeira vez), pois B era igual a zero e ficou igual a zero + 4;
d) na próxima linha o programa entrou no outro loop FOR/NEXT "embutido" dentro do primeiro; esse loop embutido será executado integralmente, antes de se chegar ao NEXT do primeiro loop (NEXT A); por isso, ***NEXT C vem antes de NEXT A***;
e) a execução do loop embutido faz imprimir os números que vão de B até B+3 (ou seja, na primeira série, de 4 até 7);
f) quando acaba o loop interno (loop da variável C), o programa passa para a linha NEXT A e volta para a linha 1Ø; dessa vez A = 2;
g) será impresso o número da série 2 (instrução da linha 2Ø);
h) B aumentará de 4, passando para 8;
i) entramos novamente no loop C; dessa vez variando de 8 a 11; esse loop imprimirá os números e, depois de terminado, passará para NEXT A;
j) o processo recomeça e vai até que os valores de A tenham terminado (A = 5);
k) depois de terminado o loop A, o programa passará para a próxima instrução (linha 6Ø) e a executará.

É muito importante entender que os loops embutidos devem ter os FOR e os NEXT colocados de maneira que a execução comece pelos loops mais internos e vá passando para os mais externos. É obvio que, nesses casos, não se pode omitir a variável depois da palavra NEXT.

Ex. 20:
```
10 FOR A= 1 TO 5: PRINT A
20 FOR B= 1 TO 5: PRINT B
30 FOR C= 1 TO 5: PRINT C
40 NEXT C
50 NEXT B
60 NEXT A
```
(veja a ordem das variáveis, de dentro para fora)

Nesse programa, o loop C será executado dentro do loop B, o qual será executado dentro do loop A.

Se houver engano na ordem o computador acusará erro, indicando que encontrou um NEXT sem o respectivo FOR.

Para completar o entendimento do uso do loop FOR/NEXT, vamos dar o exemplo da formação de uma tabela de quadrados e raízes quadradas de uma série de números.

Ex. 21:
```
10 READ A$, B$, C$
20 PRINT A$; TAB(8); B$ ; TAB(18); C$
30 FOR N= 1 TO 10
40 ? TAB(1);N ; TAB(8); N↑2; TAB(17); SQR(N)
50 NEXT N
60 DATA NÚMERO, QUADRADO, "R. QUAD."
RUN
NÚMERO    QUADRADO    R. QUAD.
  1           1         1
  2           4         1.41421
  3           9         1.73205
  4           16        2
  5           25        2.23607
  6           36        2.44949
  7           49        2.64575
  8           64        2.82848
  9           81        3
  10          100       3.16228

OK
```

Usamos as instruções READ/DATA para imprimir os títulos das colunas e o FOR/NEXT para calcular e imprimir (tabuladamente) os valores de N e do quadrado e raiz quadrada de N.

Daremos novos exemplos mais adiante. As instruções FOR/NEXT são ricas em aplicações de programação.

7.2 – INSTRUÇÕES CONDICIONAIS

As instruções de condições são aquelas que mandam o computador executar determinada tarefa ***no caso de alguma condição ser satisfeita***. Essa condição é definida no programa e, toda a vez que ela é satisfeita o programa executará determinada instrução. Caso a condição não seja satisfeita, o programa continuará normalmente, na linha seguinte.

7.2.1 – IF ---- THEN

Esse par de instruções, que é empregado sempre em conjunto, funciona da seguinte maneira:

IF (SE) acontecer determinada condição no programa, THEN (ENTÃO) execute a seguinte instrução (segue-se uma nova instrução).

Ex. 22:
```
10 PRINT "ESCOLHA UM NÚMERO";
20 INPUT A
30 IF A=5 THEN PRINT "ACERTOU": GOTO 50
35 PRINT "ERROU"
40 GOTO 10
50 END
RUN
ESCOLHA UM NÚMERO ? 3
ERROU
ESCOLHA UM NÚMERO ? 2
ERROU
ESCOLHA UM NÚMERO ? 5
ACERTOU

OK
```

Analisemos o que aconteceu: o programa pediu um número (linhas 1Ø e 2Ø). A variável A tomou esse número, isto é, ficou sendo igual ao número que o operador teclou.

Na linha 3Ø a instrução diz que se A for igual a 5 (IF A = 5), o programa deve mandar imprimir ACERTOU (THEN PRINT "ACERTOU"); além disso, o programa deve pular para a linha 5Ø, que termina a execução (END). Mas, se A não for igual a 5, a linha 3Ø não é executada e o programa prossegue para a linha 35, onde se manda imprimir ERROU. Na linha 4Ø o programa volta para a linha 1Ø (GOTO 1Ø) e pede novo número. O programa ficará pedindo novos números até o operador teclar 5, que é o número certo, para esse programa.

Na maioria dos interpretadores e compiladores BASIC, quando houver mais de uma instrução por linha, o programa só executará as instruções seguintes ao IF no caso de a condição estabelecida ser satisfeita. No exemplo acima, na linha 3Ø havia uma instrução separada do conjunto IF---- THEN por dois pontos (GOTO 5Ø). No exemplo que nós explicamos, essa instrução, apesar de separada do IF ----THEN, também ficou condicionada ao atendimento da condição A = 5.

Em alguns tipos de BASIC, no entanto, essa instrução ficaria realmente separada do IF ---- THEN, e seria executada sempre, mesmo que A não fosse igual a 5.

É necessário que o leitor verifique como funciona o BASIC com que estiver trabalhando.

Para os efeitos deste manual consideraremos que tudo o que vier depois do IF ---- THEN está sujeito à mesma condição, isto é, só será executado se o IF for satisfeito.

Ex. 23:
```
1Ø INPUT A
2Ø IF A= -4 THEN GOTO 4Ø
3Ø GOTO 1Ø
4Ø PRINT "ACABOU"
```

Nesse exemplo, quando o operador entrar com o número -4, o programa vai pular da linha 2Ø para a linha 4Ø; imprimirá ACABOU e terminará a execução. Enquanto A não foi igual a -4 o programa ficará indo da linha 3Ø para a linha 1Ø e ficará pedindo novos valores de A.

Nas versões mais recentes de BASIC não é necessário, no caso do exemplo acima, escrever THEN GOTO 4Ø, basta escrever THEN 4Ø. O programa já sabe que se trata de um desvio para a linha 4Ø.

As condições em que o IF é usado são as seguintes:

— considerando duas variáveis, A e B, podemos usar o IF nos seguintes casos:

IF	A < B	(A menor que B)
	A > B	(A maior que B)
	A = B	(A igual a B)
	A <= B	(A menor ou igual a B)
	A >= B	(A maior ou igual a B)
	A <> B	(A diferente de B)

Por mais complexas que sejam as condições estabelecidas em um programa, elas acabam caindo em uma das categorias acima (veremos diversos exemplos a seguir).

Um outro uso muito comum para o IF ---- THEN é o uso do comando PRINT:

```
IF A > B THEN PRINT "XXX"
```

Se essa condição não for satisfeita, o computador não imprimirá nada.

Ex. 24:
```
1Ø READ X
2Ø IF X < Ø THEN PRINT "VERDADEIRO:";X;
"<Ø ": GOTO 1Ø
3Ø PRINT "FALSO: "; X; "NÃO É <Ø ": GOTO 1Ø
4Ø DATA 5, Ø, 2 , -6, -2, 1
RUN
FALSO: 5 NÃO É <Ø
FALSO: Ø NÃO É <Ø
FALSO: 2 NÃO É <Ø
VERDADEIRO: -6 <Ø
VERDADEIRO: -2 <Ø
FALSO: 1 NÃO É <Ø
ERROR IN 1Ø

OK
```

Observações sobre o exemplo acima:

— inicialmente o programa atribuiu a X o primeiro valor do DATA (X = 5);

— como X não era menor do que zero, todo o conteúdo da

linha 2Ø foi abandonado e o programa passou nha 3Ø;
- imprimiu o que estava determinado e voltou para 1Ø;
- leu um novo valor para X; agora X = Ø (próximo valc linha 4Ø DATA);
- passou para a linha 2Ø e, já que X não era menor do que Ø, passou para linha 3Ø, imprimindo e voltando para 1Ø·
- foi lido um novo valor para X (X = 2), que era o próx mo do conjunto de DATA;
- na linha 2Ø X ainda não era menor do que Ø; passou para linha 3Ø, imprimiu e voltou para a linha 1Ø;
- novo valor para X; dessa vez X = -6;
- na linha 2Ø, ao verificar que X é menor que Ø, o programa mandou imprimir o que vinha depois do THEN e pulou para a linha 1Ø;
- foi lido novo valor de X (X = -2);
- a linha 2Ø funcionou, pois X era menor que Ø;
- novo valor de X (X = 1);
- na linha 2Ø, como X não era menor do que Ø, o programa passou direto para a linha 3Ø;
- na linha 3Ø, imprimiu o que estava determinado e voltou para a linha 1Ø;
- nesse momento, a linha 1Ø mandou ler o próximo valor para X, mas já acabaram os valores da linha 4Ø (DATA); assim, foi impressa uma mensagem de erro, indicando "mais READ's do que DATA's".

Um outro exemplo, variante do primeiro, seria mandar imprimir os números positivos.

Ex. 25:
```
10 READ X
20 IF X > 0 THEN PRINT X
30 GOTO 10
40 DATA 2,-5,0,3.5,-.5,.4
RUN
 2
 3.5
 .4
ERROR IN 10

OK
```

ı vez que o número não era maior do que Ø, o passava para a linha 3Ø, voltava para a linha 1Ø e valor para X (nesse caso não encontrava nenhuma ɔ PRINT, pois o PRINT está associado a X > Ø). ı mensagem de erro quando acabaram os dados.

Vejamos um programa um pouco mais complicado:

Ex. 26:
```
5  N= N+1:  IF N>8 THEN 6Ø
1Ø INPUT "VALOR DE B";B
2Ø INPUT "VALOR DE D";D
3Ø A= 3 ↑ B + 2* B : C = SQR (D) + D
4Ø IF A>C THEN PRINT " B É MELHOR" : GOTO 5
5Ø PRINT "D É MELHOR" : GOTO 5
6Ø PRINT "ACABOU O JOGO"
RUN
VALOR DE B ? 1
VALOR DE D ? 2
B É MELHOR
VALOR DE B ? 2
VALOR DE D ? 9
B É MELHOR
VALOR DE B ? 5
VALOR DE D ? 7
B É MELHOR
VALOR DE B ? .9
VALOR DE D ? 8
D É MELHOR
VALOR DE B ? -1
VALOR DE D ? .8
D É MELHOR
VALOR DE B ? -5
VALOR DE D ? 5
D É MELHOR
VALOR DE B ? .5
VALOR DE D ? 4
D É MELHOR
VALOR DE B ? 3
VALOR DE D ? 8
B É MELHOR
ACABOU O JOGO

OK
```

A explicação do programa é simples:
- cada vez que o programa volta para a linha 5 vai "contar" uma passada, isto é, N aumentará de uma unidade;
- na mesma linha dizemos que, quando N for maior que 8, isto é, depois de entrarmos 8 vezes com valores para B e D, o programa pula para a linha 60, imprime ACABOU O JOGO e termina;
- nas linhas 10 e 20 o programa pergunta os valores de B e D, que serão teclados pelo operador;
- na linha 30 o programa efetua duas contas: uma que define um valor de A, em função de B, e outra que define um valor de C em função de D;
- na linha 40, após calcular os valores de A e C, o programa determina que, se A for maior que C, deverá ser impresso a frase: B É MELHOR e o programa deve voltar para a linha 5 contando mais uma passada e perguntando novos valores de B e D;
- se A não for maior que C, o programa passa para a linha 50, que manda imprimir D É MELHOR e volta para o início (pedindo novos valores de B e D).

Nota importante: o contador que foi usado no programa do exemplo 26 para terminar depois de um certo número de passagens, pode ser usado para evitar as mensagens de erro que apareceram nos exemplos 24 e 25.

Ex. 27:
```
10 READ X
20 N=N+1 : IF  N>5 THEN END
30 IF X>0 THEN PRINT X
35 GOTO 10
40 DATA 2, -5, 0, 3.5, .4, -.5
RUN
 2
 3.5
 .4

OK
```

Essa nova versão do exemplo 25 não deu mensagem de erro porque, quando foi lido o último dado (o sexto dado, -.5), N passou a ser igual a 6 e, logo, maior do que 5, o que fez terminar o programa na linha 20 (THEN END).

Mesmo se colocarmos mais dados positivos na linha 40, o resultado será o mesmo pois o contador terminará o programa quando N>5.

Ex. 28: Colocar a nova linha 40

```
40 DATA 2, -5, 0, 3.5, .4, -4, -.5, 3, 4, -1
RUN
 2
 3.5
 .4

OK
```

Agora, se tirarmos o "contador", os valores positivos 3 e 4 que nós incluímos na linha 40 serão lidos e impressos, mas, nesse caso, aparecerá a mensagem de erro, indicando "mais READS do que DATA", depois que for lido o último dado (a linha 35 mandará voltar para a linha 10, para ler mais um valor de X, mesmo depois que os valores acabarem). Suponhamos eliminada a linha 20:

```
RUN
 2
 3.5
 .4
 3
 4
ERROR IN 10

OK
```

As comparações ou condições que influem nas instruções IF/THEN não são apenas numéricas. Também podem ser feitas comparações de STRINGS.

Vamos modificar um pouco o programa do exemplo 22:

Ex. 29:

```
10 PRINT "ESCOLHA UM NÚMERO";
20 INPUT A
30 IF A=5 THEN PRINT "ACERTOU": GOTO 50
40 PRINT "ERROU"
50 INPUT "QUER CONTINUAR";K$
60 IF K$ = "SIM" THEN 10
RUN
```

```
ESCOLHA UM NÚMERO ? 3          (1)
ERROU
QUER CONTINUAR ? SIM           (2)
ESCOLHA UM NÚMERO ? 5
ACERTOU
QUER CONTINUAR ? NÃO

OK
```

(1) teclamos o número 3;
(2) teclamos a palavra SIM, *sem aspas.*

Observações sobre o exemplo acima:

— quando queremos que o INPUT seja uma palavra, letra ou frase, a variável associada deve ter o cifrão; no caso acima definimos que a resposta será um STRING denominado K$;
— podemos comparar o STRING K$ com outro STRING, usando as alternativas >, <, >=, <=, <>, =; essa comparação será feita pela verificação da ordem alfabética da primeira letra do STRING (se a primeira letra do STRING K$ for, por ordem alfabética, menor que a primeira letra do outro STRING, o STRING K$ será considerado menor que o outro); se as primeiras letras forem iguais, a comparação se fará na próxima e, assim por diante; se todas as letras forem iguais até um certo ponto, mas um dos STRINGS tiver mais caracteres, o que tiver mais caracteres será considerado maior;
— quando queremos comparar uma variável de STRING com um STRING, este deve estar contido entre aspas (lembre-se do caso do comando LET: LET A$= "ANDRÉ");
— quando o operador vai teclar um STRING, como resposta à pergunta do computador, esse STRING não leva aspas (reveja o capítulo sobre a instrução INPUT);
— o programa comparou K$ com o que o operador teclou; se for o mesmo STRING (K$= "SIM"), o programa executará o que estiver depois do THEN (no exemplo acima, volta para a linha 1Ø, perguntando novo número);
— se K$ não for igual a SIM (isto é, se for teclado qualquer outra palavra, frase, letra ou caracter), o computador passa para a próxima linha (como não há, no exemplo

acima, nenhuma outra linha, o programa simplesmente acaba).

É claro que o programa mostrado poderia ser feito de várias outras maneiras:

Poderíamos, na linha 6Ø, colocar:

```
6Ø IF K$ = "S" THEN 1Ø
```

Nesse caso, para continuar o programa bastaria teclar a letra S, em vez da palavra SIM.

Poderíamos, também, fazer por desigualdade:

```
6Ø IF K$ <> "N" THEN 1Ø
```

Nesse caso, batendo qualquer tecla, diferente de N, o jogo iria continuar, com o programa voltando para a linha 1Ø. Se fosse teclado N, o programa terminaria.

A comparação de variáveis de STRING pode ser útil no caso das instruções READ/DATA.

Vamos adaptar um pouco o programa do exemplo 24:

Ex. 30:
```
1Ø READ X$, Y
2Ø IF X$ = "FIM" THEN PRINT "ACABOU": END
3Ø IF Y < Ø THEN PRINT "VERDADEIRO:";
X$;Y;"É<Ø":GOTO 1Ø
4Ø PRINT "FALSO:"; X$; Y; "NÃO É<Ø" : GOTO 1Ø
5Ø DATA " O NÚMERO", 5, " O NÚMERO ",
Ø, " O NÚMERO", -6, " O NÚMERO",
-.2, " O NÚMERO", 1, FIM,Ø
```

Antes de rodar o programa, observe o seguinte:

— dessa vez vamos ler duas variáveis de cada vez: uma variável de STRING e um número;
— vamos mandar imprimir as duas variáveis (linhas 3Ø e 4Ø).
— os STRINGS " O NÚMERO" estão entre aspas por causa do espaço inicial, que será necessário na hora da impressão; o STRING FIM não precisa de aspas;
— o zero foi colocado depois do FIM para poder dar um valor a Y, já que X$ e Y são lidos de uma vez e, se parássemos no FIM, o programa acusaria falta de dados;
— quando o READ X$ chegar à palavra FIM, a instrução

da linha 2Ø mandará imprimir ACABOU e terminará o programa; nesse caso é necessário colocar o END, pois, se isso não for feito, o programa prosseguirá para a linha 3Ø, depois de imprimir ACABOU; é um dos exemplos de caso em que se necessita colocar a instrução END no meio do programa. Essa palavra, FIM, é um tipo de FLAG.

Agora, vamos executar o programa:

```
RUN
FALSO: O NÚMERO 5 NÃO É < Ø
FALSO: O NÚMERO Ø NÃO É < Ø
VERDADEIRO: O NÚMERO-6 É < Ø
VERDADEIRO: O NÚMERO-2. É < Ø
FALSO: O NÚMERO 1 NÃO É < Ø
ACABOU

OK
```

Observações:

— veja que o espaço colocado nos STRINGS " O NÚMERO" serviu para separar as palavras FALSO: e VERDADEIRO: das palavras O NÚMERO;

— como não colocamos nenhum espaço depois da palavra NÚMERO, os números negativos ficaram "grudados" na palavra NÚMERO; isso não aconteceu com os números positivos que têm sempre um espaço na frente (representando o sinal de +, que não aparece).

Até aqui vimos casos em que havia uma instrução IF----- THEN em uma linha, e se a condição estabelecida não fosse satisfeita, o programa passaria para a próxima linha. Isto significa: IF (certa coisa acontecer) THEN (faça determinada coisa). Se não acontecer, apenas passe para a próxima linha.

Acontece que, em alguns casos não existe apenas a condição e o seu oposto (ou seja, "acontece" ou "não acontece").

Para exemplificar: queremos saber se um número é maior que zero, igual a zero ou menor que zero (são 3 condições e não apenas 2).

Nesse caso, teremos que usar as instruções IF- - - - THEN repetidamente. Em outras palavras, primeiro o programa verifica se a primeira condição foi satisfeita; se não foi, passa para a próxima linha, onde irá encontrar uma nova instrução IF- - - -THEN para verificar a segunda condição; se a segunda condição não for satisfeita, o programa passa para a linha seguinte.

Teoricamente podemos colocar vários IF- - - -THEN em linhas sucessivas, para verificar uma série grande de condições. Na prática isso se torna um pouco complicado e exige muita atenção do programador para não entrar em condições contraditórias.

