Data Center

MBASIC

BASIC APLICADO
UM ENFOQUE PROFISSIONAL

Esta edição foi preparada exclusivamente
para uso do DATA CENTER

APRESENTAÇÃO

Em 1984, o DATA CENTER inicia suas atividades no campo do ensino de Informática com cursos de BASIC e COBOL.

Um ano mais tarde já eram 3 unidades em São Paulo que, graças à preocupação constante com a qualidade de ensino, deram origem a uma expansão sem igual, gerando, em menos de 3 anos, uma rede de ensino com mais de 20 unidades espalhados em todo território nacional, formando e reciclando profissionais para a área de Informática.

Nesta caminhada, mais de 180.000 (cento e oitenta mil) alunos já participaram de nossos cursos. A experiência didática, adquirida pela rede DATA CENTER, não encontra, nos dias de hoje, similar no mercado.

Em meados de 1986, com o mercado de BASIC e COBOL sedimentados, estudos aprofundados revelaram a crescente procura por profissionais com conhecimentos de aplicativos para a linha IBM-PC.

Tantas foram as solicitações de alunos e empresários diante das novas exigências de mercado que o DATA CENTER apresenta, agora, seu curso integrado de aplicativos.

A metodologia de ensino e dedicação pedagógica, que fizeram do DATA CENTER a maior rede particular no ensino de Informática no Brasil, voltam-se à utilização dos aplicativos dBASE II, dBASE III, WordStar e LOTUS 1-2-3.

O caráter prático, presente nos cursos DATA CENTER, é a tônica que permite a utilização no dia-a-dia dos conhecimentos adquiridos, fazendo-o dominar, por completo, esta nova ferramenta de trabalho: o computador.

Para tanto, reuniu-se num pacote os 4 (quatro) principais aplicativos da linha IBM-PC:

dBASE II e dBASE III – ambos bancos de dados que facilitarão trabalhos como: controle de estoque, mala direta, cadastro de clientes e tudo que necessitar a pesquisa e o trabalho com grande quantidade de dados.

WordStar – o editor de texto consagrado em todo mundo como o mais versátil editor na atualidade, permitindo a emissão e confecção de memorandos, petições, contratos, cartas etc... com personalização e rapidez fantástica.

LOTUS 1-2-3 – o aplicativo que integra banco de dados, planilha eletrônica e gerador de gráficos, facilitando a análise de dados e os trabalhos de controle de vendas, fluxo de caixa, folha de pagamento, orçamento e tudo mais que depender de planilhas e cálculos.

O material didático é rico em exercícios e exemplos, escrito por autores consagrados e de qualidade inquestionável garantindo um apoio adequado ao seu aprendizado.

Tudo isto está, a partir deste momento, à sua disposição no seu DATA CENTER.

Bem-vindo

Álvaro Luis Cruz

BASIC APLICADO
UM ENFOQUE PROFISSIONAL

Rubens da Silva Prates Jr.

Analista de Sistemas do
Centro de Processamento de Dados da
Escola de Administração de Empresas de
São Paulo da Fundação Getúlio Vargas

Revisor de texto:
Raymundo Paula de Arruda

Diagramação e paginação:
José Mesquita

Capa:
AG Comunicação Visual Assessoria e Projetos Ltda.

LIVROS TECNICOS E CIENTIFICOS EDITORA LTDA.

PREFÁCIO

"Por que nós nunca temos tempo suficiente para fazer as coisas certas pela primeira vez, mas sempre temos bastante tempo para consertá-las?"

Dennie Van Tassel

Este livro é o resultado da experiência acumulada pelo autor como profissional de processamento de dados e instrutor em inúmeros cursos de BASIC.

Este material tem um enfoque profissional, ao contrário da maioria dos livros encontrados no mercado, que têm um enfoque "hobista".

Destina-se às pessoas interessadas em desenvolver sistemas em microcomputadores usando a linguagem BASIC.

Pode ser usado em cursos profissionais, em cursos de graduação voltados para a área de informática, ou ainda como material de referência da linguagem BASIC.

Tanto estudantes como profissionais de processamento de dados encontrarão neste livro "dicas" de muita utilidade, que contribuirão para o aprendizado ou aperfeiçoamento na prática da programação.

Foram colocadas inúmeras rotinas de apoio que podem dar início a uma biblioteca de rotinas, e desde que implementada, pode reduzir em muito a tarefa de programação, na medida em que evita a necessidade de se "reinventar a roda".

O conteúdo está baseado no BASIC da MICROSOFT, e por uma razão muito simples: a maior parte dos programas de uso empresarial é desenvolvida nesta versão do BASIC.

Escrever um livro não é tarefa fácil. A publicação deste somente se tornou possível graças à contribuição de Rosemeire Gumbis Dichaune e do Departamento de Treinamento do Banco Crefisul, em especial Vanda Bahcivanji. A eles, meus sinceros agradecimentos.

Um livro, bem como qualquer sistema de computador, está sujeito a erros. O autor agradece sugestões ou críticas que lhe sejam enviadas.

Rubens Prates

São Paulo, Setembro de 1984

INTRODUÇÃO AO BASIC

"Um barril novo preservará por muito tempo a coloração da bebida que primeiramente o impregnou."

Horácio

Nos anos 60, o método mais comum de entrada de dados no computador era por meio de cartões perfurados. Quem se interessasse em aprender programação, encontrava sérias dificuldades, pois os cartões perfurados propiciavam baixo nível de interatividade.

Os professores Kemeny e Kurtz, do Dartmouth College, interessados em tornar os computadores acessíveis a todos os seus alunos e não somente àqueles pertencentes ao departamento de computação, criaram a linguagem BASIC, que em princípio era bastante simplificada e fácil de aprender e usar.

Os principais motivos que fizeram do BASIC uma linguagem bastante popular, obtendo grande sucesso, foram:

– ser de natureza conversacional, e

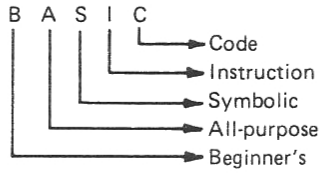
– possuir estrutura simples.

Isto quer dizer que, ao se processar um programa em BASIC, a máquina imediatamente reage, executando o que foi solicitado, e, se necessário, comunicando-se com o usuário.

A princípio, como fora criado para iniciantes em programação, contava com um vocabulário muito limitado, utilizando basicamente palavras em inglês. Com o passar do tempo foi sendo incrementado, tornando-se uma linguagem de alto nível adequada tanto para aplicações comerciais como científicas; entretanto, como não poderia deixar de ser, perdeu um pouco da simplicidade inicial.

Com a crescente popularização dessa linguagem foram sendo desenvolvidas inúmeras versões. Atualmente existem disponíveis no mercado perto de 250 dialetos do BASIC.

Apesar desta "Torre de Babel" de versões, a linguagem BASIC continua sendo a linguagem-padrão de programação para microcomputadores e é uma das linguagens de alto nível mais facilmente assimilada por principiantes em programação.



(Código de Instrução Simbólico de Uso Geral para Principiantes)

SUMÁRIO

ÍNDICE PARA CONSULTA RÁPIDA

1

CONCEITOS FUNDAMENTAIS

". . . se um programa é fácil de ler, ele é provavelmente um bom programa; se ele é difícil de ler, provavelmente ele não é bom."

Kernighan & Plauger

1.1 Conjunto de caracteres do BASIC

O BASIC permite o uso de caracteres alfabéticos, numéricos e os seguintes caracteres especiais:

CARACTER	DESCRIÇÃO
	espaço
;	ponto e vírgula
=	sinal de igual
+	sinal de adição
–	sinal de subtração ou hifen
*	asterisco ou sinal de multiplicação
/	barra inclinada ou sinal de divisão
↑	seta vertical ou sinal de exponenciação
(	abre parênteses
)	fecha parênteses
%	porcentagem
#	libra
$	cifrão
!	ponto de exclamação
[	abre colchete
]	fecha colchete
,	vírgula
'	apóstrofo
.	ponto
:	dois pontos

&	letra "E" comercial
?	ponto de interrogação
<	sinal de menor
>	sinal de maior
\	barra reversa ou sinal de divisão inteira
@	sinal de arroba
"	aspas
_	sublinhado

1.2 Colocando o interpretador na memória

Para colocar o interpretador na memória, digite:

MBASIC ou MBASIC programa

Se o programa for incluído, o interpretador é carregado e o programa é executado.

Exemplos:

(1) MBASIC

O interpretador será colocado na memória

(2) MBASIC PROGX. BAS

O interpretador será colocado na memória e o programa PROGX. BAS será executado.

Após o carregamento do interpretador na memória, será mostrado no video a versão, data da versão, o número de bytes livres e a mensagem "OK".

Neste momento o microcomputador estará pronto para a introdução de programas ou ser usado como simples calculadora.

1.3 Modos de operação do BASIC

Quando o interpretador é carregado na memória, aparece no video a mensagem "OK", significando que ele está pronto para receber comandos. Neste momento, poderá ser usádo de dois modos: imediato ou programado.

1.3.1 Usando o modo imediato

No modo imediato os comandos e declarações não são precedidos de número de linha. A execução vai sendo feita conforme a entrada de comandos ou declarações. Deste modo, os resultados de operações matemáticas ou lógicas são mostrados de imediato e armazenados para uso posterior, mas as declarações são perdidas logo após a execução.

Este modo de utilização é útil para se fazer a depuração de programas, bem como para se usar o microcomputador como uma calculadora, onde são feitos cálculos rápidos que não requerem a elaboração de um programa.

Exemplos:

PRINT 5 + 6 11	Adição
PRINT 600 – 437 163	Subtração
PRINT 10 * 23 230	Multiplicação
PRINT 108 / 12 9	Divisão
PRINT 3 ↑ 3 27	Exponenciação
PRINT 3 * 4 * 10 – 800 – 680	Combinação

1.3.2 Usando o modo programado

O modo programado é usado para a entrada de programas. As linhas dos programas são precedidas por números e ficam armazenadas na memória.

Exemplo:

```
650 LPRINT "Valor Total"; VLTT
660 PRINT "Fim normal"
670 END
```

1.4 Linhas do programa

1.4.1 Formato da linha do programa

O formato da linha de um programa BASIC é o seguinte:

nnn declaração [: declaração] . . . (CR)

Uma linha do programa sempre se inicia com um número e termina com um Carriage Return (CR) e pode conter até 255 caracteres.

Pode-se colocar mais de uma declaração por linha, mas cada declaração deve ser separada por dois pontos (:).

Exemplos:

(1) 160 X$ = INKEY$: IF X$ = " " THEN 160

(2) 300 A = 5 : B = 13 : C = 7

É possível continuar uma linha além do limite do video (80 colunas), continuando a digitar normalmente além dos 80 caracteres. Isso permite que uma linha lógica tenha continuidade na próxima linha física sem pressionar ENTER.

Exemplo:

```
100 COMMON  ALPHA, BETA, CHI, DELTA, EPSIL, ETA, GAMMA, IOTA, KAPPA,
            LAMBDA, MU, NU, OMEGA, OMICR, PHI, PI, PSI, RHO, SIGMA, TAU
```

Dicas:

(1) Evite usar mais de uma declaração por linha. Múltiplas declarações em uma mesma linha economizam espaço de memória, mas dificultam a leitura do programa. Caso sejam colocadas mais de uma instrução na mesma linha, separe-as por um espaço em branco.

Exemplo:

```
60 X$ = INKEY$ : IF X $ = " " THEN 60
```

(2) Se uma declaração deve ser continuada em duas ou mais linhas, coloque a segunda e as sucessivas linhas recuadas da margem esquerda.

Por exemplo:

Não recomendável:

```
500  COMMON ALPHA, BETA, CHI, DELTA, EPSIL, ETA, GAMMA, IOTA,
     KAPPA, LAMBDA, MU, NU, OMEGA, OMICR, PHI, PI, PSI, RHO,
     SIGMA, TAU
```

Recomendável:

```
500 COMMON  ALPHA, BETA, CHI, DELTA, EPSIL, ETA, GAMMA, IOTA,
            KAPPA, LAMBDA, MU, NU, OMEGA, OMICR, PHI, PI, PSI,
            RHO, SIGMA, TAU
```

1.4.2 Numeração das linhas do programa

Os números das linhas devem estar no intervalo de 0 a 65529.

Dicas:

(1) Use apenas os múltiplos de 10, facilitando futuras inserções.

Exemplo:

```
10 DIM A (20)
20 X = 60
```



```
30 '
40 '        Abertura de arquivos
50 '
60 OPEN "I", #1, "CADASTRO · DAT"
```

(2) Numere o programa em blocos lógicos.

Por exemplo:

O programa principal começa na linha 50 e as sub-rotinas a partir da linha 5000 com intervalos de 500 para cada sub-rotina.

1.5 Funções especiais do teclado

Estas funções variam conforme a configuração e versão do software.
A tecla CONTROL (ctl) e a letra correspondente devem ser pressionadas simultaneamente.

Teclas	Função
ctl A	Entra no modo de edição na linha atual. Aparecerá um ponto de exclamação e a linha estará no modo de edição.
ctl C	Interrompe a execução do programa. Torna correntes os disquetes inseridos nos drives.
ctl G	Aciona a campanhia do microcomputador.
ctl H	Apaga o último caracter (backspace).
ctl I	Avança o cursor para a próxima tabulação (a cada oito colunas).
ctl J	Move o cursor para o início da próxima linha (Line feed).
ctl M	Carriage return.
ctl O	Interrompe a saída no video. Outro ctl O reinicia.
ctl R	Repete a linha que está sendo digitada.
ctl S	Interrompe a execução do programa. Qualquer outra tecla reinicia.
ctl U	Cancela a linha corrente.
ctl X	Elimina a linha que está sendo digitada.

1.6 Definindo constantes

Constantes são valores que não sofrem alteração durante a execução do programa.
Há dois tipos de constantes: strings e numéricas.

1.6.1 Strings

Um string corresponde a um grupo de caracteres alfanuméricos, delimitados pelo caracter ("). O comprimento máximo de um string é de 255 caracteres.

Exemplos:

"BOM DIA"
"LTD002"
"ROTA 35"

1.6.2 Constantes numéricas

São números positivos ou negativos, sem vírgulas inseridas.
Há cinco tipos de constantes numéricas:

a) Constantes inteiras

Localizam-se no intervalo de – 32768 a 32767 e não aceitam ponto decimal.

Exemplos:

153	válido
257	válido
– 36	válido
– 999999	inválido – número muito pequeno
12,840	inválido – não são permitidas vírgulas
32840	inválido – número muito grande

Dica. Use constantes inteiras sempre que possível, pois haverá economia de memória e o programa será executado mais rapidamente.

b) Constantes com ponto fixo

São números reais positivos ou negativos, com ponto decimal.

Exemplos:

– 4.72
27180.321

c) Constantes com ponto flutuante

São números positivos ou negativos, representados na forma exponencial.
Uma constante com ponto flutuante consiste de uma constante inteira ou de ponto fixo (mantissa) com sinal opcional seguido pela letra E, mais uma constante inteira (expoente) com sinal opcional.

O expoente deve estar no intervalo de –38 a +38.

$$10^{-38} \text{ a } 10^{38}$$

Exemplos:

354.44E3 = (354.44) * 10 ↑ 3
215.753E – 7 = .0000215753
2177E6 = 2177000000

Observação. O maior valor que o BASIC pode aceitar é 1.70141E38, e o menor positivo, 2.9387E – 38.

d) Constantes hexadecimais

São números com o prefixo &H.

Exemplos:

&H55
&H11F

e) Constantes octais

São números com o prefixo &O ou &

Exemplos:

&O347
&2345

1.6.3 Precisão das constantes numéricas

a) Constantes de precisão simples

Uma constante de precisão simples é qualquer constante que possua:

- sete ou menos dígitos
- formato exponencial utilizando "E"
- ponto de exclamação (!) no final

Exemplos:

37.6
3765.0
– 6.46E – 05
17.8!

As constantes de precisão simples são armazenadas com sete dígitos de precisão e mostradas com seis dígitos.

b) Constantes de precisão dupla

Uma constante de precisão dupla é qualquer constante que possua:

- oito ou mais dígitos
- formato exponencial utilizando "D"
- um sinal libra (#) no final

Exemplos:

```
   276432197
- 1.25327D-05
     2743.0#
 2435741.0974
```

As constantes de precisão dupla são armazenadas com 16 dígitos de precisão e mostradas com até 16 dígitos.

1.7 Definindo variáveis

Variáveis são nomes usados para representar valores que podem ser alterados durante a execução do programa BASIC.

O valor de uma variável pode ser atribuído pelo programador ou ser o resultado de um determinado cálculo dentro do programa.

Antes de se atribuir valor a uma variável, ela terá valor zero se for numérica e nulo no caso de strings.

> Dica. Declare as variáveis mais usadas em primeiro lugar, pois o programa será executado mais rapidamente quando estiver sendo interpretado, ou então declare-as em ordem alfabética para facilitar a visualização.

1.7.1 Nomes de variáveis

Os nomes de variáveis podem ter qualquer tamanho, porém somente os 40 primeiros caracteres são significativos. Os nomes de variáveis podem ser compostos por letras e números, além do ponto decimal, entretanto o primeiro caracter do nome da variável deve ser letra.

Exemplos:

```
%B  nome inválido
B3  nome válido
```

O nome de uma variável não pode ser uma palavra reservada (comandos, declarações, funções e operadores) mas pode ter uma palavra reservada inserida.

Exemplos:

```
KIF     nome válido
GOTOX   nome válido
QINPUT  nome válido
MERGE   nome inválido
```

Para evitar problemas, antes de atribuir um nome a uma variável, consulte a relação de palavras reservadas no Apêndice C.

Se a variável começar com as letras FN, ela será chamada "Função de Usuário", pois é uma função definida pelo próprio usuário; portanto, não pode ser usada como nome de variável. (Verifique no Cap. 8 a criação de funções específicas.)

Exemplo:

```
FNTXT   Inicia com a palavra reservada FN, portanto é considerada uma função.
```

Lembre-se:

(1) Se o valor da variável não foi definido anteriormente, seu valor é 0 para numéricas e nulo para strings.

(2) Os nomes de variáveis não podem começar com FN.

Dicas para a escolha de nomes de variáveis:

(1) Empregue nomes significativos.

(2) Escolha nomes que não sejam confundíveis.

a) a letra S pode ser confundida com o número 5.

Exemplo: NS com N5.

b) A letra O pode ser confundida com o número 0.

Exemplo: COP com C0P.

c) A letra I pode ser confundida com o número 1.

Exemplo: KI com K1.

Muitos programadores evitam colocar numerais nos nomes de variáveis.

d) Diferem apenas por uma letra: CH e CV

e) Diferem apenas no final: VTOT e VTOTG

(3) Use abreviações populares na organização.

Exemplo: CLS$ para "clear screen",
BCKSPC$ para "backspace",
RTRN$ para "return ou carriage return", CTLIN para contador de linhas etc.

(4) As consoantes são mais memorizáveis que as vogais.

(5) Os caracteres do início do nome da variável são mais memorizáveis que os do final.

1.7.2 Algoritmo para a colocação de nomes de variáveis

A abreviação de uma palavra é feita pela eliminação sucessiva das vogais, até que todas as vogais tenham sido eliminadas (exceto quando a letra inicial da palavra seja uma vogal) ou a palavra esteja reduzida ao tamanho desejado. No entanto, se as vogais já foram eliminadas e a palavra continuar grande, repete-se o processo, eliminando-se então as consoantes até que a palavra tenha o tamanho desejado.

Exemplos:

Nome	Abreviação
CUSTO TOTAL	CSTTTL
VALOR TOTAL	VLRTTL
REGISTRO	RGSTR
TRANSAÇÃO	TRNSC

1.7.3 Caracteres que definem o tipo

As variáveis podem representar um valor numérico ou um string de caracteres. Nomes de variáveis associados a strings recebem um cifrão ($) como último caracter.

Exemplo:

```
X$ = "Código inválido"
```

Os nomes de variáveis numéricas podem declarar (conter) valores inteiros, valores de precisão simples ou dupla. Os caracteres de especificação de tipos, para estes nomes de variáveis, são os seguintes:

% Variável inteira
! Variável de simples precisão
Variável de dupla precisão

O tipo "default" para nomes de variáveis numéricas é simples precisão (!). Isto é, se não for especificado o tipo, será assumido simples precisão.

Exemplos de nomes de variáveis:

PI#	Declara um valor de dupla precisão
MINIMO!	Declara um valor de simples precisão
LIMITE%	Declara um valor inteiro
NOME$	Declara um valor string
TOTAL	Declara um valor de simples precisão

1.8 Conversão de tipos

Quando necessário, o BASIC converte uma constante numérica de um tipo para outro. Tenha sempre em mente as seguintes regras, pois lhe serão bastante úteis:

1.8.1 Quando os tipos são diferentes

Se uma constante numérica de determinado tipo for definido como sendo igual a uma variável numérica de tipo diferente, o número é armazenado como o tipo declarado no nome da variável.

Se uma variável string foi assinalada com um valor numérico ou vice-versa, ocorrerá um erro: "Type mismatch (tipos incompatíveis)".

Exemplos:

(1)
```
10 A% = 23.4
20 PRINT A%
RUN
 23
```

(2)
```
100 A$ = 13
```

Neste caso ocorrerá o erro "Type mismatch".

Observação. Quando um valor de ponto flutuante for convertido em inteiro, a parte fracionária é arredondada.

```
10 A% = 23.9
20 PRINT A%
RUN
 24
```

1.8.2 Quando é assumida a maior precisão

Durante o cálculo de uma expressão, todos os operandos em uma operação aritmética ou relacional são convertidos para o mesmo grau de precisão, isto é, o de maior precisão. Da mesma forma, o resultado da operação aritmética é devolvido com este grau de precisão.

Exemplos:

(1)
```
10 D# = 6#/7
20 PRINT D#
```

```
RUN
 .8571428571428571
```

A operação foi executada em precisão dupla e o resultado devolvido em D# foi de dupla precisão.

(2)
```
10 D = 6#/7
20 PRINT D
RUN
 .857143
```

A operação foi executada em dupla precisão, e o resultado, devolvido em D (variável de simples precisão), foi arredondado e apresentado como um valor de simples precisão.

1.8.3 Assinalando um valor de simples precisão a uma variável de dupla precisão

Se uma variável de dupla precisão é especificada como um valor de simples precisão, apenas os sete primeiros dígitos arredondados do número convertido são válidos. Isto se deve ao fato de apenas sete dígitos de precisão serem considerados em um valor de simples precisão. O valor absoluto da diferença entre o número de dupla precisão e o valor original de simples precisão é menor que 6.3E-8.

Exemplo:

```
10 A = 2.04
20 B# = A
30 PRINT A; B#
RUN
 2.04   2.03999961853027
```

1.9 Usando os operadores

Os operadores executam operações matemáticas ou lógicas sobre valores. Os operadores podem ser divididos em quatro categorias: aritméticos, relacionais, lógicos e funcionais.

1.9.1 Operadores aritméticos

Os operadores aritméticos em ordem decrescente de precedência de execução são:

↑	exponenciação
–	negação
* /	multiplicação e divisão
\	divisão inteira
MOD	módulo (retorna o resto de uma divisão)
+ –	adição e subtração

Para alterar a ordem de execução das operações, devem-se utilizar parênteses. As operações dentro dos parênteses são executadas primeiro. Dentro dos parênteses, a ordem usual das operações é mantida. No caso de "conflito", a seqüência de cálculo é da esquerda para a direita.

Exemplos:

Expressões Algébricas	Expressões BASIC
$x + yz$	X + Y * Z
$x - \frac{y}{z}$	X – Y / Z
$\frac{xy}{z}$	X * Y / Z
x^{2y}	(X ↑ 2) ↑ Y
x^{y^z}	(X ↑ (Y ↑ Z)
x (– y)	X * (– Y)
	Duas operações consecutivas devem ser separadas por parênteses.

Observação. Quando ocorrer uma divisão por zero, o programa dará como resultado o valor 1.70141E38, mostrando a mensagem "Division by zero" e a execução continuará.

Exemplos de colocação de parênteses:

Não recomendável	Recomendável
A*B*C/(D*E*F)	(A*B*C)/(D*E*F)
A↑B↑C	A↑(B↑C)
A/B/C/D	((A/B)/C)/D
A↑B*C	(A↑B)*C

a) Pegando a parte inteira

A divisão inteira é representada pelo caracter barra reversa. Os operandos são arredondados para inteiros antes da operação e o quociente é truncado para um inteiro após a operação.

Observação. Os operandos devem estar entre –32768 e 32767.

Exemplos:

```
10 \ 4 = 2
25.68 \ 6.99 = 3
269.9 \ 3 = 270 \ 3 = 90
268.9 \ 3 = 269 \ 3 = 89.666 . . . = 89
```

b) Pegando o resto

O operador MOD (módulo aritmético) retorna o resto de uma divisão. Se os operandos não forem inteiros, a parte decimal será arredondada. O valor resultante deverá estar no intervalo de −32768 a 32767.

Exemplos:

10.4 MOD 4 = 2 (10/4 = 2 com resto 2)
25.68 MOD 6.99 = 5 (26/7 = 3 com resto 5)
82 MOD 9 = 1
225 MOD 2 = 1
120 MOD 2 = 0
56 MOD 2 = 0

1.9.2 Operadores relacionais

Os operadores relacionais são usados para comparar dois valores.

Operador	Relação testada	Expressão
=	igualdade	X = Y
< >	desigualdade	X < > Y
<	menor que	X < Y
>	maior que	X > Y
<=	menor ou igual a	X <= Y
>=	maior ou igual a	X >= Y

Quando os operadores aritméticos e relacionais são combinados em uma expressão, o operador aritmético é executado em primeiro lugar.

Exemplos:

IF X = Y THEN 200 — Se o valor da variável X for igual ao valor da variável Y, então o fluxo é desviado para a linha 200.

IF A < > B THEN 500 — Se o valor de A for diferente de B, então o fluxo é desviado para a linha 500.

IF N > 100 THEN 300 — Se o valor de N for maior que 100, então o fluxo é desviado para a linha 300.

IF N >= 100 THEN 700 — Se o valor de N for maior ou igual a 100, então o fluxo é desviado para a linha 700.

IF W < 5 THEN 200 — Se o valor de W for menor que 5, então o fluxo é desviado para a linha 200.

IF W <= 5 THEN 230 — Se o valor de W for menor ou igual a 5, então o fluxo é desviado para a linha 230.

IF A > (B − C) THEN 500 — Se o valor de A for maior que a diferença de B menos C, então o fluxo é desviado para a linha 500.

1.9.3 Operadores lógicos

Os operadores lógicos executam testes em relações múltiplas. Os principais operadores são: AND, OR, NOT e XOR.

Exemplos:

(1) IF D < 200 AND F < 4 THEN GOTO 80

Se D for menor que 200 e F for menor que 4, então vá para a linha 80.

(2) IF I > 10 OR K < 0 THEN GOTO 100

Se I for maior que 10 ou K for menor que 0, então vá para a linha 100.

(3) IF NOT (A > 5) THEN 60

Se A não é maior que 5, então vá para 60.

(4) IF A = 3 XOR B = 3 THEN 80

Se A = 3 ou B = 3, mas não ambos, então vá para a linha 80.

1.9.4 Operadores funcionais

a) Concatenando strings

Exemplo:

```
10 A$ = "NOME DE " : B$ = "ARQUIVO"
20 PRINT A$ + B$
30 PRINT "NOVO " + A$ + B$
RUN
NOME DE ARQUIVO
NOVO NOME DE ARQUIVO
```

b) Comparando strings

Pode-se comparar strings fazendo-se uso dos mesmos operadores relacionais que são usados com números:

= <> < > <= >=

As comparações entre strings são feitas caracter a caracter, comparando os códigos ASCII. Se os códigos ASCII forem os mesmos, os strings são iguais. Se diferirem, o código de menor número precede o de maior número. Durante a comparação de strings, quando se encontra o fim de um deles, o string mais curto é entendido como sendo o menor. Todos os caracteres brancos são significativos.

Exemplos:

"AB" é maior que "AA"
"Z*" é maior que "Z%"
"KX$" é maior que "KX"
"AA" é menor que "aa"
"TERMINAL" é maior que "TERMINA"
"13/11/83" é menor que "22/11/83"
"CL " é maior que "CL"

Assim, as comparações entre strings podem ser usadas para testar valores ou dispô-los em ordem alfabética.

Exemplo:

Classificação de um arquivo de nomes em ordem alfabética.
Como sugestão para testar este recurso, execute o programa abaixo. Coloque todas as opções que deseje verificar.

```
10 INPUT "Entre com dois strings"; A$, B$
20 IF A$ > B$ THEN PRINT A$;" > "; B$ : GOTO 10
30 IF A$ < B$ THEN PRINT A$;" < "; B$ : GOTO 10
40 PRINT A$; " = "; B$ : GOTO 10
```

Dicas:

(1) Escala de rapidez de processamento

rápido	adição ou subtração
↓	multiplicação
	divisão
lento	exponenciação

Exemplos:

B + B é mais rápido que 2 * B
.2 * C é mais rápido que C/5. Evite a divisão quando necessitar de maior rapidez.
D * D * D * D é mais rápido que D ↑ 4
X * X é mais rápido que X ↑ 2

(2) Evite cálculos repetidos

Não recomendável	Recomendável
X = Y + A/B * C	K = A/B * C
Z = W + A/B * C	X = Y + K
	Z = W + K

(3) Um programa é freqüentemente mais legível se os operadores forem precedidos e seguidos por um espaço.
Exemplo: 500 A = B + C + D + 5

(4) Use parênteses em expressões, mesmo quando acreditar que eles não sejam necessários.

1.10 Testes

1) O que será mostrado no video?

```
10 I% = 10
20 I! = 20
30 I = 40
40 PRINT I%; I!; I; I#
```

a) 10 10 10 10
b) 10 20 40 0
c) 40 40 40 40
d) 10 40 40 20
e) nenhuma das anteriores.

2) Qual é a alternativa errada?

a) variáveis inteiras = – 32768 a 32767;
b) variáveis de simples precisão armazenam 7 dígitos e apresentam 6;
c) variáveis de dupla precisão armazenam 17 dígitos e apresentam 16;
d) variáveis string armazenam de 0 a 255 caracteres.

3) Dado:

```
10 A = 9 : B = 2
20 C = A \ B
30 D = A MOD B
```

Quais são os valores de C e D, respectivamente?

a) 4.5, 3;
b) 4, 1;
c) 4.5, 11;
d) 4.5, 7;
e) 4, 11.

4) Qual é a alternativa que contém respectivamente nomes de variáveis inteiras, simples precisão, dupla precisão e string?

a) A%, A!, A#, A$;
b) A!, A%, A$, A;
c) A$, A#, C$, B$;

d) B!, C$, J%, A;
e) A!, B%, B#, K$.

5) Dado:

```
10 A = 5 : B = 2
20 C = A \ B
30 D = A MOD B
```

Quais são os valores de C e D respectivamente?

a) 2,3;
b) 2, 1;
c) 2.5, 7;
d) 2.5, 3;
e) 2, 2.

6) O que será mostrado no video?

```
PRINT (3\2 *1.5)
```

a) 2.25
b) 1
c) 0
d) 5
e) nenhuma das anteriores.

7) Qual é a alternativa que contém um nome de variável ilegal?

a) NOME$, TOTAL#, ACUM%;
b) BMERGE, A!, XLOAD#;
c) A#, AM%, XK%;
d) NTO, FNCP$, X$.

8) X = 15 MOD 11

Qual é o valor de X?

a) 9;
b) 0.54;
c) 0;
d) 6;
e) nenhuma das anteriores.

9) Qual é o valor de D%, se D% = 5/100?

a) 0.02;
b) 0.002;
c) 0;
d) 2D – 2;
e) 2E – 2.

10) Dado: X$ = 13

Qual será o valor de X$?

a) 13;
b) nulo;
c) nenhuma das anteriores.

1.11 Exercícios propostos

1) Transforme estas equações algébricas em equações BASIC.

a) $x + y$;

b) $e + \frac{b}{c}$;

c) $a + \frac{k}{c} + d$;

d) $\frac{a + m}{c + d} + x$.

2) Verifique quais são os caracteres que o seu microcomputador e a impressora podem mostrar. **Sugestão. Use os códigos da tabela ASCII.**

3) Verifique todas as funções das teclas do teclado do seu microcomputador.

4) Verifique qual é a versão do interpretador que você utiliza.

5) Verifique, no seu microcomputador, qual é o número de bytes livres após o carregamento do interpretador.

2

USANDO OS COMANDOS

"Quando um programador é bom, ele é muito, muito bom, mas quando é ruim, ele é horrível."

Edward Yourdon

2.1 Manipulando o programa na memória

2.1.1 Limpando a memória

O comando NEW suprime o programa e todas as variáveis da memória.

Exemplo: NEW

Dicas:

(1) Execute o comando NEW sempre antes de introduzir um novo programa na memória. Se a memória não for limpa, o novo programa introduzido vai-se misturar com o anterior que estava na memória.

(2) Após a execução do comando NEW não é possível recuperar o conteúdo anterior na memória. Portanto, pense um pouco antes de utilizá-lo.

(3) Para limpar somente as variáveis, use a instrução CLEAR. Verifique o Cap. 15, para obter informações sobre a instrução CLEAR.

2.1.2 Numeração automática das linhas

Quando se introduz um programa na memória, é necessário colocar os números de linhas e as respectivas instruções.

O comando AUTO gera automaticamente um número de linha a cada ENTER, evitando assim que seja digitado o número da linha.

AUTO [linha-inicial] [incremento]

O comando AUTO inicia a numeração das linhas do programa a partir da linha-inicial e incrementa cada número subseqüente do valor do incremento.

Se o comando AUTO gera um número de linha que já existe na memória, é mostrado um asterisco após o número para advertir que a próxima linha substituirá a linha existente. Entretanto, se for pressionada a tecla ENTER imediatamente após o asterisco, o conteúdo desta linha anterior será gravado e é gerado o próximo número de linha.

Exemplos:

(1) AUTO 100, 50

gera números de linha a partir de 100, de 50 em 50.

```
100 .....
150 .....
200 .....
  .
  .
```

(2) AUTO

gera números de linha a partir de 10, de 10 em 10.

```
10 .....
20 .....
30 .....
40 .....
  .
  .
```

Observações:

(1) O "default" para ambos os parâmetros de AUTO é 10.

(2) Para interromper o comando AUTO, pressione ctl C.

(3) AUTO é abreviação de AUTOmatic (automático).

Lembre-se: se aparecer um asterisco é porque a linha já existe na memória.

2.1.3 Mostrando o programa

Para mostrar no video o programa que está na memória, digite LIST e pressione ENTER. Para mostrar na impressora, use o comando LLIST.

Caso deseje ver apenas partes do programa, use estas opções:

LIST – n	Mostra do início do programa até a linha n.
LIST n –	Mostra da linha n até o fim do programa.
LIST n – p	Mostra da linha n até a linha p.
LIST n	Mostra apenas a linha n.

Exemplo: LIST 500 –
Será mostrado as linhas do programa começando na linha 500 até a última linha do programa.

Observações:

(1) Use ctl S para pausa e ctl C para cancelar o comando.

(2) Estas opções também podem ser utilizadas no comando LLIST.

2.1.4 Eliminando linhas do programa

O comando DELETE elimina linhas do programa.
As opções são:

DELETE n	Elimina a linha n do programa.
DELETE n – p	Elimina a linha n até a linha p.
DELETE – n	Elimina do início do programa até a linha n.
DELETE n –	Não é permitido !!!

Exemplo: DELETE 50–80
Neste caso serão eliminadas as linhas compreendidas entre os números 50 e 80, inclusive.

2.1.5 Fazendo correções na linha do programa

O comando EDIT < linha > traz para o video o conteúdo da linha especificada. Desta forma é possível fazer correções ou modificações nesta linha.

Quanto maior e mais complexo for o programa, maior será a utilidade do EDIT.

Exemplo: EDIT 10
Edita a linha 10 do programa

Observação. EDIT.
Edita a última linha referenciada.

Após o comando de edição, o BASIC apresenta o número da linha a ser editada, seguido de um espaço, e aguarda um dos subcomandos do modo de edição, relacionados a seguir.

Para entrar no modo de edição na linha que está sendo digitada, pressione ctl A. O cursor será posicionado no primeiro caracter da linha. Neste momento pode ser usado qualquer dos subcomandos relacionados abaixo.

Subcomandos do modo de edição

Estes subcomandos são usados para movimentação do cursor, inserção, supressão, substituição ou pesquisa de caracteres dentro de uma linha. Os subcomandos de edição não aparecem no video quando digitados. A maioria dos subcomandos pode ser precedida por um número que determina o número de vezes que o subcomando deve ser executado.

Quando este número não é especificado, o valor 1 é assumido.

a) Movimentando o cursor

< espaços >	Move o cursor para a direita. Os caracteres vão sendo exibidos à medida que o cursor vai avançando à direita.
← – – –	Move o cursor à esquerda.

b) Inserindo caracteres na linha

I	Os caracteres serão inseridos a partir da posição onde estiver o cursor, sem alterar nenhuma outra parte da linha. Pressione ctl ESC para interromper a inserção. Em alguns computadores basta pressionar ESC.
X	Posiciona o cursor no fim da linha, para, a partir dali, ser feita qualquer inserção, como se o comando I tivesse sido então pressionado.

c) Suprimindo caracteres na linha

nD	Elimina os "n" caracteres que estiverem à direita do cursor, fazendo com que estes apareçam entre pontos de exclamação. *Observação*. (D) elete = apaga
H	Apaga o que estiver à direita do cursor e entra no modo de inserção. *Observação*. (H) ack = corta
nKc	Apaga todos os caracteres que estão entre a posição do cursor e a n-ésima ocorrência do caracter "c". *Observação*. (K) ill = elimina

d) Pesquisando caracteres na linha

nSc	Posiciona o cursor na n-ésima ocorrência do caracter "c". Se este caracter não for encontrado, o cursor se posicionará no final da linha que está sendo editada. *Observação*. (S) earch = pesquisa

e) Substituindo caracteres na linha

nC	Permite a troca de n caracteres a partir do cursor. Se n não for especificado, será assumido como sendo 1. *Observação*. (C) hange = altera

f) Edição e reinicialização do modo de edição

L	Mostra o restante da linha, reposicionando o cursor no início da linha. *Observação*. (L) ist = lista

E — Volta ao nível de comando imediato considerando as alterações efetuadas.

Observação. (E)xit = saída

Q — Volta ao nível de comando imediato cancelando as alterações efetuadas.

Observação. (Q)uit = abandona

A — Move o cursor para o início da linha cancelando as alterações feitas na linha.

Observação. (A)bend = abandona

ENTER — Acionando a tecla ENTER, o BASIC sairá do modo de edição e todas as alterações que tiverem sido feitas serão consideradas.

2.1.6 Renumerando as linhas do programa

O comando RENUM renumera as linhas do programa.

RENUM [novo-número] [, número-inicial] [, incr.]

Observações:

(1) O "default" para o incremento e novo-número é 10.

(2) Se não for fornecido, o número-inicial será renumerado a partir da primeira linha do programa.

(3) RENUM não gera um número de linha maior que 65529.

Exemplos:

(1) RENUM

Renumera o programa inteiro. O novo número inicial será 10 e as linhas subseqüentes serão incrementadas de 10.

Antes	Depois
5	10 . . . . .
20	20 . . . . .
30	30 . . . . .
40	40 . . . . .
50	50 . . . . .
55	60 . . . . .
60	70 . . . . .
62	80 . . . . .

(2) RENUM 300 ,, 50

Renumera o programa inteiro. O novo número inicial será 300 e as linhas subseqüentes serão incrementadas de 50.

Antes	Depois
10 . . .	300 . . .
20 . . .	350 . . .
30 . . .	400 . . .
40 . . .	450 . . .
50 . . .	500 . . .
60 . . .	550 . . .

(3) RENUM 60, 55, 5

Renumera as linhas a partir da linha 55. O novo número inicial será 60 e as linhas subseqüentes serão incrementadas de 5.

Antes	Depois
10	10 . . . . .
20	20 . . . . .
30	30 . . . . .
40	40 . . . . .
50	50 . . . . .
55	60 . . . . .
60	65 . . . . .
62	70 . . . . .
63	75 . . . . .
64	80 . . . . .
70	85 . . . . .
150	90 . . . . .
170	95 . . . . .
280	100 . . . . .
385	105 . . . . .

Dicas:

(1) Depois que o programa estiver pronto, renumere de 10 em 10, para facilitar manutenções futuras.

(2) Renumere em blocos lógicos.

Por exemplo: o programa principal começa na linha 500 e as sub-rotinas a partir da linha 5000 com intervalos de 500 para cada sub-rotina.

2.1.7 Executando o programa

O comando RUN linha executa o programa que está na memória. Se o número da linha for especificado, a execução começará a partir desta linha. Caso contrário, a execução começará na primeira linha do programa.

Exemplos:

(1) RUN Executará a partir da primeira linha do programa.

(2) RUN 500 Executará a partir da linha 500 do programa.

Observações:

(1) Para pausa na execução pressione ctl S.

(2) Para interromper a execução pressione ctl C.

2.1.8 Rastreando a execução do programa

A instrução TRON, executada tanto no modo imediato como no programado, imprime entre colchetes, os números das linhas que vão sendo executadas. O "rastreamento" da execução é desabilitado com a instrução TROFF.

Com este recurso pode-se seguir o fluxo de execução do programa, descobrindo assim possíveis erros de lógica.

Exemplo:

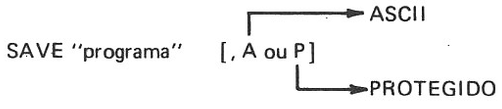
```
10 K = 10
20 FOR J = 1 TO 2
30     L = K + 10
40     PRINT J; K; L
50     K = K + 10
60 NEXT J
70 END
TRON
RUN
[10] [20] [30] [40] 1  10  20
[50] [60] [30] [40] 2  20  30
[50] [60] [70]
TROFF
```

Observação. TRON é abreviação de TRace ON e TROFF é abreviação de TRace OFF.

2.2 Usando o disquete

2.2.1 Gravando o programa no disquete

O comando SAVE grava o programa que está na memória no disquete.



A opção A grava o programa em formato ASCII, e a opção P grava em formato protegido, isto é, o mesmo não poderá ser listado ou editado, apenas poderá ser executado. Se não for usada nenhuma destas opções, o programa será gravado no formato-padrão do BASIC, ou seja, em binário compactado.

Se no nome do programa não for colocada extensão, será assumido extensão . BAS.

Exemplo: SAVE "CP05 . ASC", A

Dicas:

(1) Grave os programas no formato ASCII, pois poderão ser listados através de comandos do CP/M.

(2) Use extensão . ASC para os programas gravados no formato ASCII.

Exemplo: SAVE "CPP07 . ASC", A

(3) Use extensão . PRO para os programas gravados no formato protegido.

Exemplo: SAVE "CPP09 . PRO", P

(4) Use extensão . BAS para os programas gravados no formato padrão do BASIC.

Exemplo: SAVE "CPP13 . BAS"

(5) Lembre-se sempre de gravar o programa a cada edição, para evitar perdas de trabalho quando, por exemplo, houver interrupção de energia elétrica. Para programas extensos, é conveniente salvá-lo periodicamente, durante sua digitação, ainda que inacabado.

(6) Coloque o comando SAVE na última linha do programa. Faça com que esta linha somente seja executada com a instrução RUN linha. Isto elimina a necessidade de se digitar o comando SAVE toda vez que se queira gravar o programa. Elimine esta linha antes de compilar o programa, pois este comando não é permitido no compilador.

Exemplo:

```
1340 . . . . . .
1350 . . . . . .
1360 END
9000 SAVE "CPP14 . ASC", A
```

Para gravar este programa, apenas digite RUN 9000.

2.2.2 Verificando o conteúdo do disquete

O comando FILES mostra no video a relação de arquivos do disquete. É similar ao comando DIR do CP/M.

Exemplos:

(1) FILES Será mostrada a relação de arquivos do disquete corrente.

(2) FILES "*. BAS" Será mostrada a relação de arquivos com extensão . BAS no disquete corrente.

(3) FILES "B : * . *" Será mostrada a relação de arquivos no disquete do drive B.

2.2.3 Carregando o programa na memória

O comando LOAD carrega o programa do disquete para a memória. Para executá-lo, digite RUN.

Este comando fecha todos os arquivos abertos e limpa toda a memória antes de carregar o programa.

LOAD "programa"

Exemplo: LOAD "SGP05 . BAS"

2.2.4 Carregando e executando o programa

O comando RUN "programa" carrega e executa o programa que está gravado no disquete.

RUN "programa"

A execução do RUN fecha todos os arquivos abertos e suprime o conteúdo corrente da memória, antes de carregar o programa especificado.

Se não for fornecido a extensão, será assumido . BAS.

Exemplos:

(1) RUN "CXP07 . ASC"
(2) IF OP$ = "I" THEN RUN "FPP00 . ASC"

2.2.5 Eliminando o arquivo do disquete

O comando KILL elimina o arquivo especificado do disquete.

KILL "arquivo . ext"

Exemplos:

(1) KILL "CPP08 . $$$"

Será eliminado o arquivo "CPP08 . $$$" do disquete corrente.

(2) KILL "B : XYZ . BAS"

Será eliminado o arquivo "XYZ . BAS" do disquete que está no drive B.

Observações:

(1) Não dê KILL em um arquivo aberto, as conseqüências serão imprevisíveis. Nos Caps. 10 e 11 é explicado como abrir um arquivo.
(2) Deve ser fornecida a extensão, caso exista.

2.2.6 Alterando o nome do arquivo

O comando NAME troca o nome do arquivo que está no disquete.

NAME "nome anterior" AS "novo nome"

O "nome anterior" já deve existir no disquete e o "novo nome" não deve existir, caso contrário resultará em erro.

Após a execução do comando, o arquivo continua no mesmo lugar no disquete, porém com o novo nome.

Exemplo: NAME "XYZ . BAS" AS "CGP17 . BAS"

Neste caso, deseja-se alterar o nome do programa "XYZ . BAS" para "CGP17 . BAS"

Observações:

(1) Não altere o nome de um arquivo aberto. Pode ser perdido o arquivo. Nos Caps. 10 e 11 é explicado como abrir um arquivo.
(2) Deve ser fornecida a extensão do "nome anterior", caso exista.

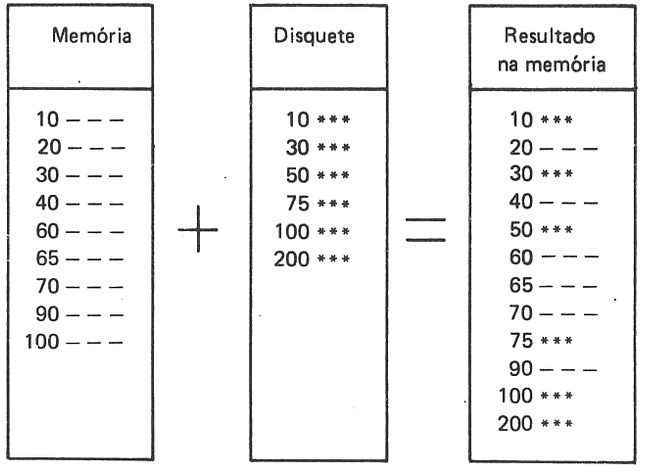
Dica. Use o artifício NAME "X" AS "X" para verificar se o arquivo X existe ou não. Se o arquivo X existe, ocorrerá erro 58; caso contrário, ocorrerá erro 53. Esta técnica deve ser usada em conjunto com a instrução ON ERROR GOTO linha.

2.2.7 Intercalando programas

O comando MERGE "programa" intercala ao programa que está na memória, um programa em disquete.

O programa em disquete deve ter sido gravado no formato ASCII.

Se qualquer linha do programa em disquete possuir o mesmo número que uma linha do programa na memória, a linha proveniente do programa em disquete substituirá a linha correspondente do programa na memória.



A operação do comando MERGE pode ser entendida como uma inserção de linhas do programa em disquete no programa na memória.

Exemplo: MERGE "SUBROT . ASC"

> Dica. Use este comando para inserir, no programa que está na memória, sub-rotinas-padrão da biblioteca de sub-rotinas.
>
> *Por exemplo*: a subrotina para verificar a validade de uma data é necessária na maior parte dos programas e usando o comando MERGE não é necessário digitá-la em todos os programas.

2.2.8 Atualizando o diretório dos disquetes

A instrução RESET reinicializa as informações de alocação do diretório. Deve ser executada após a inserção de um novo disquete.

Pode ser usada tanto no modo imediato como no programado.

Exemplo:
```
490 . . . . . .
500 RESET
600 . . . . . .
```

2.3 Testes

1) Qual é a alternativa que contém um comando inválido?
   a) LIST – 100
   b) LLIST 100 –
   c) LIST 100 –
   d) DELETE 100 –

2) Qual é o formato correto do comando NAME?
   a) NAME "nome antigo" AS "novo nome"
   b) NAME "novo nome" AS "nome antigo"

3) Qual é o formato correto do comando RENUM?
   a) RENUM novo-número, número-antigo, incremento
   b) RENUM número-antigo, novo-número, incremento

4) Dado: NAME "XYZ . BAS" AS "CPP01 . BAS"
   Qual das alternativas é correta?
   a) ocorre erro se XYZ . BAS não for encontrado no disquete;
   b) ocorre erro se CPP01 . BAS já existe no disquete;
   c) as alternativas anteriores estão corretas.

5) Qual das alternativas é correta?
   a) o comando RESET deve ser executado antes da remoção de disquetes;
   b) o comando RESET deve ser executado após a remoção de disquetes.

6) Qual das funções especiais do teclado é usada para entrar no modo de edição na linha que está sendo introduzida?
   a) ctl J
   b) ctl R
   c) ctl A
   d) ctl I
   e) nenhuma das anteriores.

7) Qual das funções especiais do teclado é usada para interromper a execução do comando AUTO?
   a) ctl J
   b) ctl R
   c) ctl A
   d) ctl C
   e) nenhuma das anteriores.

8) O que acontece se houver coincidência de números de linhas entre um programa na memória e uma rotina que foi inserida com o comando MERGE?
   a) serão assumidas as linhas que vieram da rotina em disquete;
   b) serão assumidas as linhas do programa que estava na memória;
   c) nenhuma das anteriores.

9) Dado: SAVE "EXEMPLO", P
   Qual das alternativas é inválida?
   a) o programa EXEMPLO não poderá ser listado;
   b) o programa EXEMPLO não poderá ser editado;
   c) o programa EXEMPLO não poderá ser executado.

10) "É possível recuperar um programa que estava na memória após ser executado o comando NEW."
    Esta afirmação é verdadeira ou falsa?
    a) verdadeira;
    b) falsa.

2.4 Exercícios propostos

1) Use o comando FILES para verificar a relação de arquivos nos disquetes.
2) Use o comando KILL para eliminar os arquivos desnecessários nos disquetes.
3) Use o comando RENUM para renumerar as linhas dos programas.

3

DANDO OS PRIMEIROS PASSOS

"A única documentação confiável de um programa é o próprio código fonte."

Kernighan & Plauger

3.1 Entrando com dados no microcomputador

A instrução INPUT permite a entrada de dados durante a execução do programa. Quando for encontrada a instrução INPUT, o microcomputador interrompe a execução e fica aguardando que se introduza alguma informação por intermédio do teclado.

Exemplo:

```
10 INPUT A, B
```

Quando esta instrução for executada, aparecerá no video o ponto de interrogação, indicando que o microcomputador espera que se introduza alguma informação.

Caso queira que apareça uma mensagem informando ao operador qual dado deve ser introduzido, altere a instrução anterior para:

```
10 INPUT "Entre com os valores de A, B"; A, B
```

Neste caso, irá aparecer a mensagem "Entre com os valores de A, B", seguida do ponto de interrogação, e o microcomputador ficará aguardando a entrada dos dados.

Se for colocada uma vírgula separando a mensagem da lista de variáveis, não será mostrado o ponto de interrogação.

Dicas:

(1) Não é aconselhável alterar o valor de uma variável relacionada no INPUT.

(2) Quando estiver introduzindo datas, sempre coloque o formato em que a data deve ser introduzida.

Exemplo:

```
100 INPUT "Entre com a data DDMMAA"; DT$
```

(3) Uma maneira de consistir a data é, além de pedir DDMMAA, pedir também o dia da semana, e depois verificar se coincidem.

3.2 Fazendo os primeiros cálculos

A principal declaração para cálculos é LET, que tem a seguinte sintaxe:

LET < variável > = < expressão >

O símbolo = em BASIC não é igual ao símbolo matemático.

Em BASIC o significado de = é "o valor da expressão à direita é colocado na variável à esquerda."

Exemplos:

10 LET N = N + 1	O valor 1 é somado ao valor de N. O novo resultado altera o valor original de N.
10 LET R = A + B – 63	O valor de B é somado ao valor de A e ao valor da soma é subtraído 63. O valor final é colocado na variável R.
10 LET X = 3	A variável X terá o valor 3.
10 LET D2 = 0	A variável D2 terá o valor 0.
10 LET A$ = Z$	O conteúdo de A$ é alterado para o valor da variável Z$.
10 LET D$ = "FIM"	O string "FIM" é assinalado a D$.

Observação. A colocação da palavra LET é opcional.
Exemplo: 10 N = N + 1

3.3 Mostrando informações

3.3.1 Limpando o video

Para limpar o video, use PRINT CHR$ (12). No Cap. 9 é explicada a função CHR$.

Exemplo:

```
10 PRINT CHR$ (12)
20 PRINT "SOFTBRAS Inclusão de Pedido"
```

3.3.2 Mostrando informações no video

A instrução PRINT mostra no video mensagens, constantes, variáveis ou expressões.

Por exemplo:

```
50 PRINT "O valor de X = "; X
```

Será mostrada no video a mensagem "O valor de X = " e, em seguida, o valor da variável X.

Os itens a serem mostrados podem ser separados por vírgulas, para que o cursor avance para a próxima zona de impressão. Numa linha de video existem 5 zonas de 14 posições cada. Para que o cursor não avance para a próxima zona de impressão, a lista deve ser separada por ponto e vírgula.

Exemplos:

(1)
```
10 PRINT "zona1", "zona2", "zona3"
20 PRINT "zona1"; "zona2", "zona3"
RUN
zona1          zona2     zona3
zona1zona2     zona3
```

(2)
```
10 PRINT 15; 13
RUN
(15  13)
```
É mostrado um espaço antes e depois de valores numéricos.

(3)
```
10 PRINT – 30; 18
RUN
– 30  18
```
No caso de valores numéricos negativos, não é mostrado um espaço antes do número.

Se no final da lista houver ponto e vírgula ou vírgula, a próxima instrução PRINT irá imprimir nesta mesma linha, espaçando de acordo com as regras acima.

Exemplos:

(1)
```
160 PRINT "Testando",
170 PRINT "Estou na nova zona"
RUN
Testando      Estou na nova zona
```

(2)
```
160 PRINT "Testando";
170 PRINT "Continuo na mesma linha"
RUN
TestandoContinuo na mesma linha
```

Observações:

(1) A instrução PRINT pode ser substituída por ponto de interrogação durante a digitação.

Exemplo:

```
500 ? "Codigo Invalido"
```

(2) A instrução PRINT sem nenhum parâmetro irá imprimir uma linha em branco no video.

Exemplo:

```
110 PRINT "Valor da semana"; VALSEM#
120 PRINT
130 PRINT "Valor do mês"; VLMES#
140 PRINT
150 PRINT "Valor do ano"; VLANO#
```


3.3.3 Mostrando informações na impressora

A instrução LPRINT tem as mesmas opções da instrução PRINT, entretanto o resultado será mostrado na impressora.

Por exemplo:

```
70 LPRINT "RELATORIO DE VENDAS"
```

Para imprimir na página seguinte, use LPRINT CHR$(12).

Exemplo:

```
160 IF  CONTLIN > 56 THEN LPRINT CHR$(12) "RELATORIO DE VENDAS"
```

3.4 Finalizando o programa

A instrução END encerra a execução do programa, fecha todos os arquivos abertos e retorna ao modo imediato.

Esta instrução pode ser colocada em qualquer parte do programa para terminar a execução.

Ao contrário de STOP, a instrução END não mostra nenhuma mensagem no video.

É opcional incluir a instrução END na última linha do programa.

Exemplos:

(1)
```
50 IF OP$ = "F" THEN END
```

(2)
```
200 PRINT "Fim normal"
210 END
```

Dica. Documente a execução bem ou malsucedida de um programa.

Exemplo:
```
PRINT "FIM NORMAL" ou
PRINT "FIM ANORMAL "
```

3.5 Introduzindo comentários no programa

Um programa BASIC sem comentários é geralmente um programa difícil de ser entendido por outra pessoa, ou mesmo, depois de algum tempo, pela pessoa que o desenvolveu.

A instrução REM indica ao BASIC que os caracteres que vem a seguir são comentários e, portanto, são ignorados na execução do programa.

Exemplo:

```
150 REM
160 REM      Calculando o valor total a pagar
170 REM
180 VTPG = VPRST + VJR + VJM + VMD *0.57
```

Se não houvesse o comentário ficaria muito difícil para outra pessoa descobrir que na linha 180 é calculado o valor total a pagar.

Pode-se usar o apóstrofo (') no lugar de REM.

Refazendo o exemplo anterior:

```
150 '
160 '    Calculando o valor total a pagar
170 '
180 VTPG = VPRST + VJR + VJM + VMD *0.57
```

A instrução REM não pode ser colocada antes de outras declarações numa mesma linha com múltiplas declarações, porque todo o texto que se segue é considerado comentário, portanto sendo ignorado pelo BASIC durante a execução.

Exemplo:

```
100 REM entrando com A: INPUT A
```

Não será executada a instrução INPUT pois a mesma é considerada comentário.

Dicas:

(1) Use comentários para introduzir cada função principal do programa.

Exemplo:

```
5000 '
5010 '        calculando o saldo médio
5020 '
```

(2) Use comentários para documentar quais são os dados usados no programa. Coloque um dicionário de variáveis.

Exemplo:

```
100 '
110 '        Dicionário de variáveis
120 '
130 '    TOTMES     — total do mês
140 '    TOTMANT    — total do mês anterior
150 '    CTLIN      — contador de linhas
160 '    DTVEN      — data de vencimento
```


(3) Evite redundâncias. Os comentários devem transmitir novas informações. Não devem simplesmente fazer eco ao código.

Por exemplo:

Poucas pessoas acham que o seguinte comentário seria útil.

```
10 '
20 '   ler e imprimir os valores de A a B
30 '
40 INPUT A, B
50 PRINT A, B
```

(4) Use nos comentários letras minúsculas. As letras minúsculas distinguem-se das declarações do programa.

Exemplo:

```
100 '
110 '        Abertura dos arquivos
120 '
130 OPEN "I", #1, "MESTRE . DAT"
140 OPEN "I", #2, "MOVTO . DAT"
150 OPEN "O", #3, "SAÍDA . DAT"
```

(5) Recue os comentários da margem esquerda.

Exemplo:

```
1000 '
1010 '          Consistindo a entrada
1020 '
1030 IF VLTX > 13 THEN 800
```

(6) Coloque comentários que indiquem alterações no programa original.

Exemplo:

```
500 A = A + 5  ' alterado de 3 para 5 em 12/5/83 — Rubens
```

(7) Coloque uma linha em branco antes e depois dos comentários.

Exemplo:

```
500 '
510 '          Imprimindo o resultado
520 '
530 LPRINT CHR$(12) "VALOR TOTAL"; VT#
```

(8) Use o apóstrofo no lugar de REM.

(9) Use comentários para ajudar o leitor a traduzir os segmentos difíceis do programa ou os pontos intrincados.

(10) Para eliminar temporariamente uma instrução, use o seguinte artifício.

Exemplo:

```
50 LPRINT CHR$(12)
```

Colocando um apóstrofo nesta linha, ela se transforma em comentário, sendo ignorada pelo BASIC durante a execução do programa.

```
50 ' LPRINT CHR$(12)
```

Desta maneira pode-se inutilizar uma linha sem ter que eliminá-la fisicamente.

(11) Coloque em todos os programas um prólogo. As seguintes informações poderiam ser colocadas:

. nome do programa
. sistema
. subsistema
. autor
. função
. restrições de memória, periféricos etc.
. equipamento utilizado
. memória utilizada
. sub-rotinas utilizadas
. versão do interpretador BASIC utilizada
. chaves a serem utilizadas na compilação e linkedição
. data de criação do programa
. versões

Exemplo:

```
10  '
20  '  Programa – ATUALIZA . BAS
25  '  Sistema – Controle Mercantil
30  '  Autor – RUBENS
40  '  Função – Atualizar arquivo de
45  '           clientes
50  '  Equipamento – UNITRON com cartão CP/M
60  '  Data de criação – ABRIL/83
70  '  Versão 2 – MARCIA – JULHO/83
80  '  Versão 3 – RUBENS – SETEMBRO/83
```

3.6 Desvios

3.6.1 Incondicionais

A instrução GOTO < linha > desvia o fluxo do programa para a linha especificada.

Exemplos:

(1)
```
150 PRINT "Valor total = "; VLTTO
160 GOTO 500
```



```
. . . . . .
. . . . . .
. . . . . .
500 VLAC = VLAC + VLTTO
```

(2)
```
100 PRINT "Brasil"
110 GOTO 100
```

Neste caso, o programa ficará em um "laço infinito", isto é, nunca irá parar a não ser que seja interrompido.

Observação quanto ao uso do GOTO:

Qualquer tentativa de compreender um programa que utiliza muitas instruções GOTO é frustrada pelo fato de que após as primeiras declarações há um salto para uma declaração na página 3, depois volta para uma declaração na página 2, lendo mais algumas instruções encontra-se outro desvio, e assim por diante. Após 4 ou 5 manobras deste tipo, facilmente se esquece não somente onde era o começo, como também onde se está, e ainda o que estávamos fazendo.

Sugestão. Evite a "síndrome do espaguete".

3.6.2 Condicionais

A instrução ON < expr > GOTO L1, L2, L3, . . desvia o fluxo do programa para uma das linhas relacionadas, dependendo do valor da expressão.

Se o valor da expressão for 1, o fluxo é desviado para a linha L1; se o valor for 2, para a linha L2; e assim por diante.

Exemplos:

(1) 100 ON X GOTO 500, 600, 700, 800

(2) 180 ON X + 5 GOTO 300, 350, 400

Observações:

(1) O fluxo é desviado para a instrução seguinte se o valor da expressão for 0 ou maior que o número de linhas especificadas.

(2) Ocorrerá erro se o valor da expressão for negativo ou maior que 255.

(3) A parte fracionária será arredondada se o valor da expressão não for um valor inteiro.

3.7 Selecionando ações alternativas

IF < condição > THEN < ação(ões) > ELSE < ação(ões) >

Se o resultado da condição for verdadeiro, as declarações após THEN serão executadas, senão as declarações após ELSE serão executadas.

Exemplos:

(1) 100 IF X = 5 THEN A = X ELSE A = X + 15

(2) 500 IF X = 5 THEN Y = 6 ELSE GOTO 600

(3) 150 IF X = 5 OR Y = 6 THEN 600

Observação. Não é permitido comparar um valor numérico com um alfanumérico.

Exemplo:

```
IF X$ = 13 THEN 600
```

Neste caso ocorrerá erro.

3.8 Definindo o tipo das variáveis

Vimos no Cap. 1 que os sufixos %, !, # e $ definem variáveis inteiras, simples precisão, dupla precisão e string, respectivamente.

Existe uma outra maneira pela qual os tipos das variáveis podem ser definidos.

As instruções DEFINT, DEFSNG, DEFDBL e DEFSTR podem ser incluídas no programa para declarar os tipos das variáveis.

Exemplos:

100 DEFINT A – D	Esta instrução define como inteiras todas as variáveis que tenham o nome começando com A, B, C ou D.
110 DEFSNG S	Define como simples precisão todas as variáveis que tenham o nome começando com S.
120 DEFDBL V, X	Define como dupla precisão todas as variáveis que começam com V ou X.
130 DEFSTR E – G	Define como strings todas as variáveis que começam com E, F ou G.

Observação. O tipo das variáveis pode ser redefinido por outra declaração de tipo.

Exemplo:

```
10 DEFINT I
20 DEFSNG I
```

Neste caso prevalescerá o tipo simples precisão definido na linha 20.

Dicas:

(1) Defina as variáveis inteiras como aquelas que começam com as letras no intervalo I – N. As variáveis strings no intervalo A – F, as variáveis dupla precisão com V e as de simples precisão com S.

Exemplo:

```
110 DEFSTR A – F
110 DEFDBL V
120 DEFINT I – N
130 DEFSNG S
```

(2) Sempre que possível, use variáveis inteiras, principalmente nos contadores de laços. Isto fará com que o programa seja executado mais rapidamente e ocupe menos espaço de memória.

Exemplo:

```
100 DEFINT I
  .
  .
500 FOR I = 1 TO 10000
  .
  .
600 NEXT I
```

Observação. Verifique o Cap. 5, para obter informações sobre laços.

3.9. Trocando o conteúdo de duas variáveis

A execução da instrução SWAP < var 1 >, < var 2 > troca os valores de duas variáveis entre si. As variáveis devem ser do mesmo tipo.

Exemplo:

```
100 A% = 5: B% = 10
110 SWAP A%, B%
120 PRINT A%; B%
RUN
 10   5
```

Após esta instrução, a variável A% terá o valor da variável B% e B% terá o valor de A%.

Lembre-se: as duas variáveis devem ser do mesmo tipo.

3.10 Interrompendo a execução do programa

Para interromper a execução do programa, pressione ctl C (juntos). Outra maneira é colocar a instrução STOP dentro do programa.

Exemplo:

```
160 IF VTL > 60 THEN STOP
```

Observação. Para pausa, pressione ctl S.

3.11 Retomando a execução

Para continuar a execução a partir de onde foi interrompido por ctl C ou STOP, digite CONT e pressione < ENTER >.

O comando CONT permite recomeçar a execução a partir do ponto onde o programa foi interrompido.

Esta instrução é útil quando se deseja verificar o conteúdo de alguma variável durante a execução.

O comando CONT é invalidado se o programa foi editado ou algum valor da memória foi alterado durante a interrupção.

O comando CONT também é invalidado se ocorreu erro no programa.

Observação. Pode-se também reiniciar a execução com GOTO < linha >.

3.12 Retornando ao sistema operacional

Para sair do BASIC e retornar ao sistema operacional, utiliza-se o comando SYSTEM.

Antes de retornar ao sistema operacional, este comando fecha todos os arquivos abertos.

Pode-se usar SYSTEM tanto no modo imediato como no modo programado dentro do programa.

Exemplo:

```
1000 IF OP$ = "FIM" THEN SYSTEM
```

3.13 Testes

1) Dado:

```
10 A = 10 : B = 20 : C = 30 : END
20 PRINT A; B; C
```

O que será mostrado?

a) 10 20 30
b) 10 20 30
c) 10 20 30
d) 102030
e) nenhuma das anteriores.

2) Qual alternativa define variáveis simples precisão, dupla precisão, inteiras, e strings, respectivamente?

a) DEFDBL, DEFINT, DEFSTR, DEFSNG;
b) DEFSTR, DEFDBL, DEFINT, DEFSNG;
c) DEFSNG, DEFDBL, DEFINT, DEFSTR;
d) DEFINT, DEFSNG, DEFDBL, DEFSTR;
e) nenhuma das anteriores.

3) Dado:

```
10 A% = 5: B% = 7 : C! = 9
20 SWAP A%, B%: SWAP B%, C! : SWAP A%, C!
30 PRINT A%, B%; C!
```

O que será mostrado?

a) 5, 9, 7
b) 5, 7, 9
c) 9, 7, 5

d) 7, 5, 9
e) nenhuma das anteriores.

4) Dado:

```
10 DEFINT A – F : DEFSTR D – G
```

Qual é o tipo das variáveis que tenham a letra H no primeiro caracter do nome?

a) inteiro;
b) string;
c) simples precisão;
d) dupla precisão;
e) nenhuma das anteriores.

5) Qual das alternativas contém uma instrução INPUT que executada não mostrará o ponto de interrogação?

a) INPUT "Entre com a taxa média"; TXMD
b) INPUT "Entre com a taxa média", TXMD
c) nenhuma das anteriores.

6) Dado:

```
10 A = 3
20 ON A GOTO 40, 50
30 PRINT "linha 30";
40 PRINT "linha 40";
50 PRINT "linha 50"
```

O que será mostrado?

a) linha 50
b) linha 40
c) linha 40linha 50
d) linha 30linha 40linha 50
e) nenhuma das anteriores.

7) O que será mostrado?

```
10 N = 4
20 ON N GOTO 30, 40, 50
25 STOP
30 PRINT "primeiro"
40 PRINT "segundo"
50 PRINT "terceiro"
60 PRINT "quarto"
70 END
```

a) primeiro
b) segundo
c) terceiro
d) quarto
e) nenhuma das anteriores.

8) O que será mostrado?

```
10 A$ = "Aa"
20 B$ = "AA"
30 IF A$ > B$ THEN PRINT "maior" ELSE PRINT "menor"
```

a) maior
b) menor

9) Dado:

```
10 X = 2.6
20 ON X GOTO 30, 40, 50
30 PRINT "30": END
40 PRINT "40": END
50 PRINT "50": END
```

O que será mostrado?

a) 30
b) 40
c) 50
d) ocorrerá erro
e) nenhuma das anteriores

10) Dado: 10 X = 3: Y = 5: Z = 7
20 SWAP X, Y: SWAP Y, Z: SWAP X, Z

Quais serão os novos valores das variáveis X, Y e Z?

a) 3, 5, 7;
b) 5, 3, 7;
c) 7, 3, 5;
d) 3, 7, 5;
e) nenhuma das anteriores.

3.14 Exercícios propostos

1) As notas em seus últimos testes são N1, N2, N3, N4 e N5. Elabore um programa para calcular a média das notas.
2) Escreva um programa que calcule o número de segundos em 1 min, 1 hora, 1 dia e 1 ano.
3) Escreva um programa que converta temperatura Fahrenheit para Celsius.

   $C = 5/9 (F - 32)$
4) Escreva um programa que verifique se determinado ano é bissexto.
5) Elabore um programa que leia os comprimentos dos três lados de um triângulo (L1, L2 e L3) e calcule a área do triângulo de acordo com a fórmula.

$$A = \sqrt{(T\,(T - L1)\,(T - L2)\,(T - L3))}$$

onde

$$T = \frac{L1 + L2 + L3}{2}$$

6) Elabore um programa que leia três números e imprima o maior.

4

OUTRAS DECLARAÇÕES

"O fluxograma é uma parte da documentação vendida por muito mais do que vale."

Frederick P. Brooks

4.1 Entrando com dados

4.1.1 Lendo apenas um caracter do teclado

A função INKEY$ permite que se entre com apenas um caracter do teclado por vez. Quando a seqüência de execução do programa passa pela função INKEY$, o microcomputador verifica em questão de milissegundos se existe alguma tecla pressionada; caso exista, é retornado o valor ASCII da tecla; caso contrário, é retornado o valor nulo ("").

Exemplo:
```
10 A$ = INKEY$
20 IF A$ = "" THEN 10
30 PRINT "Foi pressionado"; A$
40 GOTO 10
```

Execute-o para ver o resultado.

Perceba que não é necessário pressionar ENTER. Enquanto a máquina não encontrar uma tecla pressionada, o controle não passará para a linha 30. Este programa não verifica algumas teclas, como, por exemplo, BACKSPACE.

Refazendo o exemplo anterior, podemos também verificar outras teclas.

```
10 A$ = INKEY$
20 IF A$ = "" THEN 10
30 IFA$ = CHR$(8) THEN PRINT "BACKSPACE" : GOTO10
40 PRINT "Foi pressionado"; A$
50 GOTO 10
```

Observações:

(1) A função INKEY$ não mostra o caracter pressionado. Deve ser providenciada a sua impressão no video.

(2) Pressione ctl C para interromper a função INKEY$.

Sugestão. Use a função INKEY$ para a entrada de opções de menus.

Exemplo:

```
10 X$ = INKEY$
20 IF X$ = "" THEN 10
30 IF X$ = "I" THEN 500
40 IF X$ = "A" THEN 600
50 IF X$ = "E" THEN 700
60 GOTO 10
```

4.1.2 Entrando com uma linha inteira

A declaração LINE INPUT [;] [mensagem]; <var$ > permite entrar com uma linha inteira (até 254 caracteres) em uma variável string sem o uso de delimitadores (aspas).

Exemplo:

```
100 LINE INPUT "Entre com o endereço"; E$
```

Se LINE INPUT estiver seguido imediatamente por um ponto e vírgula, então o pressionamento da tecla ENTER ao final da entrada não ocasionará um "scroll" automático.

4.1.3 Entrando com dados sem mostrar no video

A função INPUT$(n) lê n caracteres do teclado sem mostrá-los no video.
Para interromper INPUT$(n) pressione ctl C.

Exemplo:

```
100 PRINT "Digite C para continuar ou";
105 PRINT "P para parar"
110 X$ = INPUT$(1)
120 IF X$ = "C" THEN 500
130 IF X$ = "P" THEN END
140 GOTO 110
```

Dicas:

(1) Use a instrução INPUT$(n) para introduzir senhas, pois a mesma não é mostrada no video quando se está digitando.

Exemplo:

```
600 SENHA$ = INPUT$ (5)
610 IF SENHA$ < > "X2043" THEN 600
```

(2) Não deixe as senhas à vista ou escritas em manuais de operação etc.

(3) As senhas não devem aparecer no video ou na impressora.

(4) Às vezes é interessante colocar níveis de acesso, tendo uma senha em cada nível.

Por exemplo:

Em um sistema de Folha de Pagamento coloque uma senha para verificação do salário de funcionários e outra para alterar o salário.

(5) As senhas devem ser alteradas periodicamente.

4.2 Mostrando informações

4.2.1 Usando a tabulação

PRINT TAB (coluna)

O cursor será posicionado na coluna especificada. Se a posição especificada for maior que 80, a impressão ocorrerá na próxima linha. O valor da coluna deve estar entre 1 e 255.

Exemplo:

```
10 PRINT TAB (5) "POSIÇÃO 5" TAB (25) "POSIÇÃO 25"
RUN
     POSIÇÃO 5                POSIÇÃO 25
```

Observação. Não pode haver espaços entre TAB e ().

4.2.2 Imprimindo espaços em branco

PRINT SPC(n)

Apresenta n caracteres brancos, sendo que n deve estar entre 0 e 255.
Esta instrução somente pode ser utilizada nas instruções PRINT e LPRINT.

Exemplo:

```
10 PRINT "LESTE" SPC (15) "OESTE"
RUN
LESTE                OESTE
```

Observação. Não pode haver espaço entre SPC e ().

4.2.3 Imprimindo com formato prefixado

PRINT USING "formato"; < lista de variáveis >

Permite especificar um formato para impressão de valores numéricos ou alfanuméricos. A instrução imprime a lista de variáveis de acordo com o formato especificado.

a) FORMATO PARA IMPRESSÃO DE STRINGS

"!" Indica que apenas o primeiro caracter do string fornecido será mostrado.

Exemplo:
```
PRINT USING "!"; "CAROLINA"
C
```


\espaços\ Especifica que 2 + n caracteres do string devem ser mostrados.

Exemplos:

```
PRINT USING "\\"; "CAROLINA"
CA
PRINT USING "\ \"; "CAROLINA"
CAR
PRINT USING "\  \"; "CAROLINA"
CARO
```

"&" O string será mostrado exatamente como foi introduzido.

Exemplo:

```
10 A$ = "VEJA" : B$ = "ALI"
20 PRINT USING "!"; A$;
30 PRINT USING "&"; B$
RUN
VALI
```

b) FORMATO PARA IMPRESSÃO DE NÚMEROS

\# Este sinal é usado para representar a posição de cada dígito. Se o formato for maior que o valor numérico a ser impresso, as posições não utilizadas a esquerda do ponto decimal serão preenchidas com brancos e as da direita com zeros.

Exemplos:

Formato	Valor	Mostrado
####.##	18	18.00
	−285	−285.00
	31.6	31.60
	.06	0.06
	2.138	2.14

Se o número for maior que o formato, será mostrado % na frente do número mostrado.

Exemplo:

Se o formato é "##" e o número for 123, será mostrado %123, indicando que houve "overflow".

O ponto decimal pode ser colocado em qualquer parte do formato. Se houver mais dígitos a direita do ponto decimal que o formato, será feito arredondamento.