Ex. 31:
```
10 INPUT "ESCOLHA UM NÚMERO"; N
20 IF N<0 THEN PRINT "O NÚMERO ";
N; "É<0 ": GOTO 10
30 IF N=0 THEN PRINT "O NÚMERO ";
N; "É=0 "; GOTO 10
40 PRINT "O NÚMERO "; N; "É>0":
GOTO 10
```

Vejamos o que acontece:

```
RUN
ESCOLHA UM NÚMERO ? 2
O NÚMERO  2 É > 0
ESCOLHA UM NÚMERO ? -5
O NÚMERO -5 É < 0
ESCOLHA UM NÚMERO ? 0
O NÚMERO  0 É = 0
ESCOLHA UM NÚMERO ?
```

(nesse momento vamos teclar <RETURN> e interromper o programa)

Uma rápida análise mostra o que aconteceu:

- quando entramos com o número 2, N passou a ser igual a 2;
- na linha 20 o programa verificou que N não era menor que zero e passou para a linha 30;
- na linha 30 o programa verificou que N não era igual a zero e passou para a linha 40;

— a linha 4Ø mandou imprimir o NÚMERO 2 É > Ø e voltou para a linha 1Ø;
— na linha 1Ø foi pedido um novo número; entramos com o número -5;
— na linha 2Ø o programa verificou que N<Ø e então (THEN) mandou imprimir O NÚMERO -5 É < Ø e voltar para a linha 1Ø;
— com a nova pergunta, escolhemos o número Ø;
— na linha 2Ø o programa passou direto, pois N não era menor que Ø;
— na linha 3Ø, a condição foi satisfeita (N=Ø); foi impresso Ø É = 0 e voltou para a linha 1Ø;
— quando foi pedido novo número, nós teclamos <RETURN> e interrompemos o programa.

7.2.2 — AND, OR e NOT

Essas instruções servem para fazer comparações múltiplas (se tal coisa acontecer ***E (AND)*** tal coisa acontecer ***OU (OR)*** tal acontecer THEN faça aquilo).

Em muitos casos, já vimos que gostaríamos de comparar diversas coisas simultaneamente, para, só então, chegar a uma conclusão.

Ex. 32:
```
10 A=3
20 INPUT "VALOR DE B"; B
30 INPUT "VALOR DE C"; C
40 IF B=A AND C= A THEN PRINT
"OS TRÊS NÚMEROS SÃO IGUAIS": END
50 PRINT "OS TRÊS NÚMEROS NÃO
SÃO IGUAIS"
RUN
VALOR DE B? 3   (teclamos 3)
VALOR DE C? 4   (teclamos 4)
OS TRÊS NÚMEROS NÃO SÃO IGUAIS

OK
```

O que aconteceu: B=3 (o primeiro INPUT) era igual a A, mas C=4 não era igual a A. Assim a condição dupla não foi satisfeita e o programa passou para a linha 5Ø.

A condição AND pode ser usada mais vezes:

Ex. 33:
```
10 INPUT "QUER JOGAR COMIGO"; S$
20 IF S$<>"S" THEN END
30 INPUT "DIGA UM NÚMERO";N
40 INPUT "DIGA OUTRO NÚMERO" ; M
50 INPUT "DIGA OUTRO NÚMERO"; Q
60 INPUT "QUER VÊ-LOS EM ORDEM CRESCENTE";
A$
70 IF N<M AND M<Q AND A$="S"THEN PRINT N;
M; Q
80 IF N<M AND Q<M AND N<Q AND A$="S"
THEN PRINT N; Q; M
90 IF N<M AND N>Q AND A$= "S" THEN
PRINT Q; N; M
100 IF N>M AND M>Q AND A$="S" THEN
PRINT Q; M; N
110 IF N<M AND M<Q AND Q>N AND A$="S"
THEN PRINT M; N; Q
120 IF N>M AND N>Q AND Q>M AND A$="S"
THEN PRINT M; Q; N
RUN
QUER JOGAR COMIGO? S              (teclamos S)
DIGA UM NÚMERO? 25                (teclamos 25)
DIGA OUTRO NÚMERO? 38             (teclamos 38)
DIGA OUTRO NÚMERO? 13             (teclamos 13)
QUER VÊ-LOS EM ORDEM CRESCENTE ? S  (teclamos S)

 13  25  38

OK
```

Verifica-se que o programa executou várias comparações simultâneas, inclusive com um STRING (A$). Em qualquer caso, quando uma das condições simultâneas não é satisfeita, o programa passa para a próxima linha.

No caso do OR, as condições são alternativas: se acontecer isto ***OU (OR)*** aquilo, então faça tal coisa.

Ex. 34:
```
10 INPUT "DIGA SEU NOME"; A$
20 IF A$="JOÃO" OR A$="MANOEL" OR A$=
"ANTÔNIO"THEN PRINT "SEU NOME É";A$:END
30 PRINT "NÃO ENTENDI SEU NOME—FAVOR
```

```
REPETIR";:INPUT A$
40 GOTO 20
RUN
DIGA SEU NOME? JOAQUIM          (foi teclado esse
                                        nome)
NÃO ENTENDI SEU NOME - FAVOR REPETIR?
JOÃO    (teclamos JOÃO)
SEU NOME É JOÃO

OK
```

Na linha 20 colocamos uma condição na qual o programa reconhecerá os nomes JOÃO ***OU (OR)*** MANOEL ***OU (OR)*** ANTÔNIO. Se nenhum deles foi teclado o programa passa para a linha 30, pergunta de novo e volta para a linha 20 (comparação).

Vamos, agora, fazer um exemplo interessante em que aparece a comparação de STRINGS por sua ordem alfabética.

Vamos escolher três palavras e mandar o programa colocá-las em ordem alfabética (tipo dicionário).

Esse exemplo é uma variante do exemplo 33.

Ex. 35:
```
10 INPUT "DIGA TRÊS PALAVRAS SEPARADAS
POR VÍRGULAS "; A$, B$, C$
20 ? "A ORDEM ALFABÉTICA É A SEGUINTE"
30 IF A$<B$ AND B$<C$ THEN PRINT
A$: PRINT B$: PRINT C$: GOTO 10
40 IF A$<B$ AND C$<B$ AND A$<C$
THEN PRINT A$: PRINT C$: PRINT B$: GOTO 10
50 IF A$<B$ AND A$>C$ THEN
PRINT C$: ? A$: ? B$: GOTO 10
60 IF A$>B$ AND B$>C$ THEN
PRINT C$: PRINT B$ : ? A$: GOTO 10
70 IF A$>B$ AND B$<C$ AND
C$>A$ THEN ? B$: ? A$: ? C$: GOTO 10
80 PRINT B$: PRINT C$: PRINT A$:
GOTO 10
RUN
```

```
DIGA TRÊS PALAVRAS SEPARADAS POR
VÍRGULAS? TREVO, GATO, POSTE
A ORDEM ALFABÉTICA É A SEGUINTE
GATO
POSTE
TREVO
DIGA TRÊS PALAVRAS SEPARADAS POR VÍRGU-
LAS?
```

Nesse ponto podemos continuar, tentando várias palavras, ou teclar <RETURN>, interrompendo o programa.

A instrução NOT equivale a "não é igual a"; ela é usada da seguinte maneira, em conjunto com IF e THEN:

```
IF NOT B>A THEN 7
```

O significado dessa instrução é o seguinte:
se B não for maior que A vá para a linha 7 (é o mesmo que B<=A).

Ex. 36:
```
10 INPUT "DIGA UM NÚMERO"; A
20 IF NOT A=0 THEN PRINT "A É DIFERENTE DE
ZERO"
30 PRINT "A É IGUAL A ZERO"
```

Se A for diferente de zero, a instrução condicional da linha 20 será satisfeita.

7.2.3 — Antes de prosseguir na descrição das instruções da linguagem BASIC devemos mencionar que existe uma que não tem a finalidade de executar nenhuma função no programa. É a instrução REM, oriunda da palavra "remark", ou seja, comentário.

A instrução REM serve para se colocar notas e informações no programa, para facilitar a sua compreensão. Quando o programa for escrito, impresso ou listado no vídeo, as informações contidas junto à instrução REM poderão ser de grande valor para a correção de erros ou, mesmo, para relembrar ao autor do programa a razão de ser de certas instruções (lembre-se que, após algum tempo, os programas mais longos ficam difíceis de ser entendidos até mesmo pelo autor).

A instrução REM é usada da seguinte maneira:

```
20 REM – PROGRAMA PARA CÁLCULO
DA ÁREA DE UM CÍRCULO
```

A instrução é colocada em uma linha numerada, no local do programa em que se deseja colocar a informação e é seguida de traço (hifen) e o comentário, sem necessidade de aspas.

Ex. 37:
```
10 REM – VOLUME DE UM CILINDRO
20 INPUT "RAIO DA BASE"; R:
REM – CÁLCULO DA ÁREA DA BASE
30 A= 3,14159* R ↑ 2
40 INPUT "ALTURA"; H
50 REM – CÁLCULO DO VOLUME
60 V = A * H
70 PRINT "O VOLUME É"; V
```

Quando o programa for executado, as instruções REM e os comentários que se seguem não serão considerados pelo computador. No entanto esses comentários ficam incluídos no programa e, toda a vez que se listar o mesmo, todos os REM serão também listados, facilitando o trabalho do operador.

No caso do exemplo acima não seria necessário colocar comentários, mas, quando o programa fica mais complicado, essa instrução se torna bastante útil.

Duas observações:

- a instrução REM deve sempre ser a última de uma linha com instruções múltiplas; se isso não for feito, tudo o que aparecer depois do REM, na mesma linha numerada, será considerado comentário e não será executado (veja linha 20 no exemplo acima);
- não se deve abusar da instrução REM, pois os comentários ocupam memória do computador; assim, devemos colocar o REM em lugares estratégicos, para orientação do programador ou de outras pessoas que tenham que estudar o programa.

A colocação de comentários se torna particularmente importante quando usamos sub-rotinas, como se verá no próximo capítulo.

7.2.4 — Para concluir este capítulo sobre instruções de loop e condicionais, vamos mostrar uma aplicação interessante da instrução GOTO.

Em diversas versões de BASIC é possível usar essa instrução em modo direto, dando o número de uma linha (supondo que já existe um programa armazenado no computador).

Com isso o programa será executado, a partir da linha indicada, da mesma forma que com o uso do comando RUN.

A diferença é que, enquanto o comando RUN zera as variáveis (como já vimos), a entrada por GOTO faz executar o programa sem alterar as variáveis.

Ex. 38:
```
10 INPUT " VALOR DE A ";A
20 PRINT A
RUN
VALOR DE A? 5
 5
OK
```

Se fizermos RUN 20, teremos um resultado 0, pois a variável A foi zerada pelo comando RUN. No entanto, se fizermos GOTO 20, o programa imprimirá o valor 5, que não terá sido apagado da memória.

Capítulo VIII
SUB-ROTINAS

8.1 – Uma sub-rotina pode ser definida como um pedaço de programa que pode ser usado várias vezes, dentro de um programa maior ou, mesmo, dentro de vários programas.

Vamos mostrar um exemplo simples.

Suponhamos o seguinte jogo de adivinhação de números (para duas pessoas):

Ex. 1:
```
5 N= INT (RND (1) * 10 + 1)
10 INPUT "JOGADOR 1: DIGA UM NÚMERO DE 1 A 10:";A
20 IF A=N THEN PRINT "ACERTOU – O NÚMERO É "; N: END
30 PRINT "ERROU – É A VEZ DO OUTRO JOGADOR"
40 INPUT "JOGADOR 2: DIGA UM NÚMERO DE 1 A 10:";A
50 IF A=N THEN PRINT "ACERTOU – O NÚMERO É "; N: END
60 PRINT "ERROU – É A VEZ DO OUTRO JOGADOR"
70 GOTO 10
```

A linha 5 manda o computador escolher um número aleatório entre 1 e 10. Esse número é denominado N e fica constante durante a execução do programa. Se o programa for reiniciado, através de um novo comando RUN, o número aleatório N escolhido será outro.

Quando o primeiro jogador escolhe um número (A), esse número é comparado com N, e o resultado impresso no vídeo é correspondente ao acerto ou erro. Se houve acerto,

o programa termina. Se houve erro, o programa pede a escolha de um número pelo outro jogador. A variável A pode ser usada novamente pois, como já vimos, qualquer variável pode ser mudada no meio de um programa. Quando o jogador 2 escolhe um número, a variável A passa a ter esse valor, em lugar do valor escolhido anteriormente.

Verifica-se que o programa de comparação das linhas 5Ø e 6Ø é idêntico ao das linhas 2Ø e 3Ø.

Em vez de repetir essas linhas, poderíamos simplesmente ter usado a instrução GOTO, da seguinte maneira:

```
5Ø GOTO 2Ø
```

Assim, após o jogador 2 escolher o número, a comparação seria executada novamente pelas linhas 2Ø e 3Ø.

Acontece que essa solução apresenta um sério inconveniente: o jogador 1 não seria mais chamado a jogar, pois o programa não voltaria mais para a linha 1Ø.

De fato, após a execução das linhas 2Ø e 3Ø, o jogador 2 seria chamado pela linha 4Ø e o programa voltaria para a linha 2Ø até terminar. Essa solução é boa quando existe apenas um jogador. Agora imagine o leitor se tivéssemos 3 ou 4 jogadores.

O ideal é poder usar as instruções das linhas 2Ø e 3Ø quando quisermos, após a jogada de qualquer jogador. Esse conceito básico é que caracteriza uma sub-rotina: é um pedaço de programa que pode ser usado várias vezes no mesmo programa, de maneira independente, sem estar amarrado a um determinado lugar (como aconteceu no exemplo que vimos acima, ao ser usada a instrução GOTO).

Assim, as linhas 2Ø e 3Ø, nesses programas, passam a ser consideradas como uma sub-rotina dentro do programa geral.

Vejamos, em seguida, como podem ser usadas as sub-rotinas.

8.2 – GOSUB XX

Esta instrução determina que o programa deve pular para a sub-rotina que começa na linha de número XX.

Uma sub-rotina tem que ter um fim, obviamente. Se não, seria impossível separar a sub-rotina do resto do progra-

ma. As sub-rotinas, em BASIC, terminam com a instrução RETURN. Isso significa que, após a execução da sub-rotina o programa retornará (RETURN) ao ponto onde estava o programa. Mais claramente, o programa retorna para a linha seguinte àquela em que estava a instrução GOSUB.

Assim, o esquema é o seguinte:

linha 1ØØ GOSUB 5ØØ
(ao chegar à linha 1ØØ, o programa passará para a sub-rotina que começa na linha 5ØØ)
linhas 5ØØ a 6ØØ
(sub-rotina)
linha 61Ø RETURN
(61Ø é, nesse exemplo, a próxima linha depois da sub-rotina)
O programa volta para a linha 11Ø
(Supondo, nesse exemplo, que a próxima linha, depois da 1ØØ, é a linha 11Ø).

Existem diversas particularidades a serem observadas no uso das sub-rotinas. Vamos mostrar essas particularidades nos exemplos a seguir:

Voltando ao exemplo 1, vamos fazer as seguintes alterações:

```
5 N= INT (RND (1) * 1Ø + 1)
1Ø INPUT "JOGADOR 1: DIGA UM NÚMERO DE 1 a
1Ø: "; A
2Ø GOSUB 1ØØ
3Ø INPUT "JOGADOR 2: DIGA UM NÚMERO DE 1 a
1Ø: "; A
4Ø GOSUB 1ØØ
5Ø GOTO 1Ø
1ØØ IF A=N THEN PRINT "ACERTOU – O NÚMERO
É: "; N: END
11Ø PRINT "ERROU – É A VEZ DO OUTRO JOGA-
DOR"
12Ø RETURN
```

Analisemos essa versão do programa:

É escolhido um número N; depois, o programa pede o palpite do jogador 1; em seguida pula a sub-rotina que começa na linha 1ØØ.

Na execução da sub-rotina o programa pode ser encerrado, quando houve acerto na escolha, ou voltar. Nesse caso, o programa volta para a linha 3Ø, que é a próxima linha, depois da instrução GOSUB 1ØØ.

Após a escolha feita pelo jogador 2, o programa volta para a sub-rotina da linha 1ØØ. Acontece a mesma coisa que já foi explicada. No entanto, ***desta vez o programa volta para a linha 5Ø,*** que é a próxima, depois da nova chamada da sub-rotina. Por isso colocamos a instrução GOTO 1Ø, para que o programa possa voltar ao jogador 1.

É importante entender bem os seguintes conceitos sobre as sub-rotinas:

a) após a sub-rotina, o programa volta para a linha seguinte àquela que chamou a sub-rotina (a mesma sub-rotina pode ser chamada diversas vezes no programa e voltará sempre para a linha seguinte àquela que contém a instrução GOSUB que a chamou);

b) é necessário tomar cuidado para não permitir que o programa passe pela sub-rotina sem que ela seja explicitamente chamada; não deve ser esquecido que a sub-rotina é constituída de instruções em linhas numeradas e, se o programa chegar até essas linhas, elas serão executadas; nesse caso, o programa não saberá para onde voltar quando encontrar o RETURN e acusará erro;

c) a sub-rotina pode ficar em qualquer lugar do programa, não necessariamente no fim; é necessário, apenas, tomar certas precauções para que não ocorra o que está descrito no item ***b*** acima;

d) quando a instrução GOSUB está no meio de uma linha com instruções múltiplas, o retorno será feito para a próxima instrução, após o GOSUB; nos tipos de interpretador em que isso não ocorre o GOSUB deve ficar no fim da linha, como última instrução;

Vamos mostrar mais alguns exemplos, explicando as precauções de "proteção" das sub-rotinas, para que elas não sejam executadas fora de hora.

Ainda usando o exemplo anterior, vamos alterar a colocação da sub-rotina.

```
5 N= INT (RND (1) * 1Ø + 1)
1Ø GOTO 5Ø
2Ø IF A=N THEN PRINT "ACERTOU – O NÚMERO
É: "; N: END
3Ø PRINT "ERROU – É A VEZ DO OUTRO JOGA-
DOR"
4Ø RETURN
5Ø INPUT "JOGADOR 1: DIGA UM NÚMERO DE 1 A
1Ø:";A
6Ø GOSUB 2Ø
7Ø INPUT "JOGADOR 2: DIGA UM NÚMERO DE 1 A
1Ø:";A
8Ø GOSUB 2Ø
9Ø GOTO 5Ø
```

Agora, o programa manda pular por cima da sub-rotina, logo no início (GOTO 5Ø). Se isso não fosse feito, o programa passaria da linha 5 direto para a sub-rotina e tentaria as linhas 2Ø e 3Ø (como A=0, pois não houve escolha nenhuma, A seria diferente de N e o programa imprimiria a frase da linha 3Ø); na linha 4Ø encontraria a instrução RETURN e imprimiria uma mensagem de erro:

```
RG ERROR IN 4Ø
```

(as letras RG significam RETURN sem GOSUB; esse código e a maneira de indicar o tipo de erro variam de computador para computador; daremos alguns exemplos em capítulo mais adiante)

Depois que o programa pulou para a linha 5Ø, a sub-rotina ficou "protegida", pois, daí em diante, o programa só poderá voltar para a linha 2Ø quando houver uma instrução específica (GOSUB 2Ø).

A instrução GOTO 5Ø permite passar de volta para o jogador 1; se não houvesse essa instrução, o programa terminaria depois da segunda passada pela sub-rotina, pois tentaria voltar à próxima instrução depois do GOSUB e não haveria próxima linha (o que seria interpretado, na maioria dos tipos de BASIC mais recentes, como fim de programa).

Uma compactação do programa acima pode ser feita, usando instruções múltiplas por linha (veja observação da alínea *d*, mais atrás).

```
5 N= INT(RND(1)*10+1): GOTO 30
10 IF A=N THEN PRINT "ACERTOU – O NÚMERO
É:"; N: END
20 PRINT "ERROU – É A VEZ DO OUTRO JOGA-
DOR": RETURN
30 INPUT "JOGADOR 1: DIGA UM NÚMERO DE 1 A
10:"; A: GOSUB 10
40 INPUT "JOGADOR 2: DIGA UM NÚMERO DE 1 A
10:"; A: GOSUB 10
50 GOTO 30
```

Verificamos que as instruções GOSUB 10 são as últimas das linhas 30 e 40. Se assim não fosse, todas as instruções que aparecessem na mesma linha, após a instrução GOSUB, não seriam consideradas, pois o programa já teria pulado para a sub-rotina. O mesmo ocorre com RETURN na linha 20. Se o RETURN viesse antes de uma instrução na mesma linha, esta instrução não seria executada.

É bastante comum o uso de sub-rotinas associadas a um desvio condicional; assim, se uma determinada condição for satisfeita, o programa deverá pular para uma sub-rotina.