Exemplos:

(1) PRINT USING "##.##"; 20.3, 1.668
20.30 1.67

(2) PRINT USING "Valor total = ###.##"; 987.65
Valor total = 987.65

Uma vírgula colocada no formato em qualquer lugar a esquerda do ponto decimal faz com que na apresentação do número seja inserida uma vírgula a cada três dígitos a esquerda do ponto decimal. Uma vírgula posicionada no final do formato é impressa como parte do valor a ser executado.

Exemplos:

(1) PRINT USING "####,.##"; 1234.5
1,234.50

(2) PRINT USING "####.##,"; 1234.5
1234.50,

Observação. A vírgula a cada três dígitos corresponde a notação americana.

\+ O sinal de adição colocado no início ou no final do formato faz com que o sinal do número (positivo ou negativo) seja mostrado antes ou depois do número.

Exemplo:

PRINT USING" +##.##"; – 68.95, 2.4, 55.6
– 68,95 + 2.40 + 55,60

– Um sinal de subtração no final do formato faz com que os números negativos sejam mostrados com o sinal de subtração a direita. Se o número for positivo, será mostrado um espaço.

Exemplo:

PRINT USING "##.##–"; – 68.95, 22.4
68.95 – 22.40

** Dois asteriscos no início do formato fazem com que os espaços em branco à esquerda do número sejam preenchidos com asteriscos.

Exemplo:

PRINT USING "**####.#"; 12.39, – 0.9
***12.4 ***– 0.9

$$ Dois dólares imprimem um dólar a esquerda do número.

Exemplo:

PRINT USING "$$####.##"; 456.78
$456.78

**$ Combina o efeito de ** e $$.

Exemplo:

PRINT USING "**$##.##"; 2.34
***$2.34

↑↑↑↑ Estes quatro caracteres permitem que se escreva um número no formato exponencial, ou seja, E + XX.

Exemplos:

PRINT USING "##.##↑↑↑↑"; 234.56
2.35E +02
PRINT USING "+.##↑↑↑"; 123
+.12E +03

Dica. Entre com o programa abaixo para testar todas as opções.

```
10 INPUT "Entre com o formato"; F$
15 INPUT "Entre com o número"; N#
20 PRINT USING F$; N#
30 GOTO 10
```

4.3 Definindo o tamanho da linha do video ou impressora

A instrução WIDTH < tamanho > define a largura da linha do video.

Exemplo:

10 WIDTH 80

A instrução WIDTH LPRINT < tamanho > define a largura da linha da impressora.

Exemplo:

10 WIDTH LPRINT 120

O "default" para o video é 80 colunas, e para a impressora, 132 colunas.

Dicas:

(1) WIDTH 0

Insere uma linha em branco entre as linhas do programa quando mostrado no video.

WIDTH LPRINT 0

Insere uma linha em branco entre as linhas do programa quando mostrado na impressora.

(2) WIDTH 255

O tamanho da linha será infinito. Não será inserido automaticamente um < CR >. Use-a para evitar o "scroll" automático.

4.4 Verificando a posição do cursor e do cabeçote

A função POS (X) retorna a posição atual do cursor. A posição mais à esquerda é 1. O argumento X tem apenas efeito sintático.

Exemplos:

(1) PRINT TAB (5) "Estou aqui" POS (X)
Estou aqui 15

(2) PRINT TAB (5) POS (X)
5

(3) 10 PRINT TAB (5) "Ah!!";
20 X = POS (Y)
30 PRINT X
RUN
Ah!! 9

A função LPOS (X) retorna a posição atual da cabeça de impressão da impressora no "buffer" da impressora (esta posição não é necessariamente a posição física da cabeça de impressão).

4.5 Ninhos de IFs

As declarações IF . . . THEN . . . ELSE podem ser aninhadas.

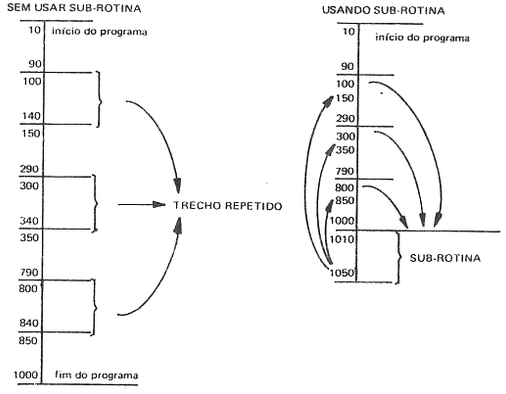
Exemplo:

```
IF X > Y THEN PRINT "maior" ELSE IF Y > X THEN
              PRINT "menor" ELSE PRINT "igual"
```

4.6 Usando sub-rotinas

4.6.1 Definindo uma sub-rotina

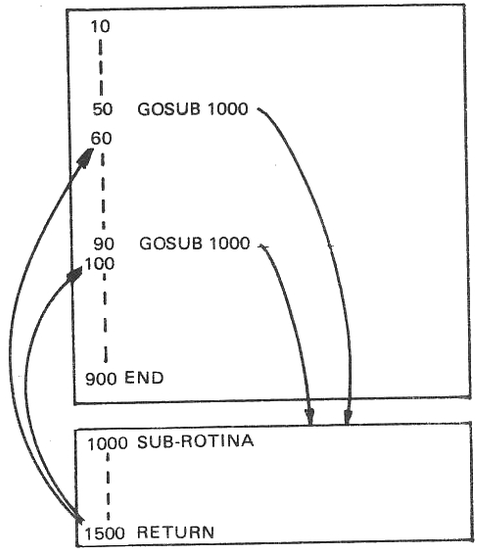
Quando estamos escrevendo um programa, pode acontecer que existam trechos que estejam repetidos várias vezes em diferentes lugares do programa. Esse trecho pode ser colocado à parte e, quando for necessário usá-lo, especificar um desvio com retorno. Esse trecho passa a se chamar sub-rotina. Desta maneira o programa fica menor e mais legível.



4.6.2 Como desviar a execução para a sub-rotina

a) Desvio incondicional

A instrução GOSUB < linha > desvia o fluxo do programa para a sub-rotina que começa na linha especificada e retorna a instrução seguinte ao GOSUB após a sub-rotina ter sido executada.



b) Desvio condicional

A instrução ON < expr > GOSUB L1, L2, L3, . . . desvia o fluxo do programa para uma das linhas especificadas dependendo do valor da expressão.

Se o valor da < expr > for 1, o fluxo será desviado para a sub-rotina que começa em L1, e assim por diante.

Exemplos:

(1) 600 ON X GOSUB 5000, 5500, 6000

(2) 100 ON X + 1 GOSUB 1000, 1500, 2000

Observações:

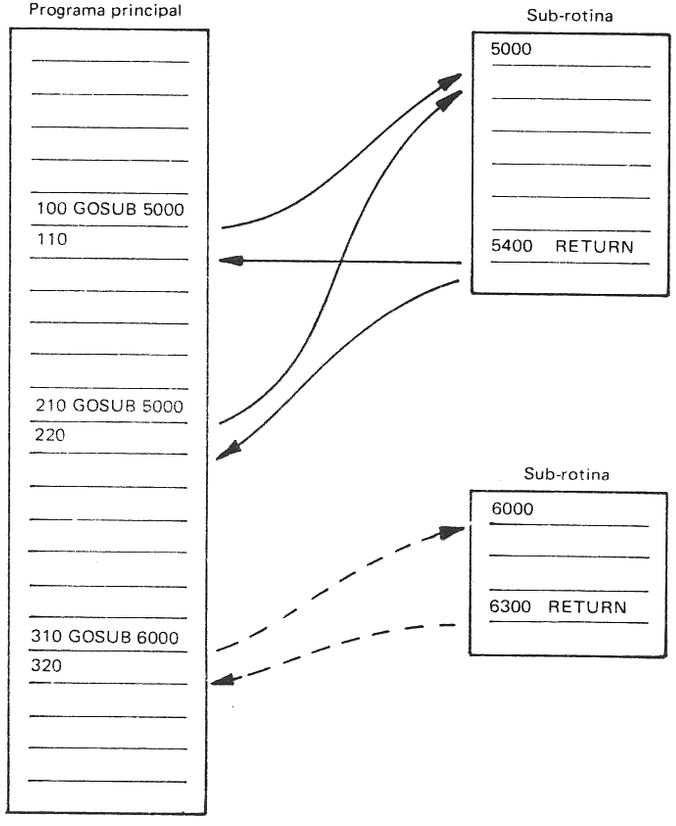
(1) O fluxo continuará na linha seguinte se o valor da expressão for 0 ou maior que o número de linhas especificado.

(2) Ocorrerá erro se o valor da expressão for negativo ou maior que 255.

(3) A parte fracionária será arredondada se o valor da expressão não for inteiro.

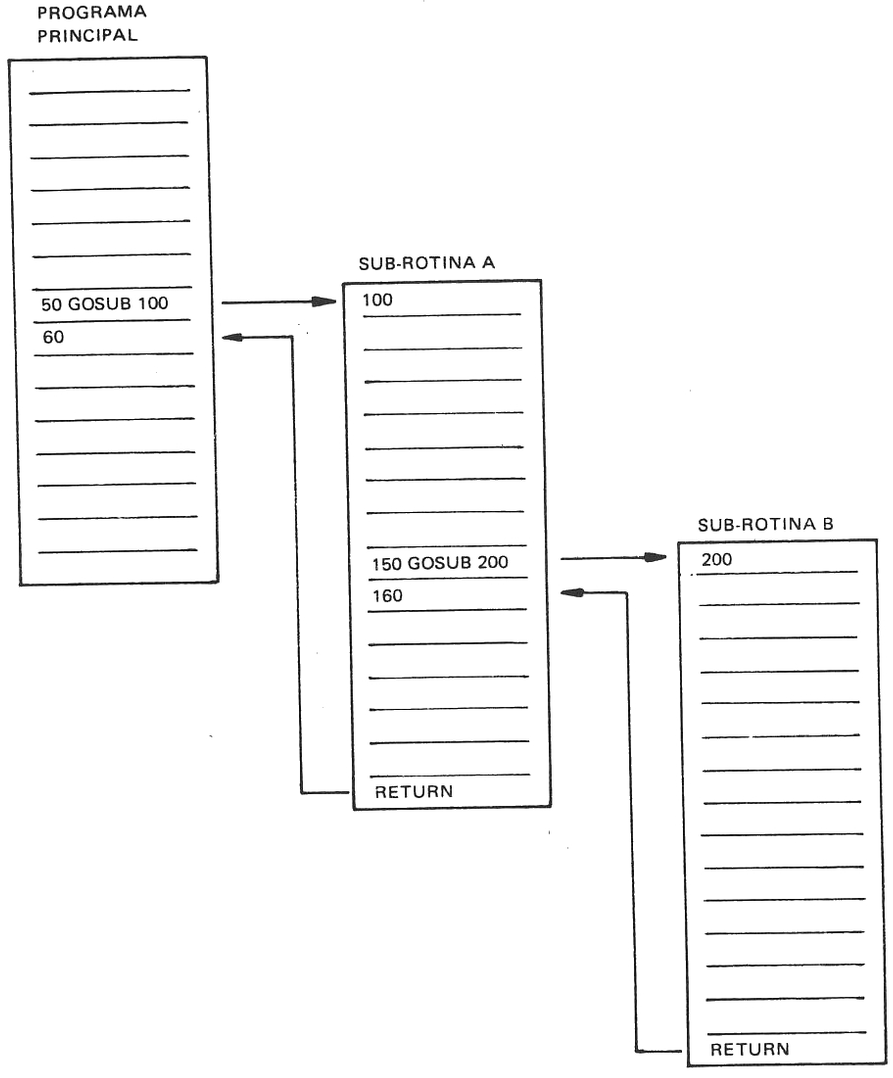
4.6.3 Encerrando a execução da sub-rotina

A instrução RETURN encerra a execução da sub-rotina e retorna o controle para a declaração seguinte ao último GOSUB.



Observações:

(1) A sub-rotina pode ser chamada várias vezes dentro de um programa e pode ser chamada ainda por uma outra sub-rotina. O número de sub-rotinas encadeadas é limitado apenas pela memória disponível.



(2) A sub-rotina pode conter mais de uma instrução RETURN. Entretanto, por questões de legibilidade, é aconselhável colocar apenas uma instrução RETURN na sub-rotina.

(3) As sub-rotinas podem ser colocadas em qualquer parte do programa; no entanto, é recomendável que elas sejam distintas em relação ao programa principal, por questões de legibilidade.

(4) Para prevenir acessos indevidos a uma sub-rotina, esta pode ser precedida pelas instruções STOP, END ou GOTO.

Exemplo:

```
500.. . . . . .
510 . . . . . .
520 END
600 '
610 '    cálculo da taxa interna
620 '
630 . . . . . .
640 . . . . . .

700 RETURN
```

Dicas:

(1) Coloque uma linha em branco antes e depois da sub-rotina, facilitando a visualização da mesma.

(2) É aconselhável que os desvios para a sub-rotina referenciem a primeira linha executável, e não a linha de comentário.

Exemplo:

```
800 GOSUB 5000
    . . . . . .
    . . . . . .
4970 '
4980 '          imprimindo o total geral
4990 '
5000 LPRINT "Total do mês ="; VLTTM#
```

(3) É aconselhável que as sub-rotinas tenham apenas um ponto de entrada e um ponto de saída.

4.7 Dicas para a criação de telas

(1) As telas devem ser projetadas de forma que o documento possa ser digitado lendo-se da esquerda para a direita e de cima para baixo.

(2) Use letras minúsculas nos títulos da tela, diferenciando assim do que está sendo digitado.

```
SOFTBRAS          Inclusão de Nota Fiscal

Número      [1234]
Produto     [PARAFUSO 15.34]
Valor       [500]
Data        [11/02/82]
ICM         [10]
IPI         [5]

                  confirma (S/N)
```


(3) Faça com que as mensagens emitidas pelo programa fiquem piscando por algum tempo no video. Isto facilita a visualização das mesmas pelo operador.

(4) Coloque a identificação da empresa e da tela na primeira linha da tela. Preencha toda a segunda e a penúltima linha com caracteres contínuos (CHR$ (95)). Os caracteres contínuos têm melhor efeito visual que os traços. Na última linha coloque as mensagens.

Exemplo:

```
Softbras Ltda        Menu principal

                  (I)nclusão
                  (A)lteração
                  (E)xclusão

        Entre com sua opção
```

(5) Coloque sempre confirmação das informações introduzidas.

Exemplo:

```
Softbras              Exclusão

Código [          ]
Nome   [          ]
Endereço [        ]
Cidade [          ]

        Confirma (S/N)
```

(6) Na criação de menus use a seguinte técnica.

(A)lterar, (I)ncluir, (E)xcluir, (F)inalizar ao invés de associar um número a cada opção.

Exemplo:

```
Modelo 1
Alterar . . . . 1
Incluir . . . . 2
Excluir . . . . 3
Finalizar . . . 4
```

```
Modelo 2
(A)lterar
(I)ncluir
(E)xcluir
(F)inalizar
```

(7) Coloque parada automática no fim de cada tela de informação e espere acionar a tecla ENTER ou outra tecla, antes de prosseguir.

(8) As informações muito agrupadas na tela dificultam a visualização. Somente coloque juntas as informações relacionadas.

(9) Cuidado com o local em que as mensagens serão mostradas. Se o local não estiver limpo, a mensagem será colocada em cima da anterior. Porém, se existirem caracteres não cobertos pela nova mensagem, eles ficarão lá. Isto pode atrapalhar o operador e levá-lo a cometer erros.

(10) Organize a entrada dos dados de forma que eles permaneçam no video por algum tempo após a digitação. Isto dá ao operador a oportunidade de corrigir erros. Se os dados desaparecem tão logo são entrados, não há oportunidade para que o operador reveja a digitação e ocasionalmente detecte algum erro.

4.8 Dicas para a criação de relatórios

(1) Em formulários pré-impressos difíceis de serem posicionados, deixe um pequeno quadrado no canto superior do formulário para o alinhamento do mesmo. Enquanto o programa não imprimir uma cruz dentro do quadrado, o formulário ainda não está posicionado corretamente.

(2) As informações mais usadas devem ser colocadas em lugar onde possam ser achadas rapidamente. Em geral nas margens esquerda e direita do formulário.

(3) Nos programas em que é gerado grande volume de relatórios, é recomendável a utilização de opção para reimpressão a partir de determinada página.

(4) Coloque opção para número de cópias quando for necessário emitir mais de uma via.

(5) Coloque a data de emissão em todos os relatórios, isto facilita a identificação do relatório mais recente.

(6) Em alguns relatórios pode ser interessante colocar a hora em que foi emitido.

(7) Nos relatórios de uso interno não coloque cabeçalhos complexos, simplifique-os.

(8) Antes de começar a impressão, verifique se a impressora está ligada e se o papel está posicionado corretamente.

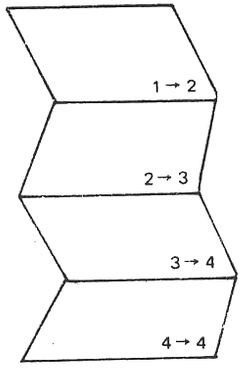
(9) Identifique sempre a última página do relatório.

Primeira opção:

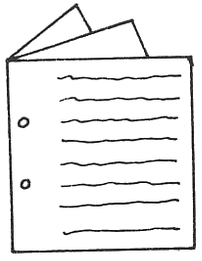
Imprimir "FIM DO RELATÓRIO" após a última linha impressa.

Segunda opção:

Numerar as páginas da seguinte maneira: No final da primeira página coloque 1 → 2, no final da segunda página coloque 2 → 3 e assim por diante. Na última página coloque n → n.
Desta forma é fácil visualizar quanto não existe continuação.



(10) Deixe espaços a esquerda no relatório quando for necessário arquivar a listagem.



(11) Sempre que possível, use apenas 80 colunas para o relatório, permitindo assim maior flexibilidade quanto ao tipo de impressora (80 ou 132 colunas) ou ao papel utilizado.

(12) Para imprimir em letras "garrafais" na impressora, use CHR$ (18), e, para voltar ao normal, CHR$ (20), (Obs.: este código pode variar de impressora para impressora).

Exemplo:

```
LPRINT CHR$ (18) "RELATÓRIO DE VENDAS" CHR$ (20)
```

(13) Para imprimir em negrito, use CHR$ (13) (Obs.: este código pode variar de impressora para impressora).

Exemplo:

```
LPRINT "SOFTBRAS" CHR$ (13) "SOFTBRAS"
```

(14) Os caracteres para controle da impressora variam de impressora para impressora; portanto, documente o uso de tais caracteres para facilitar possíveis conversões.

Exemplos:

```
10 SLPG$ = CHR$ (12) ' Salta p/próx. pág.
20 GRRF$ = CHR$ (18) ' Letras "garrafais"
30 NEGR$ = CHR$ (13) ' Imprime em negrito.
```

(15) Use letras "garrafais" para imprimir o cabeçalho. Utilize CHR$ (18).

4.9 Testes

1) Dado:

```
20 A# = 123456789.899
40 PRINT USING "##########,.##"; A#
```

O que será mostrado?

a) 123456789, . 89
b) 123456789.89
c) 123, 456, 789, 89
d) 123, 456, 789.90
e) nenhuma das anteriores.

2) O que será impresso?

```
LPRINT USING "+####"; – 68.958, 22.449, – 7.01, 33.45
```

a) – 68,22, – 7, 33
b) – 68, + 22, – 7, 33
c) 68, 22, – 7, + 33
d) 68, 22, 7, 33
e) – 69, + 22, – 7, + 33

3) O que será impresso?

```
LPRINT USING "##.## – "; 68.958, 22.449, – 7.01, 33.45
```

a) 68.95 –, 22.45, 7.01 – , 33.45
b) 68.95, 22.45, 7.01, 33.46
c) 68.96 –, 22.45,7.01 –, 33.45
d) 68.96 –, 22.45 +, 7.01 –, 33.45 +
e) 68.96, 22.45, 7.01 –, 33.45

4) O que será impresso?

```
LPRINT USING "##.## +"; – 68.958, 22.449, – 7.01, 33.45
```

a) 68.95 –, 22.45 +, 7.01 –, 33.45 +
b) 68.96 –, 22.45, 7.01 –, 33.45
c) 68.95 –, 22.45, 7.01, 33.45
d) 68.95, 22.45, 7.01, 33.45
e) 68.96 –, 22.45 +, 7.01 –, 33.45 +

5) O que será mostrado?

```
PRINT USING "##.##"; 13.563, 13.565, 100.00
```

a) 13.56, 13.57, % 100.00
b) 13.56, 13.56, 100.00
c) 13.57, 13.57, % 100.00
d) 13.56, 13.56, % 100.00
e) nenhuma das anteriores.

6) O que será mostrado?

```
PRINT 10; "AAA"; 20; "AAA"; "BBB"
```

a) 10 AAA20 AAA BBB
b) 10AAA 20AAA BBB
c) 10 AAA – 20AAA – BBB
d) 10AAA20AAABBB
e) 10 AAA 20 AAABBB

7) O que será mostrado?

```
PRINT USING "**$ ###.#"; 3.57
```

a) **#3.5
b) **$ 3.6
c) ****$ 3.6
d) ***3.5
e) $$3.5

8) O que será mostrado?

```
PRINT 3/2 * 1.5
```

a) 2.25
b) 1
c) 0
d) 5
e) nenhuma das anteriores.

9) O que será mostrado?

```
10 PRINT "50";
20 PRINT "60";
30 PRINT "70";
40 PRINT "80"
```

a) 50
b) 80
c) 50 60 70 80
d) 50607080
e) nenhuma das anteriores.

10) O que será mostrado?

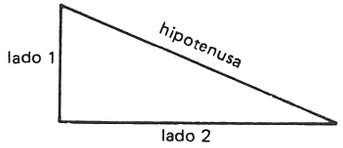
```
 10 A = 3
 20 ON A GOSUB 100, 200, 300
100 PRINT "100"
200 PRINT "200"
300 PRINT "300"
```

a) 300
b) 100 200 300
c) 3
d) ocorrerá erro;
e) nenhuma das anteriores.

4.10 Exercícios propostos

1) Escreva um programa que verifique se um número é par ou ímpar.
2) Elabore um programa que calcule a hipotenusa de um triângulo retângulo (algoritmo de Pitágoras).

X1 = Lado1 ↑ 2
X2 = Lado2 ↑ 2
Hipotenusa = $\sqrt{(X1 + X2)}$



3) Elabore um programa que leia três números e imprima o maior e o menor número.

5

LAÇOS

"Na programação, o demônio se esconde nos detalhes."

Niklaus Wirth

5.1 Definindo um laço

Laço ou "loop" é um grupo de instruções que devem ser executadas determinado número de vezes.

Por exemplo:

```
10 I = 0
20 I = I + 1
30 . . . . . .
40 . . . . . .
50 . . . . . .
 .
 .
 .
150 IF I < 100 THEN GOTO 20
```

Neste caso serão executadas as linhas começando na 20 até a 150, enquanto a variável I for menor que 100.

Um laço pode ser definido usando-se um contador que é incrementado até atingir um limite previamente determinado, como no exemplo anterior, ou pode ser definido usando-se as instruções: FOR/NEXT ou WHILE/WEND.

Por exemplo:

```
10 FOR I = 1 TO 100 STEP 1
```

```
150 NEXT I
```

ou usando WHILE/WEND

```
10 WHILE I < 101
.
.
.
150 WEND
```

Ambas as maneiras de se definir um laço produzem o mesmo resultado, entretanto a segunda maneira (usando FOR/NEXT ou WHILE/WEND) é bem mais simples.

5.2 Definindo o início e fim do laço

As instruções FOR e WHILE definem o início do laço, enquanto que as instruções NEXT e WEND definem o fim do laço.

A sintaxe das instruções FOR/NEXT é:

```
FOR contador = x TO y [STEP z]
  .
  .
NEXT contador
```

A variável ou expressão numérica X define o valor inicial do contador e a variável ou expressão numérica Y define o valor final.

O contador é uma variável que terá um novo valor a cada execução do laço. Na primeira vez que o laço for executado o contador terá o valor X, na segunda vez terá o valor X + Z, onde Z é o incremento (definido no STEP) a ser somado a cada execução do laço. Na terceira vez terá o valor X + Z + Z, e assim por diante, até ultrapassar o valor de Y.

Exemplos:

(1)
```
10 FOR I = 1 TO 10 STEP 3
20    PRINT I
30 NEXT I
RUN
 1
 4
 7
 10
```

(2)
```
10 FOR J = 3 TO 8 STEP 2
20    PRINT J
30 NEXT J
RUN
 3
 5
 7
```


Quando for encontrada a instrução NEXT, a seqüência de execução retorna para a instrução FOR, onde o contador será incrementado e será verificado se o valor do contador não ultrapassou o valor de Y. Caso tenha ultrapassado, a seqüência de execução é desviada para a instrução seguinte a NEXT. Se o incremento não for especificado, será assumido que é igual a +1.

Por exemplo:

```
10  FOR K = 1 TO 3
20      PRINT K
30  NEXT K
RUN
 1
 2
 3
```

Observação. O contador das instruções WHILE e WEND será executado enquanto a condição for verdadeira.

As instruções WHILE e WEND têm a seguinte sintaxe:

```
WHILE  condição
  .
  .
  .
WEND
```

O laço definido pelas instruções WHILE e WEND será executado enquanto a condição for verdadeira.

Por exemplo:

```
10 WHILE OP$ < > "S" AND OP$ < > "N"
20 INPUT "Os dados estão corretos (S/N)"; OP$
30 WEND
```

5.3 Quando o incremento é negativo

O valor do incremento pode ser negativo. Neste caso o valor final do contador deve ser menor que o valor inicial. O valor do incremento será subtraído do valor do contador, e o ciclo é repetido até que o valor do contador seja menor que o valor final.

Exemplos:

(1)
```
10 FOR I = 3 TO 1 STEP – 1
20    PRINT I
30 NEXT I
RUN
 3
 2
 1
```

```
(2) 10 '
    20 ' Contagem regressiva
    30 '
    40 FOR I = 50 TO 1 STEP – 1
    50    PRINT "Faltam"; I; "segundos"
    60 NEXT I
    70 PRINT "Fogo!!!"
```

5.4 Quando o incremento é fracionário

O valor do incremento pode ser um número fracionário.

Exemplos:

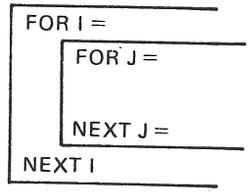
```
(1)  10 FOR J = 0.5 TO 2 STEP 0.5
     20    PRINT J
     30 NEXT J
     RUN
     0.5
     1
     1.5
     2

(2)  10 FOR K = 5 TO 3 STEP – 0.5
     20    PRINT K * K * K
     30 NEXT K
     RUN
      125
      91.125
      64
      42.875
      27
```

5.5 Quando os laços estão aninhados

Os laços podem ser aninhados, isto é, podem ser definidos dentro de um outro laço.

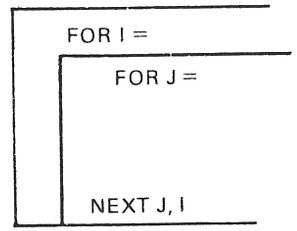
Quando há laços aninhados, cada um deve ter uma variável distinta como contador. A instrução NEXT para o laço mais interno deve aparecer antes do NEXT referente ao laço mais externo.



Exemplo:

```
10 FOR X = 1 TO 5
20    FOR L = 1 TO 10
30       FOR K = 2 TO 7
.           .......
.           .......
.           .......
70       NEXT K
80    NEXT L
90 NEXT X
```

Se os laços tiverem o mesmo ponto de encerramento, pode ser utilizada uma única instrução NEXT para todos os laços.



Exemplo:

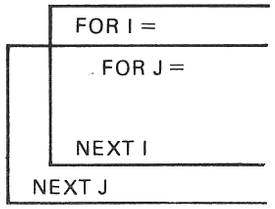
```
10 FOR I = 1 TO 10
20    FOR J = 1 TO 20
30       ............
40       ............
50 NEXT J, I
```

O contador da instrução NEXT pode ser omitido e, neste caso, a instrução NEXT será referente a instrução FOR mais recente.

Por exemplo:

```
10 FOR J = 20 TO 50
.
.
.
50 NEXT
```

Se for encontrada uma instrução NEXT antes do seu FOR, ocorrerá erro e a execução será cancelada.



Exemplo:

```
10 FOR L = 1 TO 10
20    FOR K = 1 TO 25
.
.
50    NEXT L
60 NEXT K
```

Neste caso ocorrerá erro "NEXT without FOR".

5.6 Um pouco de estilo

O laço é um dos recursos mais importantes em programação. Entretanto, se a sua colocação não for bem documentada, o programa pode se tornar muito difícil para ser lido ou entendido.

Logo abaixo, estão algumas "dicas"; desde que observadas, podem ajudar em muito a aumentar a legibilidade do programa.

Dicas:

(1) Coloque as instruções de dentro do laço deslocadas três colunas em relação as instruções FOR/NEXT ou WHILE/WEND, facilitando a visualização.

Exemplos:

a)
```
150 FOR K = N TO L
160         VLTT = VLTT + VLTDR
170 NEXT K
```

b)
```
10 FOR S = 1 TO 10
20    PRINT S ↑ 0.5
30 NEXT S
```

(2) Coloque uma linha em branco antes e depois de um laço, facilitando a visualização do mesmo.

Exemplo:

```
145 N# = 0
150 ABT# = 10000000000#
155 '
160 FOR I% = 0 TO 9
170    ABT# = ABT#/10
175    IF N# < ABT# THEN 195
180    N# = N# - ABT#
190    GOTO 175
195 NEXT I%
200 '
205 SOMA% = 0
210 GOSUB 1000
220 IF CH = 1 THEN 300 ELSE 400
```

(3) Podem-se colocar todas as instruções de um laço numa única linha de programação, o que pode ser eficiente em termos de máquina, mas é horrível para o leitor humano.

Exemplo:

```
400 FOR I = 1 TO N : PRINT I : NEXT I
```

5.7 Dicas para otimizar laços

(1) Não recalcule constantes dentro de um laço.

Exemplo:

Ineficiente

```
FOR I = 1 TO 10
  FOR J = 1 TO 10
    FOR K = 1 TO 10
      A = B + X * Y + (I * J * K) + A
```

Eficiente

```
W = B + X * Y
FOR I = 1 TO 10
  FOR J = 1 TO 10
    FOR K = 1 TO 10
      A = W + (I * J * K) + A
```

(2) Os laços mais internos têm maior prioridade para serem otimizados.

(3) Diminua o número de vezes que o laço é incrementado.

Exemplo:

Ineficiente

```
FOR I = 1 TO 1000
   B (I) = 0
NEXT I
```

Eficiente

```
FOR I = 1 TO 1000 STEP 2
   B (I) = 0
   B (I + 1) = 0
NEXT I
```

Observação. Verifique o Cap. 6 para obter informações sobre matrizes.

(4) Sempre que possível, agrupe os laços.

Exemplo:

Ineficiente

```
FOR I = 1 TO 500
   X (I) = 0
NEXT I
FOR I = 1 TO 500
   Y (I) = 0
NEXT I
```

Eficiente

```
FOR I = 1 TO 500
   X (I) = 0
   Y (I) = 0
NEXT I
```

(5) Diminua o número de vezes que os laços são inicializados.

Ineficiente

```
FOR K = 1 TO 20
   FOR J = 1 TO 10
     FOR L = 1 TO 5
       A (K, J, L) = 0
     NEXT L
   NEXT J
NEXT K
```

O laço externo será inicializado uma vez, o do meio 20 vezes, e o mais interno 200 vezes. Portanto, teremos neste exemplo 221 inicializações.

Eficiente

```
FOR L = 1 TO 5
  FOR J = 1 TO 10
    FOR K = 1 TO 20
      A (K, J, L) = 0
    NEXT K
  NEXT J
NEXT L
```

O laço externo será inicializado uma vez, o do meio 5 vezes, e o mais interno 50 vezes. Portanto, teremos neste exemplo 56 inicializações.

(6) Declare os contadores dos laços no início do programa. O interpretador mantém uma tabela das variáveis usadas no programa. Toda vez que o interpretador encontra um contador, ele pesquisa esta tabela. Se os contadores forem definidos primeiro, eles serão localizados primeiro.

Observação:

Esta técnica não tem sentido no programa compilado. Para maiores informações, verifique o Cap. 15 – Compilação.

(7) O controle não deve ser desviado de dentro do laço para fora antes do contador atingir o limite superior especificado na instrução FOR.


```
50 FOR I = 1 TO 100
   ......
   ......
100 IF X = 0 THEN 510
   ......
   ......
500 NEXT I
510 ......
```

O interpretador mantém uma área extra de memória para todos os laços não resolvidos. Dependendo do número de vezes que ocorrer este desvio para fora, pode haver falta de memória, causando o erro "Out of memory".

Para contornar este problema, use o seguinte artifício:

```
50 FOR I = 1 TO 100
 .
 .
100 IF X = 0 THEN I = 101 : GOTO 500
 .
 .
500 NEXT I
510 .....
```

5.8 Testes

1) O que será mostrado no video?

```
10 FOR I = 1 TO 2
20    FOR J = 1 TO 4
30       X = X + I * J
40    NEXT I
50 NEXT J
60 PRINT X
```

a) 0
b) 25
c) 30
d) 40
e) nenhuma das anteriores.

2) O que será mostrado no video?

```
10 FOR I = 1 TO 5
20    FOR J = 1 TO 3
30       X = X + I * J
40    NEXT J
50 NEXT I
```

```
60 PRINT X
```

a) 80
b) 70
c) 60
d) 0
e) nenhuma das anteriores.

3) Qual será o valor de X após a execução deste programa?

```
10 X = 1
20 FOR I = 1 TO 5
30     FOR J = 1 TO 3
40         X = X + 1
50     NEXT J
60 NEXT I
```

a) 5;
b) 3;
c) 15;
d) 14;
e) nenhuma das anteriores.

4) Se executado este programa, ocorrerá erro?

```
10 FOR I = 1 TO 10
20     GOTO 500
30 NEXT I
40 END
500 PRINT "500"
510 GOTO 30
```

a) sim;
b) não.

5) O que será mostrado no video?

```
10 FOR I = 0.5 TO 6.5 STEP 1.5
20     X = X + I
30 NEXT I
40 PRINT X
```

a) 18.5
b) 0.5
c) 15.5
d) 16.5
e) nenhuma das anteriores.

6) O que será mostrado no video?

```
10 FOR J = 8 TO 1 STEP – 2.5
20     X = X + J
30 NEXT J
```

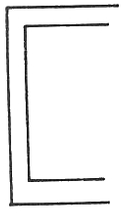


```
40 PRINT X
```

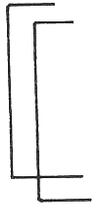
a) 15
b) 17
c) 14
d) 16
e) nenhuma das anteriores.

7) Qual dos seguintes laços aninhados não é válido?

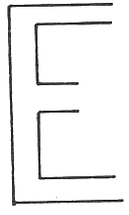
a)



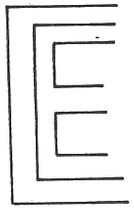
b)



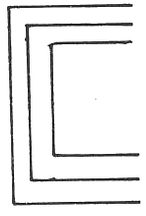
c)



d)



e)



8) Qual é o valor final de K?

```
10 K = 3
20 FOR I = 1 TO 4
30     IF K < I THEN 50
40     K = K + 1
50 NEXT I
60 END
```

a) 3;
b) 6;
c) 8;
d) 7;
e) nenhuma das antcriores.

5.9 Exercícios propostos

1) Escreva um programa que calcule o fatorial de um número.

$n! = n\,(n-1)\,(n-2)\,(n-3)\ldots(3)\,(2)\,(1)$
$0! = 1$

2) Escreva um programa que imprima a tabuada de qualquer número.

Exemplo: Tabuada do 2

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10

$2 \times 6 = 12$
$2 \times 7 = 14$
$2 \times 8 = 16$
$2 \times 9 = 18$
$2 \times 10 = 20$

3) Escreva um programa que imprima os números de FIBONACCI.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, . . .

$n_3 = n_2 + n_1 = 2$
$n_4 = n_3 + n_2 = 3$
$n_5 = n_4 + n_3 = 5$
$n_m = n_{m-1} + n_{m-2}$

4) Elabore um programa para calcular o coeficiente binomial.

Coef. binomial $= \binom{n}{r} = \frac{n!}{r!\,(n-r)!}$

6

MATRIZES

"Bons comentários não podem melhorar uma má codificação, mas maus comentários podem comprometer seriamente uma boa codificação."

Jean-Paul Tremblay
Richard B. Bunt

6.1 Introdução

Em computação, matriz é um conjunto de caracteres que são agrupados para que fiquem armazenados na memória e sejam utilizados quando necessário.

Uma matriz pode ser, por exemplo, uma lista de valores, composta por 12 resultados: 65, 63, 75, 76, 90, 83, 87, 98, 76, 75, 81 e 68.

Cada valor individual do conjunto é chamado elemento da matriz. Neste exemplo, o primeiro elemento da matriz é 65 e o último elemento é 68. Pode-se dar a esta lista de números um nome de variável, por exemplo, S. Assim, o primeiro item da lista, 65, será referenciado por S (1). Ainda usando o mesmo exemplo, qual o conteúdo de S (5)? É 90!

Ao se atribuir S para a lista, S passa a ser a matriz. Como esta matriz tem apenas um subscrito, ela é referida como sendo uma matriz unidimensional.

6.2 Dimensionando a matriz

Antes de se usar uma matriz no programa, é necessário defini-la através da instrução DIM. Esta instrução apenas reserva o espaço necessário na memória, o conteúdo da matriz deve ser fornecido posteriormente.

Exemplo:
```
10 DIM S (12)
20 FOR I = 1 TO 12
30    INPUT S (I)
40 NEXT I
```

Observação. S (13) não seria permitido.

Observe que neste programa o contador do laço FOR/NEXT foi usado com subscrito.

Depois do programa ter sido executado, a matriz está preenchida e seus elementos podem ser usados. Sendo 65 o primeiro elemento da matriz e 98 o oitavo, temos:

```
PRINT S (1)   Mostra o primeiro elemento de S
  65
PRINT S (8)   Mostra o oitavo elemento de S
  98
```

Somando-se os elementos e calculando-se a média:

```
50 SOMA = 0
60 FOR K = 1 TO 12
70    SOMA = SOMA + S (K)
80 NEXT
90 PRINT "Média e"; SOMA/12
```

Observações:

(1) A declaração DIM é opcional, se o maior subscrito for menor que 11.

(2) A declaração DIM pode ser colocada em qualquer lugar do programa, desde que venha antes da matriz a ser usada.

(3) Uma única declaração pode ser usada para dimensionar várias matrizes.

Exemplo: DIM A (30), B (40), C (50)

Dica. Coloque-as em ordem alfabética, fica mais fácil para localizá-las.

(4) Um programa pode conter várias declarações DIM, mas cada matriz deve ser dimensionada somente uma vez.

(5) O mesmo nome de variável pode ser usado no programa para uma variável subscrita ou não. Assim, S e S (17) são variáveis diferentes.

(6) O subscrito pode ser uma expressão aritmética.

Portanto, variáveis do tipo S (2 * 3 + 2) e S (A + 5) são permitidas.

Observação. Não é permitida no compilador.

(7) O subscrito não pode ser negativo.

(8) Se o valor do subscrito não for inteiro, sua parte fracionária será arredondada.

(9) Uma matriz pode conter informações alfanuméricas ou numéricas.

Exemplo:

A$ (100) é uma matriz alfanumérica.

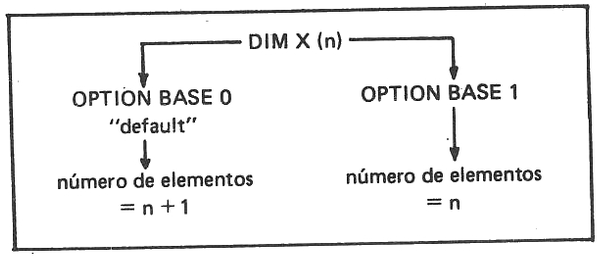
6.3 Definindo o subscrito mínimo

O subscrito 0 é permitido. Com a declaração DIM S (12), são reservadas 13 localizações na memória ou seja, para S (0), S (1), . . . , S (12); resultando em 13 elementos.

A instrução OPTION BASE n, define o subscrito mínimo.

OPTION BASE 0 Define que o subscrito mínimo é zero

OPTION BASE 1 Define que o subscrito mínimo é um



Exemplo:

```
10 OPTION BASE 1
20 DIM A (200), B (300), C (500)
```

Observações:

(1) Esta instrução deve ser executada antes de qualquer declaração DIM.

(2) Não pode ter mais de uma instrução OPTION BASE no programa.

Lembre-se. A instrução OPTION BASE deve ser usada antes da matriz ser dimensionada ou utilizada.

Dica. Use sempre a declaração OPTION BASE. Auxilia na documentação e pode otimizar o uso da memória.

6.4 Exemplos de declarações DIM

a) Matriz de uma dimensão

```
10 DIM A (15), B (X), C (D + 5)
```

O subscrito pode ser uma constante, uma variável ou uma expressão. No exemplo acima, as variáveis X e D devem ser definidas antes.

b) Matriz de duas dimensões

```
DIM A (4, 3)
```

Serão reservados 12 espaços na memória (4 * 3)

COLUNAS

LINHAS	1	2	3
1	345	687	149
2	344	689	235
3	378	499	245
4	377	568	388

c) Matriz de três dimensões

```
10 FOR K = 1 TO N3
20    FOR J = 1 TO N2
30       FOR I = 1 TO N1
40          C (I, J, K) = A (I, J, K) + C (I, J, K)
50       NEXT I
60    NEXT J
70 NEXT K
```

Observação. O número de dimensões depende do equipamento e do tamanho da memória.

Sugestão. Verifique o número máximo de dimensões em seu equipamento.

6.5 Suprimindo matrizes

Para suprimir matrizes de um programa, use a instrução ERASE (lista de matrizes).
As matrizes podem ser redimensionadas depois de suprimidas.

Exemplos:

```
.
.
.
450 ERASE A, B
460 DIM B (99)
```

Observações:

(1) Esta instrução não é permitida no compilador BASIC.

(2) Se for feita uma tentativa de redimensionar uma matriz sem antes suprimi-la, ocorrerá um erro "Duplicate Definition" (redimensionamento de matriz).

6.6 Exemplos de programas

(1) Este programa encontra o maior e o menor elemento em uma matriz.

```
80 INPUT "Entre com o núm. de elementos"; N
90 DIM A (N)
100 FOR I = 1 TO N
110    INPUT A (I)
120 NEXT I
124 '
125 ' primeiro elemento como referencial
126 '
130 MAIOR = A (1): MENOR = A (1)
140 FOR I = 2 TO N
150 IF A (I) > MAIOR THEN MAIOR = A (I) : GOTO 170
160 IF A (I) < MENOR THEN MENOR = A (I)
170 NEXT I
180 PRINT " O maior elemento e"; MAIOR
190 PRINT "O menor elemento e"; MENOR
200 END
```

(2) Este programa classifica os elementos de uma matriz.

```
10 INPUT "Entre com o núm. de elementos"; N
20 DIM MAT (N)
30 FOR I = 1 TO N
40    INPUT MAT (I)
50 NEXT I
55 CHAVE = 0
60 FOR I = 1 TO N - 1
70    IF MAT (I) < = MAT (I + 1) THEN 120
80    SWAP MAT (I), MAT (I + 1)
110   CHAVE = 1
120 NEXT I
124 '
125 '  Verificando se houve troca
126 '
130 IF CHAVE = 1 THEN 55
134 '
135 '  Imprimindo a matriz classificada
136 '
140 FOR I = 1 TO N
150   PRINT MAT (I);
160 NEXT I
170 END
```

Observação. Este algoritmo, embora seja simples, não é eficiente quando o volume de dados é muito grande. Verifique outros algoritmos mais eficientes.

6.7 Testes

1) O número total de elementos da matriz A (20, 5) com OPTION BASE 0 e OPTION BASE 1 é respectivamente:

a) 25, 27;
b) 26, 27;

c) 100, 126;
d) 126, 100;
e) 125, 126.

2) O número total de elementos da matriz X (30, 10) com OPTION BASE 0 e OPTION BASE 1 é respectivamente:

a) 300, 261;
b) 341, 300;
c) 30, 10;
d) 29, 9;
e) nenhuma das anteriores.

3) No programa abaixo

```
100 DIM A (2, 2, 2)
110 FOR K = 1 TO 2
120    FOR J = 1 TO 2
130       FOR I = 1 TO 2
140          PRINT A (I, J, K)
150       NEXT I
160    NEXT J
170 NEXT K
```

Os três últimos elementos a serem mostrados serão:

a) A (2, 1, 2), A (1, 2, 2), A (2, 2, 2);
b) A (1, 1, 2), A (1, 2, 2), A (2, 2, 2);
c) A (2, 1, 2), A (1, 1, 2), A (2, 2, 2);
d) A (1, 2, 1), A (2, 2, 1), A (2, 2, 2);
e) nenhuma das anteriores.

4) O que será mostrado no video?

```
10 DIM A$ (2, 2)
20 FOR I = 1 TO 2
30     A$ (I, 1) = "BEIRA"
40     A$ (I, 2) = "MAR"
50 NEXT I
60 PRINT A$ (1, 1); A$ (2, 2);
70 PRINT A$ (1, 2); A$ (2, 1)
80 END
```

a) BEIRAMARMARBEIRA
b) BEIRAMAR MARBEIRA
c) BEIRAMAR BEIRAMAR
d) BEIRAMARBEIRAMAR
e) nenhuma das anteriores.

5) O que será mostrado na execução deste programa?

```
10 WHILE I < 10
20     I = I + 2
30     SAL (I) = (SAL (I) + 10) * 2
40 WEND
50 ERASE SAL
60 PRINT SAL (6)
```

a) 140;
b) 60;
c) 100;
d) 1200;
e) nenhuma das anteriores.

6) Dado:

```
10 DIM A (30)
20 OPTION BASE 1
```

Ocorrerá erro?

a) sim;
b) não.

7) Dado:

```
10 A = 50
20 DIM A (A)
```

Ocorrerá erro?

a) sim;
b) não.

8) Posso colocar mais de uma declaração OPTION BASE no programa?

a) sim;
b) não.

9) Dado:

```
100 A (1) = 0
110 DIM A (200)
```

Ocorrerá erro?

a) sim;
b) não.

10) Dado:

```
10 DIM A (50)
.
.
300 ERASE A
.
.
500 A (1) = 0
```

Quantos elementos possui a matriz A?

a) 51;
b) 50;
c) 10;
d) 11;
e) nenhuma das anteriores.

11) Os dois programas abaixo têm a mesma função?

```
A) 10 FOR I = 1 TO 1000
   20     B (I) = 0
   30 NEXT I

B) 10 FOR I = 1 TO 1000 STEP 2
   20     B (I) = 0
   30     B (I + 1) = 0
   40 NEXT I
```

a) sim;
b) não.

12) Dado:

```
A) FOR J = 1 TO 5
     FOR K = 1 TO 20
       A (J, K) = 0
     NEXT K
   NEXT J

B) FOR K = 1 TO 20
     FOR J = 1 TO 5
       A (J, K) = 0
     NEXT J
   NEXT K
```

Qual dos laços é mais eficiente?

a) laço A;
b) laço B;
c) são iguais.

6.8 Exercícios propostos

1) Escreva um programa que calcule o dígito verificador do CPF

2) Escreva um programa que calcule o dígito verificador módulo 11 de um número qualquer.

Exemplo:

N.	9	4	3	4	5	7	8	4	2
PESOS	4	3	2	7	6	5	4	3	2
PROD.	36	+ 12	+ 6	+ 28	+ 30	+ 35	+ 32	+ 12	+ 4

SOMA = 195

195/11 = 17 com resto 8
DIGITO = 11 – 8 = 3

3) Elabore um programa que imprima a matriz transposta de uma dada matriz. As linhas tornam-se colunas, e vice-versa.

Exemplo:

$$\begin{bmatrix} 6 & 3 & 3 & 1 \\ 4 & 3 & 5 & 2 \end{bmatrix} \qquad \begin{bmatrix} 6 & 4 \\ 2 & 3 \\ 3 & 5 \\ 1 & 2 \end{bmatrix}$$

Matriz Introduzida — Matriz Transposta

4) Elabore um programa que imprima o resultado da adição de duas matrizes.

Exemplo:

$$\begin{bmatrix} 2 & 1 & 7 \\ 4 & -6 & 2 \\ 0 & 3 & -3 \end{bmatrix} + \begin{bmatrix} 16 & 8 & -1 \\ 4 & 6 & 9 \\ -3 & 2 & 8 \end{bmatrix} = \begin{bmatrix} 18 & 9 & 6 \\ 8 & 0 & 11 \\ -3 & 5 & 5 \end{bmatrix}$$

5) Elabore um programa que imprima o resultado da multiplicação de duas matrizes.

Exemplo:

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & -1 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 6 \\ -1 & 2 \\ 2 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1(3) + 3(-1) + 2(2) & 1(6) + 3(2) + 2(1) \\ 0(3) + (-1)(-1) + 2(2) & 0(6) + (-1)(2) + 2(1) \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 14 \\ 5 & 0 \end{bmatrix}$$

6) Elabore um programa que multiplique uma dada matriz por uma constante. (Multiplicação escalar.)

Exemplo:

$$3 \times \begin{bmatrix} 1 & 2 \\ 6 & 4 \\ 2 & 7 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 18 & 12 \\ 6 & 21 \end{bmatrix}$$

7) Elabore um programa para calcular o desvio padrão de um conjunto de dados armazenados em uma matriz unidimensional.

$$DP = \sqrt{\frac{\sum_{i=1}^{n} (xi - \bar{x})^2}{n - 1}}$$

8) Elabore um programa para calcular a matriz inversa de uma matriz qualquer.

9) Elabore uma sub-rotina que calcule o somatório dos elementos de uma matriz unidimensional.

10) Elabore um programa para imprimir os números primos.

Observação. Número primo não tem divisores, exceto 1 e ele próprio.

7

TABELAS INTERNAS

"Diagramas de blocos são geralmente um meio fraco de comunicar um projeto. Em muitos casos o programa fonte é mais preciso, mais conciso e mais conveniente."

Edward Yourdon

7.1 Introdução

Considere um problema hipotético de programação onde se deseja armazenar valores constantes em variáveis diferentes.

Exemplo:

Assume-se que em determinado lugar do programa a variável A tenha o valor esperado de 43.7, a variável B, o valor – 19.63, a variável V o valor 4, a variável V7 o valor – 9 e a variável X o valor 0.25.

```
10 A = 43.7
20 B = – 19.63
30 V = 4
40 V7 = – 9
50 X = 0.25
```

Neste exemplo, foram armazenadas apenas cinco variáveis. Quando se trabalhar com um grande número de variáveis, será muito cansativo defini-las uma a uma.

Para agilizar esse processo de definição de variáveis, existem as declarações READ e DATA.

O programa acima, por exemplo, poderia ter sido escrito em apenas duas declarações:

```
10 READ A, B, V, V7, X
20 DATA 43.7, – 19.63, 4, – 9, 0.25
```

7.2 Definindo tabelas

Observe que a declaração READ lista as variáveis a serem inicializadas e a declaração DATA lista os valores que serão usados na inicialização.

A declaração DATA pode ser colocada em qualquer parte do programa e pode conter apenas strings ou constantes numéricas, uma vez que expressões ou variáveis não são permitidas.

Pode haver várias ocorrências das declarações READ e DATA dentro de um programa.

Exemplo:
```
10 READ A, R, J3
20 DATA 4, 5
30 DATA – 3
```

Quando a declaração READ for executada, a variável A terá o valor 4, a variável R terá o valor 5 e a variável J3 terá o valor – 3.

Não é necessário limitar o programa a apenas uma declaração READ.

Exemplo:
```
10 READ A, R
20 READ J3
30 DATA 4, 5
40 DATA – 3
```

Quando o primeiro READ for executado, a variável A terá o valor 4 e a variável R o valor 5. Quando o segundo READ for executado, a variável J3 assumirá o valor – 3.

7.3 Ocorrendo falta de dados

Quando o número de variáveis do READ for maior que o número de itens relacionados nas declarações DATA, ocorrerá erro.

Exemplo:
```
10 READ A, R, J3
20 DATA 5, 15
```

Neste caso ocorrerá erro.

7.4 Colocando os dados entre aspas

Quando os itens alfanuméricos da declaração DATA contiverem:
- vírgulas
- brancos significativos
- dois pontos (:)

devem obrigatoriamente vir entre aspas.

Exemplos:

(1) brancos significativos

```
10 DATA "  PORTO ALEGRE  ","RS"
20 READ A$, B$
30 PRINT A$; B$
RUN
   PORTO ALEGRE RS
```

(2) vírgulas

```
10 DATA "XY, 30", 89
20 READ A$, B
30 PRINT A$; B
RUN
XY, 30 89
```

(3) dois pontos

```
10 DATA "RJ: BRASIL", 4567
20 READ A$, B
30 PRINT A$; B
RUN
RJ: BRASIL 4567
```


7.5 Recuperando os dados da tabela

Se for necessário reler algum comando DATA a partir de determinado ponto, deve-se fazer uso da instrução

RESTORE [linha]

Depois da execução da instrução RESTORE, a próxima instrução READ terá acesso ao primeiro item da primeira instrução DATA existente no programa. Se for especificada a linha, a próxima instrução READ terá acesso ao primeiro item da instrução DATA que se encontra na linha referida.

Exemplos:

(1)
```
100 READ X
110 RESTORE
120 READ Y
130 PRINT X; Y
140 DATA 50, 60
RUN
 50   50
```

(2)
```
10 DATA 55, 32
20 DATA 68, 97
30 DATA 22, 34
40 READ A, B, C, D
50 RESTORE 20
60 READ E, F
70 PRINT A; B; C; D; E; F
RUN
 55   32   68   97   68   97
```

Outros exemplos de READ e DATA:

(1)
```
10 READ A, B, C
20 DATA 1, 2, 3
```

Equivale a 10 A = 1 : B = 2 : C = 3

(2)
```
10 DATA 0, 1, "IRAPUA"
20 READ SOMA, CONTA, A$
```

Para cada variável existe um correspondente no DATA. Neste caso SOMA = 0, CONTA = 1 e A$ = "IRAPUA"

(3)
```
10 READ A, B
20 DATA 1, 2, 3
```

As declarações DATA podem ter mais valores do que o necessário. Os extras são ignorados. A instrução READ é que não pode ter maior número de itens que os DATA

(4)
```
10 DATA "SP", 5
20 READ C, A$
```

Neste caso ocorrerá erro, devido a incompatibilidade de tipos.

7.6 Exemplos de tabelas indexadas

(1) Tabela do número de dias de cada mês

```
 90 OPTION BASE 1
100 DIM TDM (12)
110 DATA 31, 28, 31, 30, 31, 30, 31, 31
120 DATA 30, 31, 30, 31
130 FOR I = 1 TO 12
140     READ TDM (I)
150 NEXT I
```

(2) Tabela dos meses do ano

```
 90 OPTION BASE 1
100 DIM TMES$ (12)
110 DATA JANEIRO, FEVEREIRO, MARCO,
120 DATA ABRIL, MAIO, JUNHO, JULHO,
130 DATA AGOSTO, SETEMBRO, OUTUBRO,
140 DATA NOVEMBRO, DEZEMBRO
150 FOR I = 1 TO 12
160     READ TMES$ (I)
170 NEXT I
```

(3) Tabela dos dias da semana

```
 90 OPTION BASE 1
100 DIM TDSEM$ (7)
110 DATA SEGUNDA, TERCA, QUARTA, QUINTA
120 DATA SEXTA, SABADO, DOMINGO
130 FOR I = 1 TO 7
140     READ TDSEM$ (I)
150 NEXT I
```

Dica. Em pesquisa seqüencial de tabelas, coloque os itens mais usados em primeiro lugar.

Exemplo:

Em uma tabela de siglas de Estados, coloque as siglas que têm a maior ocorrência em primeiro lugar. Deste modo a pesquisa a esta tabela pode ser mais eficiente, na medida em que os itens podem ser achados mais rapidamente.

200 DATA SP, RJ, MG, RS, PR, ... , RR

7.7 Testes

1) Dado:

```
10 READ A$, A
20 A1$ = A$
30 A1 = A
40 RESTORE
50 READ B$, B
60 B1$ = B$
70 B1 = B
80 DATA BRASIL, 82, AMERICA, 80
```

Qual o conteúdo das variáveis A1$, A1, B1$ e B1?

a) nulo, 0, nulo, 0;
b) BRASIL, 82, AMERICA, 80;
c) nulo, 80, BRASIL, 82;
d) BRASIL, 82, BRASIL, 82;
e) nenhuma das anteriores.

2) O que está armazenado no elemento X (2, 2)?

```
10 OPTION BASE 1
20 DIM X (5, 5)
30 FOR I = 5 TO 1 STEP – 1
40     FOR J = 1 TO 5
50         READ X (I, J)
60     NEXT J
70 NEXT I
80 DATA 5, 8, 9, 11, 22, 33, 15, 4, 5, 10
90 DATA 35, 99, 25, 13, 14
95 DATA 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
```

a) 15;
b) 99;
c) 16;
d) 8;
e) nenhuma das anteriores.

3) Dado o programa:

```
10 DATA 13, 20, 25
20 READ A, B, C
30 DATA 23, 5, 3
40 READ A, B, C, D
50 PRINT A; B; C; D
```

O que será mostrado no video?

a) 13 20 25 23
b) 23 5 3 0
c) 23 5 13 13
d) 25 23 5 3
e) ocorrerá erro.

4) Após a execução deste programa, qual será o valor de A (1, 3)?

```
10 DIM A (3, 3)
20 FOR I = 3 TO 1 STEP – 1
30    FOR J = 1 TO 3
40       READ A (J, I)
50    NEXT J
60 NEXT I
70 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
```

a) 9;
b) 1;
c) 4;
d) 7;
e) nenhuma das anteriores.

5) Neste programa, qual é o valor de A (7)?

```
 5 DIM A (20)
10 FOR J = 1 TO 3
20    FOR I = 1 TO 4
30       READ A (I)
40    NEXT I
50    RESTORE
60 NEXT J
70 DATA 3.08, 5.19, 3.1, 3.9, 4.24, 5.01
80 DATA 5.08, 5.55, 4.1, 3.16, 3.37, 9.92
```

a) 3.08;
b) 3.12;
c) 5.08;
d) 5.55;
e) nenhuma das anteriores.

6) O que está armazenado no elemento A (1, 3)?

```
10 OPTION BASE 1
20 DIM A (3, 3)
30 FOR I = 1 TO 3
40    FOR J = 1 TO 3
50       READ A (I, J)
60    NEXT J
70 NEXT I
80 DATA 12, 10, 8, 6, 1, 3, 5, 7
90 DATA 2, 4, 6, 11
```

a) 7
b) 8
c) 3
d) 12
e) 11

7) Dado o programa:

```
10 DATA SP RJ MG, RS, PR
20 READ S1$, S2$, S3$
```

Qual será o conteúdo de S1$, S2$ e S3$, respectivamente?

a) SP RJ MG;
b) SP RJ MG, RS, PR;
c) ocorrerá erro;
d) nenhuma das anteriores.

8) Dado o programa:

```
10 DATA SP, 2000, RJ, 1500, RS, 1000
20 READ A$, B, C$, D, E, F$
```

Qual será o conteúdo de A$, C$ e F$, respectivamente?

a) SP, RJ, 1000;
b) SP, 2000, RS;
c) nenhuma das anteriores.

7.8 Exercícios propostos

1) Elabore um programa que forneça o mês por extenso, a partir da introdução de um número (1 a 12) – (1-jan . . . 12-Dez).

Exemplo:

Entrada: 3
Saída: Março

2) Elabore um programa que forneça o dia da semana por extenso, a partir da introdução de um número (1 a 7)-(1-Seg, . . . 7-Domingo).

Exemplo:

Entrada: 2
Saída: Terça

3) Elabore um programa que verifique se uma determinada sigla de Estado é válida.

Acre – AC	Pará – PA
Alagoas – AL	Paraíba – PB
Amapá – AP	Paraná – PR
Amazonas – AM	Pernambuco – PE
Bahia – BA	Piauí – PI
Ceará – CE	Rio de Janeiro – RJ
Distrito Federal – DF	Rio Grande do Norte – RN
Espírito Santo – ES	Rio Grande do Sul – RS
Fernando de Noronha – FN	Rondônia – RO
Goiás – GO	Roraima – RR
Maranhão – MA	Santa Catarina – SC
Mato Grosso – MT	São Paulo – SP
Mato Grosso do Sul – MS	Sergipe – SE
Minas Gerais – MG	

Exemplos:

Entrada: RX
Saída: Sigla inválida

Entrada: RS
Saída: Sigla válida

4) Escreva um programa que, dado o dia e mês de nascimento, forneça o signo correspondente.

Signo	Intervalo
Carneiro	21/03 a 20/04
Leão	22/07 a 22/08
Sagitário	22/11 a 21/12
Touro	21/04 a 20/05
Virgem	23/08 a 22/09
Capricórnio	22/12 a 20/01
Gêmeos	21/05 a 20/06
Balança	23/09 a 22/10
Aquário	21/01 a 19/02
Câncer	21/06 a 21/07
Escorpião	23/10 a 21/11
Peixes	20/02 a 20/03

Exemplo:

Entrada: 01/08
Saída: Leão

5) Escreva um programa que dado o ano de nascimento, forneça o signo no horóscopo chinês.

Rato	1900, 1912, 1924, 1936, 1948, 1960, 1972
Bufalo	1901, 1913, 1925, 1937, 1949, 1961, 1973
Tigre	1902, 1914, 1926, 1938, 1950, 1962, 1974
Gato	1903, 1915, 1927, 1939, 1951, 1963, 1975
Dragão	1904, 1916, 1928, 1940, 1952, 1964, 1976
Serpente	1905, 1917, 1929, 1941, 1953, 1965, 1977
Cavalo	1906, 1918, 1930, 1942, 1954, 1966, 1978
Cabra	1907, 1919, 1931, 1943, 1955, 1967, 1979
Macaco	1908, 1920, 1932, 1944, 1956, 1968, 1980
Galo	1909, 1921, 1933, 1945, 1957, 1969, 1981
Cachorro	1910, 1922, 1934, 1946, 1958, 1970, 1982
Porco	1911, 1923, 1935, 1947, 1959, 1971, 1983

Exemplo:

Entrada: 1957
Saída: Galo

6) Elabore um programa que calcule o último dia útil de cada mês. Considere como não úteis somente sábados e domingos.

7) Elabore um programa que verifique qual é o "bicho" a partir de um número de 00 a 99.

Avestruz	1 – 4	Águia	5 – 8
Burro	9 – 12	Borboleta	13 – 16
Cachorro	17 – 20	Cabra	21 – 24
Carneiro	25 – 28	Camelo	29 – 32
Cobra	33 – 36	Coelho	37 – 40
Cavalo	41 – 44	Elefante	45 – 48
Galo	49 – 52	Gato	53 – 56
Jacaré	57 – 60	Leão	61 – 64
Macaco	65 – 68	Porco	69 – 72
Pavão	73 – 76	Peru	77 – 80
Touro	81 – 84	Tigre	85 – 88
Urso	89 – 92	Veado	93 – 96
Vaca	97 – 99		
	00		

Exemplo:

Entrada: 15
Saída: Borboleta

8

FUNÇÕES MATEMÁTICAS E DE CONVERSÃO

"Em programação sempre siga o método KISS. KISS = Keep It Simple, Stupid"

8.1 Introdução

Neste capítulo abordaremos algumas funções intrínsecas, que executam operações especiais.

O argumento de todas as funções deve estar entre parênteses e pode ser uma variável, expressão ou constante numérica.

8.2 Funções aritméticas

8.2.1 Pegando o valor absoluto

Para calcular o valor absoluto de um argumento, usamos a função ABS (X), onde X é o argumento.

Exemplos:

(1)
```
10 Y = ABS (– 45.36)
20 PRINT Y
RUN
 45.36
```

(2)
```
10 X = ABS (53.22)
20 PRINT X
RUN
 53.22
```

Lembre-se. O valor retornado depende da precisão do argumento.

Observação. ABS é abreviação de ABSolute (absoluto).

8.2.2 Calculando o logaritmo neperiano

A função LOG (X) retorna o logaritmo neperiano (base e) de X.

Exemplo:

```
10 X = LOG (5)
20 PRINT X
RUN
 1.6094
```

Para calcular o logaritmo de um número em outra base B, use a fórmula:

$LOG_B (X) = LOG_e (X)/LOG_e (B)$

Observações:

(1) LOG é abreviação de LOGarithm (logaritmo).

(2) e = 2.71828

8.2.3 Calculando o inverso do logaritmo neperiano

A função EXP (X) calcula o inverso do logaritmo neperiano de X, isto é, e ↑ X.

Exemplos:

(1)
```
10 PRINT EXP (1)
RUN
2.71828
```

(2)
```
10 Y = EXP (2)
20 PRINT Y
RUN
 7.38906
```

Observação. O argumento X deve ser menor ou igual a 87.3365

8.2.4 Verificando o sinal da variável

A função SGN (X) retorna o sinal do argumento X.

SE	RETORNA
X > 0	1
X < 0	– 1
X = 0	0

Exemplos:

(1)
```
10 PRINT SGN (– 987)
RUN
– 1
```

(2)
```
10 INPUT "Entre com um número"; A
20 ON SGN (A) + 2 GOTO 30, 40, 50
30 PRINT "O número é negativo"
35 GOTO 10
40 PRINT "O número é zero"
45 GOTO 10
50 PRINT "O número é positivo"
55 GOTO 10
```

Observação. SGN é abreviação de SiGNal (sinal).

8.2.5 Calculando a raiz quadrada

A função SQR (X) calcula a raiz quadrada de X. Equivale a X ↑ 0.5. O argumento X deve ser positivo.

Exemplos:

(1)
```
10 FOR X = 10 TO 25 STEP 5
20    PRINT X; SQR (X)
30 NEXT X
RUN
 10  3.16228
 15  3.87298
 20  4.47214
 25  5
```

(2)
```
PRINT SQR (– 25)
```

Neste caso ocorrerá erro, pois o argumento deve ser positivo.

Observação. SQR é abreviação de SQuare Root (raiz quadrada)

8.3 Funções trigonométricas

8.3.1 Calculando o seno

A função SIN (X) retorna o seno de X em radianos. Esta função é calculada em simples precisão.

Exemplo:

```
PRINT SIN (1.5)
.997495
```

Observação. SIN é abreviação de SINe (seno).

8.3.2 Calculando o co-seno

A função COS (X) retorna o co-seno de X em radianos. O cálculo é feito em simples precisão.

Exemplo:

```
10 X = 2 * COS (0.4)
20 PRINT X
RUN
 1.84212
```

Para obter o co-seno de X em graus, use:

```
COS (X * 0.0174533)
```

8.3.3 Calculando a tangente

A função TAN (X) retorna a tangente de X em radianos. O cálculo é feito em simples precisão.

Exemplo:

```
10 PRINT TAN (5)
RUN
– 3.38052
```

Para obter a tangente em graus use:

```
TAN (X * 0.0174533)
```

8.3.4 Calculando o arco tangente

A função ATN (X) retorna o arco tangente de X em radianos. O resultado estará no intervalo de $-\pi/2$ a $\pi/2$.

O argumento X pode ser qualquer tipo numérico, mas o cálculo de ATN é sempre realizado em simples precisão.

Exemplo:

```
10 PRINT ATN (3)
RUN
 1.24905
```

Observação. 1 grau = 0.0174533 radianos.

8.4 Funções de conversão

8.4.1 Convertendo decimal em hexadecimal

A função HEX$ (X) retorna o valor hexadecimal do argumento X.

Exemplos:

(1) 10 PRINT HEX$ (15)
RUN
F

(2) 10 INPUT X
20 A$ = HEX$ (X)
30 PRINT X; "em decimal e ="; A$;
35 PRINT "em hexa"
40 GOTO 10

8.4.2 Convertendo decimal em octal

A função OCT$ (X) retorna o valor octal do argumento X.

Exemplo:

PRINT OCT$ (24)
30

8.4.3 Convertendo em dupla precisão

A função CDBL (X) retorna a representação em dupla precisão de X. O valor retornado contém 16 dígitos, mas apenas aqueles contidos no argumento X serão significativos.

A função CDBL pode ser útil quando desejamos que uma função seja feita em dupla precisão.

Exemplo:

10 A = 454.67
20 PRINT A; CDBL (A)
RUN
454.67 454.6700134277344

Observação. CDBL é abreviação de Convert to DouBLe.

8.4.4 Convertendo em simples precisão

A função CSNG (X) retorna a representação em simples precisão de X. Quando o argumento X for um valor em dupla precisão, é retornado 6 dígitos significativos, com arredondamento no dígito menos significativo.

Exemplos:

(1) CSNG (.6666666666666667) = .666667
CSNG (.3333333333333333) = .333333

(2) 10 A# = 975.3421#
20 PRINT A#; CSNG (A#)
RUN
975.3421 975.342

Observação. CSNG é abreviação de Convert to SiNGle.

8.4.5 Pegando a parte inteira – CINT

A função CINT (X) retorna a parte inteira de X, arredondando a parte fracionária. O argumento X deve estar entre – 32768 e 32767.

Exemplos:

(1) PRINT CINT (1.5); CINT (– 1.5)
2 – 2

(2) PRINT CINT (45.67)
46

Lembre-se. O argumento X deve estar entre – 32768 e 32767.

Observação. CINT é abreviação de Convert to INTeger.

8.4.6 Pegando a parte inteira – INT

A função INT (X) retorna a parte inteira de X arredondando para o inteiro inferior, usando o número menor ou igual ao argumento.

O resultado é armazenado como um número de simples precisão. O argumento X não precisa estar limitado ao intervalo de – 32768 a 32767.

Exemplos:

(1) PRINT INT (2.5)
2

(2) PRINT INT (– 2.5)
– 3

(3) PRINT INT (1000101.23)
1000101

Dica. Não use expressões complexas. Separe-as em outras menores.

8.4.7 Pegando a parte inteira sem arredondar

Para obtermos a parte inteira de qualquer valor, usamos a função FIX (X).

Exemplos:

(1) PRINT FIX (2.2)
2

(2) PRINT FIX (– 2.2)
– 2

(3) PRINT FIX (58.75)
58

(4) PRINT FIX (– 58.75)
– 58

8.5 Gerando números aleatórios

A geração de números aleatórios é necessária em algumas aplicações. Por exemplo: simulações e jogos.

O BASIC possui duas instruções para realizar esta operação. São elas RANDOMIZE e RND.

8.5.1 Definindo a semente

A instrução RANDOMIZE [expressão] atribui um novo valor para o número gerador da seqüência de números aleatórios.

Se a [expressão] for omitida, a execução será interrompida e será pedido um valor através da mensagem:

Random number seed (– 32768 to 32767)?
(Número gerador entre (– 32768 e 32767)?)

Se o número gerador não for realimentado, ou seja, não lhe é atribuído um novo valor, a função RND fornece a mesma seqüência de números aleatórios cada vez que o programa é executado com o comando RUN.

Para mudar a seqüência de números aleatórios a cada execução do programa, coloque a instrução RANDOMIZE no início do programa e mude o argumento a cada RUN.

Exemplo:

```
10 RANDOMIZE
20 FOR I = 1 TO 5
30    PRINT RND;
40 NEXT I
RUN
Número gerador entre – 32768 e 32767?3
(O operador digita 3)
.88598    .484668    .586328    .119426    .709225
RUN
Número gerador entre – 32768 e 32767? 4
(O operador digita 4 para a nova seqüência)
.803506    .162462    .929364    .292443    .322921
RUN
Número gerador entre – 32768 e 32767? 3
(mesmo número que a primeira execução)
.88598    .484668    .586328    .119426    .709225
```


8.5.2 Pegando o próximo número aleatório da seqüência

A função RND [(X)] retorna um número aleatório entre 0 e 1, exclusive. Se X > 0 ou X for omitido, será gerado o próximo número aleatório. Se X = 0, será repetido o último número gerado.

Exemplo:

```
10 FOR I = 1 TO 5
20    PRINT FIX(RND * 100);
30 NEXT I
RUN
 24    30    31    51    5
```

Dica. Para gerar números aleatórios dentro de um intervalo, use o seguinte algoritmo:

VI = valor inicial do intervalo
VF = valor final do intervalo
X = VF – VI + 1
Número aleatório = FIX (X * RND) + VI

Exemplos:

(1) Intervalo de 50 a 85
W = FIX (36 * RND) + 50

(2) Intervalo de – 90 a 80
W = FIX (171 * RND) – 90

Observação. RND é abreviação de RaNDom (aleatório).

8.6 Criando funções específicas

A linguagem BASIC permite ao programador definir suas próprias funções, que podem ser tanto numéricas quanto strings.

O primeiro passo é definir a função antes que ela seja executada.

DEF FN < nome > < argumentos > = < função >

< nome > deve ser um nome válido de variável. Este nome precedido por FN torna-se o nome da função. Se a função é do tipo string, então < nome > deve ser seguido por $.

< argumentos > são as variáveis a serem substituídas quando a função for chamada.

< função > expressão a ser executada.

a) a expressão não pode ultrapassar uma linha
b) não pode conter separador de instrução " : "
c) não pode conter qualquer verbo (ex: GOTO)
d) não pode conter IF . . THEN
e) pode conter operadores lógicos (OR, AND, etc)

Exemplos:

(1)
```
10 DEF FNAB (X, Y) = (X ↑ 3)/(Y ↑ 2)
   .
   .
   .
90 PRINT FNAB (6, 3); FNAB (5, 5)
   RUN
    24     5
```

(2)
```
10 DEF FNB (X) = X + 3
20 PRINT FNB (3)
RUN
 6
```

(3)
```
10  '
20  ' Esta função transforma temperatura
30  ' de graus Fahrenheit em graus Celsius
40  '
50  DEF FNX (A) = (A – 32) * 0.5555555
60  '
70  INPUT "Entre com a temperatura em F –"; F
80  C = FNX (F)
90  PRINT F; "graus F ="; C; "graus Celsius"
```

(4) Esta função calcula a área de um círculo

```
DEF FNA (R) = R * R * 3.14159
```

(5)
```
10  '
20  ' Esta função converte data
30  ' gregoriana em juliana
35  '
40  DEF FNJU% (D%, M%, A%) = (M% – 1) * 28 + VAL (MID$ (
    "000303060811131619212426",
    (M% – 1) * 2 + 1, 2)) – ((M% > 2)
    AND ((A%ANDNOT – 4) = 0)) + D%
50  INPUT "Entre com o dia"; D%
60  INPUT "Entre com o mês"; M%
70  INPUT "Entre com o ano"; A%
80  PRINT FNJU% (D%, M%, A%)
90  GOTO 50
```

(6) O BASIC possibilita o posicionamento do cursor em qualquer ponto do video. Para tanto é necessário definir uma função para o posicionamento. No microcomputador SCOPUS a função é:

```
DEF FNCP$ (L, C) = CHR$ (27) + CHR$ (61) +
                   CHR$ (31 + L) + CHR$ (31 + C)
```

L é a linha, e C, a coluna onde se deseja colocar o cursor.

Exemplos:

(1) Para imprimir a mensagem "Código inválido" na linha 24 e coluna 10:
```
PRINT FNCP$ (24, 10) "Código inválido"
```

(2)
```
10 DEF FNCP$ (L, C) = CHR$ (27) + CHR$ (61)
          + CHR$ (31 + L) + CHR$ (31 + C)
20 INPUT "Entre com L, C"; L, C
30 INPUT "Entre com a mensagem"; M$
40 PRINT FNCP$ (L, C) M$;
50 GOTO 20
```

Execute-o!!!

Sugestão. Verifique no seu equipamento qual é a função para o posicionamento do cursor.

Observações:

(1) Estas funções devem ser definidas antes de serem utilizadas.

Sugestão. Defina-as no início do programa.

(2) Estas funções podem ser redefinidas em qualquer lugar do programa.

(3) Uma função pode referenciar uma outra.

(4) DEF FN é abreviação de DEFine FuNction.

(5) É obrigatório pelo menos um espaço entre DEF e FN.

8.7 Testes

1) Sendo X = – 45.3498

Quais são os valores respectivos de:

CINT (X), INT (X), FIX (X), CSNG (X) e ABS (X)?

a) – 45, – 45, – 45, – 45, – 45;
b) 46, – 45, – 45, – 46, 46;
c) – 46, – 46, – 46, – 45.3498, 46;
d) – 46, – 45, – 46, – 45.3498, 45.3498;
e) nenhuma das anteriores.