Ex. 2:
```
10 INPUT "QUER UMA LISTA DE NÚMEROS
PARES ENTRE 0 e 20"; S$
20 IF S$ = "S" THEN GOSUB 100
30 INPUT "QUER OS NÚMEROS ÍMPARES"; A $
40 IF A $ = "S" THEN GOSUB 200
50 END
100 FOR N=0 TO 20 STEP 2: PRINT N;
: NEXT: RETURN
200 FOR N=1 TO 19 STEP 2: PRINT N;:
NEXT: RETURN
```

Nesse programa, quando aparece na tela a pergunta QUER UMA LISTA DE NÚMEROS PARES ENTRE 0 e 20? o operador poderá teclar S, se quiser a lista, ou outro caracter qualquer, se não quiser. Nese último caso, o programa passará para a linha 30 e aparecerá a pergunta QUER OS NÚMEROS ÍMPARES?; novamente a letra S indicará se o operador quer ou não essa lista. Se não quiser, o programa terminará na linha 50.

Suponhamos que desejamos a lista dos números pares; ao teclar S, fazemos com que seja satisfeita a condição da

linha 2Ø e o programa pulará para a sub-rotina iniciada na linha 1ØØ. Essa sub-rotina produz a impressão dos números pares entre Ø e 2Ø e, depois, volta para a linha 3Ø, onde se pergunta pela lista dos números ímpares. Se quisermos essa lista, a letra S nos enviará para a sub-rotina da linha 2ØØ, que imprimirá os números ímpares. Na volta dessa sub-rotina o programa terminará, na linha 5Ø.

Um outro aspecto importante a ser mencionado é que pode haver uma sub-rotina dentro de outra sub-rotina. Aqui, também, se aplicam as restrições mencionadas no caso dos "loops" FOR/NEXT embutidos. É preciso tomar cuidado com os retornos, para evitar que as duas sub-rotinas fiquem embaralhadas e o programa não saiba para onde voltar. O retorno da sub-rotina interna fica dentro da sub-rotina externa.

Ex. 3:
```
10 REM – CÁLCULO DO VOLUME DE
PRISMAS E CILINDROS
20 INPUT "QUER O VOLUME DE UM
PRIMA QUADRADO, PRISMA RETANGULAR
OU CILINDRO"; A $
30 IF A$ = "P.Q." THEN GOSUB 100
40 IF A$ = "P.R." THEN GOSUB 150
50 IF A$ = "CIL." THEN GOSUB 200
60 IF A$ <> "P.Q." AND A$ <> "P.R." AND
A$ <> "CIL" THEN 80
70 PRINT "O VOLUME É"; V: END
80 PRINT "ESSE SÓLIDO NÃO TEM
REGISTRO": END
100 INPUT "VALOR DO LADO DA BASE E DA
ALTURA"; L, H
110 AB = L ↑ 2 : GOSUB 250
120 RETURN
150 INPUT "VALOR DOS LADOS DA BASE
E DA ALTURA"; L1, L2, H
160 AB= L1*L2 : GOSUB 250
170 RETURN
200 INPUT "VALOR DO RAIO DA BASE E DA
ALTURA";R,H
210 AB= 3.14159 * R ↑ 2: GOSUB 250
220 RETURN
250 V= AB * H : RETURN
```

Analisemos esse programa:

Na linha 2Ø vamos escolher o tipo de sólido cujo volume queremos (usando as abreviações P.Q. — P.R. — CIL. para simplificar o trabalho de teclagem). Se escolhermos um prisma de base quadrada (P.Q.), o programa irá para a sub-rotina da linha 1ØØ; se escolhermos prisma retangular (P.R.), a sub-rotina será a da linha 15Ø, e se escolhermos cilindro, a sub-rotina será a da linha 2ØØ. Se não escolhermos nenhum deles, a condição da linha 6Ø nos enviará para uma resposta indicando não haver registro e o programa terminará.

A sub-rotina 1ØØ calcula a área de base e, dentro dela, uma outra sub-rotina (linha 25Ø) calcula o produto da área da base pela altura. O valor do lado da base e da altura é indicado pelo operador (linha 1ØØ). A sub-rotina 25Ø tem apenas uma linha e retorna para a linha 12Ø, que, por sua vez, manda retornar para a linha 4Ø (isto porque, quando a sub-rotina 25Ø termina, a sub-rotina 1ØØ também está terminada). Na linha 4Ø é feita uma nova comparação, cujo resultado será negativo, pois A$ = "P.Q." e, assim, o programa passará direto, chegando à linha 6Ø; nessa linha a condição não é satisfeita, pois A$ não é diferente de P.Q.; assim, o programa prossegue para a linha 7Ø, imprime o valor do volume e termina.

A mesma mecânica ocorre se escolhermos um prisma retangular ou um cilindro. Ambas as sub-rotinas vão usar, também, a sub-rotina da linha 25Ø. O resto é igual ao já explicado.

8.3 — ON ------ GOSUB

Esta instrução consta das palavras ON e GOSUB tendo, também, um argumento, que é na realidade uma variável.

A instrução é escrita da seguinte forma:

ON K GOSUB XX,YY,ZZ,WW -----

em que K é uma variável e XX,YY,ZZ, etc. o número das linhas em que são iniciadas diversas sub-rotinas. A sub-rotina escolhida depende do valor da variável (no caso, K). Se K for igual a 1, a sub-rotina será a da linha XX, se for 2, a da linha YY e, assim por diante.

Isso permite que com apenas uma instrução se possam utilizar várias sub-rotinas, sem a necessidade de escrever, para cada uma delas, a condição em que ela deve ser escolhida.

Ex. 4: Vamos mostrar um programa que não usa a instrução ON - - - - GOSUB e, depois, mostrar como ele é simplificado por essa instrução:

```
10 INPUT "ESCREVA UM NÚMERO INTEIRO DE 1 a
5";A
20 IF A=1 THEN GOSUB 80
30 IF A=2 THEN GOSUB 100
40 IF A=3 THEN GOSUB 120
50 IF A=4 THEN GOSUB 140
60 IF A=5 THEN GOSUB 160
70 GOTO 10
80 ? "VOCÊ ESCOLHEU O PRIMEIRO NÚMERO":
GOSUB 180
90 RETURN
100 ? "O NÚMERO ESCOLHIDO É O SEGUNDO":
GOSUB 180
110 RETURN
120 ? "DESTA VEZ VOCÊ FICOU COM O TERCEIRO
NÚMERO": GOSUB 180
130 RETURN
140 ? "FOI ESCOLHIDO O NÚMERO 4": GOSUB 180
150 RETURN
160 ? "ESCOLHEU O QUINTO NÚMERO": GOSUB
180
170 RETURN
180 INPUT "QUER CONTINUAR"; S$
190 IF S$ <> "S" THEN END
200 RETURN
```

Explicação: Suponhamos que foi escolhido A=1; o programa vai para a sub-rotina 80; imprime a frase correspondente e passa para uma nova sub-rotina (180), na qual o programa vai continuar ou terminar conforme a resposta do operador. Em caso de continuação, da sub-rotina 180 o programa volta para a linha 30 (não se esqueça que estávamos

na sub-rotina 8Ø, iniciada pela linha 2Ø). Nas linhas 3Ø, 4Ø, 5Ø e 6Ø, a condição IF não é satisfeita e o programa passa para a linha 7Ø de onde volta para o começo.

A mesma operação ocorre com qualquer número inteiro escolhido. Se não for inteiro ou for menor que 1 ou maior que 5, o programa volta logo para o início, pois chega até a linha 7Ø.

Para os outros valores de A o funcionamento é igual ao já explicado.

Agora, vejamos o programa simplificado:

```
1Ø INPUT "ESCREVA UM NÚMERO INTEIRO DE 1 a
5"; A
2Ø ON A GOSUB 4Ø, 6Ø, 8Ø, 1ØØ, 12Ø
3Ø GOTO 1Ø
4Ø ? "VOCÊ ESCOLHEU O PRIMEIRO NÚMERO":
GOSUB 14Ø
5Ø RETURN
6Ø ? "O NÚMERO ESCOLHIDO É O SEGUNDO":
GOSUB 14Ø
7Ø RETURN
8Ø ? "DESSA VEZ VOCÊ FICOU COM O TERCEIRO
NÚMERO": GOSUB 14Ø
9Ø RETURN
1ØØ ? "FOI ESCOLHIDO O NÚMERO 4": GOSUB 14Ø
11Ø RETURN
12Ø ? "ESCOLHEU O QUINTO NÚMERO": GOSUB
14Ø
13Ø RETURN
14Ø INPUT "QUER CONTINUAR";S$
15Ø IF S<>"S" THEN END
16Ø RETURN
```

Explicação: quando teclamos um número inteiro, entre 1 e 5, a instrução da linha 2Ø faz o seguinte: se o número (a variável A) for 1, o programa irá para a sub-rotina 4Ø; para A = 2, a sub-rotina será 6Ø; A = 3 enviará o programa para a sub-rotina 8Ø e assim até A = 5; se A não for inteiro ou for maior do que 5 ou menor do que 1, o programa volta para a linha 1Ø e recomeça.

Dentro de cada sub-rotina existe uma outra, na linha 14Ø, para perguntar se o operador quer continuar. O restante do programa é o mesmo já visto, na primeira versão.

Note que foram reduzidas 4 linhas no programa, porque a instrução da linha 2Ø da versão simplificada substituiu as linhas 2Ø, 3Ø, 4Ø, 5Ø e 6Ø da versão anterior.

Uma observação importante é que o argumento (a variável que aparece na expressão ON - - - - variável - - - - GOSUB) pode ser obtida através de manipulações matemáticas. Assim, no meio do programa, se uma variável pode assumir valores inteiros, nós podemos associar a cada valor inteiro uma sub-rotina.

Ex. 5: Vamos fazer um programa simulando a escolha aleatória de cartas em um baralho:

```
10 PRINT SPC (72): INPUT "VAMOS ESCOLHER
CARTAS";S$
20 IF S$ <> "S" THEN 270
30 K= INT(RND(1)* 13 + 1): Q=INT (RND(1)* 4+1)
40 ON K GOSUB 100, 110, 120, 130, 140, 150, 160,
170, 180, 190, 200, 210, 220
50 ON Q GOSUB 230, 240, 250, 260
60 PRINT: PRINT: PRINT "A CARTA ESCOLHIDA
FOI ";A$; " DE "; B $
70 INPUT "QUER CONTINUAR"; G$
80 IF G$ <> "S" THEN 270
90 GOTO 30
100 A$= "ÁS": RETURN
110 A$= "2": RETURN
120 A$= "3": RETURN
130 A$= "4": RETURN
140 A$= "5": RETURN
150 A$= "6": RETURN
160 A$= "7": RETURN
170 A$= "8": RETURN
180 A$= "9": RETURN
190 A$= "10": RETURN
200 A$= "VALETE": RETURN
210 A$= "DAMA": RETURN
220 A$= "REI": RETURN
230 B$= "OUROS": RETURN
```

```
240 B$= "ESPADAS": RETURN
250 B$= "COPAS": RETURN
260 B$= "PAUS": RETURN
270 PRINT: PRINT "ACABOU O JOGO"
```

Análise do programa:

Na linha 1Ø mandamos imprimir 72 espaços (cerca de 3 linhas) para dar um espaço em branco na tela, antes de colocar o título, que é VAMOS ESCOLHER CARTAS?; a interrogação, como se sabe, é devida à instrução INPUT. Se a resposta a essa pergunta não for S, o programa pula para a linha 27Ø, que dá o jogo por encerrado (note o PRINT que dá uma linha de espaço em branco, antes de imprimir ACABOU O JOGO).

A resposta sendo S, o programa prossegue e escolhe, aleatoriamente, um número K, entre 1 e 13 e um número Q, entre 1 e 4.

Na próxima linha, dependendo do valor de K escolhido, o programa pulará para a sub-rotina correspondente (é importante notar que os números das linhas das sub-rotinas têm que estar na ordem certa: a primeira linha indicada corresponde ao valor 1 do argumento K, a segunda, ao valor 2 e assim por diante). Suponhamos que foi escolhido K = 5; a linha 4Ø mandará o programa pular para a sub-rotina da linha 14Ø.

Na sub-rotina 14Ø simplesmente fazemos A$ ser igual a 5 (note que A$ é uma variável de STRING, para ser possível escolher as cartas ÁS, VALETE, DAMA e REI).

Ao voltar da sub-rotina, o programa volta para a linha 5Ø.

Na linha 5Ø encontramos, novamente, a instrução ON --- GOSUB, dessa vez com o argumento Q. O valor de Q foi escolhido entre 1 e 4; dependendo do valor escolhido o programa passará para uma outra sub-rotina.

Suponhamos que o número Q escolhido é 2; o programa passará para a sub-rotina da linha 24Ø, na qual definimos que B$ é igual a ESPADAS. Ao retornar da sub-rotina, o programa imprimirá o nome da carta escolhida (que depende de A$ e B$). No caso que estamos exemplificando será impresso:

(note que colocamos dois PRINT na linha 6Ø, para dar duas linhas de espaço, antes de imprimir o nome da carta).

Na linha 7Ø, o programa pergunta se queremos continuar. Se a resposta não for S, passamos para a linha 27Ø e o programa termina. Se for S, o programa volta para a linha 3Ø, onde são escolhidos novos valores de K e Q.

É claro que há maneiras mais compactas de escrever este programa, mas o nosso intuito aqui foi apenas demonstrar o uso da instrução ON - - - - GOSUB.

Apenas para dar mais um exemplo, vamos introduzir uma alteração no programa, para fazer a compactação:

Vamos mudar a linha 4Ø, da seguinte maneira:

```
4Ø ON K GOSUB 1ØØ, 11Ø, 11Ø, 11Ø 11Ø, 11Ø, 11Ø,
11Ø, 11Ø, 11Ø, 12Ø, 13Ø, 14Ø
```

O que significa isso? significa que, quando o número K for 2, 3 - - - - 1Ø, o programa pulará sempre para a mesma sub-rotina, da linha 11Ø.

Agora, na linha 11Ø vamos escrever:

```
11Ø PRINT: PRINT: PRINT "A CARTA ESCOLHIDA
FOI"; K; "DE ";:RETURN
```

Nas linhas 1ØØ, 12Ø, 13Ø e 14Ø vamos fazer as seguintes alterações:

```
1ØØ A$= "AS": GOSUB 15Ø
1Ø5 RETURN
12Ø A$= "VALETE": GOSUB 15Ø
125 RETURN
13Ø A$= "DAMA": GOSUB 15Ø
135 RETURN
14Ø A$= "REI": GOSUB 15Ø
145 RETURN
15Ø PRINT: PRINT: PRINT "A CARTA ESCOLHIDA
FOI "; A$; " DE ";: RETURN
```

Agora, na linha 6Ø, vamos colocar apenas:

```
6Ø PRINT B$
```

As linhas 16Ø até 22Ø podem ser eliminadas.
Vamos ver como ficou?

```
LIST
 1Ø PRINT SPC(72): INPUT "VAMOS ESCOLHER
 CARTAS"; S$
 2Ø IF S$<>"S" THEN 27Ø
 3Ø K = INT(RND(1)* 13 + 1): Q = INT(RND(1)*4+1)
 4Ø ON K GOSUB 1ØØ, 11Ø, 11Ø, 11Ø, 11Ø, 11Ø, 11Ø,
 11Ø, 11Ø, 11Ø, 12Ø, 13Ø, 14Ø
 5Ø ON Q GOSUB 23Ø, 24Ø, 25Ø, 26Ø
 6Ø PRINT B$
 7Ø INPUT "QUER CONTINUAR"; G$
 8Ø IF G$<>"S" THEN 27Ø
 9Ø GOTO 3Ø
 1ØØ A$= "ÁS":GOSUB 15Ø
 1Ø5 RETURN
 11Ø PRINT: PRINT: PRINT " A CARTA ESCOLHIDA
 FOI"; K; "DE ";: RETURN
 12Ø A$ = "VALETE": GOSUB 15Ø
 125 RETURN
 13Ø A$ = "DAMA": GOSUB 15Ø
 135 RETURN
 14Ø A$ = "REI": GOSUB 15Ø
 145 RETURN
 15Ø PRINT: PRINT: PRINT "A CARTA ESCOLHIDA
 FOI "; A$; " DE ";:RETURN
 23Ø B$ = "OUROS": RETURN
 24Ø B$ = "ESPADAS": RETURN
 25Ø B$ = "COPAS": RETURN
 26Ø B$ = "PAUS": RETURN
 27Ø PRINT: PRINT "ACABOU O JOGO"
```

— Algumas observações adicionais sobre a instrução ON ---- GOSUB:

Quando o argumento é negativo, o programa passa para a próxima linha, não considerando a instrução ON ---- GOSUB. O mesmo ocorre quando o argumento é zero ou não inteiro. Quando o argumento atinge um valor maior do que o número de sub-rotinas indicadas na instrução, o programa passa para a próxima linha.

Ex. 6:
```
1Ø N=Ø: INPUT "ESCOLHA UM NÚMERO DE 1 A
5";A
2Ø ON A GOSUB 7Ø, 7Ø, 7Ø, 7Ø, 7Ø
```

```
30 IF N>0 THEN 50
40 PRINT: PRINT "NÚMERO ERRADO": PRINT:?:
GOTO 10
50 INPUT "QUER CONTINUAR"; S$
60 IF S$<>"S" THEN ? "O JOGO ACABOU": END
65 GOTO 10
70 PRINT: PRINT "O NÚMERO ESCOLHIDO
FOI"; A: N= N + 1: RETURN
```

Explicação:

Na linha 20, se o número escolhido for um inteiro de 1 a 5, o programa pulará para a sub-rotina 70 (veja que tivemos que repetir o número 70 5 vezes); será impressa a mensagem dizendo qual o número escolhido e o contador N incrementado de 1 unidade; daí retornamos para a linha 30.

O contador N serve para o seguinte:

Se for escolhido um número negativo, nulo, fracionário ou maior que 5, o programa passa para a linha 30; nesse caso, N é igual a 0 e, portanto, o programa pula para a linha 40; nessa linha será indicado o erro e o programa recomeça na linha 10. Cada vez que o programa recomeça na linha 10 o valor de N passa a ser 0, para permitir que a mensagem de erro seja impressa (linha 40).

Quando o programa retorna para a linha 30, após passar pela sub-rotina 70, o valor de N será maior que 0; assim, o programa passa para a linha 50, onde se pergunta se o jogo vai continuar. O resto o leitor já deverá ter entendido.

Vamos tentar executar esse programa:

```
RUN
ESCOLHA UM NUMERO DE 1 A 5 ? 0

NÚMERO ERRADO


ESCOLHA UM NUMERO DE 1 A 5 ? 2

O NÚMERO ESCOLHIDO FOI 2
QUER CONTINUAR ? S
ESCOLHA UM NÚMERO DE 1 A 5 ? 6
```

```
NÚMERO ERRADO

ESCOLHA UM NÚMERO DE 1 A 5 ? 3

O NÚMERO ESCOLHIDO FOI 3
QUER CONTINUAR ? N
O JOGO ACABOU

OK
```

O fato de, às vezes, ser necessário colocar mais números de sub-rotinas do que cabem em uma linha de programa pode ser contornado da seguinte maneira:

Ex. 7:
```
10 (instrução)
20 (instrução)
30 ON K GOSUB 100, 110, 120, 130,
140, 150, 160, 170, 180, 190
```
(vamos supor que não dá mais para colocar as sub-rotinas restantes — de 200 em diante — nessa linha)
```
40 ON K - 10 GOSUB 200, 210, 220,
230, 240
```

Explicação da linha 40: quando K é maior que 10 (que é a quantidade de linhas de sub-rotinas que foram escritas na linha 30), o programa pula para a linha 40. Nessa linha K-10 será, novamente, 1, 2, 3, etc.; assim, o programa prosseguirá indo para as sub-rotinas 200, 210, etc.

Finalmente, é importante notar que a instrução ON ---- GOSUB deve ser sempre a última de uma linha de várias instruções:

Ex. 8:
```
10 N= N + 1
20 PRINT N: ON N GOSUB 50,
60, 70 ----
```

Veja que, na linha 20, a instrução ON ---- GOSUB ficou depois do PRINT. Caso contrário o PRINT não seria executado.

8.4 – ON ---- GOTO

Essa instrução não é, na realidade, uma instrução de sub-rotina. Estamos colocando-a neste capítulo apenas em razão das semelhanças que ela apresenta com a instrução ON --- GOSUB.

O funcionamento é muito semelhante ao já explicado, com a única diferença que, sendo um GOTO, não há retorno e, portanto, instrução RETURN. Assim, quando escrevemos:

```
ON K GOTO XX, YY, ZZ
```

isso significa que, para K=1 o programa pulará para a linha XX; para K=2 irá para a linha YY, e assim por diante. No entanto, depois de pular para essas linhas, o programa prosseguirá, na ordem crescente de numeração de linhas, a menos que uma outra instrução (tipo GOTO, GOSUB, etc.) determine de maneira diferente.