2) O que será mostrado no video?

PRINT FIX (– 68.75); SGN (– 68.75); INT (ABS (– 68.75))

a) – 68 – 68 68;
b) – 68 – 1 – 68;
c) 68 – 1 68;
d) – 68 – 68 – 68;
e) nenhuma das anteriores.

3) O que será mostrado no video?

```
10 PRINT FIX (– 65.32); CINT (– 65.32);
20 PRINT INT (65.32); SGN (– 65.32)
```

a) 65 −65 65 65.32
b) −65 −66 65 −1
c) −65 −66 65 0
d) −65 −65 65 −1
e) nenhuma das anteriores.

4) Quais são as funções que retornam logaritmo natural, valor em dupla precisão, número aleatório e valor absoluto?
a) ABS (X), SIN (X), LOG (X), CDBL (X);
b) LOG (X), ABS (X), RND, CDBL (X);
c) LOG (X), CDBL (X), RND, ABS (X);
d) SIN (X), ABS (X), SIN (X), RND;
e) LOG (X), CDBL (X), RND, FIX (X).

5) X = SGN (−50)

Qual é o valor de X?

a) 50;
b) −50,
c) 1;
d) 0;
e) −1.

6) Dado X = FIX (−99.89)
Y = CINT (−99.89)
Z = INT (−99.89)

Quais são os valores de X, Y, e Z, respectivamente?

a) −100, −99, −99;
b) −99, −100, −100;
c) 99, 100, 100;
d) −99, −99, −99;
e) 100, 100, 100.

7) PRINT CINT (−45000); CINT (45000)

O que será mostrado no video?

a) −45000 45000
b) 45000 45000
c) 45000 −45000
d) 0 0
e) nenhuma das anteriores.

8) 10 DEF FNX (A, B) = (A + B)/3
20 PRINT FNX (1, 2)

O que será mostrado no video?

a) 0
b) 2
c) 3

d) 1
e) nenhuma das anteriores.

9) 10 DEF FND (X) = X ↑ 2 − X ↑ 3 + 362
20 S = FND (3) + 16

Qual será o valor de S?

a) 360;
b) 362;
c) 370;
d) 344;
e) nenhuma das anteriores.

10) Dado S = FIX (INT (13.6) + 12 + ABS (7 − 16)/3)

Qual será o valor de S?

a) 26;
b) 27;
c) 28;
d) 29;
e) nenhuma das anteriores.

8.8 Exercícios propostos

1) Escreva um programa que calcule o número de dias entre duas datas.

Por exemplo: de 01/01/84 a 31/12/84 existem 366 dias

2) Escreva um programa que, dada uma data base mais n dias, calcule a data resultante.

Por exemplo: data data base = 311283
n = 3
data resultante = 030184

3) Escreva um programa que gere cinco números aleatórios entre 0 e 99, sem repetição, simulando o jogo da LOTO.

4) Escreva um programa que ache o dia da semana de uma data qualquer.

Por exemplo:

dia 29/12/83 Quinta-feira
dia 01/01/85 Quarta-feira

5) Escreva um programa que, dada uma data qualquer, indique qual é a fase da LUA. (Minguante, Cheia, Crescente e Nova.)

6) Escreva um programa que forneça o biorritmo.

7) Escreva um programa que calcule o tempo, em segundos, entre dois instantes.

8) Elabore um programa que resolva uma equação de segundo grau.

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

9) Elabore um programa que calcule a taxa interna de retorno de um investimento.

9

MANIPULAÇÃO DE STRINGS

"Nada como ser um VIP
(Very Important Programmer)."

9.1 Introdução

"String" é uma cadeia de caracteres ou seqüência de letras e/ou números.

Um string pode ter de 0 a 255 caracteres. É composto por caracteres alfanuméricos delimitados por aspas (" ").

9.2 Calculando o tamanho do string

A função LEN (string) fornece o tamanho exato do string, incluindo os espaços.

Exemplos:

(1)
```
10 A$ = "SÃO PAULO"
20 T = LEN (A$)
30 PRINT T
RUN
 9
```

(2)
```
10 B$ = "XYZ"
20 PRINT B$; LEN (B$)
RUN
XYZ 3
```

(3)
```
10 INPUT "Entre com uma palavra"; A$
20 PRINT "A palavra "; A$;" tem";
30 PRINT LEN (A$); "caracteres"
40 GOTO 10
```

Execute-o para ver o resultado!

Observação. LEN é abreviação de LENght (tamanho).

Dica. Use esta função para verificar se o tamanho de um campo está dentro dos limites estabelecidos.

9.3 Concatenando strings

Exemplo:
```
10 A$ = "BOM "
20 B$ = "DIA "
30 C$ = "COMPUTADOR"
40 D$ = A$ + B$ + C$
50 PRINT D$
RUN
BOM DIA COMPUTADOR
```

Na linha 40 foi feita uma concatenação, que nada mais é do que um agrupamento de strings.

9.4 Repetindo um caracter

Supondo que seja necessário imprimir uma linha preenchida com asteriscos, uma opção seria:

```
PRINT "**********************************************************************"
```

Dado que esta forma de preencher uma linha toda com asteriscos não é nada prática, a função STRING$ simplifica esta tarefa.

```
PRINT STRING$ (39, "*")
```

Execute-a para ver o que acontece!

Pode-se ainda usar o código ASCII do caráter "*", ao invés de usar "*". Neste caso, a função ficaria assim:

```
PRINT STRING$ (39, 42)
```

Outro exemplo:

```
10 B$ = STRING$ (30, 61)
20 PRINT "Existem 30 caracteres - - > "; B$
```

Lembre-se. O comprimento máximo de caracteres do string é 255.

Sugestão. Verifique na tabela ASCII (Apêndice B) todos os códigos com os caracteres correspondentes.

9.5 Pegando a parte esquerda do string

Existem ocasiões onde é necessário segmentar um string.
A função LEFT$ separa n posições iniciais do string.

Exemplos:

(1)
```
10 A$ = "CAROLINA"
20 PRINT LEFT$ (A$, 4)
RUN
CARO
```

(2) Queremos apenas as cinco primeiras posições do nome

```
10 INPUT "Entre com o nome"; NOME$
20 A$ = LEFT$ (NOME$, 5)
30 PRINT "Os 5 primeiros caracteres";
40 PRINT " de "; NOME$; " são: "; A$
```

Observação. LEFT = esquerda.

9.6 Pegando a parte direita do string

Para separar as últimas posições do string, usamos a função RIGHT$.

Exemplo:

```
10 A$ = "MARCIA"
20 PRINT RIGHT$ (A$, 3)
RUN
CIA
```

Observação. RIGHT = direita.

9.7 Pegando a parte central do string

Para separar as posições do meio do string, usamos a função MID$ (string, início, tamanho).

Exemplo:

```
10 A$ = "ABCDEFGHIJ"
20 PRINT MID$ (A$, 4, 3)
```

Neste exemplo serão impressos três caracteres a partir do quarto caracter. O resultado é DEF.

Se a quantidade de caracteres não for informada, serão retornados todos os caracteres a partir da posição de início.

Exemplo:

```
10 A$ = "ABCDEFGHIJ"
20 PRINT MID$ (A$, 8)
RUN
HIJ
```

Outro uso de MID$ é:

```
10 A$ = "XYZW"
20 MID$ (A$, 1, 2) = "AB"
RUN
ABZW
```

Observação. MID é abreviação de MIDdle (meio).

9.8 Verificando o código ASCII

Para se saber qual é o código correspondente a um caracter na tabela de códigos ASCII, deve-se recorrer a função ASC (string).

Exemplo:

```
PRINT ASC ("A")
```

Será impresso 65, que corresponde ao código ASCII de A.

Se o parâmetro (string) da função ASC tiver mais de um caracter, será considerado somente o primeiro caracter.

Exemplos:

(1)
```
A$ = "BRASIL"
PRINT ASC (A$)
```

Será impresso 66, que corresponde a B.

(2)
```
10  ' exercício de criptografia.
20  INPUT "Entre com um número"; N
30  INPUT "Entre com a mensagem"; A$
40  FOR K = 1 TO LEN (A$)
50     B$ = MID$ (A$, K, 1)
60     B = N + ASC (B$)
70     CODE$ = CODE$ + CHR$ (B)
80  NEXT K
90  PRINT "A mensagem codificada é"; CODE$
```

Observações:

(1) Se o string for nulo, ocorrerá erro.

Exemplo:

```
PRINT ASC (" ")
```

Neste caso ocorrerá erro.

(2) ASC é abreviação de ASCII.

9.9 Transformando o código ASCII em carácter

Dado um código ASCII para se obter o caracter, usa-se a função CHR$ (código).

Exemplo:
```
PRINT CHR$ (65)
A
```

Observando a tabela ASCII, notamos que os códigos de 0 a 31 têm funções especiais. Os códigos de 32 a 126 representam caracteres. O código 127 tem também função especial.

Exemplos: PRINT CHR$ (7) aciona a campainha do computador.
PRINT CHR$ (8) funciona como Backspace.
PRINT CHR$ (12) limpa o video.

Dicas:

(1) Use sempre mnemônicos para os caracteres de controle, pois há muita variação de uma máquina para outra. Coloque-os no início do programa.

Exemplo:

```
10 BIP$ = CHR$ (70          ' campainha
20 BCKSPC$ = CHR$ (8)       ' backspace
30 CLS$ = CHR$ (12)         'limpa o vídeo
40 ENTER$ = CHR$ (13)       ' enter ou < CR >
 .
 .
500 IF X > 5 THEN PRINT BIP$
 .
 .
800 IF OP$ = "S" THEN PRINT CLS$
```

Nestes casos fica mais fácil para efetuar qualquer alteração, pois é necessário fazê-la somente uma vez no início do programa.

(2) Acione a "campainha" quando desejar advertir o operador. Use PRINT CHR$ (7) para apenas um sinal, ou PRINT STRING$ (n, 7) para n sinais.

9.10 Convertendo strings

Em algumas ocasiões é necessário converter um string para numérico, ou vice-versa.

9.10.1 De numérico para string

Para converter uma variável ou constante numérica para string, usamos a função STR$ (número).

Exemplo:
```
10 A = 58.5: B = – 58.5
20 PRINT STR$ (A); STR$ (B)
30 PRINT STR$ (A + B)
40 PRINT STR$ (A) +STR$ (B)
RUN
```

```
58.5 – 58.5
0
58.5 – 58.5
```

Observação. A função STR$ deixa um espaço à esquerda devido ao sinal do número.

9.10.2 De string para numérico

A função VAL (string) faz a operação contrária, isto é, retorna o valor numérico dos caracteres do string.

Exemplo:
```
10 A$ = "12"
20 B$ = "34"
30 PRINT VAL (A$ + "." + B$)
RUN
 12.34
```

Se o primeiro caracter do argumento não for +, –, & ou um dígito numérico, a função VAL retorna o valor zero.

Exemplo:
```
10 PRINT VAL ("100 cruzeiros")
20 PRINT VAL ("vale 100 cruzeiros")
30 PRINT VAL ("345 nome 538")
40 PRINT VAL ("vale 100   200")
RUN
 100
 0
 345
 0
```

Observação. A função VAL ignora espaços.
Exemplo: VAL (" – 3") retorna – 3.

9.11 Pesquisando a ocorrência de Y$ em X$

A função INSTR (I, X$, Y$), procura a primeira ocorrência do string Y$ dentro do string X$. Se for colocado I, a pesquisa começa a partir da posição I, dentro de X$. A variável I deve estar entre 0 e 255.

Se o string Y$ for encontrado, será retornado a posição onde ele inicia. Caso não seja encontrado, será retornado 0. A função sempre retorna a localização do string encontrado a partir do primeiro caracter.

Exemplos:

(1)
```
10 X$ = "SsNn"
20 Y$ = "N"
30 PRINT INSTR (X$, Y$)
RUN
 3
```


(2)
```
10 A$ = "ABCDEF"
20 PRINT INSTR (3, A$, "F")
RUN
 6
```

(3)
```
10 INPUT "Entre com sua opção"; OP$
20 X = INSTR ("IEAF", OP$) + 1
30 ON X GOTO 40, 50, 60, 70
40 PRINT "Opção inválida": GOTO 10
```

Dica. Use esta instrução para verificar as opções fornecidas em menus.

Por exemplo:

```
ON INSTR ("SNF", OP$) GOTO 150, 200, 300
```

9.12 Preenchendo um string com brancos

A função SPACE$ (X) preenche com X brancos um string. Tem a mesma função de STRING$ (X, 32), onde 32 é o código ASCII de branco.

Exemplo:
```
10 FOR I = 1 TO 5
20     X$ = SPACE$ (I)
30     PRINT X$; I
40 NEXT I
RUN
 1
  2
   3
    4
```

Observação. O argumento da instrução deve estar entre 0 e 255.

9.13 Testes

1) Sendo 65 o código ASCII de "A".

```
10 A$ = CHR$ (ASC ("A"))
20 B = ASC (CHR$ (32))
```

Quais os valores de A$ e B, respectivamente?

a) 32, "A"
b) "A", 32
c) 32, 32
d) "A", "A"

2) O que será mostrado?

```
10 A$ = "SsNnFf"
20 B$ = "J"
```

```
30 X = INSTR (A$, B$)
40 PRINT X
```

a) 1
b) 0
c) 6
d) "J"
e) nenhuma das anteriores.

3) O que será mostrado?

```
10 C$ = "ABCDEFG"
20 PRINT LEFT$ (C$, 3)
30 END
```

a) CDE
b) EFG
c) ABC
d) ABCDEFG
e) nenhuma das anteriores.

4) Qual das seguintes instruções imprime 51?

a) PRINT VAL ("5, 1")
b) PRINT VAL ("51, uma boa idéia")
c) PRINT VAL ("uma boa idéia, 51")
d) PRINT VAL ("5100, uma boa idéia")
e) nenhuma das anteriores.

5) Dado A$ = "12345678", Y$ = MID$ (A$, 3, 2) e X$ = RIGHT$ (A$, 2)

Quais são os valores de Y$ e X$, respectivamente?

a) 23,12
b) 34,12
c) 23,78
d) 34,78
e) nenhuma das anteriores.

6) 10 A$ = "252"
20 PRINT VAL (A$) + 48

O que será mostrado?

a) 252
b) 0
c) 300
d) 204
e) nenhuma das anteriores.

7) 10 X = VAL ("100A")
20 Y = VAL ("A100")
30 Z = VAL ("A100A")
40 W = VAL ("100A100")

Quais serão os valores de X, Y, Z e W?

a) 100, 0, 100, 0
b) 0, 100, 100, 0
c) 100, 0, 0, 100
d) 100, 100, 0, 100
e) nenhuma das anteriores.

8) 10 PRINT MID$ (RIGHT$ ("CAROLINA", 4), 3, 2)

O que será mostrado?

a) NA
b) CA
c) RO
d) LI
e) nenhuma das anteriores.

9) Se CEP$ = "CEP: 02134", então VAL (CEP$) é:

a) 02134
b) CEP02134
c) 0
d) 1
e) 2134

10) Tendo N$ = "ONE FOR ALL", após a instrução
MID$ (N$, 9) = "ONE"

Qual é o novo conteúdo de N$?

a) N$ = "ONE FOR ALL";
b) N$ = "ONE ONE ALL";
c) N$ = "ALL FOR ALL";
d) N$ = "ALL ALL ALL";
e) N$ = "ONE FOR ONE".

11) Dado:

```
10 X$ = "ABCDEB"
20 Y$ = "B"
30 Z = INSTR (X$, Y$)
40 W = INSTR (4, X$, Y$)
```

Quais são os valores de Z e W, respectivamente?

a) 2,6; 2, 6;
b) B, B;
c) 2, B;
d) B, 2;
e) nenhuma das anteriores.

12) O que será mostrado?

```
10 A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 B$ = MID$ (A$, 4, 5)
```

```
30 C$ = LEFT$ (B$, 2)
40 PRINT RIGHT$ (C$, 1)
```

a) D
b) E
c) F
d) G
e) nenhuma das anteriores.

13) O que será mostrado?

```
10 A$ = "123456789"
20 B$ = MID$ (A$, 4, 4)
30 C = VAL (B$)
40 PRINT C
```

a) 3456
b) 5678
c) 567
d) 4567
e) nenhuma das anteriores.

9.14 Exercícios propostos

1) Elabore um programa que verifique se todos os caracteres de um string são alfabéticos (A – Z). O espaço é considerado alfabético.

2) Elabore um programa que verifique se todos os caracteres de um string são numéricos (0 – 9). Os caracteres "+" e "–" somente são válidos se estiverem na primeira posição.

3) Elabore um programa que transforme um número em notação americana, a qual utiliza ponto decimal no lugar de vírgula, e vice-versa, para a nossa notação.

4) Elabore um programa que receba um texto (telegrama) e calcule o custo do mesmo. Cada caracter do texto custa CR$ 5.00. (Considerar brancos.)

5) Elabore um programa que receba um texto (telegrama) e calcule o custo do mesmo. Os 15 primeiros caracteres custam CR$ 5.00 cada e os caracteres restantes CR$ 3.00 cada um. (Considerar brancos.)

6) Elabore um programa que imprima de trás para frente uma palavra introduzida pelo teclado.

7) Elabore um programa que transforme uma palavra com letras minúsculas em letras maiúsculas.

8) Elabore um programa que imprima no video e na impressora a tabela ASCII. (Somente os códigos de 32 a 126.)

9) Elabore um programa que imprima cheques de até 999.999.999, 99.

Exemplos:

1130.00 Um mil cento e trinta cruzeiros
100.00 Cem cruzeiros
0.32 Trinta e dois centavos

Este programa poderá ser usado para emitir cheques, duplicatas, faturas etc.

10) Elabore um programa que identifique se é letra ou número um caracter entrado via teclado.

11) Elabore um programa que dado DDMMAA, imprima DD/MM/AA.

12) Elabore um programa que imprima em letras "garrafais" uma palavra de até 10 caracteres.

Exemplo:

```
FFFFFFFF   GGGGGG    VV      VV
FFFFFFFF   GGGGGGGG  VV      VV
FF         GG        VV      VV
FF         GG        VV      VV
FFFFFF     GG        VV      VV
FFFFFF     GG  GGGG  VV      VV
FF         GG  GGGG  VV      VV
FF         GG    GG   VV    VV
FF         GGGGGGGG     VVVV
FF           GGGGGG       VV
```

13) Elabore um programa que transforme "ddmmaa" em "dd mmm aa".

Exemplo: 130183 em 13 Jan 83.

14) Elabore um programa que converta ddmmaa em sss, dd mmm.

Exemplo: 271283 em Ter, 27 Dez.

15) Elabore um programa para consistir datas. Jan, Mar, Maio, Jul, Ago, Out e Dez possuem 31 dias
Abr, Jun, Set e Nov possuem 30 dias
Quando o ano for bissexto, o mês de fevereiro terá 29 dias, caso contrário terá 28 dias.

16) Elabore um programa que codifique/decodifique uma mensagem em código MORSE.

17) Elabore um programa que transforme um número arábico em número romano.
(M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, I = 1)

Símbolo romano	Símbolo arábico
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

18) Escreva um programa que imprima somente o prenome de um nome qualquer.

Exemplo: José da Silva → José

19) Escreva um programa que imprima alinhando a direita os nomes introduzidos via teclado.

Exemplo:
MARCELO
CLAUDIA
RENATA
ANTONIO
EDUARDO
JOSE
FABIO

20) Escreva um programa que imprima somente as iniciais de um nome.

Exemplo: Entrando "Rubens Prates" será impresso RP.

21) Escreva um programa para converter um número binário em decimal.

22) Escreva um programa para converter um número decimal em binário.

23) Elabore um programa para transformar número romano em número arábico.

24) Elabore um programa que elimine os brancos em excesso dentro de um string.

25) Elabore um programa que verifique se um string é palíndromo. Um palíndromo é um string cuja leitura tem o mesmo sentido se feita da esquerda para a direita ou vice-versa. Por exemplo, 'OSSO' e 'OVO' são exemplos de palíndromos.

10

MANIPULAÇÃO DE ARQUIVOS SEQÜÊNCIAIS

"Na programação, quase sempre a melhor solução é a mais simples."

10.1 Introdução

Ao trabalhar com arquivos em BASIC, o programador conta com duas opções para o armazenamento de dados:

- modo seqüencial
- modo randômico

O modo randômico será visto no Cap. 11.

Cabe a ele analisar qual o tipo de arquivo mais adequado para o trabalho que estiver realizando.

10.2 Definindo arquivo seqüencial

A principal vantagem que o programador obtém com arquivos seqüenciais é uma maior facilidade de criação e acesso ao arquivo.

No entanto, o armazenamento dos dados é feito de forma seqüencial (item após item), de acordo com a ordem de entrada. Este tipo de armazenamento limita bastante a flexibilidade e a velocidade de acesso aos dados, uma vez que a leitura é também de forma seqüencial, item após item.

10.3 Abrindo o arquivo

Antes de usar um arquivo no programa é necessário abri-lo. A abertura não é nada mais do que deixar o arquivo disponível para o programa. O CP/M obtém informações sobre o arquivo, se existe e onde se encontra. É reservada uma área da memória para o "buffer".

A instrução OPEN executa a abertura do arquivo.

OPEN "modo", [#] < numarq >, "nomearq"

"modo"	
"O"	modo "(O) utput" o arquivo será criado
"I"	modo "(I) nput" o arquivo será lido
#	delimitador sintático opcional
< numarq >	número do arquivo (de 1 a 15)
"nomearq"	nome pelo qual o arquivo é conhecido

Se o arquivo vai ser criado, isto é, serão gravadas informações, deverá ser aberto como arquivo de saída "(O) utput".

```
OPEN "O", # 1, "SAIDA . DAT"
```

declara que o arquivo será de saída — nome do arquivo — extensão ao nome do arquivo

Se o arquivo já existe, isto é, as informações gravadas serão utilizadas no programa, então deverá ser aberto como arquivo de entrada "(I) nput".

```
OPEN "I", 1, "CADASTRO . DAT"
```

declara que o arquivo será de entrada

Observações:

(1) Caso use simultaneamente mais de três arquivos no programa, é necessário especificar o número de arquivos quando chamar o interpretador. (maximo = 15)

Exemplo: > MBASIC /F : 13

(2) Não se pode usar um arquivo seqüencial como entrada e saída ao mesmo tempo.

(3) O número do arquivo deve ser uma expressão inteira no intervalo de 1 a 15.

10.4 Encerrando o acesso ao arquivo

Depois que o arquivo foi usado, é necessário fechá-lo. A instrução CLOSE encerra todas as operações com o arquivo, isto é, qualquer operação pendente com o arquivo é completada.

Exemplos:

(1) CLOSE#1 — fecha o arquivo #1

(2) CLOSE — fecha todos os arquivos abertos

(3) CLOSE#1, #2 — fecha os arquivos #1 e #2

As instruções END, RESET, SYSTEM e RUN sem a opção R também fecham todos os arquivos.

10.5 Lendo um arquivo seqüencial

10.5.1 Lendo um registro

A instrução INPUT# lê os dados de um arquivo seqüencial em disquete e os atribui as variáveis relacionadas.

INPUT# < numarq >, < lista de variaveis >

< numarq > número pelo qual o arquivo foi aberto.

O arquivo deve ser aberto com a opção "(I)nput, isto é, entrada.

```
10 OPEN "I", #1, "DADOS . DAT"
20 INPUT#1, A$, B$, C
 .
 .
 .
```

Os dados no arquivo devem estar no mesmo formato que seriam digitados em resposta a uma declaração INPUT.

Com valores numéricos, os espaços à esquerda, Carriage Return e Line Feed, são ignorados. O primeiro caracter encontrado que não for espaço, Carriage Return ou Line Feed, será o início do número. O número termina com um espaço, Carriage Return, Line Feed ou vírgula.

10.5.2 Lendo uma linha inteira

A instrução LINE INPUT# lê uma linha inteira (até 254 caracteres), sem delimitadores, de um arquivo seqüencial em disquete, para uma variável string.

LINE INPUT# < numarq >, < variavel string >

< numarq > número pelo qual o arquivo foi aberto
< variável string > variável a qual a linha será atribuída

A instrução LINE INPUT# lê seqüencialmente todos os caracteres até encontrar um delimitador Carriage Return ou Line Feed.

Esta instrução é útil quando um programa BASIC, gravado no formato ASCII, estiver sendo lido como dados por outro programa.

Exemplo: 160 LINE INPUT#1, E$

10.5.3 Verificando o fim do arquivo

A função EOF (numarq) verifica se o próximo registro a ser lido contém a marca de fim de arquivo.

A marca de fim de arquivo é colocada pelo sistema operacional após o último registro.

A função EOF (numarq) deve ser usada antes da leitura, pois EOF evita o erro "Input past end", que ocorre quando se tenta ler após o fim do arquivo.

Exemplo:

```
10 DIM M (100)
20 OPEN "I", # 1, "DADOS"
```

```
30 C = 0
40 IF EOF (1) THEN GOTO 100
50 INPUT# 1, M (C)
60 C = C + 1 : GOTO 40
   .
   .
   .
   .
```

10.6 Criando um arquivo seqüencial

Existem três opções para gravação de registros. São elas:

10.6.1 Usando a declaração PRINT#

PRINT# < numarq >, < lista de variaveis >

Com esta instrução a gravação no arquivo será exatamente como mostrado no video.

Exemplo:

```
PRINT#1, A; B; C
```

O inconveniente neste tipo de gravação é o fato das variáveis não estarem delimitadas.

Quando esta instrução for usada, deve-se providenciar a separação das variáveis de tal forma que elas sejam lidas corretamente no arquivo.

Exemplo:

```
PRINT#1, A; B; C; D; E; F
```

Dado A$ = "BRASIL" e B$ = "5354". A instrução PRINT#1, A$, B$ grava no arquivo : BRASIL5354, pois não há nenhum delimitador. Para resolver este problema, devem-se introduzir delimitadores explícitos na instrução PRINT#, da seguinte maneira:

```
PRINT#1, A$; " , "; B$
```

A imagem gravada no arquivo é BRASIL, 5354, que pode ser lida novamente a partir do arquivo em A$ e B$.

Se os strings contém vírgulas, pontos e vírgulas ou brancos significativos, a gravação deve ser feita entre aspas, através de CHR$ (34).

Exemplo:

A$ = "Rua PIO XI, 1553" e B$ = "01313"

A instrução PRINT#1, A$, B$ grava no arquivo Rua PIO XI, 1553 01313.

E a instrução: INPUT#1, A$, B$ inicializa A$ com "Rua PIO XI" e B$ com "1553 01313"

Para separar estas cadeias adequadamente, basta gravar a imagem dos dados no arquivo entre aspas, usando CHR$ (34).

```
10 D$ = CHR$ (34)
20 PRINT#, D$; A$; D$; D$; B$; D$
```


grava no arquivo "Rua PIO XI, 1553" " 01313" e a instrução INPUT#1, A$, B$ inicializa A$ com "Rua PIO XI, 1553" e B$ com "01313".

10.6.2 Usando a declaração WRITE#

WRITE# < numarq >, < lista de variáveis >

A diferença entre WRITE# e PRINT# é que WRITE# introduz vírgulas entre itens quando eles são gravados no disquete e delimita os strings com aspas. Portanto, não é necessário colocar delimitadores específicos na lista de expressões.

Exemplos:

(1) A$ = "JOSE DA SILVA" e B$ = "2993544"

A instrução WRITE#1, A$, B$ grava a seguinte imagem no arquivo: "JOSE DA SILVA", "2993544"

A instrução INPUT# subseqüente, tal como: INPUT#1, A$, B$ inicializa A$ com "JOSE DA SILVA" e B$ com "2993544"

(2) X$ = "SAO PAULO" e Y = 16
WRITE#1, X$, Y irá gravar "SAO PAULO", 16 pois as variáveis numéricas não são colocadas entre aspas.

> Dica. Sempre que possível use WRITE# no lugar de PRINT#, pois WRITE# coloca os delimitadores, evitando assim que o programador tenha que defini-los.

10.6.3 Usando a declaração PRINT#nUSING

A declaração PRINT#nUSING "formato", < lista de variáveis > grava os dados no arquivo, usando um formato específico.

Por exemplo, a declaração:

```
PRINT#1 USING "####.##,"; A, B, C, D
```

Pode ser usada para gravar dados numéricos no arquivo sem delimitadores explícitos. Para separar os dados no arquivo em disquete, use uma vírgula no final do formato.

10.7 Como saber o número de setores lidos ou gravados no arquivo

A função LOC (numarq) retorna o número de setores (blocos de 128 bytes) lidos ou gravados no arquivo, desde sua abertura.

Exemplo:

```
200 IF LOC (1) 50 THEN GOTO 1000
```

Observação. Em arquivos randômicos têm função diferente.

10.8 Dicas

(1) Como acrescentar dados ao final de um arquivo seqüencial.

Suponhamos que este arquivo chama-se "MESTRE".

a) Abrir "MESTRE" no modo "I"
b) Abrir um segundo arquivo chamado "COPIA" no modo "O"
c) Ler os dados, em "MESTRE" e gravá-los em "COPIA"
d) Fechar o arquivo "MESTRE" e eliminá-lo.
e) Gravar as novas informações em "CÓPIA".
f) Renomear o arquivo "COPIA" como "MESTRE".

Observação:

Quando um arquivo é aberto como "O", se o arquivo não existe, será criado. Caso exista, o conteúdo anterior será perdido, pois será gravada uma marca de fim de arquivo no primeiro registro (EOF).

(2) Abra o arquivo apenas quando for utilizá-lo. Feche-o logo após sua utilização. No caso de queda de energia ou remoção acidental do disquete, se o arquivo estiver aberto ele poderá ser perdido.

(3) Não remova um disquete que tenha um arquivo aberto.

(4) Qualquer programa de atualização de arquivo deve prever, além da listagem de ocorrências, a emissão de totais de controle, tais como:
- número de registros lidos no arquivo mestre
- número de registros lidos no arquivo movimento
- número de registros incluídos
- número de registros excluídos
- número de registros alterados
- número de registros inválidos

(5) Os nomes de arquivos e drives podem ser variáveis.

Exemplo:

```
490 INPUT "Entre com o drive"; DR$
500 INPUT "Entre com o nome do arq."; NA$
510 OPEN "I", #1, DR$ + ":" + NA$
```

(6) As vezes é interessante fundir duas ou mais chaves em uma única para efeitos de classificação. Isso pode ser feito colocando os campos chaves contínuos no lay-out do registro.

(7) Como criar um arquivo "vazio". Para tanto, use somente o modo imediato.

```
OPEN "O", #1, "ARQUIVO.DAT"
CLOSE
```

Isto faz com que o arquivo "ARQUIVO . DAT" seja criado e tenha uma marca de fim de arquivo no primeiro registro.

(8) Sempre que possível, coloque em drives diferentes os programas e arquivos. Deixe em um drive os programas e arquivos "Tabelas", isto é, arquivos que não sofrem atualização constante. Esta técnica facilita a tarefa de tirar backups, mas, por outro lado, gera uma alta improdutividade na atuação dos drives. Enquanto o de programas fica praticamente inativo, o de arquivos fica sobrecarregado.

(9) Quando existirem muitos cálculos e pouca impressão, crie um arquivo com a saída impressa (use extensão . PRN) e em seguida liste-o. Desta maneira a impressora não precisa ficar alocada durante todo o processamento.

(10) Coloque etiquetas em código nos disquetes que contém informações confidenciais.

Por exemplo: Não escreva no disquete "VENDAS 1983", em vez disso, chame-o de "ABC".

(11) Na atualização de arquivos seqüenciais, crie um arquivo com extensão . $$$ (temporário) para o resultado da atualização, e somente no fim do processamento renomeie-o, caso não tenha ocorrido nenhum erro.

(12) Padronize as extensões nos nomes de arquivos

Exemplos:

. ASC para programas gravados em ASCII
. BAK para backups
. BAS para programas gravados no formato padrão BASIC
. PRO para programas gravados no formato protegido
. REL para programas em formato relocável
. TXT para arquivos textos
. $$$ para arquivos temporários

(13) Sempre verifique se um arquivo definido como classificado está realmente classificado. Em algumas ocasiões a lógica do programa está baseada na certeza que o arquivo está classificado, mas, por algum erro anterior na execução do programa, o arquivo pode não estar classificado. Elabore uma rotina inicial que verifique se os registros estão na seqüência correta.

(14) Cuidado com arquivos vazios (sem nenhuma informação). Coloque opção no programa para testar tal ocorrência. O operador pode demorar para descobrir que o arquivo está vazio.

(13) Quando o arquivo é pequeno, é melhor fazer a atualização na memória. Coloque todo o arquivo na memória e faça todas as atualizações. Somente quando for fornecida a opção para finalização é que o arquivo atualizado será gravado de volta.

10.9 Testes

1) Dado A$ = "SAO PAULO" e B = 456

O que estará gravado no disquete após a execução da instrução WRITE#1, A$, B?

a) SAO PAULO, 456
b) SAO PAULO 456

c) "SAO PAULO", "456"
d) "SAO PAULO", 456
e) nenhuma das anteriores.

2) Dado A$ = "GOIAS" e B$ = "BRASILIA".

O que estará gravado no disquete após a execução da instrução WRITE#1, A$, B$?

a) "GOIAS", "BRASILIA"
b) GOIASBRASILIA
c) GOIAS, BRASILIA
d) GOIAS , BRASILIA
e) nenhuma das anteriores.

3) Onde a instrução EOF (n) deve ser colocada?

a) antes do INPUT #;
b) depois do INPUT #;
c) antes do WRITE#;
d) depois do WRITE#;
e) nenhuma das anteriores.

4) Qual é a alternativa que mostra os valores mínimo e máximo para n, sendo que n está contido na instrução OPEN "modo", [#] n, "arquivo"?

a) 0,255;
b) 0,15;
c) 1,16;
d) 1,15;
e) nenhuma das anteriores.

5) A função LOC (n) na manipulação de arquivos seqüenciais retorna:

a) número de arquivos usados no programa;
b) número de bytes do registro;
c) número de registros lidos ou gravados;
d) número de setores lidos ou gravados;
e) nenhuma das anteriores.

6) Qual é a alternativa errada? (*Observação*: Considere um programa interpretado)

a) A instrução CLOSE sem nenhum parâmetro fecha todos os arquivos;
b) A instrução END fecha todos os arquivos e retorna o controle ao CP/M;
c) A instrução END fecha todos os arquivos e continua dentro do BASIC;
d) A instrução LOC (n) retorna o número de setores lidos ou gravados.

7) Qual é a associação errada?

a) OPEN "I" com INPUT #;
b) OPEN "O" com WRITE#;
c) OPEN "O" com PRINT#;
d) OPEN "O" com EOF (n);
e) OPEN "O" com PRINT#USING.

8) O que acontece com o conteúdo de um arquivo já existente, aberto com OPEN "O"?

a) o conteúdo permanece inalterado;
b) o conteúdo é perdido.

9) Pode-se usar simultaneamente um arquivo seqüencial como entrada e saída?

a) sim;
b) não.

10) Qual o "default" para o número de arquivos que se pode usar concorrentemente em um programa BASIC?

a) 15;
b) 255;
c) 32767;
d) 8;
e) nenhuma das anteriores.

10.10 Exercícios propostos

1) Elabore um programa para verificar se dois arquivos são iguais.

2) Elabore um programa para verificar se um arquivo está classificado.

3) Elabore um programa que armazene e recupere todos os feriados anuais.

11

MANIPULAÇÃO DE ARQUIVOS RANDÔMICOS

"Um vaso de flores quebrado é freqüentemente mais fácil de ser substituído do que reparado."

Henry F. Ledgard

11.1 Introdução

Para se criar e acessar arquivos randômicos é necessário executar mais passos de programa do que quando se trabalha com arquivos seqüenciais.

No entanto, a utilização de arquivos randômicos é muito vantajosa. As principais vantagens são:

- Um arquivo gravado randomicamente utiliza menor espaço no disquete, pois o armazenamento é feito em formato binário compactado.
- Os dados armazenados podem ser acessados aleatoriamente não sendo necessária a leitura de todas as informações anteriores, a exemplo do que é feito em arquivos seqüenciais.
- É eficiente quando o volume de dados é pequeno.

Sua única desvantagem é a difícil recomposição no caso de perda de arquivo.

11.2 Abrindo o arquivo

Antes de começar a ler ou gravar no arquivo randômico, é necessário abrí-lo.

OPEN "R", [#] < numarq >, "nomearq", [tamreg]

[#] delimitador sintático opcional

< numarq > número pelo qual o arquivo será referenciado, devendo estar no intervalo de 1 a 15.

"nomearq" nome pelo qual o arquivo é conhecido

[tamreg] tamanho do registro. Pode ter até 255 bytes, sendo que o "default" = 128.

Exemplo:

```
650 OPEN "R", #3, "CLIENTES.DAT", 160
```


Os arquivos randômicos podem ser usados simultaneamente como entrada e saída, ao contrário de arquivos seqüenciais que somente podem ser usados de um modo (entrada ou saída) num dado momento.

Observações:

(1) O tamanho do registro não pode ultrapassar o tamanho definido na opção/S:< tamreg > do MBASIC. Esta estabelece o tamanho máximo do registro que poderá ser utilizado. O "default" é 128.

Exemplo: MBASIC /S:225

(2) Caso use simultaneamente mais de três arquivos no programa, é necessário especificar o número de arquivos quando chamar o interpretador.

Exemplo: MBASIC /F:13 (maximo = 15)

> Dica. Coloque sempre o tamanho do registro, evitando que o "default" seja assumido.

11.3 Definindo o "lay-out" e o tamanho do "buffer"

Após ter aberto o arquivo randômico, é necessário definir o lay-out do registro. Para isso usa-se a instrução FIELD, que tem a seguinte sintaxe:

FIELD [#] < numarq >, < tamanho-campo > AS < var$ >, ...

[#] delimitador sintático opcional

< numarq > número pelo qual o arquivo foi aberto

A instrução FIELD, além de definir o lay-out do registro, aloca espaço para o "buffer", que irá conter o que será gravado ou o que foi lido do arquivo.

Exemplo:

```
FIELD#1, 30 AS NOME$, 20 AS ENDERECO$, 5 AS CEP$
```

Neste exemplo o arquivo #1, tem como lay-out de registro:

NOME – 30 posições
ENDERECO – 20 posições
CEP – 5 posições

Observações:

(1) O número total de bytes reservados na declaração FIELD não deve ultrapassar o comprimento do registro definido na instrução OPEN.