Ex. 9:
```
1 ?  "VOU ESCOLHER UM NÚMERO – ADIVINHE EM 5 TENTATIVAS"
5 N= INT (RND(1) * 10 + 1)
10 INPUT "ESCOLHA UM NÚMERO"; A
20 T = T + 1
30 K = SGN (A-N)+2
40 ON K GOTO 50, 60, 70
50 ? "ESTÁ ABAIXO": IF T>4 THEN 100
55 GOTO 10
60 ? "ACERTOU! O NÚMERO É"; N: END
70 ? "ESTÁ ACIMA": IF T>4 THEN 100
75 GOTO 10
100 ? "ESGOTOU AS 5 TENTATIVAS"
```

Explicação: na linha 5 o programa escolhe um número aleatório entre 1 e 10.

Na linha 10 o operador escolhe um número (A); na linha 20 colocamos um contador de tentativas.

Na linha 30 usamos a função SGN, que, como já explicamos, é igual a –1, se o argumento (o que está entre parênteses) for negativo, 0, se o argumento for nulo e +1, se for positivo. Assim, se A for menor que N, K será igual a 1; se A=N, K=2 e se A for maior que N, K=3.

Na próxima linha, dependendo de K, o programa pulará para as linhas 5Ø, 6Ø ou 7Ø. Se o contador (T) ainda não tiver atingido o valor 5, o programa voltará a pedir um número. Isto porque colocamos a instrução GOTO 1Ø. Se T já for igual a 5, passaremos para a linha 1ØØ, onde o programa termina.

Verifica-se que, se não houver um comando tipo GOTO (como nas linhas 55 e 75) o programa ficará ou prosseguirá a partir da linha para onde foi mandado pelo ON- - - -GOTO - - -. (p. ex: linha 6Ø).

As características operacionais e as restrições da instrução ON - - - - GOTO são iguais às do ON - - - - GOSUB, de modo que não insistiremos no assunto. Vamos apenas dar mais um exemplo, para fixar bem a idéia.

Ex. 10:
```
1Ø REM – EXEMPLOS DE MULTIPLICAÇÃO,
DIVISÃO, POTENCIAÇÃO E RAIZ QUADRADA
2Ø FOR T= 1TO 32: ? : NEXT
3Ø ? "TABELA DE OPERAÇÕES"
4Ø ?:? " 1- MULTIPLICAÇÃO"
5Ø ?:? " 2- DIVISÃO"
6Ø ?:? " 3- POTENCIAÇÃO"
7Ø ?:? " 4- RAIZ QUADRADA"
8Ø ?: INPUT "ESCOLHA UMA OPERAÇÃO"; A
9Ø ON A GOTO 11Ø, 15Ø, 19Ø, 23Ø
1ØØ ? "ESCOLHEU ERRADO – TENTE OUTRA VEZ":
GOTO 2Ø
11Ø FOR T= 1TO 32 : ? : NEXT
115 ? "MULTIPLICAÇÃO"
12Ø ? : INPUT "PRIMEIRO NÚMERO"; B
13Ø INPUT "SEGUNDO NÚMERO"; C
14Ø D = C * B: PRINT: PRINT "O RESULTADO É";
D : GOTO 28Ø
15Ø FOR T= 1TO 32 : ? : NEXT
155 ? "DIVISÃO"
16Ø ? : INPUT "DIVIDENDO"; E
17Ø INPUT "DIVISOR"; F
18Ø G = E/F : PRINT: PRINT " O RESULTADO
É"; G: GOTO 28Ø
19Ø FOR T=1TO 32 :?: NEXT
195 ? "POTENCIAÇÃO"
```

```
200 ? : INPUT "NÚMERO"; H
210 INPUT "POTÊNCIA"; I
220 J= H↑I: PRINT: PRINT "O RESULTADO
E"; J: GOTO 280
230 FOR T= 1TO 32 : ? : NEXT
235 ? "RAIZ QUADRADA "
240 ? : INPUT "NÚMERO "; K
250 L = SQR (K)
260 PRINT: PRINT "O RESULTADO É"; L
280 ? : ? "QUER TENTAR DE NOVO";S$
290 IF S$ = "S" THEN 20
```

Vamos explicar, rapidamente, este programa:

Na linha 10 aparece apenas o título do programa (veja instrução REM).

Nas linhas 20, 110, 150, 190 e 230 usamos um processo simples para apagar a tela, antes de imprimirmos o nome do exercício ou a tabela de operações.

Nas linhas 30 a 70 colocamos uma tabela de operações na tela, para que o operador escolha aquela que deseja efetuar. Na linha 80 perguntamos qual a operação desejada, pelo seu número.

A linha 90 usa a instrução que estamos explicando: conforme o número escolhido, o programa vai pular para a linha onde é feita essa operação. Se o número escolhido for diferente dos que aparecem na tabela, o operador é convidado a recomeçar (linha 100).

As linhas 110 até 140, 150 até 180, 190 até 220 e 230 até 260 contêm os programas das operações. Cada uma delas termina com a instrução GOTO 280, exceto a 260 (onde não é necessário mandar pular para a linha 280, que é a próxima linha no programa).

A linha 280 pergunta se o operador deseja continuar o programa.

Acreditamos que esse exemplo dê uma idéia clara sobre como se pode usar a instrução ON - - - - GOTO - - - -.

Aconselhamos o leitor a programar e executar o exemplo acima, assim como os anteriores, para se familiarizar melhor com as técnicas envolvidas.

Capítulo IX
VARIÁVEIS INDEXADAS

9.1 — Em capítulo anterior vimos como a linguagem BASIC trabalha com variáveis numéricas e variáveis de STRING. Essas variáveis, descritas mais atrás, são as variáveis simples, isto é, cada variável assume um valor de cada vez.

Assim a variável A pode assumir, por exemplo, os valores 1, -2, .3, etc., um de cada vez.

Agora vamos tratar de um outro tipo de variável, extremamente útil, denominada variável indexada. As variáveis indexadas podem ser de índice simples ou de índice duplo.

Uma variável de índice simples pode ser indicada da seguinte maneira:

A(N)

em que A é o nome da variável e N é o seu índice. A variável indexada pode assumir um valor diferente para cada valor do índice.

Isso significa que A(1) pode ter um valor e A(3) pode ter outro.

Naturalmente, já que os índices são indicadores da "ordem" de uma variável, esses índices devem ser inteiros e positivos.

Uma variável de duplo índice pode ser designada da seguinte forma:

Z (M, N)

em que Z é a variável e M e N são os índices. A variável Z(3,1) pode ser diferente da variável Z(1,3), Z(1,1), e assim por diante.

O conjunto de valores das variáveis indexadas é denominado, em inglês, "ARRAY", o que poderia ser traduzido li-

vremente como matriz. Uma matriz de índice simples ou matriz unidimensional é chamada, também, vetor.

O conjunto AB(P), em que AB é o nome da variável e P é o índice, que pode assumir valores inteiros e positivos (inclusive Ø), é denominado um vetor. O conjunto C1(R,S), em que C1 é o nome da variável e R e S são os índices, é denominado "array" bidimensional, ou, simplesmente, "array" (ou matriz, se o leitor preferir).

Vejamos como podem ser utilizadas as variáveis indexadas.

9.2 — **Variáveis unidimensionais** (índice simples)

Como já mensionamos, as variáveis podem assumir os mesmos nomes das variáveis indicadas em capítulo anterior, isto é, ter uma máximo de dois caracteres, dos quais o primeiro tem que ser uma letra e o segundo (quando houver) pode ser uma letra ou um dígito, de Ø a 9. Além disso, as variáveis podem ser de STRING. Os índices são valores inteiros e positivos, inclusive Ø.

A alocação de valores a variáveis indexadas pode ser feitas da mesma maneira que para as variáveis simples, isto é, através de:

a) Comando LET (que não precisa ser escrito)

Ex. 1:
```
10 A(Ø) = 28 : A(1) = .3 : A(2) = -5
20 PRINT A(Ø):?A(1):?A(2)
RUN
 28
 .3
-5

OK
```

b) Instrução INPUT

Ex. 2:
```
10 INPUT " A(Ø) = "; A(Ø)
20 INPUT " A(1) = "; A(1)
30 INPUT " A(2) = "; A(2)
40 ? A(Ø); A(1); A(2)
RUN
```

```
A(Ø)=?  13
A(1)=?  -8
A(2)=?  .Ø4
 13 -8  .Ø4

OK
```

ou, ainda,

Ex. 3:
```
1Ø ? "DÊ OS VALORES DE A(Ø), A(1)
E A(2)"
2Ø INPUT A(Ø), A(1), A(2)
3Ø ? A(Ø); A(1); A(2)
RUN
DÊ OS VALORES DE A(Ø), A(1) E A(2)
? 25,-1,Ø
 25 -1  Ø

OK
```

c) Instrução READ/DATA

Ex. 4:
```
1Ø READ A(Ø), A(1), A(2), A(3)
2Ø ? A(Ø); A(1); A(2); A(3)
3Ø DATA 3, -2, .8, 6.5
RUN
 3  -2  .8  6.5

OK
```

ou,

Ex. 5:
```
1Ø READ A$(Ø), A$(1), A$(2)
2Ø PRINT A$(1); A$(Ø); A$(2)
3Ø DATA "BOM ", "MUITO", DIA
RUN
MUITO BOM DIA

OK
```

Até aqui, tudo fácil. Na realidade não há necessidade de se dizer claramente todos os índices das variáveis, como

foi feito nos exemplos acima (pedimos ou lemos diretamente os valores de A(Ø), A(1), etc.).

Os índices também podem ser considerados como variáveis. É melhor exemplificar.

Ex. 6:
```
1Ø FOR N = 1 TO 1Ø : READ A(N) : NEXT
2Ø FOR K = 1 TO 1Ø : ? A(K);: NEXT
3Ø DATA 3, -12, .5, 8, -8, 1Ø, 5ØØ, -.Ø3, -1ØØ, 3Ø
RUN
 3 -12  .5 8 -8 1Ø 5ØØ
-.Ø3 -1ØØ  3Ø

OK
```

Explicação: na linha 1Ø a instrução FOR/NEXT manda N variar de 1 a 1Ø, o que significa que, na primeira passada do "loop", a instrução READ vai ler A(1); na segunda passada N = 2 e, logo, será lido o valor de A(2); e assim por diante, até N = 1Ø. Os valores lidos são os constantes da linha 3Ø (DATA).

Usamos novamente a instrução FOR/NEXT para mandar imprimir os valores de A indexados de 1 a 1Ø. Verifique que, nesta linha nós chamamos a variável de K, em vez de N, e fazemos variar K de 1 a 1Ø. É óbvio que não há nenhuma diferença, pois o que interessa é o valor do índice e não o seu nome.

Explicando melhor: A(N) é a mesma coisa que A(K), se N e K tiverem os mesmos valores. Na linha 3Ø, na primeira passada do "loop" K = 1 e, assim, será impresso o valor de A(1), que é o mesmo valor que foi lido na linha 1Ø, quando N = 1.

Ex. 7: (vamos refazer parte do ex. 10 do item 8.4)
```
3Ø FOR K= Ø TO 4: READ A$(K): NEXT
4Ø FOR K= Ø TO 4: ? : ?  A$ (K) : NEXT
5Ø DATA "TABELA DE OPERAÇÕES", " 1 – MULTIPLICAÇÃO", "2 – DIVISÃO", "3– POTENCIAÇÃO"
6Ø DATA "4 – RAIZ QUADRADA"
8Ø ? : INPUT " ESCOLHA UMA OPERAÇÃO"; A
85 FOR T = 1 TO 32 : ? : NEXT
87 ? A$ (A)
9Ø ON A GOTO 12Ø, 16Ø, 2ØØ, 24Ø
```

Verifique o seguinte (comparando com o programa do ex. 10, item 8.4):

A tabela de operações é colocada na tela, atribuindo-se cada título a um índice da variável de STRING A$(K).

Assim, ao fazer o "loop" FOR/NEXT, da linha 3Ø, o programa vai pegar os dados da linha 5Ø e fazer com que:

A$(Ø) = TABELA DE OPERAÇÕES
A$(1) = 1 – MULTIPLICAÇÃO
A$(2) = 2 – DIVISÃO
A$(3) = 3 – POTENCIAÇÃO
A$(4) = 4 – RAIZ QUADRADA

Na linha 4Ø, usamos o FOR/NEXT para imprimir os valores de A$(K); repare que usamos a variável K, nessa linha, como poderíamos ter usado qualquer outra. Não há problema em repetir as variáveis desde que isso não cause confusão. No caso aqui mostrado o "loop" da linha 3Ø fez K variar de Ø a 4. Na linha 4Ø a instrução manda K variar novamente de Ø a 4, e, dessa maneira, não haverá confusão. É preciso, apenas, tomar cuidado para não usar novamente o valor de K tal como ele se encontra, sem mandá-lo mudar de valor, pois, depois da instrução da linha 4Ø o valor de K ficou sendo igual a 4 e assim continuará até ordem em contrário.

As linhas 85 e 87 foram introduzidas para substituir as linhas 11Ø, 115, 15Ø, 155, 19Ø, 195, 23Ø e 235, que podem ser eliminadas do programa.

Quando o operador escolhe o número da operação e, portanto, dá um valor para A, o programa, em seguida, apaga a tela e manda imprimir A$(A); dependendo do valor de A, o título da operação será impresso na tela.

Em seguida o programa passa para a linha 9Ø onde vai buscar a sub-rotina escolhida. Veja que nós mudamos os números das linhas, já que não é necessário ter as linhas 11Ø, 115, etc.

O uso de variáveis indexadas é muito útil na confecção de tabelas e manipulação de dados diversos.

As variáveis de STRING e numéricas podem ser misturadas, desde que seja observada a ordem correta de alocação de valores (veja o capítulo sobre entrada de dados).

Ex. 8:
```
5-REM-TABELA DE DADOS PESSOAIS
1Ø FOR K = 1 TO 5 : READ A$(K), B$(K), C(K),
D$(K) : NEXT
2Ø ? : ? " TABELA DE NOMES"
3Ø FOR K = 1 TO 5 : ? : ? A$(K): NEXT
4Ø ? : INPUT "ESCOLHA UM NOME"; N$
5Ø FOR K = 1 TO 5 : IF A$(K)=N$
THEN GOSUB 9Ø
55 NEXT K
6Ø ? : INPUT "QUER ESCOLHER OUTRO NOME";S$
7Ø IF S$ = "S" THEN 2Ø
8Ø END
9Ø FOR N = 1 TO 32: ? : NEXT
1ØØ PRINT "NOME : "; A$(K)
11Ø PRINT: PRINT "ENDEREÇO : RUA "; B$(K);
"NÚMERO"; C(K)
12Ø ?: PRINT "TELEFONE: "; D$(K)
13Ø K = 5: RETURN
14Ø DATA ROBERTO, AMAZONAS, 15, "267-1255",
PAULO, PARAÍBA, 32, "250-3827", RENATO, PIAUÍ,
1Ø, "275-8828"
15Ø DATA PEDRO, BELÉM, 178, "225-3018", ANTÔ-
NIO, NATAL, 42, "236-8888"
```

Explicação do programa: na linha 1Ø o programa vai ler os valores de A$(K),B$(K),C(K),D$(K), em que A$(K) é o nome de uma pessoa, B$(K) é o nome de uma rua, C(K) é o número da casa e D$(K) é o número do telefone. Quando K = 1, no "loop" FOR/NEXT, o programa vai ler, de uma vez, os valores de A$(1), B$(1), C(1), D$(1). Por isso, nas linhas que contêm DATA, os dados devem ser colocados na mesma ordem da leitura: um nome, um nome de rua, um número de casa, um número de telefone; outro nome, outro nome de rua, outro número de casa, telefone e, assim por diante. D$(K) está como STRING por causa do tracinho no número do telefone.

Na linha 2Ø, será impresso o título da tabela.

Na linha 3Ø são impressos os nomes que constam da tabela: quando K varia de 1 a 5, serão impressos os valores A$(1), A$(2), A$(3), A$(4) e A$(5).

Quando o programa pergunta qual o nome escolhido, o operador tecla o nome da pessoa cujo endereço e telefone deseja. Esse nome é, temporariamente, chamado de N$.

Na linha 5Ø o programa vai procurar o nome que foi escolhido. Para isso, vai lendo os valores de A$(1), A$(2), etc.

Suponhamos que o operador escolheu o nome PEDRO. A operação das instruções da linha 5Ø funciona da seguinte maneira:

Quando K = 1, o programa compara N$ com A$(1), ou seja, ROBERTO. No caso, não é igual, pois A$(1) é igual a ROBERTO e N$ é igual a PEDRO. Com o NEXT K, K passa a ser igual a 2 e A$(K) igual a A$(2), que é PAULO. A condição ainda não foi satisfeita. Quando K chega ao valor 4, A$(K) passa a ser igual a A$(4), que é PEDRO.

Nesse momento A$(K) é igual a N$ e o programa pula para a sub-rotina 9Ø. É importante notar que, agora, K é igual a 4, até ordem em contrário.

A linha 9Ø apaga a tela e a linha 1ØØ imprime o nome escolhido (que, nesse caso, é A$(4)).

A linha 11Ø imprime uma linha em branco e depois a frase:

ENDEREÇO: RUA BELÉM NÚMERO 178

As palavras ENDEREÇO: RUA e NÚMERO são fixas, isto é, estão pré-programadas. Quando o programa passa pela linha 11Ø, as únicas coisas que podem mudar são os valores de B$(K) e C(K). No nosso exemplo, B$(4) é igual a BELÉM e C(4) é igual a 178.

O mesmo ocorre na linha 12Ø que manda imprimir o número do telefone. No nosso exemplo, D$(4) é igual a 225-3018.

Quando o programa passa para a linha seguinte (13Ø), nós fazemos K = 5 e retornamos da sub-rotina. Quando o programa volta da sub-rotina, ele vai para a linha 55, que é a linha seguinte ao GOSUB. Se nós não fizéssemos N = 5, o programa prosseguiria com o NEXT K e rodaria novamente a linha 5Ø, inutilmente, gastando tempo do computador. Como nós já fizemos K = 5, o programa passa direto para a linha 6Ø onde pergunta se o operador quer continuar.

Se não for teclada a letra S, o programa terminará na linha 8Ø.

Para completar este item, sobre variáveis de índice simples devemos mencionar que a variável que define o índice pode ser resultado de uma operação matemática, desde que esse resultado seja positivo e inteiro ou nulo.

Ex. 9:
```
1Ø REM – SALADA DE FRUTAS
2Ø FOR N= 1 TO 1Ø: READ F$ (N): NEXT
25? "VAMOS FAZER UMA SALADA DE FRUTAS"
3Ø P= INT (RND(1)*1Ø + 1)
4Ø PRINT F$ (P) : M = M+1
5Ø IF M>4 THEN ? "QUER OUTRA SALADA?":
GOTO 7Ø
6Ø GOTO 3Ø
7Ø INPUT "S OU N"; S$
8Ø IF S$<>"S" THEN END
9Ø M= Ø : GOTO 25
1ØØ DATA LARANJA, ABACAXI, MELÃO,
ABACATE, MAMÃO, MAÇÃ, PÊRA
11Ø DATA UVA, CEREJA, MANGA
```

Como funciona esse programa?

Em primeiro lugar o programa lê o nome de 1Ø frutas, que estão nas linhas 1ØØ e 11Ø.

Na linha 3Ø é escolhido, ao acaso, um número inteiro entre 1 e 1Ø. Nas linhas 4Ø, 5Ø e 6Ø mandamos imprimir cinco vezes o nome da fruta escolhida (veja o contador M= M+1 e a condição IF M>4); suponhamos que o primeiro valor escolhido (P) é 3; será impresso o nome da fruta A$(3) que, no caso, é MELÃO (ver linha 1ØØ).

Como M ainda não é maior do que 4, o programa vai para a próxima linha (6Ø) que manda voltar para a linha 3Ø, onde outro número P é escolhido: suponhamos que seja 8. A fruta será A$(8), que é UVA.

Quando M é maior do que 4, o programa pergunta se o operador quer outra salada e pula para a linha 7Ø, onde o operador vai responder S ou N. Com S, o programa volta para a linha 3Ø, mas, antes, faz M = Ø, pois, caso contrário, ele já seria maior que 4 de saída (devido à passada anterior pelo programa). Se for qualquer outra letra, que não S, o programa terminará.