(2) As variáveis relacionadas na instrução FIELD devem ser necessariamente string.

(3) Podem-se definir vários FIELDs para um mesmo arquivo.

Exemplos:

(a)
```
FIELD#1, 30 AS COD$, 20 AS CGC$
FIELD#1, 15 AS CID$, 35 AS NOME$
```

(*b*) FOR I = 1 TO 8
FIELD#1, (I – 1) * 8 AS F$, 8 AS A$ (I)
NEXT I

Neste caso está sendo definido um "buffer" com 8 subcampos, que devem estar dimensionados no DIM.

(4) Uma variável para a qual está alocado um espaço no "buffer" não deve ser usada em uma declaração INPUT ou LET.

11.4 Etapas na gravação de um registro

Para gravar um registro é preciso primeiramente, transformar todas as variáveis numéricas em strings, pois, quando se trabalha com arquivos randômicos, todas as variáveis da instrução FIELD devem, necessariamente, ser strings.

11.4.1 Convertendo em strings

Esistem três funções que realizam esta conversão:

MKI$ – (Make Integer) transforma um valor inteiro em um string de 2 bytes. Se o valor não estiver no intervalo de – 32768 a 32767, ocorrerá erro.

Exemplo: A$ = MKI$ (K%)

MKS$ – (Make Single) transforma um valor simples precisão em um string de 4 bytes.

Exemplo: B$ = MKS$ (J!)

MKD$ – (Make Double) transforma um valor dupla precisão em um string de 8 bytes.

Exemplo: C$ = MKD$ (L#)

11.4.2 Alinhando a esquerda ou a direita

Depois que todas as variáveis numéricas forem convertidas em strings é preciso movê-las para o "buffer", definido na instrução FIELD, através das seguintes instruções:

LSET < varstr > = < expressão string >

Move o dado para o "buffer" definido pela declaração FIELD, alinhando-o a esquerda (Left SET).

Exemplos:

```
LSET A$ = N$
LSET B$ = MKD$ (B#)
LSET Z$ = MKI$ (K%)
```

RSET < varstr > = < expressão string >

Move o dado para o "buffer", alinhando-o a direita (Right SET).

Exemplo: RSET X$ = MKI$ (K%)

Observações:

(1) Se a < expressão string > for maior que varstr (variável string), os caracteres da direita da < expressão string > serão ignorados.

(2) LSET e RSET também podem ser usados com uma variável não relacionada na declaração FIELD, a fim de alinhar um literal pela esquerda ou pela direita em dado campo.

Exemplo:

```
10 A$ = "BRASIL"
20 B$ = SPACE$ (10)
30 RSET B$ = A$
40 PRINT B$
RUN
          BRASIL
```

11.4.3 Gravando o registro

Após a conversão em string e a colocação no "buffer", podemos então gravar o registro no arquivo em disquete.

PUT [#] < numarq > [, numreg]

[#] delimitador sintático opcional

< numarq > número assinalado ao arquivo quando foi aberto.

[numreg] número do registro dentro do arquivo. Se for omitido o número do registro, será assumido o número do PUT anterior + 1.

Exemplo: 100 PUT#2, REG%

Observações:

(1) PUT diz onde será gravado o registro, e não o que será gravado.

(2) O número máximo de registros permitido é 32767.

11.5 Etapas na leitura de um registro

11.5.1 Lendo o registro

A instrução GET lê o registro no arquivo em disquete e coloca-o no "buffer" especificado.

GET [#] < numarq > [, numreg]

[#] delimitador sintático opcional

< numarq > número assinalado ao arquivo quando foi aberto.

[numreg] número do registro dentro do arquivo.
Se for omitido, será lido o próximo registro após o último GET.

Exemplo: 100 GET#2, REG%

Observações:

(1) A instrução GET apenas coloca o conteúdo do registro no "buffer".

(2) O número máximo de registros permitido é 32767.

11.5.2 Convertendo em numéricos

Os valores numéricos que foram convertidos em strings durante a gravação devem ser convertidos novamente em numéricos, usando-se as funções de conversão.

CVI converte um string em um valor inteiro
CVS converte um string em um valor de simples precisão
CVD converte um string em um valor de dupla precisão

Exemplos:

```
K% = CVI (A$)
J! = CVS (B$)
L# = CVD (C$)
```

Pode-se, então, usar o conteúdo das variáveis K%, J! e L# no programa.

11.6 Encerrando o acesso ao arquivo

Para encerrar as operações de entrada e saída, usamos a instrução CLOSE. Qualquer operação pendente com o arquivo é completada.

Exemplos:

(1) CLOSE fecha todos os arquivos abertos

(2) CLOSE #1, #2 fecha os arquivos #1 e #2

(3) CLOSE #3 fecha o arquivo #3

As instruções END, RESET, SYSTEM e RUN sem a opção R também fecham todos os arquivos.

11.7 Verificando o próximo registro

A função LOC(n) retorna o número do próximo registro a ser usado após um GET ou PUT sem o número de registro.

Observação. n é o número do arquivo especificado na instrução OPEN.

Exemplo:

```
IF LOC (1) > 70 THEN END
```

11.8 Criando um arquivo randômico

É necessário executar os seguintes passos, para se criar um arquivo randômico.

1) Abra o arquivo no modo "R".

```
OPEN "R", #1, "ARQ1", 32
```

Neste exemplo, o registro tem 32 bytes.

2) Use FIELD para alocar espaço no "buffer" para as variáveis que serão gravadas no arquivo randômico.

```
FIELD #1, 20 AS N$, 4 AS A$, 8 AS SP$
```

3) Use LSET para colocar os dados no "buffer". Os valores numéricos devem ser transformados em strings quando colocados no "buffer".

```
LSET N$  = X$
LSET A$  = MKS$ (Q)
LSET SP$ = TEL$
```

4) Use PUT para gravar os dados do "buffer" no arquivo em disquete.

```
PUT #1, CODIGO%
```

11.9 Acessando um arquivo randômico

É necessário executar os seguintes passos para acessar um arquivo randômico.

1) Abra o arquivo no modo "R".

```
OPEN "R", #1, "ARQ1", 32
```

2) Use FIELD para alocar espaço no "buffer" para as variáveis que serão lidas a partir do arquivo.

```
FIELD #1, 20 AS N$, 4 AS A$, 8 AS SP$
```

3) Use GET para mover o registro desejado para o "buffer"

```
GET#1, CODIGO%
```

4) Os dados do "buffer" podem agora ser acessados pelo programa. Os valores numéricos (convertidos em strings na gravação) devem ser convertidos em numéricos, usando-se as funções de conversão.

```
PRINT N$
PRINT CVS (A$)
```

11.10 Testes

1) Qual é a alternativa que contém instrução(ões) que convertem um string em valor numérico, quando usamos arquivos randômicos?

a) MKI$, MKS$, MKD$;
b) CVI, CVS, CVD;
c) VAL;
d) STR$;
e) nenhuma das anteriores.

2) Os dados antes de serem movimentados para variáveis relacionadas na declaração FIELD devem ser convertidos para:

a) valores numéricos;
b) valores alfanuméricos;

c) valores inteiros;
d) valores constantes;
e) nenhuma das anteriores.

3) Qual é a alternativa errada?

a) LSET A# = N$;
b) LSET B$ = MKD$ (B#);
c) LSET C$ = D$;
d) LSET Z$ = MKS$ (Z!);
e) LSET W$ = MKI$ (W%).

4) Qual é a instrução FIELD correta?

a) FIELD #16,4 AS A$;
b) FIELD #17,5 AS B;
c) FIELD #3,15 AS X$;
d) FIELD #3,X$ AS 15.

5) Quantos bytes ocupam as variáveis inteiras, simples precisão e dupla precisão, respectivamente:

a) 2, 4, 6;
b) 1, 2, 4;
c) 4, 6, 8;
d) 2, 4, 8;
e) nenhuma das anteriores.

6) Qual é o maior número de registros permitido em um arquivo randômico?

a) 32768;
b) 32766;
c) 32767;
d) 32769;
e) nenhuma das anteriores.

7) A função LOC (n), quando usada em arquivos randômicos, retorna:

a) número de bytes lidos ou gravados;
b) número de setores lidos ou gravados;
c) número de setores do arquivo;
d) número do próximo registro;
e) nenhuma das anteriores.

8) (1) É permitido usar em uma declaração INPUT ou LET, as variáveis definidas na declaração FIELD.
(2) É permitido definir vários FIELDs para um mesmo arquivo

Qual é a alternativa correta?

a) (1) falsa, (2) verdadeira;
b) (1) verdadeira, (2) falsa;
c) (1) falsa, (2) falsa;
d) (1) verdadeira, (2) verdadeira.

9) Qual é o formato correto da instrução OPEN para arquivos randômicos?

a) OPEN "R", "nomearq", < numarq >, [tamreg];
b) OPEN "R", < numarq >, [tamreg], "nomearq";
c) OPEN "I", < numarq >, "nomearq", [tamarq];
d) OPEN "R", < numarq >, < nomereg >, [tamreg];
e) nenhuma das anteriores.

10) (1) O número total de bytes reservados em uma declaração FIELD pode ultrapassar o comprimento do registro, definido na instrução OPEN.
(2) As variáveis no FIELD devem, necessariamente, ser alfanuméricas.

Qual é a alternativa correta?

a) (1) falsa, (2) verdadeira;
b) (1) verdadeira, (2) falsa;
c) (1) falsa, (2) falsa;
d) (1) verdadeira, (2) verdadeira.

11) As instruções LSET e RSET somente podem ser utilizadas para alinhar valores nas variáveis do FIELD.

Esta afirmação é falsa ou verdadeira?

a) falsa;
b) verdadeira.

12) Se for omitido o número do registro na instrução PUT#, reg., que acontecerá?

a) ocorrerá erro;
b) será assumido o número do último registro acessado;
c) será assumido o próximo registro;
d) nenhuma das anteriores.

11.11 Exercícios propostos

1) Elabore um programa que administre um arquivo de ORTNs.
O programa poderá incluir uma ORTN ou pesquisar a ORTN de determinado mês e ano.

2) Elabore um programa que controle o consumo de álcool/gasolina de seu carro.

3) Elabore um programa que faça sua declaração de imposto de renda.

12

ENCADEAMENTO DE PROGRAMAS

"Programas longos são como um prato de espaguete; puxa aqui e alguma coisa se move do outro lado."

Dennie Van Tassel

12.1 Introdução

A linguagem BASIC permite que programas possam ser encadeados sem a intervenção do operador.

Desta forma podemos desenvolver programas usando a técnica modular, que consiste em projetar programas como um conjunto de pequenas unidades inter-relacionadas que podem ser agrupadas para formar um programa completo.

O livro "Programação Modular" de Jeff Maynard é uma ótima referência para quem se interessar em desenvolver programas modulares, que apresentam vantagens sobre os métodos monolíticos tradicionais.

As razões para encadear programas são muitas.

Por exemplo:

1) O programa é muito grande e não cabe na memória.
2) O programa é muito complexo, sendo necessário segmentá-lo.
3) Evitar a ação do operador entre a execução de um programa e outro.
4) Usar sub-rotinas externas ao programa.

Vantagens quando se definem programas pequenos:

1) As unidades de trabalho menores são mais fáceis de gerenciar, pois são apenas uma pequena parte de um programa grande e complexo.
2) Dois programadores trabalham mais rápido que um só.

Desvantagens quando se definem programas longos:

1) É difícil determinar o quanto falta para terminar o programa.
2) O teste completo de programas longos é virtualmente impossível, dado o grande número de caminhos lógicos.

3) A continuação do programa por outra pessoa é muito difícil.
4) Os testes de programas longos são demorados.

12.2 Chamando o programa seguinte

Para fazer encadeamento de programas, usa-se o comando CHAIN, que tem a seguinte sintaxe:

CHAIN [MERGE] "programa" [, início]

A função do comando CHAIN é chamar um programa, e passar variáveis do programa corrente para o programa chamado.

[início] linha inicial a ser executada no programa chamado. Se esta informação for omitida, a execução começará na primeira linha do programa. Este número de linha não é afetado pelo comando RENUM.

Exemplo: CHAIN "PROG1", 5000

12.2.1 Passando algumas informações

COMMON Var1, Var2...

A finalidade desta declaração é passar as variáveis para um programa encadeado.

A declaração COMMON é usada juntamente com CHAIN. As declarações COMMON podem aparecer em qualquer lugar do programa, sendo, entretanto, recomendável que apareçam no início. A mesma variável não pode aparecer em mais de uma declaração COMMON.

As variáveis matriciais são especificadas anexando-se "()" ao nome da variável. A ordem das variáveis no COMMON não é significativa.

Exemplo:
```
100 COMMON A, B, C, D (), G$
110 CHAIN "PROG3", 10
```

Neste caso, serão passadas para o programa PROG3 as variáveis A, B, C, G$ e a matriz D.

Dicas:

(1) Coloque uma mensagem para informar ao operador, durante o carregamento do próximo programa, pois o operador poderá tomar alguma medida inadequada, julgando que o equipamento parou, pois, dependendo do tamanho do programa a ser carregado, esta operação poderá demorar algum tempo.

(2) Coloque as variáveis do COMMON em ordem alfabética. Ficará mais fácil localizá-las.

Exemplo: COMMON A( ), B( ), C( ), X, Y, Y

(3) Somente a opção COMMON é permitida no compilador (versão 5.3). As opções ALL, MERGE e DELETE não são permitidas.

12.2.2 Passando todas as informações

Para passar todas as informações de um programa para outro, use a opção ALL da declaração CHAIN.

CHAIN "programa", início, ALL

Neste caso não poderá ser usada a declaração COMMON.

Exemplos:

(1) CHAIN "PROG3.BAS", 1000, ALL

Serão passadas todas as informações para o programa "PROG3.BAS" e o mesmo será executado a partir da linha 1000.

(2) CHAIN "PROG10.BAS",, ALL

Serão passadas todas as informações para o programa "PROG10.BAS" e o mesmo será executado a partir da primeira linha.

12.2.3 Sobrepondo rotinas ("overlay")

É permitido que uma sub-rotina seja sobreposta no programa BASIC. Para fazer isto, use a instrução CHAIN com a opção MERGE.

CHAIN MERGE "programa", início

Exemplo:

CHAIN MERGE "ROT2.BAS", 1000

Depois da execução da rotina chamada, é comum suprimi-la para que uma nova rotina possa ser incorporada de maneira semelhante.

CHAIN MERGE "programa", início, DELETE I1-I2

Após a execução da rotina chamada, serão eliminadas as linhas compreendidas no intervalo de I1-I2.

Observações:

(1) Se a opção MERGE for omitida, a instrução CHAIN não preservará o tipo das variáveis ou funções definidas pelo usuário.

(2) Para a rotina ser sobreposta, é necessário que tenha sido gravada no formato ASCII.

(3) As linhas no intervalo da opção DELETE são afetadas pelo comando RENUM.

Dica. Faça com que as variáveis da rotina chamada não coincidam com as variáveis do programa principal.

Exemplo: Na rotina para transformação de data gregoriana em juliana, comece todas as variáveis da rotina com JUL.

12.3 Passando informações por meio de arquivos

Podemos também encadear programas usando as instruções RUN e LOAD.

A instrução RUN "programa", R executa o programa especificado mantendo os arquivos abertos.

Desta forma, é possível passar informações de um programa para outro em um sistema em que os programas estão encadeados.

Exemplo:

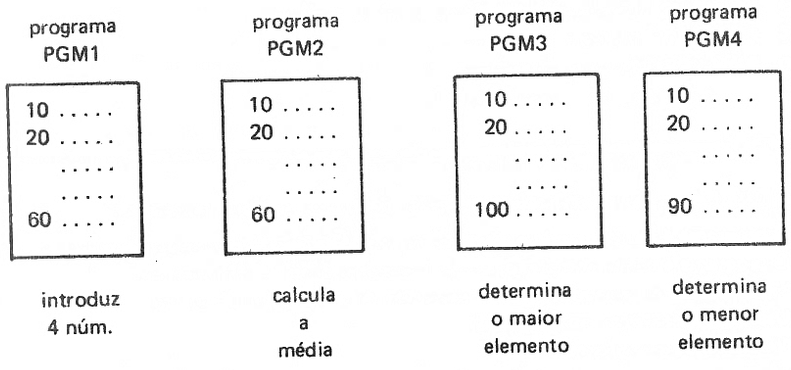
```
500 IF OPCAO = 1 THEN RUN "INCLUSAO.BAS", R
```

A instrução LOAD "programa", R tem a mesma função de RUN "programa", R.

Exemplo:

```
400 IF OPCAO = 1 THEN LOAD "INCLUSAO.BAS", R
```

12.4 Exemplo de um sistema encadeado




```
10 '    Programa : PROG1
20 '    Introduz 4 números
30 '
40 COMMON A, B, C, D
50 INPUT "Entre com 4 números"; A, B, C, D
60 CHAIN "PROG2"
```

```
10 '    Programa : PROG2
20 '    Calcula a média de A, B, C e D
30 '
40 COMMON A, B, C, D
50 PRINT "A média de.; A; B; C; D; "e = ";
55 PRINT (A + B + C + D)/4
60 CHAIN "PROG3"
```

```
10 '    Programa : PROG3
20 '    Determina o maior elemento
30 '
40 COMMON A, B, C, D
50 MAIOR = A
60 IF B > MAIOR THEN MAIOR = B
70 IF C > MAIOR THEN MAIOR = C
```

```
80 IF D > MAIOR THEN MAIOR = D
90 PRINT "O maior elemento de"; A; B; C; D;
95 PRINT "e"; MAIOR
100 CHAIN "PROG4"
```

```
10 '    Programa : PROG4
20 '    Determina o menor elemento
30 '
40 MENOR = A
50 IF B < MENOR THEN MENOR = B
60 IF C < MENOR THEN MENOR = C
70 IF D < MENOR THEN MENOR = D
80 PRINT "O menor elemento de"; A; B; C; D;
85 PRINT "e"; MENOR
90 RUN "PROG1"
```

12.5 Testes

1) Qual das alternativas não contém uma razão válida para encadear programas?

   a) O programa é muito grande e não cabe na memória, sendo necessário segmentá-lo.
   b) O programa é muito complexo, sendo necessário dividi-lo em módulos.
   c) Evitar a ação do operador, entre a execução de um programa e outro.
   d) Usar rotinas externas ao programa.
   e) Nenhuma das anteriores.

2) Qual alternativa é inválida?

   a) COMMON X, Y, W (), Z$
   b) COMMON X, Y, W (5), Z

3) "Uma mesma variável pode aparecer em mais de um COMMON no mesmo programa."

   Esta afirmação é falsa ou verdadeira?

   a) falsa;
   b) verdadeira.

4) Na instrução CHAIN "programa", início, se omitida o início, qual será a linha assumida?

   a) linha 10;
   b) linha 0;
   c) a primeira linha do programa;
   d) a última linha do programa;
   e) nenhuma das anteriores.

5) O número da linha inicial a ser executada no programa que será chamado é afetado (alterado) pelo comando RENUM?

   a) sim;
   b) não.

6) "Para que uma rotina possa ser sobreposta com a opção MERGE é necessário que tenha sido gravada no formato ASCII."

   Esta afirmação é verdadeira ou falsa?

   a) verdadeira;
   b) falsa.

7) Qual das opções da instrução CHAIN é permitida no compilador?

   a) MERGE;
   b) ALL;
   c) DELETE;
   d) COMMON.

12.6 Exercícios propostos

1) Desenvolver um sistema encadeado composto dos seguintes programas.

   Programa 1 – Entra com uma matriz A de n elementos.
   Programa 2 – Classifica a matriz A em ordem crescente e imprime.
   Programa 3 – Classifica a matriz A em ordem decrescente e imprime.
   Programa 4 – Calcula a média aritmética.

2) Desenvolver um sistema estatístico onde os programas estão encadeados.

   Programa 1 – Entra com uma matriz A de n elementos.
   Programa 2 – Calcula o desvio padrão.
   Programa 3 – Calcula a mediana.

13

ACESSANDO A MEMÓRIA

"Nunca sacrificar clareza por eficiência; nunca sacrificar clareza pela oportunidade de revelar sua inteligência."

Jean-Paul Tremblay
Richard B. Bunt

13.1 Introdução

Várias declarações permitem acessar diretamente o conteúdo da memória através de um programa BASIC.

13.2 Lendo a memória

A função PEEK retorna o conteúdo de um endereço da memória.

X = PEEK (endereço)

O valor de X estará no intervalo de 0 a 255, o endereço no intervalo de 0 a 65535, ou seja, 64K.

Exemplo: 10 VALOR = PEEK (27323)

Dica. Como obter a data e as horas do sistema operacional, no microcomputador SCOPUS.

```
DIA$ = CHR$ (PEEK (64977)) + CHR$ (PEEK (64978))
MES$ = CHR$ (PEEK (64980)) + CHR$ (PEEK (64981)
       CHR$ (PEEK (64982))
ANO$ = CHR$ (PEEK (64984)) + CHR$ (PEEK (64985))
HORA$ = CHR$ (PEEK (64988)) + CHR$ (PEEK (64989))
MIN$ = CHR$ (PEEK (64991)) + CHR$ (PEEK (64992))
SEG$ = CHR$ (PEEK (64994)) + CHR$ (PEEK (64995))
```

Observação. PEEK, em inglês, significa espiar.

13.3 Escrevendo na memória

Se você deseja colocar um determinado valor em um endereço de memória, então use a declaração POKE.

POKE < endereço >, < valor >

Será colocado o < valor > no < endereço > especificado.
O valor deve estar no intervalo de 0 a 255, e o endereço, no intervalo de 0 a 65535.

Exemplo: 1020 POKE 26328, 132

Antes de usar esta instrução, verifique no manual do microcomputador os endereços reservados pelo sistema operacional ou pelo interpretador BASIC.

Dica. Os endereços de memória mudam de máquina para máquina. As instruções PEEK e POKE devem ser documentadas para facilitar futuras conversões para outras máquinas.

13.4 Estabelecendo o topo da memória para o BASIC

Quando se usam sub-rotinas em linguagem Assembler dentro de um programa BASIC, é conveniente colocá-las no topo da memória para que não haja invasão por parte do interpretador BASIC.

Existem duas maneiras de bloquear o acesso do interpretador às posições mais altas da memória.

(*a*) Usando a opção / M: < endereço > quando chamamos o interpretador. Sendo que o endereço especificado seria o limite para o interpretador. Se omitido, será usada toda a memória pelo interpretador.

Exemplo: MBASIC / M:32768

Serão usados os primeiros 32K de memória.

(*b*) Usando a declaração CLEAR [, < expr1 >]. Esta instrução atribui valor zero à todas as variáveis numéricas e nulo para as variáveis strings e opcionalmente indica o topo da memória para o interpretador.

< expr1 > indica a mais alta posição da memória disponível para o uso do interpretador.

Exemplo: 100 CLEAR, 32000

Em algumas versões a declaração CLEAR reserva espaço de memória para strings. No caso do MBASIC, não é necessário reservar espaço para strings, pois estes são alocados dinamicamente. Não havendo memória disponível, ocorre erro "Out of string space".

Lembre-se. A instrução CLEAR limpa todas as variáveis.

13.5 Como saber o endereço das variáveis

A função VARPTR (variável) retorna o endereço do primeiro byte da variável. Qualquer tipo de variável pode ser usado, e o endereço retornado estará no intervalo de – 32768 a 32767. Se o endereço retornado for negativo, some 65536 ao endereço retornado para obter o endereço correto.

Exemplo: E = VARPTR (X#)

Esta função é normalmente usada para passar o endereço de uma variável a uma rotina em Assembler.

Para localizar o endereço de uma matriz é necessário localizar o endereço do primeiro elemento e, a partir dele, os endereços dos demais.

Exemplo: ED = VARPTR (X (0))

Observação. Os endereços de matrizes mudam constantemente, a cada definição de uma nova variável.

Para localizar o endereço do "buffer" de um arquivo, use VARPTR (#n), onde #n é o número do arquivo.

Exemplo: X = VARPTR (#1)

13.6 Verificando o número de bytes livres

A função FRE (X$) retorna o número de bytes livres, isto é, os bytes que não estão sendo usados pelo interpretador ou pelo programa que está na memória.

Exemplo: PRINT FRE (A$)

Observações:

(1) O argumento desta função tem apenas efeito sintático.

(2) Quando o interpretador é carregado na memória, é mostrada a quantidade de memória disponível para o programa BASIC.

(3) Se o argumento for nulo (FRE (" ")), será feita uma reestruturação do programa com a finalidade de otimizar o espaço ocupado pelo programa, e em seguida será mostrado o número de bytes livres.
Esta reestruturação pode ser demorada. O BASIC não inicia esta operação enquanto existir memória livre. No entanto, o uso de FRE (" "), periodicamente, resultará em atrasos menores.

13.7 Testes

1) Qual é o formato correto da instrução POKE?

a) POKE < valor >, < endereço >
b) POKE < endereço >, < valor >
c) X = POKE (endereço)

2) Qual é o formato correto da função PEEK?

a) PEEK < endereço >, < valor >
b) PEEK < valor >, < endereço >
c) X = PEEK (endereço)

3) Quais são os limites para endereço e valor nas instruções PEEK e POKE, respectivamente?

a) – 32768 a 32767, 0 a 256;
b) – 32768 a 32767, 0 a 255;
c) 0 a 65535, 0 a 255;
d) 0 a 65535, 0 a 256;
e) nenhuma das anteriores.

13.8 Exercícios propostos

1) Verifique o mapa da memória do seu microcomputador.

2) Verifique os endereços onde o sistema operacional coloca as horas e a data.

14

TRATAMENTO DE ERROS

"Se alguma coisa pode dar errado, dará errado."

Murphy

14.1 Introdução

É muito difícil que um programa ao ser concluído não contenha erros. Erros num programa, muitas vezes, somente são identificados quando o mesmo está sendo executado.

Ainda assim, pode acontecer de um programa funcionar por um bom período de tempo sem que haja falhas e em dado momento uma condição não prevista inicialmente pode provocar a sua interrupção.

A depender do tempo em que o programa foi desenvolvido, pode ser difícil ao programador localizar o problema, pois:

- provavelmente ele não se lembre mais da lógica completa do programa.
- o programa não previa o fornecimento de informações que possibilitassem ao programador a imediata localização da falha.

Para minimizar este problema, a linguagem BASIC dispõe do recurso para que o erro seja "tratado", sem interrupção da execução do programa. Quando o sistema se deparar com um erro, o fluxo do programa é desviado para uma rotina que verifica o erro ocorrido e aponta a sua localização.

Há alguns tipos de erro que podem ser facilmente corrigidos nesta rotina dando ao programa a possibilidade de continuar a execução a partir de determinada linha por ele especificada.

Outros erros que não possam ser facilmente corrigidos e recuperados recebem um tratamento especial, podendo ser emitidas mensagens explicativas, facilitando a rápida localização do erro.

14.2 Acionando a rotina de tratamento de erros

Para contar com este recurso, o programador deve colocar a instrução:

ON ERROR GOTO < linha >

Onde < linha > é a primeira linha da rotina de tratamento de erros.

Quando ocorrer um erro em qualquer lugar do programa, o fluxo do programa é desviado para a rotina de tratamento de erros, que foi especificada na instrução ON ERROR GOTO < linha >.

Exemplo:

```
10 ON ERROR GOTO 5000
.
.
.
4960 END
4970 '
4980 ' rotina de tratamento de erros
4990 '
5000 . . . . . .
```

Antes da rotina para tratamento de erros ser elaborada é necessário que o programador analise cada linha do programa, tentando prever os erros que podem ocorrer.

Para citar apenas um exemplo, quando o programa efetuar uma divisão por zero, ocorrerá um erro 11. Prevendo esta situação, o programador deverá colocar na rotina de tratamento de erros um procedimento a ser executado pelo programa sempre que houver um erro deste tipo.

Caso ocorra algum erro dentro da rotina de tratamento de erros, o programa será interrompido e emitirá mensagem de erro no video. Em outras palavras, este recurso não pode ser aplicado dentro da própria rotina de tratamento de erros.

14.3 Localizando o código do erro e a linha onde ocorreu o erro

A variável reservada ERR (ERRor) informa o código do erro.

A variável reservada ERL (ERror Line) informa a linha onde ocorreu o erro. Consulte o Apêndice D para verificar os possíveis códigos de erros.

Exemplo:

```
520 KILL "ARQORTN.DAT"
```

Se o arquivo "ARQORTN.DAT" não constar no disquete, aparecerá no video a mensagem "File not found". As variáveis ERR e ERL terão os valores 53 e 520, respectivamente.

14.4 Saindo da rotina de tratamento de erros

Para sair da rotina de tratamento de erros, deve-se usar a instrução RESUME, que conta com as seguintes opções:

Opção	Retorna
RESUME	Para a instrução onde ocorreu o erro
RESUME 0	Para a instrução onde ocorreu o erro
RESUME NEXT	Para a instrução seguinte a do erro
RESUME [linha]	Para a linha especificada

Exemplo:

```
10 ON ERROR GOTO 5000
.
.
500 KILL "B:ORTN.DAT"
.
.
4960 END
4970 '
4980 ' rotina de tratamento de erros
4990 '
5000 IF ERL = 500 AND ERR = 53 THEN RESUME NEXT
```

14.5 Desativando a rotina de tratamento de erros

Caso o programador queira desativar a rotina de tratamento de erros em alguma parte do programa, basta colocar a instrução ON ERROR GOTO 0. A partir do momento em que esta instrução for executada, a rotina de tratamento será desativada.

Após isto, caso ocorra algum erro, o programa emitirá mensagem de erro e será interrompido.

Exemplo:

```
300 ON ERROR GOTO 5000
310 OPEN "I", #1, "CADASTRO.DAT"
320 ON ERROR GOTO 0
.
.
4960 END
4970 '
4980 ' rotina de tratamento de erros
4990 '
5000 PRINT "Arquivo CADASTRO.DAT";
5010 PRINT " não foi encontrado."
5020 END
```

14.6 Criando seus próprios códigos de erro

O programador pode também criar seus próprios códigos de erro. A instrução ERROR < código > permite simular a ocorrência de um erro.

Exemplo:

```
110 ON ERROR GOTO 1000
120 INPUT "Qual a sua aposta"; B
130 IF B > 5000 THEN ERROR 210
```



```
960 END
965 '
970 ' rotina de tratamento de erros
980 '
1000 IF ERR = 210 THEN PRINT "Limite é Cr$ 5000"
1010 RESUME 120
```

Observações:

(1) O valor do código deve estar entre 0 e 255.

(2) Um código de erro criado pelo programador deve ter valor maior que aqueles usados nos códigos de erro do MBASIC (vide Apêndice D).
Se o código do erro já existe no BASIC, a declaração ERROR irá simular a ocorrência deste erro, e a mensagem correspondente será mostrada.

(3) A execução de uma declaração ERROR para qual não há nenhuma sub-rotina de tratamento de erros provoca a emissão de uma mensagem de erro e a interrupção da execução do programa.

14.7 Exemplos

(1)
```
10 ON ERROR GOTO 1000
   .........
   .........
100 OPEN "I", #1, "B:CADASTRO.DAT"
   .........
   .........
960 END
970 '
980 ' rotina de tratamento de erros
990 '
1000 IF ERL = 100 AND ERR = 53 THEN 1010 ELSE 1050
1010 PRINT "Arquivo B:CADASTRO.DAT";
1015 PRINT "não encontrado"
1020 LPRINT "Arquivo B: CADASTRO.DAT";
1025 LPRINT "não encontrado"
1040 GOTO 1065
1050 PRINT "Erro = "; ERR; "na linha"; ERL
1060 LPRINT "Erro = "; ERR; "na linha"; ERL
1065 PRINT "Programa cancelado"
1067 LPRINT "Programa cancelado"
1070 CLOSE: END
```

(2)
```
10 ON ERROR GOTO 1000
   .........
   .........
 90 NAME "B:SAIDA.DAT" AS "B:SAIDA.DAT"
100 OPEN "O", #1, "B:SAIDA.DAT"
   .........
   .........
```

```
960 END
970 '
980 ' rotina de tratamento de erros
990 '
1000 IF ERL = 90 AND ERR = 58 THEN1010 ELSE1060
1010 PRINT "O conteúdo anterior de";
1012 PRINT "B:SAIDA.DAT";
1015 PRINT "será perdido"
1020 INPUT "Confirma (S/N)"; OP$
1030 IF OP$ = "S" THEN RESUME 100
1040 IF OP$ = "N" THEN 1080
1050 GOTO 1020
1060 PRINT "Erro"; ERR; "na linha"; ERL
1070 LPRINT "Erro"; ERR; "na linha"; ERL
1080 PRINT "Programa cancelado"
1085 LPRINT "Programa cancelado"
1090 CLOSE : END
```

(3)

```
10 ON ERROR GOTO 1000
   .........
   .........
500 KILL "INDEX1.NDX"
   .........
   .........
960 END
970 '
980 ' rotina de tratamento de erros
990 '
1000 IF ERL = 500 AND ERR = 53 THEN RESUME NEXT
```

14.8 Dicas

(1) Alguns erros comuns:

Divisão por zero (código = 11)
Arquivo não é encontrado no disquete (código = 53)
Arquivo já existe no disquete (código = 58)
Disquete cheio (código = 61)
Índice de matriz fora do intervalo (código = 9)

(2) Use NAME "antigo" AS "antigo" para verificar se o arquivo existe ou não. Neste caso sempre ocorrerá erro. Se o arquivo existe ocorrerá erro 58, caso contrário ocorrerá erro 53. Esta técnica deve ser usada em conjunto com a instrução ON ERROR GOTO < linha >.

(3) Sempre verifique a validade dos dados de entrada. Não assuma que o usuário irá introduzir sempre dados corretos.

(4) Na ocorrência de erros, emitir mensagem clara, inteligível, que possua suficiente informação e permita a identificação da falha ou acidente que provocou o erro.

A mensagem deve deixar claro, também, o ponto no programa, onde o erro foi detectado. Imprima o código do erro e a linha de ocorrência, na impressora e no video. Isto elimina a necessidade de se anotar o que está no video.

(5) Sempre que houver alteração no programa é recomendável fazer o teste do programa inteiro.

(6) Na fase de testes do programa calcule manualmente os resultados que deverão ser fornecidos pelo computador antes da execução do programa. Há uma predisposição dos programadores de considerarem o resultado do computador como válidos.

(7) Quando o programa estiver pronto, não abandone imediatamente o método antigo de execução do trabalho. É recomendável fazer execução paralela dos dois sistemas por um tempo, para maior segurança.

(8) Na fase de testes do programa, escolha dados que acionem todas as linhas do fluxo do programa.

(9) Conserve as listagens das versões parciais do programa durante a fase de desenvolvimento do programa; jogue-as fora somente após o término do projeto.

(10) A depuração aleatória de programas deve ser evitada. Os testes devem ser feitos com base em reflexão, e não em intuição. Evite a "achologia".

(11) Mensagens de erro que obrigam o usuário a consultar seguidamente o manual para decifrar os problemas não são boas mensagens de erro.

(12) Coloque as mensagens do programa em uma tabela.

Exemplo:

```
100 '
110 '  Tabela de mensagens
120 '
130 ME$(1) = "CODIGO INVALIDO"
140 ME$(2) = "ARQUIVO XYZ NAO EXISTE"
150 ME$(3) = "DATA INVALIDA"
   .
   .
   .
   .
2000 IF X > 10 THEN PRINT ME$(1)
```

14.9 Testes

1) Dado o programa, o que será mostrado?

```
10 ON ERROR GOTO 500
20 A = 0
30 B = 10
40 C = B/A
```

```
50 PRINT C
60 END
500 A = 2: RESUME 0
```

a) 10
b) 0
c) 5
d) 2
e) nenhuma das anteriores.

2) Qual dos RESUME retorna o controle para a declaração onde ocorreu o erro?

a) RESUME
b) RESUME 0
c) RESUME NEXT
d) As alternativas a e b estão corretas;
e) nenhuma das anteriores.

3) Qual é a função da instrução ON ERROR GOTO 0?

a) desviar para a linha 0 em caso de erro;
b) ativar a rotina de tratamento de erros;
c) desativar a rotina de tratamento de erros;
d) nenhuma das anteriores.

4) O que acontece se ocorrer um erro dentro da rotina de tratamento de erros?

a) o programa ignora o erro;
b) o programa será interrompido;
c) nenhuma das anteriores.

5) Dado o programa, o que será mostrado?

```
10 ON ERROR GOTO 500
20 A = 0
30 B = 10
40 C = B/A
50 PRINT C
60 END
500 PRINT ERL; ERR: RESUME 60
```

a) 5
b) 10
c) 0
d) 40, 11
e) nenhuma das anteriores.

14.10 Exercícios propostos

1) Implemente nos programas o recurso para tratamento de erros.
2) Verifique no Apêndice D todos os códigos de erros.
3) Verifique quais as teclas que podem ser inibidas durante a execução.

Por exemplo: para neutralizar a tecla RESET no APPLE com cartão CP/M, proceda da seguinte maneira:

```
10 ON ERROR GOTO 1000
.
.
.
1000 IF ERR = 31 THEN RESUME
```

15

COMPILAÇÃO

"Em programação, não basta ser inventivo e engenhoso. É necessário também ser disciplinado e controlado a fim de não se emaranhar em sua própria complexidade."

Harlan Mills

15.1 Introdução

Este capítulo está baseado na versão 5.3 do compilador BASIC da MICROSOFT.

Existem diferenças significativas entre as versões disponíveis no mercado, portanto verifique qual é a versão que você possui e quais as diferenças com a versão 5.3.

Este capítulo não esgota o assunto compilação. Caso deseje mais informações, leia o manual de compilação da MICROSOFT.

15.2 Compiladores "versus" interpretadores

Um microprocessador somente pode executar suas próprias instruções de máquina; ele não pode executar declarações BASIC diretamente. Portanto, antes da declaração BASIC ser executada, algum tipo de tradução deve ocorrer.

Compiladores e interpretadores são dois tipos de programas que executam esta tradução.

O compilador converte o programa fonte em programa objeto que está em linguagem de máquina que o microcomputador pode entender e, portanto, executar.

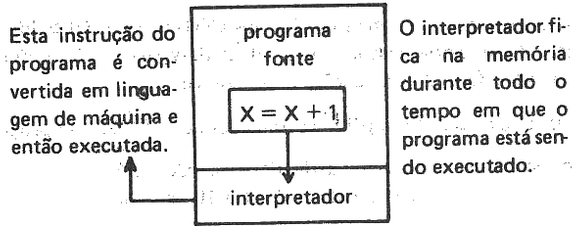
O interpretador, por outro lado, executa a conversão a cada vez que o programa é executado.