Vejamos um exemplo de execução desse programa:

```
RUN
VAMOS FAZER UMA SALADA DE FRUTAS
CEREJA
UVA
ABACAXI
ABACATE
ABACATE
QUER OUTRA SALADA?
S OU N? S
VAMOS FAZER UMA SALADA DE FRUTAS
MAÇÃ
LARANJA
MELÃO
CEREJA
QUER OUTRA SALADA?
S OU N? N

OK
```

Note que na primeira rodada o ABACATE apareceu duas vezes. Isto porque a função P é aleatória e pode dar o mesmo número duas vezes seguidas.

Existem truques de programação para evitar isso, mas deixaremos esse caso para mais adiante.

9.3 — Variáveis de índice duplo (matrizes)

As regras que mencionamos acima são, em geral, válidas para as variáveis duplamente indexadas.

É necessário, apenas, prestar bastante atenção para o fato de que A(1,2) é diferente de A(1,3) e diferente de A(2,1). Veremos como funcionam essas variáveis, através de exemplos:

Ex. 10: (antes de executar, veja item 9.4)

```
10 REM-LISTA DE DESPESAS DOMÉSTICAS
20 FOR M=1 TO 4: FOR N = 0 TO 5: READ A $
(M,N): NEXT N,M
30 FOR T = 1 TO 32: ? : NEXT : ? "TABELA DE
DESPESAS"
```

```
40 FOR K = 1 TO 4 : ?: ? A$ (K,0) : NEXT
50 ? : INPUT "ESCOLHA UM TIPO DE DESPESA"; A
60 FOR T= 1 TO 32: ? : NEXT : ? A$ (A,0)
70 FOR Q= 1 TO 5 : ? : ? A$(A,Q) : NEXT
80 ? : INPUT "QUER OUTRA ESCOLHA"; S$
90 IF S $<>"S" THEN END
100 GOTO 30
110 DATA " 1 - ALIMENTAÇÃO" , CARNE, FRUTAS, LEGUMES, LEITE, PÃO
120 DATA "2 - SERVIÇOS PÚBLICOS", ELETRICIDADE, GÁS, LUZ, TELEFONE, ÁGUA
130 DATA "3 - DESPESAS DE CASA", EMPREGADOS, ALUGUEL, CONDOMÍNIO, CONSERTOS, IMPOSTOS
140 DATA "4 - DIVERSOS", GASOLINA, DIVERSÕES, ROUPAS, PRESENTES, "DESPESAS MÉDICAS"
```

Analisemos o programa: na linha 20 temos dois "loops" FOR/NEXT embutidos.

Assim, quando M = 1, o segundo "loop" lerá os dados A$(1,0), A$(1,1), A$(1,2), A$(1,3), A$(1,4) e A$(1,5); quando M = 2, serão lidos, em seqüência, A$(2,0), A$(2,1), A$(2,2), A$(2,3), A$(2,4), e A$(2,5); a leitura termina quando M = 4 e foi lido o dado A$(4,5).

Na linha 30 apagamos a tela e colocamos o título. Na linha 40 mandamos imprimir, usando FOR/NEXT, os valores A$(1,0), A$(2,0), A$(3,0) e A$(4,0): esses valores são os títulos das despesas.

Na linha 50 podemos escolher um tipo de despesa, para ter a lista de suas subdivisões. Quando escolhemos o valor A, correspondente ao número da despesa escolhida, o programa vai apagar a tela e imprimir o nome da despesa (linha 60). Lembre-se que os nomes das despesas têm sempre N = 0, e, assim, quando A é, por exemplo, igual a 2, o título impresso será A$(A,0), ou seja A$(2,0), que é o título da segunda despesa.

Na linha 70 vamos listar os itens que constituem a despesa número 2: com o "loop" FOR/NEXT listamos, sucessivamente, A$(2,1), A$(2,2), A$(2,3), A$(2,4) e A$(2,5). Esses valores são os itens desejados

Nas linhas seguintes podemos fazer nova escolha ou terminar o programa.

Agora, vamos analisar cuidadosamente as linhas de DATA:

Na linha 11Ø o primeiro dado é "1 – ALIMENTAÇÃO" (as aspas aparecem por causa dos espaços e do traço depois do 1). Esse dado é o primeiro a ser lido pelo programa, e já sabemos que o primeiro dado lido se chamará A$(1,Ø); então, esse primeiro dado é, obrigatoriamente, um título de despesa. Como os próximos dados a serem lidos são A$(1,1), A$(1,2), A$(1,3), A$(1,4) e A$(1,5), os dados seguintes, na linha 11Ø, são os itens que compõem a despesa ALIMENTAÇÃO.

Depois de lido o valor A$(1,5), o programa passará a ler A$(2,Ø). Assim, na linha 12Ø o primeiro item, que é o próximo a ser lido, também será o título de um item de despesas. Em seguida são colocados os subitens desse segundo item de despesa. Assim vamos até o fim.

O esquema geral é o seguinte:

– primeiro item de despesa:	A$ (1,Ø)
sub itens:	A$ (1,1)
	A$ (1,2)
	A$ (1,3)
	A$ (1,4)
	A$ (1,5)
– segundo item de despesa:	A$ (2,Ø)
sub itens:	A$ (2,1)
	A$ (2,2)
	A$ (2,3)
	A$ (2,4)
	A$ (2,5)

e assim por diante.

Vamos experimentar o programa?

```
RUN
TABELA DE DESPESAS

1 – ALIMENTAÇÃO

2 – SERVIÇOS PÚBLICOS
```

3 – DESPESAS DE CASA

4 – DIVERSOS

ESCOLHA UM TIPO DE DESPESA? **3**

3 – DESPESAS DE CASA

EMPREGADOS

ALUGUEL

CONDOMÍNIO

CONSERTOS

IMPOSTOS

QUER OUTRA ESCOLHA? **S**

TABELA DE DESPESAS

1 – ALIMENTAÇÃO

2 – SERVIÇOS PÚBLICOS

3 – DESPESAS DE CASA

4 – DIVERSOS

ESCOLHA UM TIPO DE DESPESA? 4

4 – DIVERSOS

GASOLINA

DIVERSÕES

ROUPAS

PRESENTES

```
DESPESAS MÉDICAS

QUER OUTRA ESCOLHA? N

OK
```

O exemplo acima mostra a grande utilidade das variáveis duplamente indexadas, ou "arrays", na preparação de tabelas ou demonstração de dados.

Vamos mostrar um outro exemplo, de contagem de votos em uma eleição:

Vamos supor que os mesmos candidatos podem ser votados para três cargos diferentes em um clube: Presidente, Vice-Presidente e Tesoureiro. Os votos já estão colocados nas linhas de dados. O programa faz a contagem e impressão dos resultados.

Ex. 11:
```
10 REM – CONTAGEM DE VOTOS EM UMA ELEIÇÃO
15 FOR N= 1 TO 10: READ N$ (N): NEXT N
20 READ A$, B$ : IF A$="PRESIDENTE" THEN C=1:GOTO 40
25 IF A$ = "VICE-PRESIDENTE" THEN C= 2: GOTO 40
30 IF A$ = "FIM" AND B$ = "FIM" THEN 150
35 C= 3
40 IF B$ = "ANTÔNIO" THEN D= 1: GOTO 140
50 IF B$ = "JOAQUIM" THEN D= 2: GOTO 140
60 IF B$ = "MARCELO" THEN D= 3: GOTO 140
70 IF B$ = "ROBERTO" THEN D= 4: GOTO 140
80 IF B$ = "PAULO" THEN D= 5: GOTO 140
90 IF B$ = "PEDRO" THEN D=6: GOTO 140
100 IF B$= "JOÃO" THEN D= 7: GOTO 140
110 IF B$ = "MANOEL " THEN D=8: GOTO 140
120 IF B$ = "RAUL" THEN D= 9: GOTO 140
130 D= 10
140 T (C,D) = T(C,D) + 1: GOTO 20
150 PRINT "NOME"; TAB(7); "PRES"; TAB (12);
"V.PRES" ; TAB(19); "TES"
160 FOR N= 1 TO 10: PRINT N$(N);
TAB (7); T(1,N); TAB(12); T(2,N); TAB(19); T(3,N)
```

```
170 NEXT N
180 DATA ANTÔNIO, JOAQUIM, MARCELO, ROBER-
TO, PAULO, PEDRO, JOÃO, MANOEL, RAUL, JOSÉ
```

Até esse ponto do programa estabelecemos as regras para a contagem e impressão do resultado.

As próximas linhas de DATA vão conter os dados na forma que forem sendo lidos nas cédulas: cargo e nome do candidato. Se houver 20 eleitores e cada um votar em 3 nomes para os 3 cargos, teremos uma seqüência de 60 cédulas, cada uma com um cargo e um nome.

Da linha 190 em diante os dados vão sendo colocados no programa conforme as cédulas vão sendo lidas.

```
190 DATA PRESIDENTE, PEDRO, "VICE-PRESIDEN-
TE", ANTÔNIO, TESOUREIRO, JOÃO, PRESIDENTE,
ROBERTO
200 DATA "VICE-PRESIDENTE", RAUL, TESOUREI-
RO, PEDRO, PRESIDENTE, MANOEL
```

E assim por diante.

Como vai funcionar esse programa?

Na linha 15 fazemos uma leitura dos nomes dos candidatos:

```
N$(1) = ANTÔNIO
N$(2) = JOAQUIM
  -
  -
  -
  -
N$(10)= JOSÉ
```

Na linha 20 vamos ler os votos: um cargo (A$) e um nome(B$); vamos associar o cargo PRESIDENTE com o valor de C= 1, VICE-PRESIDENTE com C=2 e, se não for nem PRESIDENTE nem VICE-PRESIDENTE, só pode ser TESOUREIRO e C fica igual a 3.

O mesmo ocorre com os nomes: quando o nome votado é ANTÔNIO, D fica sendo igual a 1; se for JOAQUIM fica sendo 2; se não for nenhum dos indicados nas linhas 40 até 120, só pode ser JOSÉ e D será igual a 10.

Na linha 140 será montada a matriz de contagem:

T(C,D) é equivalente a um voto dado ao candidato D, para o cargo C.

Assim, cada vez que aparecer o mesmo conjunto C, D (o mesmo candidato votado para o mesmo cargo) o valor de T(C,D) aumenta de uma unidade. Dessa forma, no fim da contagem, quando A$ ="FIM" e B$ = "FIM" (linha 30), teremos o total de votos para cada conjunto de valores C e D.

Na linha 150 colocamos o título da tabela, e a linha 160 manda imprimir os nomes dos candidatos e o número de votos que cada um obteve para cada cargo:

Por exemplo: (antes de executar, veja item 9.4)

```
RUN
NOME        PRES    V.PRES    TES
ANTÔNIO      6        3        0
JOAQUIM      3        0       15
MARCELO      0       12        7
ROBERTO      0        2        0
PAULO        2        5        0
PEDRO        5        5        5
JOÃO         2        0        0
MANOEL       0       10        8
RAUL         1       13        2
JOSÉ         1        0       10

OK
```

Recomendamos ao leitor fazer vários exemplos semelhantes a este, com variantes, para entender bem o processo do uso das variáveis indexadas e do contador. No caso, o ponto mais importante desse contador foi o fato de que T (C,D) também é uma variável de duplo índice e os valores de C e D apareceram no programa por processamento interno. Isto mostra que os índices podem ser definidos dentro do programa, por um meio qualquer, desde que sejam inteiros e positivos ou nulos.

9.4 — Instrução DIM

Quando trabalhamos com variáveis indexadas, a linguagem BASIC estabelece uma certa quantidade de memórias para armazenar os valores dessas variáveis.

Em outras palavras, o programa precisa saber, antes de começarem a aparecer as variáveis indexadas, qual é o tamanho máximo do "array" que se terá, ou, em outras palavras, quantos elementos existirão nas matrizes ou nos vetores (índices bidimensionais e unidimensionais).

Na maior parte das versões de BASIC não é necessário declarar tamanhos de "arrays" com menos de 11 elementos. Isto significa que, se tivermos uma variável do tipo:

```
A(K)
```

e, se K na execução do programa não for maior do que 1Ø, não será preciso dar uma instrução especial declarando qual o tamanho das matrizes (ou seja, declarando qual o maior valor que se espera que K possa atingir).

A instrução DIM (oriunda de "DIMENSION" ou DIMENSÃO) é usada da seguinte maneira:

Logo no início do programa deve ser escrito:

```
DIM A(2Ø)
```

Nesse exemplo, estamos declarando que a matriz A(K) poderá, na execução do programa, ter valores A(Ø), A(1) - - - - A(2Ø). O programa vai guardar 2Ø posições para armazenar os valores de A(K). Quando não é usada a instrução DIM, o programa, automaticamente, guarda 11 posições (de Ø a 1Ø).

No caso de índice duplo, a instrução será do tipo:

```
DIM B(3Ø, 5Ø)
```

o que significa que a variável B poderá assumir índices desde B(Ø,Ø) até B(3Ø, 5Ø).

No caso de variáveis com duplo índice é sempre necessário declarar a dimensão. Dessa forma os exemplos anteriormente mostrados têm que ter o seguinte acréscimo:

Ex. 1Ø: Acrescentar: **15 DIM A$(4,5)**

Ex. 11: Acrescentar: **5 DIM T(3,1Ø)**

É importante notar o seguinte: quando se faz uma declaração de dimensão, especialmente de variável duplamente

indexada, o número de casas de memória reservadas pelo programa pode se tornar bastante grande.

Exemplo: quando declaramos DIM A(2Ø, 3Ø), estamos reservando 600 valores para a variável A(M,N). Além disso o computador precisa escrever o próprio nome da variável o que toma outra tantas vagas de memória. Na realidade, uma dimensão 2Ø, 3Ø poderá atingir mais de 600 Bytes de memória.

Em resumo, o uso de matrizes é muito útil mas ocupa muita memória. Por isso é necessário se calcular com razoável aproximação a dimensão dos "arrays", para evitar declarar dimensões maiores do que o realmente necessário.

As dimensões podem ser declaradas em conjunto, misturando-se variáveis de índice simples, duplo e de STRING.

Ex.: DIM A(1ØØ), B$(1Ø,15), T(8,14)

As dimensões das variáveis são colocadas depois da instrução DIM e separadas umas das outras através de vírgulas.

Em conclusão a este tópico vamos mencionar dois pontos que devem ser bem gravados pelo leitor:

a) quando os índices da variável atingem um valor maior do que o que está indicado na declaração DIM, o programa emitirá uma mensagem de erro:

Ex.:
```
1Ø DIMK(12)
2Ø FOR N= 1 TO 15: K(N) = N:
NEXT
3Ø FOR M = 1 TO 15: ? K(M):
NEXT
RUN
DD ERROR IN 2Ø

OK
```

O que ocorreu foi que, quando o loop da linha 2Ø tentou fazer K(13)=13, o número 13 já era maior do que a dimensão indicada na linha 1Ø e, assim, o programa parou e acusou erro de dimensão.

b) o programa deve ser feito de maneira que a instrução DIM relativa à mesma variável não seja lida mais de uma vez. O programa fica confuso com duas indica-

ções de DIM da mesma variável. Isto ocorre mesmo que a dimensão indicada seja a mesma:

Ex.:
```
10 DIM A(15)
20 FOR N= 1 TO 15: READ A(N):NEXT
30 FOR M= 1 TO 15: PRINT A(M);: NEXT
40 INPUT "QUER CONTINUAR"; S$
50 IF S$ <> "S" THEN END
60 RESTORE : GOTO 10
70 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
80 DATA 11, 12, 13, 14. 15
RUN
 1  2  3  4  5  6  7  8
 9  10  11  12  13  14  15
QUER CONTINUAR ? S
DD ERROR IN 10

OK
```

Quando mandamos o programa continuar, ele voltou para a linha onde encontrou novamente a declaração DIM A(15) e, assim, acusou erro de dupla dimensão (dimensão declarada duas vezes no mesmo programa).

Capítulo X
FUNÇÕES DIVERSAS DE STRING

10.1 – Já vimos que as variáveis de STRING da forma A$ ou A$(K) ou A$(M,N) podem ser tratadas "matematicamente", nos programas em BASIC, embora os STRINGS sejam, apenas, conjunto de caracteres, sem valor numérico.

Já demos, inclusive, diversos exemplos do tratamento que pode ser dado a essas variáveis, em capítulos anteriores. Agora vamos mostrar algumas funções específicas para a manipulação dessas variáveis, ou seja, diversos tipos de transformações e operações que podem ser feitas em STRINGS e que são muito úteis na preparação de certos programas.

10.2 – LEFT$

Essa função, que é escrita da seguinte forma:

```
LEFT $ (A$,N)
```

representa os N primeiros caracteres do STRING A$, a partir da esquerda.

Ex. 1:
```
10 CD$="TEMPERATURA"
20 PRINT LEFT $ (CD$,2)
RUN
TE

OK
```

Quando escrevemos LEFT$(CD$,2), estabelecemos uma função que representa os dois primeiros caracteres do

STRING CD$, ou seja, as letras T e E. Como mandamos imprimir essa função, foi impresso TE.

O valor do parâmetro que indica a quantidade de letras do STRING que serão representadas pela função LEFT$ pode ser uma variável, desde que positiva e inteira, variável essa determinada dentro do próprio programa:

Ex. 2:
```
10 A$ = "ABCDEFG"
20 INPUT "QUANTAS LETRAS"; N
30 K$ = LEFT$(A$,N)
40 PRINT K$: GOTO 20
RUN
QUANTAS LETRAS? 3
ABC
QUANTAS LETRAS? 5
ABCDE
QUANTAS LETRAS? 10
ABCDE
QUANTAS LETRAS?  (<RETURN>)

OK
```

Cada vez que foi dado um valor a N, o programa imprimiu o número correspondente de caracteres. Quando foi pedido um valor maior do que o comprimento do STRING, só imprimiu os caracteres existentes no STRING.

Ex. 3:
```
10 A$ = "ABCDEFGHIJ"
20 N = INT(RND(1)*10+1)
30 ? LEFT$(A$,N) : M = M + 1
40 IF M> 6 THEN END
50 GOTO 20
RUN
ABC
A
ABCDEFG
ABCD
AB
ABCDE
A
OK
```

Desta vez fizemos N ser um valor inteiro, aleatório, entre 1 e 1Ø e mandamos o programa imprimir 7 vezes a função LEFT$, a cada vez com um valor de N.

Ex. 4:
```
5 A$ = "ABCDEFG"
1Ø FOR N= 1 TO 5
2Ø ? LEFT$(A$,N) : NEXT
RUN
A
AB
ABC
ABCD
ABCDE

OK
```

Neste programa, usamos o "loop" FOR/NEXT para variar os valores de N de 1 a 5 e imprimir o número de caracteres correspondente.

Se nós lembrarmos a função LEN(A$), que foi explicada em capítulo anterior, poderemos usar o próprio comprimento do STRING como valor máximo do parâmetro, no "loop" FOR/NEXT e, assim, escrever o STRING inteiro. Lembrando que LEN(A$) é o número de caracteres do STRING A$ (incluindo espaços, vírgulas, sinais, etc.), podemos fazer o seguinte:

Ex. 5:
```
1Ø A$ = "CAFÉ COM LEITE"
2Ø FOR N = 1 TO LEN (A$)
3Ø ? LEFT$(A$ ,N) : NEXT
RUN
C
CA
CAF
CAFÉ
CAFÉ                    (*)
CAFE C
CAFE CO
CAFE COM
CAFE COM                (*)
CAFE COM L
```

```
CAFE COM LE
VAFE COM LEI
CAFE COM LEIT
CAFE COM LEITE

OK
```

(*) o número de caracteres não é igual ao anterior; há também, o espaço depois de última letra, que é invisível.

Ex. 6:
```
50 INPUT "QUER CONTINUAR"; A$
60 IF LEFT$(A$,1) <> "S" THEN END
```

Este é um pedaço de programa igual a muitos que já exemplificamos; a diferença é que não é necessário responder apenas com a letra S. Podemos teclar SIM (ou qualquer outra palavra começada por S), e o efeito será o mesmo, porque a função LEFT$(A$,1) só leverá em consideração a primeira letra da palavra que for teclada pelo operador, em resposta à pergunta da linha 50.

10.3 — RIGHT $

É uma função simétrica à anterior, com a diferença de contar os caracteres da direita para a esquerda.

Ex. 7:
```
10 A$ = "ABCDEFGH"
20 PRINT RIGHT$(A$,2)
RUN
GH

OK
```

Foram impressos os dois últimos valores à direita STRING.

Sugerimos ao leitor experimentar os exemplos mostrados para a função LEFT$, dessa vez usando RIGHT$.

10.4 – MID$ (A$,K,I)

Essa função representa os I caracteres do STRING A$, a partir do caracter K, ou seja, um sub-STRING de A$.

Ex. 8:
```
10 A$ = "ABCDEFGHIJKL"
20 ? MID$ (A$, 3, 2)
RUN
CD

OK
```

Foram impressos dois caracteres, a partir do terceiro (3,2).

Vejamos uma aplicação interessante desta função.

Ex. 9:
```
10 FOR N = 1 TO 10: READ A$(N): NEXT
20 INPUT "SOBRENOME"; Q$
25 Q = LEN(Q$)
30 FOR M= 1 T 10 : FOR T= 1 TO LEN
(A$(M))
40 IF MID$ (A$(M), T, Q)= Q $ THEN T =
LEN(A$(M)): ? A$(M)
50 NEXT T, M
60 DATA "JOÃO DA SILVA, "ANTÔNIO
SOARES", "PEDRO SILVEIRA", "JOSÉ DA
SILVA"
70 DATA "ANTÔNIO DA SILVA", "RENATO
SOARES", "LUIS SILVEIRA", "MANOEL
SOARES"
80 DATA "JOAQUIM DA SILVA", "FERNAN-
DO SOARES"
90 GOTO 20
RUN
SOBRENOME? SILVA
JOÃO DA SILVA
JOSÉ DA SILVA
ANTÔNIO DA SILVA
JOAQUIM DA SILVA
SOBRENOME? SOARES
ANTÔNIO SOARES
```

```
RENATO SOARES
MANOEL SOARES
FERNANDO SOARES
SOBRENOME?
```
(teclamos <RETURN>)

```
OK
```

Vejamos o que se passou:
Na linha 1Ø mandamos o programa ler 1Ø nomes:

```
A$(1) = JOÃO DA SILVA
A$(2) = ANTONIO SOARES
```

e assim por diante.

Na linha 2Ø o programa pergunta um sobrenome, que será teclado pelo operador. Esse sobrenome, nesse momento, passa a ser representado pela variável de STRING Q$. No exemplo mostrado, a primeira resposta do operador fez Q$ ser igual a SILVA.

Na linha 3Ø temos 2 "loops" FOR/NEXT embutidos. A variável M faz com que o primeiro "loop" (mais externo) "chame" as variáveis A$(1), A$(2)... A$(1Ø). A variável T vai fazer o primeiro caracter da função MID$ variar de 1 até o comprimento do A$(M) referente ao M em que estiver o "loop" externo. A variável Q do MID$ é igual ao comprimento de Q$, isto é, o número de letras do sobrenome teclado (linha 25).

Quando houver igualdade entre o sobrenome (Q$) e a função MID$, será impresso o nome inteiro A$(M), para o valor de M em que essa igualdade aconteceu.

Assim, de uma lista de nomes, nós podemos fazer uma lista daqueles que têm o mesmo sobrenome.

Os "loops" da linha 3Ø funcionam da seguinte maneira:

Quando M = 1, A$(M) é igual a A$(1), que é JOÃO DA SILVA. Enquanto M continua sendo igual a 1, o "loop" interno faz T variar de 1 até LEN(A$(M)), isto é, até o número de caracteres do nome JOÃO DA SILVA (incluindo os espaços). Quando o sobrenome SILVA foi teclado, Q$ passou a ser igual a SILVA e Q = LEN(Q$) passou a ser igual ao número de caracteres do nome SILVA (no caso, igual a 5).

Assim quando T = 1, MID$(A$(M),T,Q) é igual a MID$(A$(1), 1,5). Então, nesse momento, a função MID$ é igual a JOÃO + um espaço. Como isso não é igual a SILVA, o loop interno continua e faz T = 2. Agora, temos MID$(A$(1), 2,5), que é igual a OÃO D. Ainda não é igual a SILVA. Quando T for igual a 9, teremos MID$(A$(1), 9,5), que é exatamente igual a SILVA.

Nesse momento será impresso o nome A$(M), que é A$(1), ou seja, JOÃO DA SILVA. Como não adianta mais fazer o valor de T aumentar, pois já analisamos o que interessa com M = 1, fazemos T ser igual a LEN(A$(1)), ou seja, o último valor de T, no loop interno.

Com NEXT M, recomeçamos tudo no segundo nome, A$(2), que é ANTONIO SOARES. Quando T variar de 1 até o comprimento desse nome, não vai encontrar nenhum MID$ que seja igual a SILVA. Assim, o "loop" externo passa para M = 3 e o nome ANTONIO SOARES não é impresso. Com M = 3 acontece a mesma coisa. Com M = 4, vamos encontrar um valor de MID$, que é igual a SILVA e, assim, será impresso o valor de A$(4), que é JOSÉ DA SILVA.

O programa continua até M = 1Ø, quando terminam os loops. Agora, voltamos para a linha 2Ø onde um novo sobrenome é perguntado e serão impressos todos os nomes que tiverem esse sobrenome.

Veja bem que não é necessário que o sobrenome seja o último. A busca de MID$, andando de um em um caracter permite que se escolha um nome do meio ou, até, parte de um nome.

No exemplo acima vamos responder com a palavra DA, quando for perguntado o sobrenome.

```
SOBRENOME ? DA
JOÃO DA SILVA
JOSÉ DA SILVA
ANTÔNIO DA SILVA
```

e assim por diante. . .

Acontece que é preciso tomar um certo cuidado, pois, se houvesse alguém na lista chamado DAGOBERTO ANTUNES, o MID$ reconheceria o DA do nome DAGOBERTO e esse nome seria listado junto com os DA SILVA. Isso pode ser evitado, nesse exemplo, teclando um espaço, as letras

DA e outro espaço. Agora o nome DAGOBERTO não será listado pois o sobrenome que nós queremos tem 4 caracteres: espaço, D, A, espaço.

Uma observação adicional:

Quando não colocamos o segundo parâmetro do MID$ (A$,M,N), isto é, quando escrevemos

```
MID$(A$, M)
```

essa função vai representar todos os caracteres do STRING A$, começando no caracter de ordem M até o fim do STRING.

10.5 — CHR$(N)

Essa função efetua a conversão de um código numérico (N), para o símbolo representativo desse código.

Nos sistemas de computação e, principalmente, de transmissão de dados a distância, todas as letras, sinais, caracteres especiais, assim como sinais de funções especiais, têm um código numérico equivalente. A tabela de códigos mais usada é a denominada ASCII (American Standard Code for Information Interchange), mostrada abaixo.

Verificamos que a letra A é representada pelo número decimal 65, B é 66, e assim por diante. A armazenagem dos caracteres no computador é feita através do código. Assim, quando o computador armazenar na memória uma palavra ou um STRING, ele está, na realidade, armazenando os códigos numéricos correspondentes às letras e aos sinais (inclusive espaço).

Ex.: quando escrevemos

```
A$ =" JOGO "
```

o computador armazena o seguinte:

65 36 61 34 74 79 71 34

CÓDIGO DE CARACTERES ASCII

Código ASCII	Caracteres	Código ASCII	Caracteres	Código ASCII	Caracteres
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X

003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	←
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

Os códigos ASCII estão na forma decimal
LF = próxima linha
FF = alimentação de formulário
C.R. = retorno de carro
DEL = rubout

Além disso existem funções especiais que têm código. Mencionaremos algumas que não são ligadas exclusivamente à transmissão de dados e que podem, portanto, ser usadas no trabalho com um microcomputador em BASIC:

— Código 1Ø — LINE FEED
é uma instrução que faz o cursor do vídeo ou a cabeça de impressão permanecer no mesmo lugar enquanto o papel de impressora ou a linha no vídeo avança de uma posição.

— Código 13 — CARRIAGE RETURN
faz o cursor ou o cabeça da impressora voltar para o início da linha (a mesma linha que estava sendo impressa).

Vamos dar alguns exemplos:

Ex. 10:
```
1Ø P= INT(RND(1) * 26+1)
2Ø PRINT CHR$ (P+64);: M= M+1
3Ø IF M>1Ø THEN END
4Ø GOTO 1Ø
RUN
AGDETXPXKJU

OK
```

Verifique o seguinte: P é um número aleatório, entre 1 e 26. Assim P+64 é um número aleatório entre 65 e 90.

Quando mandamos imprimir CHR$(P+64) o computador converte o número P+64 no seu caracter correspondente e o imprime. Por isso, obtivemos uma seqüência aleatória de letras, ao executar o programa.

Ex. 11: (em modo direto)
```
PRINT "ABCDE"; CHR$(1Ø);"FGHIJ"
      (<RETURN>)
ABCDE
      FGHIJ

OK
```

A função CHR$(1Ø) fez pular uma linha (não esqueça que, nesse momento, o cursor já tinha avançado uma casa, após a impressão da letra E — isso ocorre automaticamente após cada impressão).

Ex. 12: (em modo direto)

```
PRINT "ABCDE"; CHR$(13); "FGH"
        (<RETURN>)
FGHDE

OK
```

Na tela do vídeo acontece o seguinte:

Primeiro aparecem as letras ABCDE e, depois da função CHR$(13), o cursor volta para o início da linha e faz aparecer FGH, apagando os caracteres que estavam nesse lugar. Em uma impressora as letras FGH serão impressas em cima das letras ABC.

Vamos repetir um exemplo já mostrado em capítulo anterior (aspas dentro do STRING).

Ex. 13:

```
1Ø PRINT CHR$(34); "COMPUTADOR"; CHR$(34)
RUN
"COMPUTADOR"

OK
```

O Código 34 representa aspas.

10.6 — ASC (A$)

É a função oposta à anterior. Ela dá o valor do código ASCII do caracter A$.

Ex. 14:

```
PRINT ASC("A")
        (<RETURN>)
 65

OK
```

As aspas servem para mostrar que A é um STRING e não uma variável. Se colocarmos dentro dos parênteses da função ASC um conjunto de caracteres, será impresso o código ASCII do primeiro caracter.

Ex. 15:
```
10 FOR N=1 TO 6: READ X$(N)
20 PRINT ASC (X$(N));: NEXT
30 DATA C,O,D,I,G,O
RUN
 67  79  68  73  71  79

OK
```

O programa leu os valores das linhas de dados, transformou cada um dos símbolos no seu valor ASCII e imprimiu esses valores.

10.7 – STR$ (X)

Essa função transforma um número X em um STRING. Depois dessa transformação o STRING não pode sofrer operações matemáticas, apesar de estar representando um número. No entanto pode ser tratado como um STRING, usando todas as funções aplicáveis a STRINGS.

Ex. 16:
```
10 N= 34.65
20 A$ = STR$(N)
30 ? N * 2 : ? A $
RUN
 69.3
 34.65

OK
```

Nesse caso, quando mandamos imprimir N*2, o computador efetuou normalmente a multiplicação e quando mandamos imprimir A$, ele imprimiu o STRING 34.65 (não esqueça que o espaço na frente do número faz parte do STRING).

Ex. 17:
```
10 FOR X = 1 TO 8: ? X; : NEXT:?
20 FOR X = 1 TO 8: K$ = STR$(X):
PRINT MID$ (X$,2);: NEXT
RUN
 1  2  3  4  5  6  7  8
12345678

OK
```

Explicação: quando mandamos imprimir X, apareceram os espaços normais na frente e atrás dos números. A instrução PRINT (?), no fim da linha 1Ø faz com que o próximos PRINT fiquem em outra linha.

Quando transformamos cada número em um STRING, mesmo que mandássemos imprimir o STRING (K$, no caso), os espaços apareceriam. Então, usamos a função MID$, para mandar imprimir a partir do segundo caracter do STRING.

Desse modo pulamos o primeiro caracter, que é o espaço do sinal.

Em conseqüência, os números foram impressos sem espaços entre eles.

Ex. 18:
```
10 Q= 15.47
20 PRINT LEN(STR$(Q))
RUN
 6

OK
```

Neste exemplo mandamos imprimir o número de caracteres do STRING em que o número Q foi transformado (o espaço do sinal também contou como um caracter).

Vamos dar um exemplo em que queremos imprimir números com a posição da vírgula decimal (ponto, no caso da língua inglesa) sendo a mesma para todos eles.

Primeiro, vejamos o que acontece normalmente:

Ex. 19:
```
10 FOR K = 1 TO 5 : READ A(K) : NEXT
20 FOR K = 1 TO 5 : ? A(K) : NEXT
30 DATA 15.5, 1384.53,.4, 1.538, 12
RUN
```

```
 15.5
 1384.53
 .4
 1.538
 12

OK
```

O ponto decimal não ficou na mesma posição vertical. Cada impressão começou no primeiro caracter do número e na primeira posição da linha.

Agora, vamos fazer a seguinte alteração no programa:

```
20 FOR K = 1 TO 5 : B(K) = LEN (STR$ (INT(A(K)))):
NEXT
25 FOR K = 1 TO 5 : ? TAB (10-B(K));
A(K): NEXT
```

Vamos executar:

```
RUN
                15.5
              1384.53
                .4
                1.538
               12

OK
```

Explicação: A função B(K) vai dar um número que é igual ao comprimento do STRING em que foi transformada a parte interira de A(K). Para exemplificar, se K = 1, A(1) é o número 15.5. Sua parte inteira é 15. Essa parte inteira é transformada em um STRING e o comprimento desse STRING é 3 (dois algarismos mais o espaço do sinal). Assim, B(1) é igual a 3.

Escolhemos uma posição no vídeo para a colocação do ponto decimal, de modo que haja espaço, à sua esquerda, para colocar as partes inteiras dos números. Nesse caso, escolhemos a posição 10.

Agora, mandamos imprimir cada número, A(K), começando na posição 10 menos o comprimento de sua parte in-

teira. Assim, o ponto ficará sempre no mesmo lugar. No caso de K = 2 o número A(2) é igual a 1384.53. Sua parte inteira tem 5 caracteres (com o sinal). Ele será impresso a partir da posição TAB(1Ø-5) o que significa que o ponto ficará na posição TAB(1Ø).

Esse programa apresentou um defeito: o número .4 ficou com a "vírgula" fora do lugar. Isso ocorreu pela seguinte razão:

A parte inteira de .4 é Ø. Assim, o número de caracteres da parte inteira é 2 (zero + sinal). A impressão começa na casa 1Ø-2, ou seja, 8. Nessa casa é colocado o sinal (como o sinal é positivo, não aparece nada).

Na casa seguinte deve aparecer o primeiro algarismo da parte inteira. Acontece que a parte inteira é zero e a linguagem BASIC não imprime o zero à esquerda do ponto decimal. Assim o ponto ficou deslocado de uma casa.

Para corrigir essa deficiência vamos fazer as seguintes alterações no programa:

Nas linhas da DATA vamos deixar todos os números menores que 1 e maiores que -1 (ou seja, que tem a parte inteira nula) no fim da listagem.

```
3Ø DATA 15.5, 1384.53, 1.538,12, .4,
.Ø4, -.2345
```

Agora, vamos separar a linha 2Ø em duas:

```
2Ø FOR K = 1 TO 4: B(K) = LEN(STR$(INT(A(K)))):
NEXT
22 FOR K = 5 TO 7 : B(K) = LEN (STR$(INT(A(K))))
-1: NEXT
```

E, nas linhas 1Ø e 25, fazemos K variar de 1 a 7, já que mudamos a quantidade de números a serem lidos.

O truque é que, na linha 22, subtraímos uma unidade dos valores B(K) correspondentes aos últimos números — B(5), B(6) e B(7). Assim, o ponto decimal avançará de uma casa, ficando no lugar certo (apenas para os últimos números — A(5), A(6) e A(7)).

Vamos listar e executar o programa modificado e ver o que acontece:

```
LIST
 1Ø FORK=1TO7:READA(K):NEXT
 2Ø FORK=1TO4:B(K)=LEN(STR$(INT
(A(K)))):NEXT
```

```
 22 FORK=5TO7:B(K)=LEN(STR$(INT
(A(K))))-1:NEXT
 25 FORK=1TO7:PRINTTAB(10-B(K));
A(K):NEXT
 30 DATA 15.5 , 1384.53 , 1.538
, 12 , .4 , .04 ,-.2345

OK
RUN
              15.5
            1384.53
               1.538
              12
                 .4
                 .04
                -.2345

OK
```

10.8 — VAL (A$)

Essa função serve para voltar a dar a característica de número a um STRING, obtido por STR$, ou a um número que foi transformado em STRING por algum outro método.

Isso é muito útil, pois podemos ter um número, transformá-lo em STRING, fazer manipulação de STRING e, depois transformá-lo, de novo, em número:

Ex. 20:
```
10 A=183:B=438
20 A$=MID$(STR$(A),2):B$=MID$
(STR$(B),2)
30 C$=A$+B$:?C$
40 C=VAL(C$):?C
RUN
183438
 183438

OK
```

Nesse exemplo usamos uma característica dos STRINGS denominada concatenação: o sinal **+**, entre dois STRINGS,

simplesmente faz juntar os dois em um só. Dessa forma conseguimos juntar os dois números e, agora, voltamos a ter um número só, com todas as características matemáticas. Veja que a primeira linha impressa não tem sinal: é o STRING C$. Já a segunda linha é o número C.

Vamos supor que queremos colocar todos os números com apenas duas casas decimais (p. ex.: quando tratamos de valores monetários).

Ex. 21:
```
10 FORN=1TO5:READA(N):NEXT
20 FORN=1TO5:C(N)=LEN(STR$
(INT(A(N)))):NEXT
30 FORN=1TO5:A$(N)=MID$(STR$
(A(N)),1,C(N)+3):NEXT
40 FORN=1TO5:A1(N)=VAL(A$(N)):
?A1(N):NEXT
50 DATA1438.555,12.3,135.3764,1.32,
18.478
RUN
 1438.55
 12.3
 135.37
 1.32
 18.47

OK
```

Explicação:

Na linha 10 lemos A(1)=1438.555,A(2)=12.3, etc.

Na linha 20 definimos os valores C(1), C(2), etc., que representam o número de caracteres do STRING em que é transformada a parte inteira de A(N). Isto significa que, quando N = 1:

A(1)=1438.555
C(1)=5 (quatro inteiros mais o sinal)

Na linha 30 definimos os STRINGS A$(N), que são as partes dos números A(N), que vão do primeiro caracter (sinal) até o comprimento da parte inteira mais 3. Esse valor + 3 representa o ponto decimal e as duas casas decimais que nós queremos como limite.

No caso de N=1:

A(1)=1438.555
C(1)=5
STR$(A(1))= espaço, 1438.555 (é um STRING)
A$(1)=MID$(STR$(A(1)),1,8)= espaço + 1438.55

Na linha 40 definimos novos números A1(N), que são a representação numérica dos A$(N). Desse modo:

A1(1)= sinal, 1438.55 (é um número e não mais um STRING)

Finalmente mandamos imprimir os valores de A1(N). Note que, quando o número só tinha uma casa decimal (12.3), ficou só com essa casa, pois o programa não pode acrescentar o zero, para ficar igual a 12.30 (pelo menos esse tipo de programa).

10.9 — A$(M,N): em algumas versões de BASIC a função MID$(A$,M,Q) é representada por A$(M,N) em que:

A$ = STRING em questão.
M = ordem do primeiro caracter do sub-STRING desejado.
N = ordem do último caracter do sub-STRING desejado (Q=N-M+1).

Ex.:
```
10 A$= "ABCDE "
20 ? A$ (2,4)
RUN
 BCD

 OK
```

Capítulo XI
INSTRUÇÕES E COMANDOS DIVERSOS

11.1 — Daremos, neste capítulo, um breve apanhado sobre outras instruções e comandos encontrados na linguagem BASIC, mas que não são universais, isto é, muitas versões dessa linguagem não possuem uma ou mais dessas facilidades.

Além daquilo que vamos mostrar existem muitas outras instruções encontradas em diferentes versões do BASIC, que é uma linguagem que tem crescido e se adaptado aos novos equipamentos, sistemas e periféricos, assim como às necessidades dos usuários de microcomputadores. Em particular, não abordaremos neste volume as instruções referentes ao uso de discos magnéticos como meios de armazenamento externo de dados.

Essas instruções constituem um tópico à parte, sob a denominação genérica Sistemas Operacionais de Disco, e esperamos poder tratar das mesmas em outro volume, futuramente.