A execução de um programa interpretado é muito lenta, se comparada a um programa compilado. Entretanto, o uso do interpretador reduz drasticamente o tempo de desenvolvimento de programas na medida em que qualquer modificação no programa fonte poderá ser feita a qualquer momento em que o programa estiver sendo executado, enquanto que um programa compilado, quando alterado, deve ser recompilado, antes que possa ser executado novamente.

Outra vantagem do interpretador é que os erros de sintaxe são informados no momento em que a instrução é executada, ao contrário do compilador que informa os erros somente após a compilação. Levando-se em conta que poucos programas se executam na primeira vez após terem sido escritos, sendo mais comum que o programa contenha erros e requeira muitas alterações antes de ser considerado como pronto, esta vantagem é muito significativa.

15.3 Como funciona a interpretação

A função do interpretador é executar a tradução do programa, instrução por instrução, durante a sua execução. Para executar uma instrução BASIC, o interpretador analisa-a, verificando se há erro, e em seguida executa a função BASIC requerida.



Se uma declaração deve ser executada repetidamente (dentro de um laço FOR/NEXT, por exemplo), esta tradução deverá ser repetida cada vez que a declaração for executada.

O programa BASIC é armazenado como uma lista encadeada de números de linhas e cada linha não é disponível como um endereço absoluto de memória durante a interpretação. Portanto, desvios como GOTOs e GOSUBs fazem com que o interpretador examine cada linha do programa, começando com a primeira até que a linha referenciada seja encontrada.

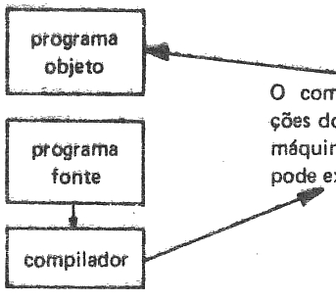
Também é mantida pelo interpretador uma lista de todas as variáveis utilizadas no programa. Quando for feita uma referência a uma variável, esta lista será pesquisada desde o início, até que a variável referida seja encontrada.

Deste modo, as variáveis do programa não estão associadas com endereços absolutos de memória.

> Dica. Quando estiver usando o interpretador, defina as variáveis mais usadas nas primeiras linhas do programa, pois as mesmas serão encontradas mais rapidamente.

15.4 Como funciona a compilação

O compilador transforma um programa fonte em um programa objeto, isto é, em forma de código de máquina. O programa objeto contém o código de máquina relocável. Todas as "traduções" devem ser feitas antes da execução, pois nenhuma "tradução" ocorre durante a execução do programa.



O compilador converte as instruções do programa em linguagem de máquina que o microcomputador pode executar.

As variáveis e os endereços referenciados nos GOTOs e nos GOSUBs estão associados a endereços absolutos de memória. Desta ferma, a lista de variáveis ou os números de linhas não precisam ser pesquisados durante a execução do programa.

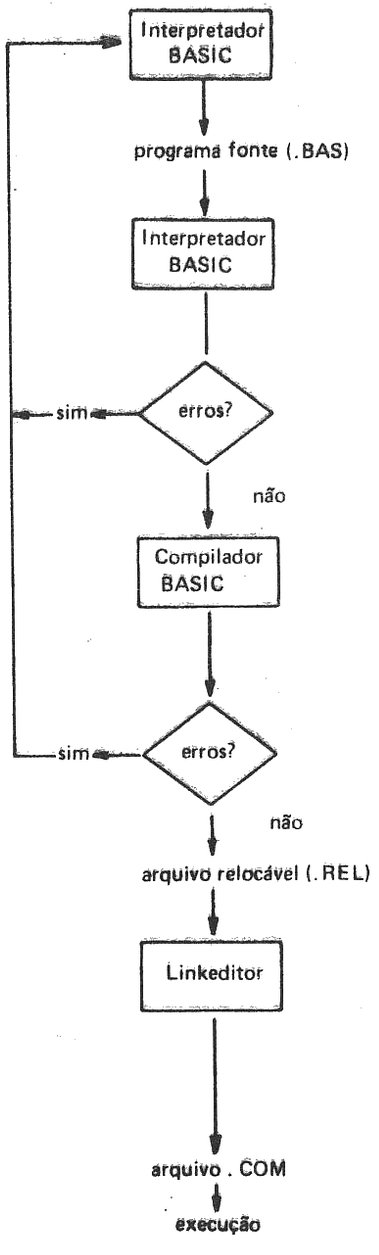
As principais vantagens obtidas pelo programador ao compilar um programa são:

- aumento significativo na velocidade de execução do programa. Em muitos casos a execução do programa compilado é de 3 a 10 vezes mais rápida que a do programa interpretado;
- diminuição do tamanho do programa;
- segurança no programa fonte.

Para ser compilado, é necessário que o programa fonte tenha sido gravado no formato ASCII.

> Dica. Habitue-se a compilar seus programas, pois estará resguardando seu trabalho de alterações indevidas ou mesmo de fraude. Além disso, o programa será executado com maior rapidez.

15.5 Etapas na compilação e linkedição



1) Cria e edita o programa fonte.

2) Executa e depura o programa fonte.

3) Compila o programa fonte e cria o arquivo relocável.

4) Linkedita o programa relocável e cria o arquivo . COM.

15.6 Usando o compilador

O formato geral para a linha de comando para chamar o compilador é:

BASCOM [relocável], [listagem] = < fonte >/[chaves]

arquivos de saída	arquivo de entrada
[relocável]	nome do arquivo relocável (.REL)
[listagem]	nome do arquivo listagem (.PRN)
< fonte >	nome do programa fonte (.BAS)
[chaves]	Verifique o item 15.6.4.

Exemplo:

BASCOM CGP11.REL, CGP11.PRN = CGP11.BAS/N

Serão criados os arquivos CGP11.REL e CGP11.PRN. Não serão gravadas no arquivo CGP11.PRN as instruções correspondentes em Assembler, devido a utilização da chave /N.

Caso não queira criar o arquivo listagem em disquete, e sim mostrar no video ou impressora, use uma das opções relacionadas abaixo:

a) usando o video

BASCOM [relocável], TTY: = < fonte >

b) usando a impressora

BASCOM [relocável], LST: = < fonte >

No final da compilação será mostrada a mensagem:

numero-de-erros) Fatal Error(s)
(bytes-livres) Bytes Free

Lembre-se. O programa a ser compilado deverá ter sido gravado no formato ASCII.

Exemplo: SAVE "EXEMPLO.ASC", A

15.6.1 Como criar somente o arquivo relocável

BASCOM = < fonte >

Será criado um arquivo relocável no mesmo disquete em que está o < fonte >.

Exemplo:

BASCOM = EXEMPLO.BAS

Neste caso será criado apenas o arquivo relocável EXEMPLO.REL.

Outra maneira é:

BASCOM [relocável] = < fonte >

Neste caso, o arquivo relocável será criado com o nome e drive especificados.

Exemplo:

BASCOM B:MENU5.REL = A:MENU5.BAS

15.6.2 Como criar somente o arquivo listagem

BASCOM , [listagem] = < fonte >

O arquivo [listagem] contém a listagem linha por linha do programa fonte. O código objeto gerado para cada declaração BASIC é "desassemblado" e listado junto com as declarações BASIC.

Para suprimir a listagem do código objeto, use a chave de compilação /N.

Exemplo:

BASCOM , CPP22.PRN = CPP22.BAS/N

15.6.3 Apenas compilando sem criar nenhum arquivo

Algumas vezes estamos somente interessados em compilar o programa para verificar se ocorrerá algum erro e quantos bytes livres ainda restam na memória.

BASCOM , = < fonte >

Esta é uma maneira rápida de verificar erros de compilação no programa.

Exemplo: BASCOM , = INCLUSAO.BAS/N

15.6.4 Chaves usadas na compilação

As chaves devem ser colocadas após o programa fonte ou após outras chaves.

CATEGORIA	CHAVE	AÇÃO
ERROS	/E	Programa tem ON ERROR GOTO com RESUME [linha]
	/X	Programa tem ON ERROR GOTO com RESUME, RESUME 0 ou RESUME NEXT
CODIGOS ESPECIAIS	/Z	Usar conjunto de instruções do Z80
	/N	Não imprime o código objeto desassemblado
	/D	Programa contém TRON/TROFF
	/S	Otimizar uso de strings entre aspas durante a compilação. Deverá ser usado quando, durante a compilação, for necessário usar muita memória.

Exemplo:

BASCOM CPP01.REL, CPP01.PRN = CPP01/E/N/D

Observação. Não confundir estas chaves com as usadas na linkedição.

Dica. A compilação deverá ser a última etapa no desenvolvimento de um programa. Somente após ter-se depurado o programa totalmente é que se deve fazer a compilação.

15.7 Diferenças entre a linguagem BASIC usada no interpretador e no compilador

15.7.1 Diferenças operacionais

Comandos não aceitos pelo compilador:

AUTO CONT DELETE EDIT LIST LLIST
LOAD MERGE NEW RENUM SAVE

15.7.2 Diferenças de linguagem

CHAIN Não aceita ALL, MERGE e DELETE.

CLEAR O argumento deve ser inteiro.

COMMON Deve aparecer antes de qualquer declaração executável.
Lista de declarações não executáveis:
COMMON
DEFDBL, DEFINT, DEFSNG, DEFSTR
DIM
OPTION BASE
REM
Todas as outras declarações são executáveis.
As matrizes devem ser declaradas antes de COMMON.

DIM O subscrito deve ser uma constante inteira.
Exemplos de declarações DIM ilegais:
DIM A (I)
DIM A (3 + 4)
DIM A (3 *4E5)

END Fecha todos os arquivos e volta ao CP/M. (No programa interpretado, não volta ao CP/M.)

ERASE Não é permitido.

ON ERROR GOTO Se o programa contém ON ERROR GOTO e RESUME linha, deve ser fornecida a chave / E. Se for usado RESUME NEXT, RESUME ou RESUME 0, a chave / X deve ser usada no lugar de / E.

RUN O compilador aceita as formas RUN e RUN linha, mas não aceita a opção "R".

STOP É idêntico a instrução END. Fecha todos os arquivos e retorna ao sistema operacional.

TRON/TROFF Deve ser usada a chave / D.

15.8 Códigos de erros na compilação

15.8.1 Erros fatais

CÓDIGO	SIGNIFICADO
BS	Subscrito inválido
CD	Variável duplicada no COMMON
CN	Matriz no COMMON não está dimensionada
CO	COMMON fora de seqüência
DD	Redimensionamento de matriz
FD	Função já foi definida
FN	Erro em FOR/NEXT
LL	Linha muito grande
LS	Constante string muito grande
OM	Estouro de memória
OV	Overflow
SN	Erro de sintaxe
SQ	Erro de seqüência
TC	Expressão muito complexa
TM	Tipos incompatíveis
UC	Comando não reconhecido
UF	Função não definida
WE	Erro em laço WHILE/WEND
/ 0	Divisão por zero
/ E	Chave / E ausente
/ X	Chave / X ausente

15.8.2 Erros não fatais

CÓDIGO	SIGNIFICADO
ND	Matriz não dimensionada
SI	Declaração ignorada

15.9 Usando o linkeditor

O formato geral para a linha de comando para chamar o linkeditor é:

L80 < programa >/[chaves], < programa >/[chaves]

Exemplos:

a) linkedita e executa

L80 EXEMPLO/G

Não será gravado o arquivo . COM . O programa EXEMPLO será executado após a linkedição.

b) linkedita e grava

L80 EXEMPLO1, EXEMPLO2/N/E

Será gravado o arquivo EXEMPLO2. COM e o controle volta ao CP/M após a linkedição

c) linkedita, grava e executa

L80 XYZ, ABC/N/G

Será gravado o arquivo ABC. COM e o mesmo será executado após a linkedição.

Após a linkedição é emitida a mensagem no video:

DATA (início-programa) (fim-programa) (bytes)

(bytes-livres) Bytes free
((endereço-inicial) (fim-programa) (num.-pag.)

(início-programa)	endereço em hexadecimal do início do programa
(fim-programa)	endereço em hexadecimal do fim do programa
(bytes)	tamanho do programa (em decimal)
(bytes-livres)	espaço de memória não utilizado durante a linkedição (decimal)
(endereço-inicial)	endereço inicial de execução do programa, não necessariamente o mesmo que (início-programa)
(núm.-pág.)	número de páginas usadas pelo programa. Página = 256 bytes.

Observação. Para interromper a linkedição, pressione ctl C.

15.9.1 Chaves usadas na linkedição

CHAVE	AÇÃO
/ E	Retorna o controle ao CP/M após a linkedição
/ G	Executa o arquivo . COM que foi criado após a linkedição
/ N	Grava em disquete o programa especificado com a extensão . COM

Dicas:

(1) Coloque no prólogo do programa fonte as chaves que devem ser usadas na compilação e linkedição.

(2) Use extensão . REL para programas em formato relocável e extensão . COM para programas em módulo de carga.

15.9.2 Mensagens de erro do linkeditor

?No Start Address	Foi usada a chave / G, mas nenhum programa foi carregado.
?Loading Error	O arquivo de entrada não está em código objeto correto.
?Out of Memory	Memória insuficiente para carregar o programa.
?Command Error	Comando inválido.
? < arquivo > Not Found	Arquivo não encontrado.

15.10 Testes

1) Se o programa contém as instruções ON ERROR GOTO e RESUME linha, qual das chaves de compilação deve ser usada?

a) / E;
b) / X;
c) / Z;
d) / D;
e) nenhuma das anteriores.

2) Se o programa contém as instruções ON ERROR GOTO e RESUME, ou RESUME 0 ou RESUME NEXT, qual das chaves de compilação deve ser usada?

a) / N;
b) / X;
c) / Z;
d) / E;
e) nenhuma das anteriores.

3) Se o programa contém TRON/TROFF, qual das chaves de compilação deve ser usada?

a) / T;
b) / E;
c) / X;
d) / S;
e) / D.

4) Qual das chaves de linkedição deve ser usada para que o controle volte ao CP/M após a linkedição?

a) / G;
b) / N;
c) / D;
d) / E;
e) nenhuma das anteriores.

5) Qual das chaves de linkedição deve ser usada para que o programa . COM seja executado em seguida?

a) / D;
b) / N;
c) / P;
d) / E;
e) nenhuma das anteriores.

6) Qual é o formato correto para chamar o compilador?

a) BASCOM listagem, relocável = < fonte >
b) BASCOM < fonte > = [relocável], [listagem]
c) BASCOM [relocável], < fonte > = [listagem]
d) BASCOM < fonte >, [relocável] = [listagem]
e) nenhuma das anteriores.

7) Qual dos comandos fará com que seja criado apenas o arquivo listagem?

a) BASCOM = < fonte >
b) BASCOM [relocável] = < fonte >
c) BASCOM, [listagem] = < fonte >
d) BASCOM = < fonte >
e) nenhuma das anteriores.

8) "A declaração STOP é idêntica a instrução END dentro de um programa compilado."

Esta afirmação é verdadeira ou falsa?

a) falsa;
b) verdadeira.

9) Qual das alternativas contém uma declaração DIM válida para o compilador?

a) DIM A (J)
b) DIM A (3)
c) DIM A (J + 3)
d) DIM A (3.4)

15.11 Exercícios propostos

1) Compare os tempos de execução de programas interpretados e compilados.

2) Compare se o tamanho do programa compilado é menor que o do interpretado.

A

SUMÁRIO DO BASIC

COMANDOS DO BASIC

COMANDO	FUNÇÃO
AUTO [início] [, incr]	Numerar automaticamente as linhas do programa.
CONT	Continuar a execução do programa após ctl C, STOP ou END.
DELETE [início] [– fim]	Eliminar linhas do programa.
EDIT linha	Editar uma linha do programa.
FILES ["d: *. *"]	Mostrar a relação de arquivos no disquete.
KILL "d: arquivo.ext"	Eliminar o arquivo do disquete.
LIST [início] [– fim]	Mostrar no video as linhas do programa.
LLIST [início] [– fim]	Mostrar na impressora as linhas do programa.
LOAD "programa" [, R]	Carregar um programa na memória.
MERGE "d: programa"	Intercalar um programa em disquete com o programa que está na memória.
NAME "anterior" AS "novo"	Mudar o nome do arquivo em disquete.
NEW	Eliminar o programa e variáveis da memória.
RENUM [nova] [, iníc.] [, incr]	Renumerar as linhas do programa.
RESET	Fechar os arquivos em disquete, atualizando seu diretório. Usado antes de trocar de disquete.
RUN [linha]	Executar o programa a partir da linha especificada.
RUN "d: programa" [, R]	Carregar e executar um programa.
SAVE "d: programa" [, A ou P]	Gravar o programa da memória em disquete.
SYSTEM	Fechar todos os arquivos e retornar ao CP/M.
TRON	Imprimir, entre colchetes, as linhas do programa que estão sendo executadas.
TROFF	Desativar o comando TRON.

DECLARAÇÕES BASIC

DECLARAÇÃO — FUNÇÃO

CHAIN [MERGE] "prog." [, linha] [, ALL] [, DELETE interv.]
Chamar um programa e, eventualmente passar variáveis do programa corrente.

CLEAR [, expr1]
Limpar todas as variáveis e, opcionalmente, definir o topo da memória.

CLOSE [#] numarq1 [, numarq2] . . .
Encerrar acesso a arquivos.

COMMON var1 [, var2] . . .
Passar uma lista de variáveis para o programa chamado pela instrução CHAIN.

DATA item1, item 2]. . .
Armazenar uma lista de dados que vão ser acessados através da instrução READ.

DEF FNnome (argumentos) = expressão
Definir uma função numérica ou string.

DEFDBL letra [– letra] . . .
Identificar as variáveis do intervalo como sendo de dupla precisão.

DEFINT letra [– letra] . . .
Identificar as variáveis do intervalo como sendo inteiras.

DEFSNG letra [– letra] . . .
Identificar as variáveis do intervalo, como sendo de simples precisão.

DEFSTR letra [– letra] . . .
Identificar as variáveis do intervalo como sendo strings.

DIM matriz1 [, matriz2] . . .
Definir e dimensionar matrizes. Ex.: DIM A (10), C (20), . . .

END
Terminar a execução do programa, fechar todos os arquivos e voltar ao modo imediato.

ERASE matriz1 [, matriz2] . . .
Eliminar as matrizes especificadas, liberando o espaço na memória por ela(s) ocupado.
Ex.: ERASE A, B, . . .

ERROR codigo
Simular a ocorrência de um erro do BASIC e também permitir definir seus próprios códigos de erro. O valor do código deve estar entre 0 e 255.

FIELD [#] numarq, tam. AS var$, . . .
Alocar espaço para as variáveis em um "buffer" de arquivo randômico.
Ex.: FIELD#1, 20 AS N$, 10 AS ID$

FOR var = x TO y [STEP Z]
A variável é usada como um contador, onde x é o valor inicial e y o último valor do contador. As instruções contidas dentro deste laço serão executadas enquanto "var" não for maior que y.

GET [#] numarq [, reg]
Ler um registro de um arquivo randômico em disquete.

GOSUB linha
Desviar o fluxo do programa para a sub-rotina começando em "linha"

GOTO linha
Desviar o fluxo do programa para a linha especificada.

IF condição THEN ação1 ELSE ação2
O fluxo do programa será seguido conforme as condições.

INPUT ["mensagem";] var1 [, var2] . . .
Permitir a entrada de dados pelo teclado.

INPUT [#] numarq, var1 [, var2] . . .
Ler dados de um arquivo seqüencial em disquete.

[LET] variavel = expressão
Assinala o valor da expressão a variável.

LINE INPUT [;] ["mensagem"]; var$
Atribuir uma linha inteira (até 254 caracteres) a uma variável string sem usar delimitadores.

LINE INPUT [#] numarq, var$
Ler um string de um arquivo seqüencial em disquete.

LPRINT [USING "formato"]; var1 [, var2] . . .
Imprimir os valores das variáveis na impressora.

LSET var$ = exprstr
Mover a "exprstring" para uma variável do "buffer" de arquivo randômico, alinhando-a a esquerda.

ON ERROR GOTO linha
Desviar no caso de erro para a rotina começando em "linha"

ON expr GOSUB linha1 [, linha2] . . .
Desviar a execução do programa para uma das linhas, dependendo do resultado da expressão.

ON expr GOTO linha1 [, linha2] . . .
Desviar para uma das linhas, dependendo do resultado da expressão.

OPEN "modo", # numarq, "nomearq" [, tamreg]
Permitir a entrada e saída para um arquivo em disquete, onde:

"modo"	expressão string onde:
"O"	especifica modo de saída para arquivo seqüencial
"I"	especifica modo de entrada para arquivo seqüencial
"R"	especifica modo de entrada/saída para arquivo randômico
numarq	expressão inteira entre 1 e 15.
"nomearq"	nome do arquivo
tamreg	define o comprimento do registro para arquivos randômicos. (default = 128).

OPTION BASE n
Declarar o valor mínimo para índices de matriz, sendo n igual a 0 ou 1.

POKE endereço, valor
Colocar o valor especificado no endereço de memória definido.

PRINT [USING "formato";] var1 [, var2] . . .
Mostrar string ou números no video, de acordo com o formato especificado.

formato p/string	Pode-se usar um dos seguintes caracteres para formatação:
"!"	Somente o primeiro caracter do string será impresso.
"\ n espaços \ "	2 + n caracteres serão impressos.
"&"	Imprime todos os caracteres do string.
formato p/números	Os seguintes caracteres podem ser usados para formatar um campo numérico:

#	Representa um dígito numérico.
.	Ponto decimal.
+	Um sinal positivo no início ou final do formato fará com que o sinal da expressão numérica seja impresso antes ou depois do número.
−	Um sinal negativo no final do formato fará com que os números negativos sejam impressos com sinal de menos.
**	Dois asteriscos no começo do formato farão com que todas as posições não utilizadas a esquerda do ponto decimal sejam preenchidas com asterisco.
$$	Um sinal duplo de dólar fará com que seja colocado o sinal dólar a esquerda do número formatado.
**$	Os primeiros espaços serão preenchidos com asteriscos e o sinal de dólar será impresso antes do número.
,	Uma vírgula a esquerda do ponto decimal de um formato imprimirá uma vírgula a cada três dígitos a esquerda do ponto decimal.
↑↑↑↑	Estes quatro caracteres permitem que se escreva um número no formato exponencial, ou seja, E + XX.

PRINT [#] numarq [USING "formato",] var1 [, var2] . . .
Gravar dados em um arquivo seqüencial em disquete.

PUT [#] numarq [, numreg]
Gravar dados do "buffer" de arquivo randômico em um arquivo em disquete.

RANDOMIZE [expressão]
Inicializar o gerador de números aleatórios.

READ var1 [var2] . . .
Ler os valores contidos numa instrução DATA, atribuindo-os as variáveis especificadas.

REM [comentários]
Permitir a inserção de comentários num programa.

RESTORE [linha]
Permitir que a instrução DATA seja relida a partir da linha especificada.

RESUME
Retornar da rotina chamada em ON ERROR GOTO linha para a instrução que causou o erro.

RESUME 0
Retornar da rotina chamada em ON ERROR GOTO linha para a instrução que causou o erro.

RESUME NEXT
Retornar da rotina chamada em ON ERROR GOTO linha para a instrução seguinte a do erro.

RETURN
Volta para a instrução seguinte ao último GOSUB.

RSET var$ = exprstring
Mover a expressão string para uma variável do "buffer" de arquivo randômico, alinhando-a a direita.

STOP
Interrompe a execução do programa.

SWAP var1, var2
Trocar os valores de duas variáveis.

WEND
Finaliza um laço definido pela instrução WHILE

WHILE condição
As instruções entre WHILE e WEND serão executadas enquanto a condição for verdadeira

WIDTH [LPRINT] tamanho
Define o tamanho da linha do video ou impressora.

WRITE [#] numarq, var1 [, var2] . . .
Grava dados em um arquivo de acesso seqüencial.

FUNÇÕES DO BASIC

FUNÇÃO	RETORNA
ABS (expr)	Valor absoluto da expressão.
ASC (str)	Valor em ASCII do primeiro caracter do string.
ATN (expr)	Valor do arco tangente da expressão.
CDBL (expr)	Valor da expressão com dupla precisão.
CHR$ (expr)	Caracter cujo valor em ASCII é = expr.
CINT (expr)	Valor inteiro da expressão.
COS (expr)	Cos-seno da expressão (deve estar em radianos).
CSNG (expr)	Valor da expressão em simples precisão.
CVD (str)	Converte um string em um valor de dupla precisão.
CVI (str)	Converte um string em um valor inteiro.
CVS (str)	Converte um string em um valor de simples precisão.
EOF (expr)	Valor − 1, se for encontrado o fim do arquivo.
ERL	Linha onde ocorreu o erro.
ERR	Código do erro.
EXP (expr)	Inversão do logaritmo da expressão.
FIX (expr)	Parte inteira da expressão.
FRE (expr)	Área disponível na memória.
INKEY$	Caracter pressionado no teclado.
INPUT$ (n)	String de n caracteres, lidos do teclado.
INSTR (n, X$,Y$)	Posição de Y$ dentro de X$ a partir de n.
INT (expr)	Número inteiro da expressão.
LEFT$ (str, n)	Substring do lado esquerdo do string.
LEN (str)	Tamanho do string.
LOC (f)	Número de setores lidos ou gravados em arquivos seqüenciais. Próximo número de registro em arquivos aleatórios.
LOG (expr)	Logaritmo neperiano da expressão.
LPOS (X)	Posição do cabeçote de impressão na linha.
MID$ (str, n, t)	Substring com início na posição n e tamanho t.
MKD$ (expr)	String de 8 caracteres de expressão dupla precisão.
MKI$ (expr)	String de 2 caracteres da expressão inteira.
MKS$ (expr)	String de 4 caracteres da expressão simples precisão.
PEEK (endereço)	Byte armazenado no endereço especificado.

POS (X)	Posição do cursor na linha do video.
RIGHT$ (str, n)	Substring do lado direito do string.
RND (expr)	Número aleatório entre 0 e 1.
SGN (expr)	Sinal da expressão: 1 se a expr. for positiva 0 se a expr. for zero – 1 se a expr. for negativa
SIN (expr)	Seno da expressão (que está em radianos).
SPACE$ (expr)	String de espaços de comprimento = "expr".
SPC (n)	n espaços (usado nas instruções PRINT e LPRINT).
SQR (expr)	Raiz quadrada da expressão.
STR$ (expr)	Representação string da expressão.
STRING$ (n, chr)	String de um caracter ASCII repetido n vezes.
TAB (expr)	Move o cursor para a posição da expressão.
TAN (expr)	Tangente da expressão em radianos.
VAL (str)	Valor numérico do string.
VARPTR (var)	Endereço da variável na memória.

SUBCOMANDOS DE EDIÇÃO

A	Cancela edições feitas e volta o cursor ao início da linha.
n Barra-espaços	Move o cursor "n" posições a direita.
n BACKSPACE	Move o cursor "n" posições a esquerda.
n C	Substitui "n" caracteres.
n D.	Apaga "n" caracteres começando pela posição do cursor.
E	Encerra a edição, mantendo as modificações.
H	Apaga o que estiver a direita do cursor e entra no modo de inserção.
I	Insere "n" caracteres que vierem a seguir.
nKc	Apaga caracteres até a n-ésima ocorrência de "c".
L	Mostra a linha editada e volta ao início dela.
Q	Sai da edição cancelando as modificações efetuadas.
nSc	Move o cursor para a n-ésima ocorrência de "c".
X	Move o cursor para o fim da linha e entra no modo de inserção.
ctl ESC	Sai dos subcomandos de edição.
ENTER	Introduz a linha editada.

B

CÓDIGOS DOS CARACTERES ASCII

DEC	CARACT	DEC	CARACT	DEC	CARACT	DEC	CARACT
0	ctl @	24	ctl X	48	0	72	H
1	ctl A	25	ctl Y	49	1	73	I
2	ctl B	26	ctl Z	50	2	74	J
3	ctl C	27	ctl [	51	3	75	K
4	ctl D	28	ctl \	52	4	76	L
5	ctl E	29	ctl]	53	5	77	M
5	ctl F	30	ctl ↑	54	6	78	N
7	ctl G	31	ctl –	55	7	79	O
8	ctl H	32	space	56	8	80	P
9	ctl I	33	!	57	9	81	Q
10	ctl J	34	"	58	:	82	R
11	ctl K	35	#	59	;	83	S
12	ctl L	36	$	60	<	84	T
13	ctl M	37	%	61	=	85	U
14	ctl N	38	&	62	>	86	V
15	ctl O	39	'	63	?	87	W
16	ctl P	40	(	64	@	88	X
17	ctl Q	41	)	65	A	89	Y
18	ctl R	42	*	66	B	90	Z
19	ctl S	43	+	67	C	91	[
20	ctl T	44	,	68	D	92	\
21	ctl U	45	–	69	E	93	]
22	ctl V	46	.	70	F	94	↑
23	ctl W	47	/	71	G	95	–

DEC	CARACT	DEC	CARACT	DEC	CARACT	DEC	CARACT
96	'	104	h	112	p	120	x
97	a	105	i	113	q	121	y
98	b	106	j	114	r	122	z
99	c	107	k	115	s	123	{
100	d	108	l	116	t	124	|
101	e	109	m	117	u	125	}
102	f	110	n	118	v	126	~
103	g	111	o	119	w	127	DEL

Observações:

ctl = tecla CONTROL
ctl G (bip) = Campainha
ctl I (tab) = Tabulação a cada 8 colunas
ctl H (BS) = Backspace
ctl J (LF) = Line Feed
ctl L (FF) = Form Feed
ctl M (CR) = Carriage Return
ctl [(ESC) = ESCape

C

PALAVRAS RESERVADAS DO BASIC

ABS	DIM	INSTR	OCT$	SPACE$
ALL	EDIT	INT	ON	SPC
AND	END	KILL	OPEN	SQR
ASC	EOF	LEFT$	OPTION	STEP
ATN	ERASE	LEN	OR	STOP
AUTO	ERL	LET	OUT	STR$
CALL	ERR	LINE	PEEK	STRING$
CDBL	ERROR	LIST	POKE	SWAP
CHAIN	EQV	LLIST	POS	SYSTEM
CHR$	EXP	LOAD	PRINT	TAB
CINT	FIELD	LOC	PRINT#	TAN
CLEAR	FILES	LOF	PUT	THEN
CLOSE	FIX	LOG	RANDOMIZE	TO
COMMON	FN	LPOS	READ	TROFF
CONT	FOR	LPRINT	REM	TRON
COS	FRE	LSET	RENUM	USING
CSNG	GET	MERGE	RESET	USR
CVD	GOSUB	MID$	RESTORE	VAL
CVI	GOTO	MKD$	RESUME	VARPTR
CVS	HEX$	MKI$	RETURN	WAIT
DATA	IF	MKS$	RIGHT$	WEND
DEF	IMP	MOD	RND	WHILE
DEFDBL	INKEY$	NAME	RSET	WIDTH
DEFINT	INP	NEW	RUN	WRITE
DEFSNG	INPUT	NEXT	SAVE	XOR
DEFSTR	INPUT$	NOT	SGN	
DELETE	INPUT$	NULL	SIN	

D

MENSAGENS E CÓDIGOS DE ERROS

Este apêndice lista todas as mensagens de erro com os respectivos códigos.

1 NEXT without FOR
NEXT sem o FOR correspondente.

2 Sintax error
Erro de sintaxe.

3 RETURN without GOSUB
RETURN sem GOSUB correspondente.

4 Out of DATA
Faltam dados para a instrução READ.
Exemplo: READ A, B, C
DATA 53, 57

5 Illegal function call
Parâmetro ilegal para uma função strings ou matemática.
Exemplo: (*a*) subscrito maior que 32767
(*b*) subscrito negativo
(*c*) função SQR com parâmetro negativo
(*d*) função LOG com argumento negativo ou zero
(*e*) número de registro negativo para GET ou PUT
(*f*) função ou declaração com argumento inválido
(*g*) tentativa de listar ou editar um programa BASIC gravado em formato protegido.

6 Overflow
A magnitude do número é muito grande para ser armazenado. Se ocorrer "underflow", o resultado será zero e a execução continuará.
Exemplo: um número inteiro não pode exceder 32767.

7 Out of memory
O programa é muito grande, ou tem muitos laços, ou GOSUBs, ou muitas variáveis, ou expressões muito complexas.

8 Undefined line number
Referência a linha inexistente.

9 Subscript out of range
Pode ser causado por:
(*a*) subscrito fora do limite da matriz
(*b*) número de subscritos maior do que o dimensionado
Exemplo: DIM A (10, 10)
A (5, 5, 3) = Z
Observe que a matriz A não tem a terceira dimensão especificada

10 Duplicate Definition
Pode ser causado por:
(*a*) duas declarações DIM para a mesma matriz
(*b*) matriz dimensionada pelo "default" e depois pela declaração DIM
Exemplo: 10 A (1) = 14
20 DIM A (20)
(*c*) Encontrado OPTION BASE após a matriz ter sido dimensionada

11 Division by zero
Pode ser causado por:
(*a*) dividir uma expressão por zero. Neste caso, o resultado será 1.70141E +38 (maior valor que pode ser armazenado) com o sinal do numerador.
(*b*) elevar zero a uma potência negativa. O resultado também será 1.70141E + 38. Nos dois casos, a execução do programa continua.

12 Illegal direct
Uso incorreto do modo imediato.
Exemplo: uso da instrução DEF FN no modo imediato.

13 Type mismatch
Uma variável string é assinalada com um valor númerico, e vice-versa.
Exemplo: (*a*) SWAP A$, B%
(*b*) X = Y$

14 Out of string space
Memória insuficiente para strings. O BASIC aloca dinamicamente memória para strings, até terminar a memória disponível.

15 String too long
Tentativa de criar um string com mais de 255 caracteres.

16 String formula too complex
Expressão string muito complexa ou longa. Divida em várias outras de menor complexidade.

17 Can't continue
A execução não pode prosseguir usando CONT.
Exemplo: (*a*) não tem programa na memória
(*b*) o programa foi alterado durante a interrupção
(*c*) ocorreu erro

18 Undefined user function
Função de usuário (DEF FN) não definida.

19 No RESUME
Faltou RESUME na rotina de tratamento de erros.

20 RESUME without error
Encontrado RESUME sem ON ERROR GOTO <linha>.

21 Unprintable error
Código de erro sem mensagem correspondente. É usualmente causado por uma declaração ERROR com código indefinido.

22 Missing operand
Expressão contém um operador sem o operando correspondente.
Exemplo: 10 A =

23 Line buffer overflow
Linha do programa muito longa.

26 FOR Without NEXT
FOR sem NEXT correspondente.

29 WHILE without WEND
WHILE sem WEND correspondente.

30 WEND without WHILE
WEND sem WHILE correspondente.

50 FIELD overflow
Tamanho do "buffer" definido pelo FIELD excede o tamanho reservado pelo OPEN.

51 Internal error
Erro interno do BASIC.

52 Bad file number
Número de arquivos maior que o especificado na opção /F:na do MBASIC ou o arquivo não foi aberto anteriormente.

53 File not found
Arquivo não foi encontrado no disquete.

54 Bad file mode
Pode ser causado por:
(*a*) PRINT# em um arquivo randômico;
(*b*) PUT ou GET em um arquivo seqüencial;
(*c*) OPEN com modo diferente de "O", "I" ou "R";
(*d*) MERGE em um programa que não foi gravado em ASCII.

55 File already open
Arquivo já aberto ou KILL em um arquivo aberto.

57 Disk I/O error
Erro de entrada ou saída no disquete.

58 File already exists
Exemplo: NAME "A" AS "B" sendo que "B" já existe.

61 Disk full
Disquete cheio de dados.
Observação: Os arquivos abertos serão fechados quando ocorrer este erro.

62 Input past end
Tentativa de ler após o fim do arquivo ou tentativa de ler um arquivo vazio.
Sugestão: Use a instrução EOF (n) para detectar o fim do arquivo.

63 Bad record number
Número de registro incorreto no GET ou PUT. O número do registro deve estar no intervalo de 1 a 32767.
Exemplo: GET #1, 32850

64 Bad file name
Nome incorreto de arquivo.
Sugestão: Verifique no manual do CP/M as regras para formação de nomes.

66 Direct statement in file
LOAD, RUN ou MERGE em um arquivo que não é um programa BASIC.

67 Too many files
Excesso de arquivos no disquete.

E

OUTRAS DICAS

1) Manutenção do equipamento

1.1) Não deixe o equipamento descoberto quando não o estiver utilizando.

1.2) Não deixe fios expostos, nem atravessados nas áreas de passagem. Pode-se perder o trabalho de horas se alguém esbarrar em algum fio.

1.3) Quando o computador apresentar algum problema não se ponha a consertá-lo, solicite a presença do técnico.

2) Manuseio de disquetes

2.1) Não toque a superfície magnetizável do disquete.

2.2) Não dobre o disquete.

2.3) Não retire o disquete do drive quando a luz do drive estiver acesa.

2.4) Não deixe dois ou mais disquetes numa mesma capa protetora.

2.5) Não deixe os disquetes perto de campos magnéticos e de materiais ferromagnéticos. *Por exemplo*: video.

2.6) Não exponha o disquete ao Sol.

2.7) Não coloque objetos sobre o disquete.

2.8) Não use o disquete até sua capacidade máxima de armazenamento.

2.9) Evite manusear os disquetes em ambiente sujo.
Por exemplo: perto de cinzeiros.

2.10) Coloque os disquetes numa caixa de metal antimagnético, ao invés de caixa plástica. O plástico não intercepta campos magnéticos, enquanto o metal antimagnético sim.
Exemplo: o alumínio é um bom isolante.

2.11) Se possível, coloque apenas uma aplicação por disquete.

2.12) Não deixe no disquete em operação as versões anteriores de programas. O operador pode se confundir e utilizar uma versão anterior.

2.13) Coloque o protetor contra gravação nos disquetes que contenham apenas os programas.

2.14) Introduza o disquete suave e cuidadosamente no drive.

2.15) Compre apenas disquetes de boa qualidade.

2.16) Identifique todos os disquetes. Use regras mnemônicas para dar nomes aos disquetes. *Exemplo*: em vez de "CONSUMIDORES", utilize "CNSM".