11.2 — PEEK e POKE

Essas instruções têm a finalidade de colocar e ler dados diretamente de endereços de memória definidos na instrução.

11.2.1 — Leitura direta de memória — PEEK

Essa instrução, oriunda de palavra inglesa que significa "espiar", é utilizada da seguinte maneira:

```
PEEK(X)
```

em que X é o endereço absoluto, em numeração decimal, do qual queremos saber o conteúdo.

Ex. 1: (em modo direto)

```
PRINT PEEK (5347)
   (<RETURN>)
127

OK
```

Mandamos o computador imprimir (ou colocar na tela) o valor que está armazenado no endereço de memória nº 5347 (note que nunca devemos colocar pontos, vírgulas ou espaços para separar os milhares, no endereço; o número deve ser escrito de maneira corrida). Esse valor é indicado sob a forma decimal, embora a maneira de armazenar números nas memórias não seja decimal.

Essa instrução é mais usada em relação a endereços que têm alguma coisa a ver com periféricos (no caso, o teclado e o vídeo são considerados periféricos).

A instrução é bastante usada para "limpar" programas, isto é, descobrir erros de programação em casos mais complexos.

Não daremos maiores detalhes sobre o uso dessa instrução, porque as maneiras de utilizá-la dependem significativamente da arquitetura do microcomputador usado e da versão de BASIC existente.

11.2.2 — Escrita direta em memória — POKE

Essa instrução permite ao operador mandar armazenar um determinado número em um endereço de memória especificado. A forma geral da instrução é:

```
POKE X,N
```

em que X é o endereço desejado e N é o número a ser armazenado (tanto X quanto N são escritos sob a forma decimal, embora a armazenagem seja na forma binária).

É preciso, de saída, notar o seguinte:

Um endereço de memória só comporta, em geral, um Byte, que é uma palavra de oito bits (cada bit é um valor binário 1 ou Ø, armazenado eletricamente na memória).

Assim, o maior número decimal que pode ser armazenado na memória é 255.

Portanto, o valor de N não pode ser superior a 255 (de Ø a 255 há 256 valores diferentes que um Byte pode ter).

Além disso, é necessário tomar cuidado com o endereço onde vamos mandar armazenar um número:

- se esse endereço pertencer a uma memória do tipo READ ONLY (apenas leitura), o valor que indicamos não será escrito, pois esse tipo de memória não admite escrita; essas memórias são as que armazenam os programas permanentes do computador, tal como o interpretador BASIC, por exemplo;
- se o endereço pertence a um tipo de memória onde se pode escrever (apagando o que estava escrito antes) é preciso verificar se nesse endereço não existem instruções de operação do computador. Quando o computador é ligado, algumas instruções da memória permanente são passadas para um pedaço da memória não permanente.

  Se essas instruções forem apagadas por um comando POKE, o computador pode parar de funcionar corretamente e terá que ser desligado e ligado novamente. Com isso, todos os programas que estavam em sua memória de trabalho (ou seja, os programas com que o operador estava trabalhando) serão apagados, o que pode ser um grande transtorno.

É necessário que o leitor verifique, no manual do seu computador, quais os endereços de memória disponíveis para o seu trabalho e somente nessas memórias deve ser escrito algum número, através de POKE.

Ex. 2:
```
10 POKE 5347,65
20 PRINT CHR$(PEEK(5347))
RUN
A

OK
```

Na linha 1Ø mandamos armazenar o número 65 no endereço 5347. Na linha seguinte mandamos imprimir o símbolo ASCII do que está na memória 5347 (veja o PEEK (5347)). Esse símbolo é a letra A.

Uma aplicação interessante do comando POKE é o uso da memória de vídeo para colocar gráficos, jogos e outras coisas diretamente em certos pontos da tela.

Quando um computador tem memória de vídeo mapeado (o leitor deve verificar se é o caso do seu computador), isso significa que existe um certo trecho de memória onde cada endereço corresponde a uma posição bem definida na tela.

Vamos exemplificar com o microcomputador do autor:

Os endereços de 53381 até 54141 referem-se a pontos bem determinados na tela. O ponto mais acima, à esquerda, tem, na memória, o endereço 53381; o ponto mais abaixo, à direita, tem o endereço 54141.

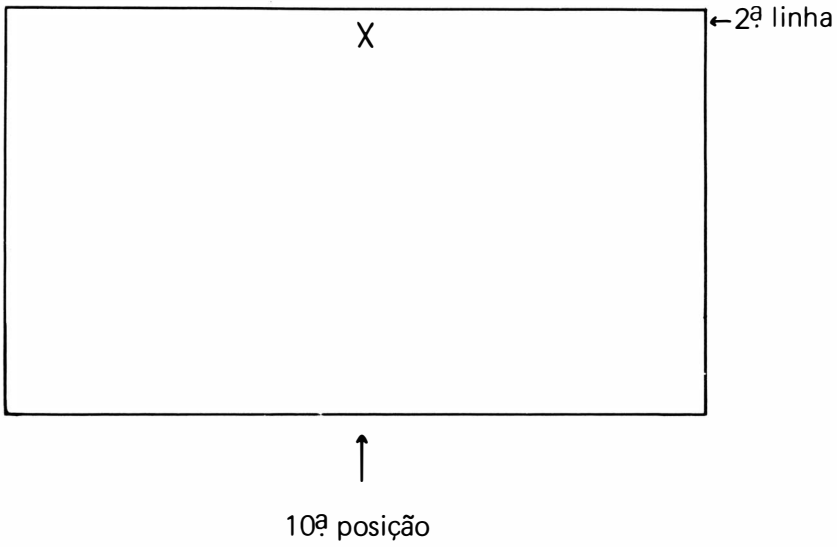
Assim, tudo o que estiver armazenado nesse pedaço de memória estará, automaticamente, colocado na tela.

Ex. 3: (em modo direto)

```
POKE 53423,88
```

(<RETURN>)

a tela ficará assim:



Aparecerá a letra X na 10ª posição da segunda linha da tela (correspondente ao endereço escolhido). O valor 88 é o código ASCII da letra X.

Além disso, o comando que nós escrevemos ainda estará na tela:

```
POKE 53423,88

OK
```

Usando o manual do seu computador o leitor poderá descobrir muitos usos para a instrução POKE, especialmente para o caso de jogos de vídeo.

11.3 – USR(X)

Essa função permite trabalhar com sub-rotinas em linguagem de máquina, dentro de um programa em BASIC. Também nesse caso, a maneira de usar essa função varia conforme a versão do BASIC usado.

De maneira geral quando a função USR é colocada no meio de um programa em BASIC, o programa passa a funcionar em linguagem de máquina, segundo um programa preparado pelo operador e cuja primeira instrução tem o endereço colocado pelo mesmo em um certo endereço fixo, que depende do computador usado.

Suponhamos que os endereços onde devem ser armazenados os Bytes, que indicam o endereço onde começa a rotina em linguagem de máquina, sejam 574 e 575. Lembre-se de que os endereços podem ir de Ø a 65536, o que significa que dois Bytes são, em geral, necessários para definir um endereço. Assim um endereço completo ocupa dois endereços de memória. Em outras palavras, é necessário usar dois endereços consecutivos para escrever o número de um endereço.

Se a sub-rotina em linguagem de máquina começa no endereço 8ØØ (decimal), esse endereço tem que ser transformado em dois Bytes, de oito bits.

Ora, o número 8ØØ, em forma binária, tem o seguinte aspecto:

```
ØØØØØØ11ØØ1ØØØØØ
```

Se dividirmos esse número binário em quatro grupos de 4 bits, teremos o seguinte:

```
ØØØØ  ØØ11    ØØ1Ø  ØØØØ
Byte Alto      Byte Baixo
```

Cada grupo de 4 bits pode ser transformado em um número em notação hexadecimal.

```
Ø     3     2     Ø
```

Assim, o endereço 8ØØ (decimal) é equivalente a Ø32Ø hexadecimal, constituído de dois Bytes: alto = Ø3 e baixo = 2Ø.

Assim, devemos colocar no endereço 574 o Byte Baixo (2Ø) e, em 575 o Byte Alto (Ø3).

Dependendo do tipo de BASIC usado, a instrução

```
X = USR(N)
```

faz com que seja chamada a sub-rotina em linguagem de máquina que começa, nesse exemplo, no endereço 8ØØ (decimal) ou Ø32Ø (hexadecimal. O parâmetro N será transferido para a sub-rotina em linguagem de máquina e o valor de X será transferido para o programa em BASIC, após o término das operações da sub-rotina. A sub-rotina tem que terminar com uma instrução em linguagem de máquina que signifique "retorno de sub-rotina", para o programa poder voltar para BASIC.

Ex. 4:
```
10 A=16
20 X=USR(16)
30 C=A+X:?C
```

Vamos supor que a sub-rotina em linguagem de máquina faça o seguinte:

Adiciona ao valor do parâmetro um valor igual a 16 (decimal) e devolve o valor de X = 32. Teremos, então:

```
RUN
 48

OK
```

Haveria muitas outras coisas a comentar sobre a função USR, mas o leitor deverá consultar o manual do BASIC do seu próprio computador para verificar as peculiaridades de sua utilização no seu caso particular.

Resta mencionar, apenas, que nem sempre é necessário passar um número para a sub-rotina em linguagem de máquina e receber um valor de volta. Às vezes a sub-rotina executa alguma função do tipo "limpar a tela", por exemplo, que não precisa dos valores de N e X.

Em outros casos é necessário passar algum parâmetro para a sub-rotina de linguagem de máquina, porém sem receber nenhum valor de volta, isto é, o valor N será passado para a sub-rotina, mas não haverá nenhum valor de X a receber (ex: colocação de gráficos na tela).

11.4 – ELSE

Essa instrução trabalha sempre em conjunto com IF/THEN, sendo um complemento da mesma.

Já vimos que IF/THEN faz uma comparação e uma escolha (SE acontecer tal coisa ENTÃO faça isso).

Há casos, porém, em que há mais de uma possibilidade, ou seja, uma simples comparação e escolha não são suficientes. Nesse caso será usada a instrução complementar ELSE (que significa "caso contrário"), ficando a expressão completa com o seguinte formato:

IF (certa coisa acontecer) THEN (faça isto) ELSE (caso contrário faça aquilo)

Vejamos alguns exemplos:

Ex. 5:
```
10 INPUT"ESCREVA O NOME JOSÉ";A$
20 IF A$="JOSÉ" THEN?"VOCÊ ESCREVEU O NOME
CERTO: JOSÉ"ELSE?"VOCÊ NÃO ESCREVEU
JOSÉ, ESCREVEU ";A$
30 GOTO 10
RUN
ESCREVA O NOME JOSÉ? JOSÉ
VOCÊ ESCREVEU O NOME CERTO: JOSÉ
ESCREVA O NOME JOSÉ? PEDRO
VOCÊ NÃO ESCREVEU JOSÉ, ESCREVEU PEDRO
```

ESCREVA O NOME JOSÉ? (<RETURN>)

OK

Na linha 2Ø o programa comparou o que foi teclado com a palavra JOSÉ; se a palavra teclada foi JOSÉ, a condição IF foi satisfeita e será impressa a frase correspondente. O programa pula para a próxima linha, que mandava voltar ao início.

Se a palavra teclada não foi JOSÉ, a condição ELSE é satisfeita e será impressa a frase correspondente. O programa pula para a próxima linha e volta para o início.

O importante é notar que, nesse caso, as duas condições contraditórias foram escritas na mesma linha e o programa prosseguiu para a próxima linha em ambos os casos. Não sendo usada a instrução ELSE, o programa teria que ser escrito da seguinte forma:

Ex. 6:
```
1Ø INPUT "ESCREVA O NOME JOSÉ"; A$
2Ø IF A$= "JOSÉ" THEN ? "VOCÊ ESCREVEU O NOME CERTO: JOSÉ": GOTO 1Ø
3Ø PRINT "VOCÊ NÃO ESCREVEU JOSÉ, ESCREVEU "; A$ : GOTO 1Ø
```

Agora a instrução GOTO 1Ø teve que ser escrita duas vezes: uma para cada condição (satisfeita ou não satisfeita).

No exemplo acima a coisa ainda é relativamente simples. Suponhamos, agora, que queremos enviar o programa para uma certa sub-rotina, no caso de IF ser satisfeito e para outra sub-rotina, no caso de não ser satisfeito. Mas queremos que a volta (RETURN) de ambas as sub-rotinas seja para a mesma linha (ou seja, em ambos os casos, o programa continuará a partir do mesmo lugar).

Ex. 7:
```
1Ø INPUT "SUB-ROTINA 1 ou 2 "; A
2Ø IF A=1 THEN GOSUB 5Ø ELSE GOSUB 1ØØ

3Ø (continuação do programa)
```

Na linha 2Ø, se for escolhido A = 1, o programa irá para a sub-rotina 5Ø e voltará para a linha 3Ø. Se A não for igual a 1 o programa irá para a sub-rotina 1ØØ e voltará também para a linha 3Ø, onde o programa continua.

Para fazer a mesma coisa sem a instrução ELSE, teríamos que preparar o programa da seguinte maneira:

```
1Ø INPUT " SUB-ROTINA 1 OU 2"; A
2Ø IF A=1 THEN GOSUB 5Ø
3Ø IF A=1 THEN GOTO 45
4Ø GOSUB 1ØØ
45 (continuação do programa)
```

Verifica-se que o programa ficou mais complicado: agora é necessário, na linha 3Ø, fazer nova comparação de A = 1 e, depois, passar para a linha 45. Isto porque, quando o programa volta da sub-rotina 5Ø, ele vai para a linha 3Ø e nós não queremos colocar a instrução GOSUB 1ØØ nessa linha, pois isso faria com que a sub-rotina 1ØØ fosse executada sempre, logo após a sub-rotina 5Ø (o que não é o desejo do programador). Por outro lado, se nós escrevermos apenas GOTO 45, quando A não for igual a 1, o programa passará sempre para a linha 3Ø e, com essa instrução, não iria nunca para a sub-rotina 1ØØ. Por isso, colocamos nova comparação IF A=1. Quando a condição da linha 2Ø não é satisfeita, o programa passa para a linha 3Ø onde testa outra vez (e, obviamente, não é satisfeita), passando para a linha 4Ø, onde recebe a ordem de ir para a sub-rotina 1ØØ.

Em alguns casos o uso da instrução complementar ELSE facilita ainda mais a leitura dos programas, como o leitor aprenderá por experiência própria.

11.5 — CLS

Em algumas versões de BASIC existe a instrução CLS, que tem a função de limpar a tela. O leitor se recorda de exemplos anteriores em que tivemos de usar do recurso:

```
FOR N= 1 TO 32 : ? : NEXT
```

para limpar a tela.

Usando essa instrução tudo se resume apenas numa palavra.

Ex. 8:
```
10-----
20 CLS
30-----
```

Na linha 20 a tela será apagada.

Essa instrução também pode ser colocada, com outras instruções, na mesma linha.

11.6 – RANDOMIZE ou RANDOM

Em certas versões de BASIC a função RND(X), que já mostramos em capítulo anterior, apresenta sempre o mesmo valor ou conjunto de valores, mesmo quando ela é chamada várias vezes seguidas. Na primeira passada, a função RND escolhe os números aleatórios e, depois, usará os mesmos números, nas próximas chamadas.

Nessas versões do BASIC existe a instrução RANDOMIZE, que indica ao programa que cada passada pela função RND deve produzir um novo número aleatório.

Ex. 9: (sem RANDOMIZE)
```
10 ? "VOU ESCOLHER UM NÚMERO"
20 N= INT (RND(1)*10+1)
30 ? N;: GOTO 20
RUN
VOU ESCOLHER UM NÚMERO
 8  8  8  8  8  8-------
    (<CTRL>C)
BREAK IN LINE 20

OK
```

O "loop" contínuo teve que ser interrompido com o uso de CONTROL C, como já vimos em capítulo anterior. Mas o importante é notar que a função RND escolheu um número aleatório entre 1 e 10, na primeira passada e, quando o programa mandou escolher outro número, o mesmo número 8 foi repetido.

Agora, usando RANDOMIZE, isso não acontecerá:

Ex. 10:
```
10 RANDOMIZE
20 N= INT(RND(1)*10+1)
30 ? "VOU ESCOLHER UM NÚMERO"
40 ? N:GOTO 20
RUN
VOU ESCOLHER UM NÚMERO
 4
VOU ESCOLHER UM NÚMERO
 6
VOU ESCOLHER UM NÚMERO
 8
(<CTRL>C)
BREAK IN LINE 40

OK
```

Dessa vez, cada passada pela linha 20 produziu um número diferente.

11.7 – SET (PLOT)/RESET

Essa instrução existe de maneiras diferentes, conforme a versão de BASIC usada. No caso mais comum, a instrução manda colocar determinado caracter ou desenho em ponto bem especificado da tela.

A posição do ponto é definida por duas variáveis, sob a forma de coordenadas cartesianas.

```
SET (X,Y)
```

Os valores que as variáveis X e Y podem assumir dependem do microcomputador usado e do seu sistema de mapeamento de vídeo e seleção de caracteres. Tipicamente, X sendo a coordenada horizontal, poderá assumir valores de 0 a 127 (ou menos), correspondendo a 128 posições por linha de vídeo; a variável Y pode ir de 0 até 32, em grande parte dos casos, correspondendo a igual número de linhas na tela.

Quando é dada uma instrução

```
SET(X,Y)
```

um ponto, na coordenada X,Y será aceso, na tela.

A instrução RESET(X,Y) manda apagar o ponto na coordenada X,Y.

11.8 – DELETE

Essa instrução permite alterar programas mandando apagar uma linha ou conjunto de linhas de um programa armazenado na memória do computador.

Ex. 11: (em modo direto)

DELETE 1ØØ

(será cancelada a linha 1ØØ do programa)

Ex. 12: (em modo direto)

DELETE 5Ø – 1ØØ

(serão apagadas as linhas de 5Ø a 1ØØ)

Ex. 13: (modo direto)

DELETE – 5Ø

(serão apagadas todas as linhas até a do número 5Ø)

Ex. 14: (modo direto)

DELETE 6Ø –

(apagará as linhas do número 6Ø até o fim do programa)

É natural que esse comando seja usado sempre em modo direto, pois não há sentido em colocar dentro de um programa uma instrução que manda apagar uma ou mais linhas do mesmo programa.

11.9 – PRINT USING

É uma instrução que permite a formatação de um valor a ser impresso.

A maneira mais usada, nas versões de BASIC que tem a instrução PRINT USING, é a formatação de números, indicando quantas casas tem a parte inteira e a decimal ou separando os milhares por meio de vírgulas (isso é útil para imprimir valores monetários).

A maneira de informar a quantidade de casas inteiras e decimais é, normalmente, a seguinte:

```
PRINT USING "##.##";A
```

em que os sinais # correspondem às casas inteiras e decimais, separadas pelo ponto. A é o número ou variável numérica a ser impressa.

Ex. 15:
```
5 FOR N= 1 TO 5: READ A(N): NEXT
10 FOR N= 1 TO 5
20 PRINT USING "###.###"; A(N):NEXT
30 DATA 138:65, 1.38, .13865, 1386.5, 13865
RUN
138.650
1.380
.139          (houve arredondamento)
1386.500      (*)
13865.000     (*)

OK
```

Serão obedecidos, na impressão, os tamanhos dos campos inteiro e decimal indicados na linha 20, exceto quando o número tem mais casas inteiras do que a formatação indica. Nos casos marcados com asterisco, dependendo da versão de BASIC usada, poderá aparecer uma mensagem ou sinal de erro, porque colocamos mais algarismos inteiros do que o PRINT USING pede.

Note-se o seguinte: o símbolo "##.##" é um STRING e pode receber o nome de uma variável de STRING.

O exemplo acima poderia ser escrito da seguinte forma:

```
0 A$ ="###.###"              (colocamos a linha 0)
20 PRINT USING A$; A(N): NEXT
```

Outros tipos de formatação:

— Para STRING

```
PRINT USING "%    %"
```

Os sinais % separam uma certa quantidade de espaços, o que significa que estamos mandando imprimir um STRING com aquela quantidade de espaços + 2 (esses 2 adicionais correspondem aos símbolos %).

Ex. 16:
```
10 A$ = "%    %"          (4 espaços)
20 PRINT USING A$;"ABCDEFG"
RUN
 ABCDEF

OK
```

A$ corresponde a 4 espaços mais dois sinais %. Assim, foram impressos 6 caracteres do STRING "ABCDEFG".

— Com cifrão

Ex. 17:
```
PRINT USING "$$##.##"; 33.474
                (<RETURN>)
$ 33.47

OK
```

Um duplo cifrão dentro do STRING faz aparecer um cifrão na impressão.

— Com notação exponencial

Ex. 18:
```
PRINT USING "##.##[[[[";15.18
                (<RETURN>)
 1.518E+01

OK
```

Existem, possivelmente, muitas outras formas de usar instrução PRINT USING. O leitor deve consultar o manual do BASIC do seu computador para ver quais as facilidades que o mesmo apresenta para essa instrução (se existir).

11.10 — OPERAÇÕES LÓGICAS

Já vimos anteriormente que as operações lógicas AND, OR e NOT podem ser usadas em relação à instrução IF/THEN.

Agora mostraremos que esses operadores podem ser usados para a manipulação de bits.

Quando comparamos dois números, por meio dos operadores AND, OR e NOT, esses números são convertidos em números binários, na notação "complemento de dois".

A notação "complemento de 2" em 16 bits significa que o maior número positivo que pode ser escrito é 32767, que tem a forma

0111111111111111

Os números negativos, na forma "complemento de 2", são escritos com 1 na 16ª casa (mais à esquerda) e com os valores dos 1 e 0 se posicionando da seguinte maneira:

A forma binária normal dos números positivos é simetrizada, isto é, cada 0 é transformado em 1 e vice-versa e ao resultado é somado o valor 1.

Assim, - 2 equivale a

1111111111111110

(2 é igual a 0000000000000010, seu recíproco bit a bit, também chamado complemento de 1, é equivalente a 1111111111111101 e, somando 1 a esse número, encontramos o complemento de 2, negativo, mostrado acima).

Quanto usamos os operadores AND, OR e NOT, essas operações são efetuadas nos bits de mesma localização correspondentes aos números comparados entre si.

Ex. 19:
```
10 A=16 : B = 7
20 C= A OR B
30 PRINT C
RUN
 23

OK
```

Explicação:

A = 16	equivale a	10000
B = 7	equivale a	00111
A OR B	equivale a	10111

ou seja, C = 23

Ex. 20:
```
10 A = -1 : B = 12
20 C = A AND B
30 ? C
RUN
 12

OK
```

Explicação:

A= -1 equivale a 1111111111111111
A= 12 equivale a 0000000000001100
A AND B equivale a 0000000000001100

ou seja, C = 12

Ex. 21:
```
10 A = 8 : B = 7
20 C = A AND B
30 PRINT C
RUN
 0

OK
```

Explicação:

A= 8 equivale a 1000
B= 4 equivale a 0100
A AND B equivale 0000

ou seja, C = 0

O operador NOT, usado na forma bit a bit, inverte os bits de um número, dando o complemento de 1 desse número em binário de 16 bits.

Dessa forma:

NOT 1 = -2
1 equivale a 0000000000000001
NOT 1 equivale a 1111111111111110

que é igual a --2 (em complemento de 2)

De forma geral, NOT N = -(N+1)

Ex. 22: NOT -1 equivale a Ø
NOT Ø equivale a -1
NOT 3 equivale a -4

Com este capítulo encerramos o que se poderia denominar BASIC fundamental. Há muitas outras instruções, variando conforme a versão do BASIC, mas todas elas razoavelmente compatíveis entre si.

O leitor deverá procurar a melhor forma de compatibilizar as explicações dadas nos capítulos anteriores e neste com o conjunto de instruções da linguagem BASIC que roda no computador com que estiver trabalhando.

Capítulo XII
MENSAGENS DE ERRO

12.1 — Conforme já foi mencionado anteriormente, o interpretador BASIC "imprime" uma mensagem de erro toda a vez que uma instrução ou comando ilegal ou ininteligível é introduzido, seja pelo teclado, seja através de um meio externo (p. ex., fita cassete).

Quando qualquer coisa é escrita, fora de uma linha numerada e é apertada a tecla <RETURN>, isto é interpretado como um comando, em modo direto. Se o computador não entender ou não puder executar o comando emitirá imediatamente uma mensagem de erro.

Quando um erro é cometido dentro de uma linha numerada, ou seja, dentro de um programa, o computador não faz a crítica imediatamente. O erro só será detectado quando o programa for executado, isto é, depois do comando RUN.

O formato geral da mensagem de erro pode variar de computador para computador e depende do tipo de BASIC usado.

Genericamente, a mensagem de erro tem a forma

```
XX ERROR IN YY
```

em que XX é um código (em geral de duas letras) que indica qual o tipo de erro e YY é o número da linha onde se encontra o erro. No caso de comando direto, não aparece o número da linha:

```
XX ERROR
```

Vamos indicar, a seguir, os tipos de erro mais usualmente encontrados nos programas em BASIC, mencionando os seus códigos (que, como já dissemos, podem variar conforme a máquina usada).

12.2 — DD — (Double Dimension)

Essa indicação de erro ocorre quando uma variável indexada é dimensionada mais de uma vez; isso significa que a instrução DIM, para a mesma variável, é lida mais de uma vez no mesmo programa ou que há mais de uma instrução DIM para a mesma variável.

Ex. 1: DD ERROR IN 5Ø
(é indicada a linha em que é lida a instrução DIM adicional).

12.3 — FC — (Function Call Error)

Significa que o parâmetro associado a uma função ficou fora dos limites admissíveis para o BASIC ou que o parâmetro é incompatível com a função a ser executada.

Ex. 2: **? SQR(-1)** (modo direto)
(<RETURN>)
FC ERROR

OK
(o programa não pode computar raiz quadrada de número negativo)

Ex. 3: **1Ø A$ = "AAAAAAA"**
2Ø ? MID$(A$,Ø,2)
RUN
FC ERROR IN 2Ø

OK
(o programa não aceitou o parâmetro Ø, na função MID$).

12.4 — ID — (Illegal Direct)

Esse erro é indicado quando se tenta entrar em modo direto uma instrução que não é permitida nesse modo.

Ex. 4: **INPUT A** (em modo direto)
(<RETURN>)
ID ERROR

OK
(o BASIC não aceita INPUT em modo direto)

12.5 – NF – (NEXT Without FOR)

Significa que o programa encontrou uma instrução NEXT sem ter havido antes um FOR correspondente.

Ex. 5:
```
10 READ A(K)
20 NEXT K
30 DATA 1,2,3,4,5
RUN
NF ERROR IN 20

OK
```
(foi esquecido o FOR correspondente ao NEXT que aparece na linha 20)

12.6 – OD – (Out of Data)

Significa que em uma instrução READ/DATA o programa encontrou mais instruções READ do que dados disponíveis em DATA.

Ex. 6:
```
10 READ A(K)
20 PRINT A(K): K = K+1
30 GOTO 10
40 DATA 1,2,3,4,5
RUN
 1
 2
 3
 4
 5
OD ERROR IN 10

OK
```

(o programa funcionou até acabarem os dados; no próximo retorno para linha 1Ø ocorreu o erro)

12.7 – OM – (Out of Memory)

Significa que o programa foi longo demais ou contém muitos "loops" (FOR/NEXT, GOSUB, etc.) ou variáveis demais (especialmente as duplamente indexadas).

Esse erro pode ocorrer no meio da execução de um programa, pois muitos programas vão gerando e armazenando dados durante sua execução. No momento em que acabar o espaço de memória será emitida a mensagem de erro.

12.8 – OV – (Overflow)

Introdução de um parâmetro ou resultado de um cálculo sendo igual a um número grande demais para o BASIC.

(Esse valor pode variar conforme a versão da linguagem usada).

12.9 – SN – (Syntax Error)

Erro de sintaxe, significa que uma instrução foi mal teclada, com erro de ortografia ou que o BASIC não entendeu a instrução ou qualquer outro dado fornecido.

Ex. 7:
```
A                (<RETURN>)
SN ERROR
```
(entramos com a letra A e teclamos RETURN ; o BASIC não entendeu do que se trata)

Ex. 8:
```
1Ø INPUT "TECLE UMA LETRA ATÉ J"; A$
2Ø IF A$ "J" THEN END
3Ø ? "TECLOU A LETRA"; A$
RUN
SN ERROR IN 2Ø

OK
```

(faltou um sinal entre A$ e "J", indicando a desigualdade)

12.10 – RG – (RETURN Without GOSUB)

Significa que o programa encontrou uma instrução de RETURN sem que, antes, tivesse havido um envio para sub-rotina (GOSUB). O programa não sabe para onde retornar, já que deveria retornar para a linha seguinte ao GOSUB.

12.11 – US – (Undefined Statement)

É uma instrução não completamente definida. Por exemplo, mandamos o programa pular para uma linha cujo número não existe.

Ex. 9:
```
10 GOTO 30
20 END
RUN
US ERROR IN 10
```
(a linha 10 manda pular para a linha 30, que não existe)

12.12 – /Ø (Divisão por Zero)

Quando um cálculo qualquer conduz a uma divisão por zero, essa mensagem é emitida.

Ex. 10:
```
10 FOR N= 1 TO 5
20 PRINT 5/(N-3): NEXT
RUN
-2.5
-5
/Ø ERROR IN 20
```
(o programa foi sendo executado até que houve a tentativa de dividir 5 por zero, no caso de N = 3)

12.13 – CN – (Continue Errors)

Esse código aparece quando se tenta usar o comando CONT para dar prosseguimento em um programa que foi interrompido (por exemplo, com a instrução STOP) e que não pode ser continuado.

Esse caso pode ocorrer se durante a interrupção o programa foi alterado, de modo que não possa haver prosseguimento. Há outros casos em que não se pode continuar o programa, como, por exemplo, se a interrupção foi devida a um erro.

12.14 – LS – (Long String)

Esse erro aparece quando se tenta processar um STRING maior que 255 caracteres.

12.15 – OS – (Out of String Space)

É um código que indica que o espaço reservado para STRINGS na memória foi excedido. Isso ocorre quando se introduz ou processa uma quantidade muito grande de STRINGS, os quais ocupam muito espaço de memória.

12.16 – ST – (String Temporaries)

Esse tipo de erro ocorre quando se tenta colocar uma fórmula muito complicada envolvendo STRINGS; nesses casos é preferível tentar subdividir a expressão em outras menos complexas.

12.17 – TM – (Type Mismatch)

Esse código é produzido quando há incompatibilidade de variáveis, isto é, quando se tenta dar um valor numérico a um STRING ou associar um STRING a uma variável numérica.

Ex. 11:
```
10 A=18: B$= STR$(2)
20 ? A; B$
30 C = A+B$: ? C
RUN
18 2
TM ERROR IN 30
OK
```
(quando a linha 30 tentou somar uma variável numérica com um STRING ocorreu o erro)

12.18 — UF — (Undefined Function)

Esse tipo de erro ocorre quando não se define corretamente uma função; por exemplo, quando não são fornecidos todos os parâmetros necessários.

Ex. 12:
```
10 DEF FNA(X) = FNB(X) + 5
20 ? FNA(1)
RUN
UF ERROR IN 20
```
(quando o programa tentou imprimir o valor de FNA(1) descobriu que o valor de FNB(X) não havia sido definido)

12.19 — BS — (Subscript Out of Range)

Esse código indica que uma variável excedeu a quantidade de índices dimensionada ***a priori*** ou que uma variável de índices simples excedeu o índice 10, sem a indicação de DIM.

Ex. 13:
```
10 DIM A(13)
20 FOR N = 1 TO 20 : READ A (N)
30 ? A (N); : NEXT
40 DATA 1, 2, 3, 4, 5, 6, ----------20
RUN
 1   2   3   4   5   6   7
 8   9   10   11   12   13
BS ERROR IN 20
```
(depois de ler 13 valores o programa acusou erro na linha 20, pois não podia ler A(14), já que tinha sido estabelecido que DIM A(13)

Antes de concluir, vamos relembrar ao leitor que as várias versões de BASIC apresentam códigos de erro às vezes um pouco diferentes entre si. No entanto, não será difícil associar as explicações acima com o caso particular do seu computador.

TABELA DE ERROS (coloque na primeira coluna os códigos de seu computador)		
CÓDIGO		D E S C R I Ç Ã O
	DD	Variável dimensionada mais de uma vez.
	FC	Parâmetro não aceitável para a função.
	ID	Comando direto não admitido.
	NF	NEXT sem FOR.
	OD	Fim de dados; mais READ do que dados.
	OM	Memória totalmente ocupada.
	OV	Parâmetro ou resultado grande demais.
	SN	Erro de sintaxe.
	RG	RETURN sem GOSUB.
	US	Instrução mal definida.
	/Ø	Divisão por zero.
	CN	Impossível continuar o programa.
	LS	STRING maior que 255 caracteres.
	OS	Foi excedido o espaço para STRING na memória.
	ST	Fórmula de STRING muito complexa.
	TM	Variáveis numéricas e de STRING incompatíveis com os dados.
	UF	Função indefinida.
	BS	Índice maior do que a dimensão declarada.

APÊNDICE

VARIANTES E COMPLEMENTOS DAS INSTRUÇÕES DO BASIC

Nos capítulos deste livro, tal como composto em sua primeira edição, foram explicados diversos comandos e instruções usualmente encontradas no BASIC versão MICROSOFT.

Com o advento dos microcomputadores nacionais tornou-se necessário acrescentar ao texto uma série de variantes usuais das instruções explicadas anteriormente, assim como complementar o "vocabulário" do BASIC com aqueles comandos ou instruções que não haviam sido incluídas na primeira edição.

1. Comandos de gravação e leitura de fita cassete

Além da forma geral explicada anteriormente, com o uso dos comandos LOAD e SAVE, é muito usual a particularização de um comando exclusivo para fita cassete, na forma CLOAD e CSAVE; no caso mais geral, em diversos tipos de microcomputadores os comandos LOAD e SAVE podem, também, ser usados para gravação em disco, desde que a máquina possua o sistema operacional adequado.

CLOAD — é a forma mais usual do comando de leitura de fita cassete; em alguns tipos de BASIC, o comando CLOAD pode ser seguido de um rótulo, entre aspas, constituído de um ou mais caracteres (em geral letras do alfabeto). Dessa forma, ao se dar o comando CLOAD " AB ", por exemplo, o único programa que será lido da fita será aquele que tiver o título "AB".

Uma observação importante é que o comando CLOAD provoca, na maioria dos casos, o apagamento dos programas

já armazenados na memória do computador. Assim, a menos que se usem artifícios (que dependem do computador usado), não é possível ler dois programas seguidos, de fita. O segundo CLOAD apagará o primeiro programa lido.

A maneira de se ver, no vídeo, o processo de leitura depende, usualmente, da velocidade de gravação.

Na maioria dos casos, em que a velocidade de gravação é superior a 300 baud (usualmente 500, 1 200 ou até 2 000), o programa lido da fita não aparece no vídeo; o que aparece é um sinal que indica que está havendo leitura; quando termina a leitura o computador o indica, através de READY (ou OK).

Também é comum encontrar-se microcomputadores em que o controle remoto do gravador é controlado por aqueles, fazendo o motor do gravador andar, logo após o comando CLOAD, e parar quando o programa está todo carregado.

CSAVE – é a maneira mais usual de comandar a gravação de um programa em fita cassete; em alguns tipos de BASIC esse comando pode ser seguido de um título, entre aspas, contendo um ou mais caracteres; esse título será reconhecido pelo comando CLOAD "título".

A forma geral do comando CSAVE é:

CSAVE ou CSAVE "XX"

O comando CSAVE provoca a gravação em fita dos programas armazenados na memória do computador, não permitindo a gravação de outras instruções, como mencionado em relação a algumas aplicações de SAVE.

2. Outros Comandos

– CLEAR N – Em alguns tipos de BASIC a instrução CLEAR pode ser seguida de um número, indicando que o computador reservará aquela quantidade de Bytes para a armazenagem de STRINGS:

Ex. 1: (modo direto)
CLEAR 100

significa que serão reservados 100 Bytes para a armazenagem de STRINGS.

É bastante comum que os interpretadores BASIC deixem reservados 50 Bytes para a armazenagem de STRINGS; nesse caso, se o total de Bytes do STRINGS usados no programa não exceder 50, não será preciso indicar CLEAR N.

O comando CLEAR é, freqüentemente, usado com a abreviação CLR.

– LPRINT – é um comando que permite fazer a impressão de STRINGS, números e variáveis em uma impressora, ao invés de colocar esses caracteres no vídeo. Todas as características do comando PRINT, já vistas, são em geral, aplicáveis ao comando LPRINT.

Ex. 2: (comando direto)
LPRINT "ABACAXI"

Ex. 3: 1Ø LPRINT "BOM DIA"

Ao serem executados esses comandos, as palavras ABACAXI e BOM DIA serão impressas no papel e não aparecerão no vídeo.

É preciso prestar atenção pois, dependendo do tipo de impressora usada, o comando LPRINT pode variar um pouco ou apresentar outras características.

– LLIST – quando se deseja listar um programa em uma impressora, o comando LIST é um pouco modificado: LLIST indica ao computador que a listagem não será feita no vídeo e sim impressa em papel. Todas as demais características do comando LIST são válidas para LLIST.

– FRE (A$) – Em alguns tipos de BASIC pode-se colocar uma variável de STRING, como argumento de função FRE, da forma FRE(A$), o que dará como resultado a quantidade de Bytes livres para a armazenagem de STRINGS; a variável A$ não tem maior significado, indicando apenas que se trata de uma função de STRING; em vez de A$ pode-se usar também uma letra entre aspas: FRE ("B").

Para concluir, é necessário mencionar que o elenco de comandos e instruções utilizadas na linguagem BASIC cresce a cada dia, com o lançamento de novos computadores.

É nossa intenção, em um outro volume, complementar a descrição da linguagem, mostrando os acréscimos e adaptações mais usualmente encontradas, em relação ao núcleo explicado neste texto.

BIBLIOGRAFIA

BROWN, Jerald R. ***Instant Basic***. Dilithium Press, 1978.

ALBRECHT, Finkel & Brown. ***Basic — A Self Teaching Guide,*** John Wiley & Sons, Inc, 1979.

TRACTON, Ken. ***The Basic Cookbook,*** TAB Books, 1978.

PEREIRA FILHO, Jorge da C. ***Basic Básico.*** Editora Campus, 1981.

Manual do Usuário, Microcomputador Ohio Scientific, Modelo Superboard II.

ALBRECHT, Finkel & Brown. ***Atari Basic — A Self Teaching Guide,*** John Wiley & Sons, Inc, 1969.

LIEN, David A. ***The Basic Handbook***, Compusoft Publishing, 1981.

ÍNDICE ANALÍTICO

ROBERTO KRESCH

Engenheiro de Telecomunicações.

Formado em Engenharia Elétrica, especialidade Telecomunicações, pela Escola Nacional de Engenharia da Universidade do Brasil – atual UFRJ.

Cursos de aperfeiçoamento em Engenharia Econômica, Programa de Computadores Eletrônicos e outros na área de eletrônica – na UFRJ, PUC e Universidade de Colúmbia.

Professor da cadeira de Telefonia na Escola de Engenharia da UFRJ; e da Mini Universidade do Congresso Intelcom 80.

Trabalhou na Standard Electrica S/A, ocupando diversas posições no Departamento de Telefonia; Engenheiro de equipamentos de microondas na Western Electric Co, Bell System, Nova York; Superintendente Geral de Engenharia de Comutação da CTB – atual Telerj – durante vários anos, sendo responsável pelo planejamento, projeto, acompanhamento de implantação e testes de todos os equipamentos de comutação.

Secretário de Planejamento e Tecnologia da Secretaria Geral do Ministério das Comunicações.

Participou ativamente da elaboração das Normas Brasileiras de Sinalização Telefônica, Numeração, Encaminhamento e Comutação; coordenou os estudos que definiram as diretrizes para a fabricação e utilização de Centrais Telefônicas Controladas por Programa Armazenado no País; atuou como Coordenador do Grupo de Trabalho que estudou as diretrizes e a política geral para o desenvolvimento da área de transmissão de dados no Brasil.

Exerceu ainda a função de Diretor Técnico da Coencisa – Indústria de Comunicações.

Atualmente ocupa o cargo de Gerente de Desenvolvimento Industrial, na Diretoria Técnica da DIGIBRÁS – Empresa Digital Brasileira S/A.

ER
Editora Rio
ESTÁCIO DE SÁ
Rua Dona Cecília, 25 – Rio Comprido – RJ
Tels.: 273-2994 e 273-2743