2.17) Não use o disquete original. Tire um backup e use-o em lugar do original.

2.18) Use em todos os disquetes a mesma versão do sistema operacional.

2.19) Arquive os disquetes na posição vertical.

2.20) Escreva na etiqueta do disquete somente com canetas de ponta porosa. Outro tipo de caneta pode danificar a superfície magnetizável do disquete.

3) Backups

3.1) Mantenha os disquetes backup e os pouco usados longe do computador.

3.2) Coloque na identificação dos disquetes backup a data em que foi gerado o backup e a data de expiração, caso exista.

4) Segurança

4.1) Toda saída impressa que contenha informações confidenciais que não se queira guardar deve ser queimada.

5) Conjunto de instruções

5.1) Use somente as instruções padrão para todas as máquinas. Nos casos onde isto não é possível, documente o fato no programa, para facilitar manutenções ou conversões futuras.

5.2) Explore novos comandos da linguagem BASIC. Existe uma tendência para o uso de um conjunto mínimo de instruções desperdiçando as vezes instruções muito mais poderosas.

RESPOSTAS DOS TESTES

Capítulo 1

1) *e* A variável I é de simples precisão, pois foi assumido o "default". Portanto, as variáveis I! e I são a mesma coisa. O valor de I# é zero, pois a variável não foi definida.
2) *c* As variáveis de dupla precisão armazenam 16 dígitos e apresentam 16.
3) *b*.
4) *a*.
5) *b*.
6) *b* A multiplicação tem prioridade sobre a divisão inteira.
7) *d* FNCP$ é considerada uma função, e não uma variável.
8) *e* O valor de X e' 4.
9) *c*.
10) *c* Neste caso ocorrerá o erro "Type mismatch". Foi assinalado um valor numérico a uma variável alfanumérica.

Capítulo 2

1) *d* O comando DELETE n— não é permitido.
2) *a*.
3) *a*.
4) *c*.
5) *a*.
6) *c*.
7) *d*.
8) *a*.
9) *c* O programa poderá ser executado. Não poderá ser listado ou editado.
10) *b* Não é possível recuperar o programa após ter dado o comando NEW.

Capítulo 3

1) *e* A instrução END na linha 10 encerra a execução.
2) *c*.
3) *e* Ocorrerá o erro "Type mismatch". As variáveis devem ser do mesmo tipo.
4) *c* Quando não for definido o tipo, é assumido como simples precisão.
5) *b* A vírgula após a mensagem faz com que não seja mostrado o ponto de interrogação.
6) *d* Se o valor da expressão for 0 ou maior que o número de linhas especificado, o fluxo continuará na instrução seguinte.
7) *e* O programa será interrompido na linha 25.
8) *a* O código ASCII de "a" é maior que o código de "A".
9) *c* O valor 2.6 será arredondado para 3.
10) *d*.

Capítulo 4

1) *d*.
2) *e*.
3) *e*.
4) *e*.
5) *a*.
6) *e*.
7) *c*.
8) *a* A divisão e a multiplicação tem a mesma prioridade, entretanto, nestes casos, o cálculo é feito da esquerda para a direita.
9) *d*.
10) *a* O BASIC insere automaticamente a instrução END no fim do programa, caso não tenha sido colocada pelo programador.

Capítulo 5

1) *e* Ocorrerá erro, causado pela associação errada entre as instruções FOR e NEXT.
2) *e* Será mostrado 90.
3) *e* Será mostrado 16.
4) *b* É permitido sair de um laço, embora não seja recomendável. Também é permitido desviar para dentro do laço, desde que anteriormente tenha saído do laço.
5) *e* Será mostrado 17.5.
6) *e* Será mostrado 16.5.
7) *b*.
8) *d*.

Capítulo 6

1) *d* Com OPTION BASE 0 o elemento zero é definido tanto na linha quanto na coluna.
2) *b*.
3) *a*.
4) *a*.

5) *e* A matriz SAL é eliminada na linha 50 e é criada novamente na linha 60 através do "default". Portanto será mostrado 0, pois a matriz foi apenas dimensionada pelo "default" e contém apenas zeros, por se tratar de uma matriz numérica.
6) *a* A instrução OPTION BASE deve vir antes de qualquer declaração DIM.
7) *b* São variáveis distintas.
8) *b*.
9) *a* A matriz será criada na linha 100 usando o "default".
10) *d* A matriz será criada na linha 500 usando o "default".
11) *a* Tem a mesma função, embora o laço B seja mais eficiente.
12) *a* Haverá 6 inicializações no laço A e 21 inicializações no laço B, portanto o laço A é mais eficiente.

Capítulo 7

1) *d* A instrução RESTORE sem o número de linha recupera todas as instruções DATA do programa.
2) *e* O valor armazenado em X (2, 2) e 11.
3) *e* Ocorrerá erro "Out of DATA", isto é, faltam dados para a instrução READ.
4) *b*.
5) *e* O valor de A (7) será zero.
6) *b*.
7) *b*.
8) *c* Ocorrerá o erro "Type mismatch". A variável E é numérica e foi assinalado RS que é alfanumérico.

Capítulo 8

1) *e* CINT (X) = – 45, INT (X) = – 46, FIX (X) = – 45
CSNG (X) = – 45.3498, ABS (X) = 45.3498.
2) *e* Será mostrado 68 – 1 68
3) *d*.
4) *c*.
5) *e*.
6) *b*.
7) *e* Ocorrerá "Overflow". O argumento da função CINT deve estar no intervalo – 32768 a 32767.
8) *d*.
9) *a*.
10) *c*.

Capítulo 9

1) *b*.
2) *b*.
3) *c*.
4) *b*.
5) *d*.
6) *c*.

7) *c*.
8) *a*.
9) *c*.
10) *e*.
11) *a*.
12) *b*.
13) *d*.

Capítulo 10

1) *d* Com a instrução WRITE# somente os strings são gravados entre aspas.
2) *a*.
3) *a*.
4) *d*.
5) *d*.
6) *b* A instrução END fecha todos os arquivos e continua no BASIC.
7) *d*.
8) *b*.
9) *b*.
10) *e* O "default" é 3.

Capítulo 11

1) *b*.
2) *b*.
3) *a*.
4) *c* O valor máximo para o número de arquivos é 15.
5) *d*.
6) *c*.
7) *d*.
8) *a*.
9) *e* O formato correto é
OPEN "R", #<numarq>, "nomearq" [, tamreg]
10) *a*.
11) *a*.
12) *c*.

Capítulo 12

1) *e* Todas as alternativas estão corretas.
2) *b* Na passagem de matrizes entre programas não é especificado o subscrito.
3) *a*.
4) *c*.
5) *b*.
6) *a*.
7) *d*.

Capítulo 13

1) *b*.
2) *c*.
3) *c*.

Capítulo 14

1) *c* O valor de A é alterado de 0 para 2.
2) *d* RESUME e RESUME 0 têm a mesma função.
3) *c*.
4) *b* O recurso para tratar o erro não é aplicado dentro da rotina de tratamento de erros.
5) *d* O código de erro causado pela divisão por zero é 11.

Capítulo 15

1) *a*.
2) *b*.
3) *e*.
4) *d*.
5) *e* Chave /G:.
6) *e* O formato é BASCOM [relocavel], [listagem] = < fonte >/[chaves].
7) *c*.
8) *b*.
9) *b*.

RESPOSTAS DOS EXERCÍCIOS PROPOSTOS

CAPÍTULO 1

Exercício 1

```
a) X + Y
b) E + B/C
c) A + K/C + D
d) (A + M) / (C + D) + X
```

CAPÍTULO 3

Exercício 1

```
10 '
20 '   Este programa  calcula a media
30 '
40 INPUT "ENTRE COM AS CINCO NOTAS ";N1,N2,N3,N4,N5
50 MEDIA=(N1+N2+N3+N4+N5)/5
60 PRINT "MEDIA = ";MEDIA
```

Exercício 2

```
10 '
20 '    Este  programa calcula o numero  de
30 '    segundos em 1 minuto, 1 dia e 1 ano
40 '
50 PRINT "UM MINUTO TEM ";60;" SEGUNDOS"
60 PRINT "UMA HORA  TEM ";60*60;" SEGUNDOS"
70 PRINT "UM DIA    TEM ";60*60*24;" SEGUNDOS"
80 PRINT "UM ANO    TEM ";60*60*24*365;" SEGUNDOS"
```

Exercício 3

```
10 '
20 '    Este programa transforma temperatura em
30 '    Fahrenheit para graus Celsius
40 '
50 INPUT "Entre com a temperatura em Fahrenheit";F
60 C= 5*(F-32)/9
70 PRINT "A temperatura em Celsius e'";C;" graus"
80 GOTO 50
90 END
```

Exercício 4

```
10 '
20 '     Este programa verifica se o ano e' bissexto
30 '
40 INPUT "Entre com o ano"; A
50 PRINT A;
60 IF A/4 = INT(A/4) THEN PRINT "e' bissexto": GOTO 40
70 PRINT "nao e' bissexto": GOTO 40
```


Exercício 5

```
10 '
20 '   Este  programa calcula a area de um triangulo
30 '
40 INPUT "ENTRE COM O COMPRIMENTO DOS TRES LADOS ";L1,L2,L3
50 T=(L1+L2+L3)/2
60 AREA=(T*(T-L1)*(T-L2)*(T-L3))^0.5
70 PRINT "AREA = ";AREA
```

Exercício 6

```
10 '
20 ' Este  programa le tres  numeros
30 ' e imprime o maior
40 '
50 INPUT "ENTRE COM OS TRES VALORES ";A,B,C
60 MA=A
70 IF B > MA THEN MA=B
80 IF C > MA THEN MA=C
90 PRINT "MAIOR  = ";MA
100 END
```

CAPÍTULO 4

Exercício 1

```
10 '
20 '   Este programa verifica se um numero e' par ou impar
30 '
40 INPUT "Entre com o numero"; N
50 PRINT N;
60 IF N/2 <> INT(N/2) THEN PRINT "e' impar": GOTO 40
70 PRINT "e' par": GOTO 40
```

Exercício 2

```
10 '
20 '  Este  programa calcula a hipotenusa
30 '  de um triangulo retangulo
40 '
50 INPUT "ENTRE COM O COMPRIMENTO DO LADO 1";L1
60 INPUT "ENTRE COM O COMPRIMENTO DO LADO 2";L2
70 X1=L1^2
80 X2=L2^2
90 HIP=(X1+X2)^0.5
100 PRINT "HIPOTENUSA = ";HIP
```

Exercício 3

```
10 '
20 '  Este programa le os tres numeros e
30 '  imprime o maior e o menor
40 '
50 INPUT "ENTRE COM OS TRES NUMEROS";A,B,C
60 MA=A
70 ME=A
80 IF B > MA THEN MA=B
90 IF B < ME THEN ME=B
100 IF C > MA THEN MA=C
110 IF C < ME THEN ME=C
120 PRINT "MAIOR = ";MA
130 PRINT "MENOR = ";ME
```


CAPÍTULO 5

Exercício 1

```
10 '
20 '    Este programa calcula o fatorial de um numero
30 '
40 '
50 INPUT "Entre com o numero desejado";N
60 FAT#=1
70 FOR I=2 TO N
80    FAT#=FAT#*I
90 NEXT I
100 PRINT "O fatorial de";N;"e'";:PRINT USING " ##########";FAT#
110 GOTO 50
```

Exercício 2

```
10 '
20 '    Este programa imprime a tabuada de um numero qualquer
30 '
40 INPUT "Entre com o numero desejado";N
50 FOR I=1 TO 10
60    PRINT N;"X";I;"=";N*I
70 NEXT I
80 GOTO 40
```

Exercício 3

```
10 '
20 '   Este programa calcula os numeros de FIBONACI
30 '
40 '
50 X=1
60 Y=0
70 Z=X+Y
80 PRINT Z;
90 INPUT "Quer continuar (S)im (N)ao ";OP$
100 IF OP$="N" THEN END
110 X=Y
120 Y=Z
130 GOTO 70
```

Exercício 4

```
10 '
20 '   Programa para Calculo de Coeficiente Binomial
30 '    Onde:    CB=n!/r!(n-r)!
40 '
50 INPUT "ENTRE COM O (N) ";N
60 INPUT "ENTRE COM O (R) ";R
70 IF N<R THEN CB#=0 : GOTO 180
80 IF N=R OR R=0 THEN CB#=1 : GOTO 180
90 IF R=1 THEN CB#=N : GOTO 180
100 '
110 '   Calculo do Coeficiente Binomial
120 '
130 Z=N-R
140 A=N:GOSUB 260  :BN#=FT#
150 A=R:GOSUB 260  :BR#=FT#
160 A=Z:GOSUB 260  :BZ#=FT#
170 CB#=BN#/(BR#*BZ#)
180 PRINT "O COEFICIENTE BINOMIAL E ";CB#
190 INPUT "NOVAMENTE (S/N) ";S$
200 IF S$="S" THEN 50
210 IF S$="N" THEN 220
220 END
230 '
240 '  Calculo do Fatorial
250 '
260 FT#=1
270 FOR I=2 TO A
280     FT#=FT#*I
290 NEXT I
300 RETURN
```


CAPÍTULO 6

Exercício 1

```
10 '
20 '   Calculo do digito de controle do CPF
30 '
40 DIM K(11)
50 INPUT "Entre com o numero do CPF (999999999)";CPF$
60 FOR J=1 TO 9
70     K(J)=VAL(MID$(CPF$,J,1))
80 NEXT J
90 '
100 '  Calculo do primeiro digito
110 '
120 L=1
130 PMRSM=0
140 FOR J=9 TO 1 STEP -1
150     L=L+1
160     PMRSM=PMRSM+(L*K(J))
170 NEXT J
180 K(10)=11-(PMRSM-(INT(PMRSM/11)*11))
190 IF K(10) > 9 THEN K(10)=0
200 '
210 '   Calculo do segundo digito
220 '
230 L=1
240 SGDSM=0
250 FOR J=10 TO 2 STEP -1
260     L=L+1
270     SGDSM=SGDSM+(L*K(J))
280 NEXT J
290 K(11)=11-(SGDSM-(INT(SGDSM/11)*11))
300 IF K(11) > 9 THEN K(11)=0
310 '
320 '   Impressao do CPF + Digito
330 '
340 FOR J=1 TO 11
350     IF J=4 OR J=7 THEN PRINT " ";
360     IF J=10 THEN PRINT "-";
370     PRINT K(J);
380 NEXT J
390 END
```

Exercício 2

```
10 '
20 '  Este programa verifica os digitos de controle
30 '  de um codigo de ate' 10 digitos (Modulo 11)
40 '
50 INPUT "ENTRE COM O CODIGO ";CDG$
60 IF LEN(CDG$) >10 THEN 50
70 INPUT "ENTRE COM O DIGITO ";DGT$
80 IF LEN(DGT$) >1 THEN 70
90 CDG$=RIGHT$(STRING$(10,"0")+CDG$,10)
100 PESO%=2
110 FOR I%=10 TO 5 STEP -1
120     SOMA=SOMA+VAL(MID$(CDG$,I%,1))*PESO%
130     PESO%=PESO%+1
140 NEXT I%
150 PESO%=2
160 FOR I%=4 TO 1 STEP -1
170     SOMA=SOMA+VAL(MID$(CDG$,I%,1))*PESO%
180     PESO%=PESO%+1
190 NEXT I%
200 QCNT=INT(SOMA/11)
210 RST=SOMA-QCNT*11
220 IF RST=0 THEN DC=0:GOTO 250
230 IF RST=1 THEN DC=1:GOTO 250
240 DC=11-RST
250 IF VAL(DGT$) <> DC THEN PRINT "INVALIDO":END
260 PRINT "VALIDO":END
```

Exercício 3

```
10 '
20 '   Este programa imprime a matriz transposta
30 '
40 INPUT "Entre com o numero de linhas e colunas";LI,CO
50 DIM N(LI,CO)
60 FOR L=1 TO LI
70    FOR C=1 TO CO
80       INPUT N(L,C)
90    NEXT C
100 NEXT L
110 FOR C=1 TO CO
120    FOR L=1 TO LI
130       PRINT N(L,C);
140   NEXT L
150 PRINT
160 NEXT C
170 END
```


Exercício 4

```
10 '
20 '   Este programa calcula a soma de duas matrizes
30 '
40 INPUT "Entre com o numero de linhas e colunas";LI,CO
50 DIM N1(LI,CO),N2(LI,CO)
60 PRINT "Entre com os elementos da 1a. matriz "
70 FOR L=1 TO LI
80    FOR C=1 TO CO
90       INPUT N1(L,C)
100   NEXT C
110 NEXT L
120 PRINT "Entre com os elementos da 2a. matriz"
130 FOR L=1 TO LI
140    FOR C=1 TO CO
150       INPUT N2(L,C)
160    NEXT C
170 NEXT L
180 PRINT "Matriz resultante"
190 FOR L=1 TO LI
200    FOR C=1 TO CO
210       PRINT N1(L,C) + N2(L,C);
220    NEXT C
230 PRINT
240 NEXT L
250 END
```

Exercício 5

```
10 '
20 '  Este programa calcula a multiplicacao de duas
30 '  matrizes(linhas da matriz 1 = colunas da matriz 2)
40 '
50 DIM A(10,10),B(10,10)
60 INPUT "ENTRE COM A DIMENSAO DA MATRIZ 1 (L,C)";R1,C1
70 INPUT "ENTRE COM A DIMENSAO DA MATRIZ 2 (L,C)";R2,C2
80 IF C1=R2 THEN 140
90 PRINT "AS MATRIZES NAO PODEM SER MULTIPLICADAS"
100 GOTO 60
110 '
120 '  Entrada de valores
130 '
140 PRINT "MATRIZ 1 :"
150 FOR J=1 TO R1
160    PRINT "LINHA ";J
170    FOR I=1 TO C1
180        PRINT "VALOR DA COLUNA ";I;
190        INPUT A(J,I)
200    NEXT I
210 NEXT J
220 PRINT
230 PRINT "MATRIZ 2:"
240 FOR J=1 TO R2
250    PRINT "LINHA ";J
260    FOR I=1 TO C2
270        PRINT "VALOR DA COLUNA ";I;
280        INPUT B(J,I)
290    NEXT I
300 NEXT J
310 PRINT
320 '
330 '  Multiplicacao das matrizes
340 '
350 FOR I=1 TO R1
360    FOR J=1 TO C2
370        S=0
380        FOR K=1 TO C1
390          S=S+A(I,K)*B(K,J)
400        NEXT K
410        PRINT S;"  ";
420    NEXT J
430    PRINT
440 NEXT I
450 END
```


Exercício 6

```
10 '
20 '   Este programa efetua a multiplicacao escalar
30 '
40 DIM A(20)
50 INPUT "ENTRE COM A CONSTANTE ";C
60 FOR I=1 TO 20
70     INPUT A(I)
80 NEXT I
90 '
100 '  Efetuando a multiplicacao
110 '
120 FOR I=1 TO 20
130     A(I)=A(I)*C
140 NEXT I
150 '
160 '  Imprimindo a matriz resultante
170 '
180 FOR I=1 TO 20
190     PRINT A(I)
200 NEXT I
```

Exercício 7

```
10 '
20 '  Este programa calcula o desvio padrao
30 '
40 DIM A(20)
50 '
60 '  Entrada dos elementos
70 '
80 FOR I=1 TO 20
90     INPUT A(I)
100 NEXT I
110 '
120 '  Calculando a media
130 '
140 SOMA=0
150 FOR I=1 TO 20
160     SOMA=SOMA+A(I)
170 NEXT I
```

```
180 MEDIA=SOMA/20
190 '
200 '  Calculando o desvio padrao
210 '
220 SOMA=0
230 FOR I=1 TO 20
240     SOMA=SOMA + (A(I)-MEDIA)^2
250 NEXT I
260 DP=(SOMA/19)^0.5
270 PRINT "DESVIO PADRAO = ";DP
```

Exercício 8

```
10 '
20 '  Este programa calcula a Matriz Inversa
30 '
40 DIM A(10,10),B(10,10)
50 INPUT "DIMENSAO DA MATRIZ ";R
60 PRINT "ELEMENTOS DA MATRIZ"
70 FOR J=1 TO R
80     PRINT "LINHA ";J
90     FOR I=1 TO R
100        PRINT "VALOR DA COLUNA ";I;
110        INPUT A(J,I)
120    NEXT I
130    B(J,J)=1
140 NEXT J
150 '
160 '  Inversao da Matriz( Linhas 150-420)
170 '
180 FOR J=1 TO R
190    FOR I=J TO R
200       IF A(I,J)<>0 THEN 240
210    NEXT I
220    PRINT "MATRIZ SINGULAR "
230    GOTO 560
240    FOR K=1 TO R
250       S=A(J,K)
260       A(J,K)=A(I,K)
270       A(I,K)=S
280       S=B(J,K)
290       B(J,K)=B(I,K)
300       B(I,K)=S
310    NEXT K
320    T=1/A(J,J)
```



```
330    FOR K=1 TO R
340       A(J,K)=T*A(J,K)
350       B(J,K)=T*B(J,K)
360    NEXT K
370    FOR L=1 TO R
380       IF L=J THEN 440
390       T=-A(L,J)
400       FOR K=1 TO R
410          A(L,K)=A(L,K)+T*A(J,K)
420          B(L,K)=B(L,K)+T*B(J,K)
430       NEXT K
440    NEXT L
450 NEXT J
460 '
470 '  Impressao da Matriz Inversa
480 '
490 PRINT
500 FOR I=1 TO R
510    FOR J=1 TO R
520       PRINT INT(B(I,J)*1000+.5)/1000;"  ";
530    NEXT J
540    PRINT
550 NEXT I
560 END
```

Exercício 9

```
10 '
20 '   Esta rotina calcula a somatoria
30 '
40 DIM A(20)
50 FOR I=1 TO 20
60     INPUT A(I)
70 NEXT I
80 FOR I=1 TO 20
90     S=S+A(I)
100 NEXT I
110 PRINT "A SOMATORIA E' = ";S
120 END
```

Exercício 10

```
10 '
20 '  Programa gerador de numeros primos
30 '
40 DIM A(400)
50 R=1
60 A(1)=2
70 PRINT A(1);
80 FOR X=3 TO 400 STEP 2
90     FOR Y=1 TO R
100       ,IF INT(X/A(Y))*A(Y) = X THEN 150
110    NEXT Y
120    R=R+1
130    A(R)=X
140 PRINT X;
150 NEXT X
160 END
```

CAPÍTULO 7

Exercício 1

```
10 '
20 '     Este programa imprime o mes por extenso
30 '     a partir da introducao do numero do mes
40 '
50 DIM M$(12)
60 DATA Janeiro,Fevereiro,Marco,Abril,Maio,Junho,Julho
70 DATA Agosto,Setembro,Outubro,Novembro,Dezembro
80 FOR I% = 1 TO 12
90    READ M$(I%)
100 NEXT I%
110 INPUT "Entre com o numero do mes";NM%
120 PRINT M$(NM%)
130 GOTO 110
```


Exercício 2

```
10 '
20 '    Este programa imprime o dia da semana por extenso
30 '    a partir da introducao do numero do dia da semana
40 '
50 DIM S$(7)
60 DATA Segunda,Terca,Quarta,Quinta,Sexta,Sabado,Domingo
70 FOR I% = 1 TO 7
80    READ S$(I%)
90 NEXT I%
100 INPUT "Entre com o numero do dia";ND%
110 PRINT S$(ND%)
120 GOTO 100
```

Exercício 3

```
10 '
20 '   Este programa verifica se uma sigla de estado
30 '   e valida
40 '
50 DIM SIGLA$(27)
60 DATA SP,RJ,MG,PR,RS,BA,DF,PA,PB
70 DATA MS,GO,MT,SC,PE,ES,SE,AM,CE
80 DATA AL,MA,PI,AC,RO,RR,RN,FN,AP
90 FOR I%=1 TO 27
100    READ SIGLA$(I%)
110 NEXT I%
120 INPUT "Entre com a sigla a ser verificada";S$
130 FOR I%=1 TO 27
140    IF S$=SIGLA$(I%) THEN GOTO 170
150 NEXT I%
160 PRINT "Sigla invalida":GOTO 120
170 PRINT "Sigla valida":GOTO 120
```

Exercício 4

```
10 '
20 '     Este programa fornece o signo a
30 '     a partir da data de nascimento
40 '
50 DIM SIGNO$(24),LIM(24)
60 DATA Carneiro,Leao,Sagitario,Touro,Virgem,Capricornio
70 DATA Gemeos,Balanca,Aquario,Cancer,Escorpiao,Peixes
80 DATA 321,420,722,822,1122,1221,421,520,823,922
90 DATA 1222,120,521,620,923,1022,121,219,621,721
100 DATA 1023,1121,220,320
110 FOR I = 1 TO 24 STEP 2
120    READ S$
130    SIGNO$(I) = S$
140    SIGNO$(I+1) = S$
150 NEXT I
160 FOR I = 1 TO 24 STEP 2
170    READ LIM(I),LIM(I+1)
180 NEXT I
190 INPUT "Entre com o dia e mes (dd,mm)";DIA,MES
200 CHAVE = (MES * 100) + DIA
210 FOR I = 1 TO 24 STEP 2
220    IF CHAVE > LIM(I+1) THEN 250
230    IF CHAVE < LIM(I) THEN 250
240 PRINT "O signo e ";SIGNO$(I): GOTO 190
250 NEXT I
260 PRINT "Data invalida: Entre novamente": GOTO 190
```

Exercício 5

```
10 '
20 '     Este programa acha o signo no horoscopo chines
30 '     a partir do entrada do ano de nascimento
40 '
50 DIM SG$(12)
60 DATA Rato,Bufalo,Tigre,Gato,Dragao,Serpente
70 DATA Cavalo,Cabra,Macaco,Galo,Cachorro,Porco
80 FOR I%=1 TO 12
90    READ SG$(I%)
100 NEXT I%
110 INPUT "Entre com o ano de nascimento";ANO%
120 N% = ANO% - 1900
130 R% = N% - INT(N% / 12) * 12 + 1
140 PRINT "As pessoas nascidas em";ANO%;
150 PRINT "sao do signo ";SG$(R%)
160 GOTO 110
```


Exercício 6

```
10 '
20 '    Este programa calcula o ultimo dia util de
30 '    cada mes. Somente Sabado e Domingo nao sao
40 '    considerados dias uteis.
50 '
60 DEFINT A - Z
70 DIM TMES(12)
80 DATA 31,28,31,30,31,30,31,31,30,31,30,31
90 FOR I = 1 TO 12
100    READ TMES(I)
110 NEXT I
120 SCL = 19
130 INPUT "Entre com o mes e ano (mm,aa)";IMES,ANO
140 IF IMES <> 2 THEN 160
150 IF (ANO/4) <> INT(ANO/4) THEN TMES(2)=28 ELSE TMES(2)=29
160 DIA = TMES(IMES)
170 MES = IMES
180 GOSUB 270
190 ON DSEM GOTO 220  ,220  ,220  ,220  ,220  ,220  ,210
200 DUTIL = TMES(IMES) - 2: GOTO 230    'domingo
210 DUTIL = TMES(IMES) - 1: GOTO 230    'sabado
220 DUTIL = TMES(IMES)    'segunda-sexta
230 PRINT "O ultimo dia util e";DUTIL: GOTO 130
240 '
250 '   subrotina para calcular o dia da semana
260 '
270 IF MES < 3 THEN MES = MES + 12: ANO = ANO - 1
280 MES = MES - 2
290 W = INT((13 * MES) - 1) / 5)
300 X = INT(ANO / 4)
310 Y = INT(SCL / 4)
320 Z = W + X + Y + DIA + ANO - (2 * SCL)
330 DSEM = Z - (INT(Z / 7) * 7)
340 RETURN
```

Exercício 7

```
10 '
20 '    Este programa acha qual e o "bicho"
30 '
40 DIM BICHO$(25)
50 DATA Avestruz,Aguia,Burro,Borboleta,Cachorro,Cabra
60 DATA Carneiro,Camelo,Cobra,Coelho,Cavalo,Elefante
70 DATA Galo,Gato,Jacare,Leao,Macaco,Porco,Pavao,Peru
80 DATA Touro,Tigre,Urso,Veado,Vaca
90 FOR I%=1 TO 25
100   READ BICHO$(I%)
110 NEXT I%
120 INPUT "Entre com o numero desejado";N%
130 IF N%=0 THEN X%=25: GOTO 150
140 X%=INT((N%+3)/4)
150 PRINT "O numero";N%;"e ";BICHO$(X%)
160 GOTO 120
```


CAPÍTULO 8

Exercício 1

```
10 '
20 '  Este programa calcula o numero de dias
30 '  entre duas datas
40 '
50 INPUT "ENTRE COM A DATA INICIAL ";DI$
60 INPUT "ENTRE COM A DATA FINAL ";DF$
70 DIA=VAL(LEFT$(DF$,2))
80 MES=VAL(MID$(DF$,3,2))
90 ANO=VAL(RIGHT$(DF$,2))
100 GOSUB 190
110 X1=A
120 DIA=VAL(LEFT$(DI$,2))
130 MES=VAL(MID$(DI$,3,2))
140 ANO=VAL(RIGHT$(DI$,2))
150 GOSUB 190
160 NDIAS=INT(X1-A)
170 PRINT "EXISTEM ";NDIAS;" DIAS"
180 END
190 '
200 '  Rotina de calculo do numero de dias
210 '
220 IF MES=1 OR MES=2 THEN ANO=ANO-1:MES=MES+13 ELSE MES=MES+1
230 A=365.25*ANO
240 A=A+30.6*MES
250 A=A+DIA-621049
260 RETURN
```

Exercício 3

```
10 '  Este programa gera 5 numeros aleatorios entre 0 e
20 '  sem repeticao, simulando o jogo da LOTO
30 '
40 N1=INT(100*RND)
50 N2=INT(100*RND)
60 IF N2=N1 THEN 50
70 N3=INT(100*RND)
80 IF N3=N1 OR N3=N2 THEN 70
90 N4=INT(100*RND)
100 IF N4=N1 OR N4=N2 OR N4=N3 THEN 90
110 N5=INT(100*RND)
120 IF N5=N1 OR N5=N2 OR N5=N3 OR N5=N4 THEN 110
130 PRINT N1;N2;N3;N4;N5
```

Exercício 4

```
10 '
20 '     Este programa acha o dia da semana
30 '     de uma data qualquer
40 '
50 DEFINT A - Z
60 INPUT "Entre com a data (dd,mm,ss,aa)";DIA,MES,SCL,ANO
70 IF MES < 3 THEN  MES = MES + 12: ANO = ANO - 1
80 MES = MES - 2
90 W = INT(((13 * MES) - 1) / 5)
100 X = INT(ANO / 4)
110 Y = INT(SCL / 4)
120 Z = W + X + Y + DIA + ANO - (2 * SCL)
130 DSEM = Z - (INT(Z / 7) * 7)
140 ON  DSEM  GOTO 160  ,170  ,180  ,190  ,200  ,210
150 PRINT "Domingo": GOTO 60
160 PRINT "Segunda": GOTO 60
170 PRINT "Terca": GOTO 60
180 PRINT "Quarta": GOTO 60
190 PRINT "Quinta": GOTO 60
200 PRINT "Sexta": GOTO 60
210 PRINT "Sabado": GOTO 60
```


Exercício 6

```
10 '
20 '  Este programa calcula o Biorritmo
30 '
40 '  Armazenamento dos dias da semana  na tabela
50 '
60 DIM DS$(7)
70 DATA "Segunda-feira","Terca-feira","Quarta-feira"
80 DATA "Quinta-feira","Sexta-feira","Sabado","Domingo"
90 FOR I%=1 TO 7
100     READ DS$(I%)
110 NEXT I%
120 '
130 '  Entrada dos dados
140 '
150 INPUT "Entre com data de nascimento ";DN$
160 INPUT "Entre com a data do biorritmo ";DB$
170 DIA=VAL(LEFT$(DN$,2))
180 MES=VAL(MID$(DN$,3,2))
190 ANO=VAL(RIGHT$(DN$,2))
200 GOSUB 500  : NA$=DS$(DSEM)
210 GOSUB 420
220 X1=A
230 DIA=VAL(LEFT$(DB$,2))
240 MES=VAL(MID$(DB$,3,2))
250 ANO=VAL(RIGHT$(DB$,2))
260 GOSUB 500  : BI$=DS$(DSEM)
270 GOSUB 420
280 NDIAS=A-X1
290 '
300 '   Calculo  do  Biorritmo
310 '
320 CF=INT(100*SIN(2*3.1416*NDIAS/23))
330 CE=INT(100*SIN(2*3.1416*NDIAS/28))
340 CI=INT(100*SIN(2*3.1416*NDIAS/33))
350 PRINT "Dia do Nascimento  ->   ";NA$
360 PRINT "Dia do Biorritmo   ->   ";BI$
370 PRINT
380 PRINT "Ciclo Fisico       ->   ";CF;" (%) "
390 PRINT "Ciclo Emocional    ->   ";CE;" (%)  "
400 PRINT "Ciclo Intelectual  ->   ";CI;" (%) "
410 END
420 '
430 '   SuBbrotina para calculo de numero de dias
440 '
```

```
450 IF MES=1 OR MES=2 THEN ANO=ANO-1:MES=MES+13 ELSE MES=MES+1
460 A=365.25*ANO
470 A=A+30.6*MES
480 A=A+DIA-621049
490 RETURN
500 '
510 '   Subrotina para calculo do dia da semana
520 '
530 Di=DIA
540 Mi=MES
550 Ai=ANO
560 SCL=19
570 IF Mi<3 THEN Mi=Mi+12:Ai=Ai-1
580 Mi=Mi-2
590 W=INT(((13*Mi)-1)/5)
600 X=INT(Ai/4)
610 Y=INT(SCL/4)
620 Z=W+X+Y+Di+Ai - (2*SCL)
630 DSEM=Z-(INT(Z/7)*7)
640 IF INT(DSEM)=0 THEN DSEM=7
650 RETURN
```

Exercício 7

```
10 '
20 '   Este programa calcula o tempo em segundos
30 '   entre 2 instantes
40 '
50 PRINT "Forneca os dois instantes na forma hh,mm,ss"
60 INPUT "Entre com o primeiro instante";Hi,Mi,Si
70 INPUT "Entre com o segundo instante";H2,M2,S2
80 D = (H2-Hi) * 3600 + (M2-Mi) * 60 + (S2-Si)
90 PRINT "A diferenca e' de ";D;"segundos"
100 GOTO 50
```

Exercício 8

```
10 '
20 '   Este programa calcula as raizes de uma equacao
30 '   do segundo grau
40 '
50 INPUT "Entre com os valores de A,B e C";A,B,C
60 IF A = 0 THEN PRINT "nao e' de segundo grau": GOTO 50
70 DELTA = B^2 - 4*A*C
80 ON SGN(DELTA) + 2 GOTO 90   ,100  ,100
90 PRINT "nao existem raizes reais": GOTO 50
100 PRINT "x1= ";(-B + SQR(DELTA))/(2*A)
110 PRINT "x2= ";(-B - SQR(DELTA))/(2*A)
120 GOTO 50
```

CAPÍTULO 9

Exercício 1

```
10 '
20 '   Este programa verifica se um texto tem apenas
30 '   caracteres alfabeticos (A - Z) (a - z)
40 '
50 DEFINT I
60 INPUT "Entre com um texto";A$
70 FOR I = 1 TO LEN(A$)
80   IF (MID$(A$,I,1)=>"A") AND (MID$(A$,I,1)<="Z") THEN 110
90   IF (MID$(A$,I,1)=>"a") AND (MID$(A$,I,1)<="z") THEN 110
100   PRINT "O texto contem caracteres nao alfabeticos":GOTO 60
110 NEXT I
120 PRINT "O texto contem apenas caracteres alfabeticos"
130 GOTO 60
```

Exercício 2

```
10 '
20 '   Este programa verifica se um numero tem apenas
30 '   caracteres numericos (0 - 9)
40 '
50 DEFINT I
60 INPUT "Entre com um numero";A$
70 FOR I = 1 TO LEN(A$)
80   IF (MID$(A$,I,1)=>"0") AND (MID$(A$,I,1)(="9") THEN 100
90   PRINT "O texto contem caracteres nao numericos":GOTO 60
100 NEXT I
110 PRINT "O texto contem apenas caracteres numericos"
120 GOTO 60
```

Exercício 3

```
10 '
20 '   Este programa coloca ponto e virgulas em um numero
30 '
40 INPUT "Entre com o numero";ENT$
50 SAI$=""
60 ENT$=RIGHT$((STRING$(17,"0")+ENT$),17)
70 FOR I=1 TO 10 STEP 3
80    SAI$=SAI$+MID$(ENT$,I,3)+"."
90 NEXT I
100 SAI$=SAI$+MID$(ENT$,13,3)+","+MID$(ENT$,16,2)
110 FOR I=1 TO LEN(SAI$)-4
120    IF MID$(SAI$,I,1) =) "1" THEN 150
130    MID$(SAI$,I,1)=CHR$(32)
140 NEXT I
150 PRINT SAI$
160 GOTO 40
```


Exercício 4

```
10 '
20 '   Este programa calcula o valor de um telegrama
30 '
40 INPUT "Entre com a mensagem"; M$
50 X=LEN(M$)                    'calculando o numero de letras
60 VT=X*5.00                    'calculando o valor a ser pago
70 PRINT "O valor a ser pago e" ";
80 PRINT USING "#####.##";VT
90 GOTO 40
```

Exercício 5

```
10 '
20 '   Este programa calcula o custo de um telegrama.
30 '   Os 15 primeiros caracteres custam cr$ 5.00 cada
40 '   e os restantes cr$ 3.00 cada.
50 '
60 INPUT "Entre com o texto";TE$
70 CST = 5 * LEN(TE$)
80 DSCNT = (LEN(TE$) - 15) * 2
90 IF LEN(TE$) > 15 THEN CST = CST - DSCNT
100 PRINT "O custo de";LEN(TE$);"caracteres e";
110 PRINT USING "######.##";CST
120 GOTO 60
```

Exercício 6

```
10 '
20 '   Este programa imprime de tras para frente
30 '   uma palavra introduzida via teclado
40 '
50 INPUT " Entre com uma palavra";PA$
60 FOR I%=LEN(PA$) TO 1 STEP -1
70    PRINT MID$(PA$,I%,1);
80 NEXT I%
90 GOTO 50
```

Exercício 7

```
10 '
20 '   Este programa transforma letras
30 '   minusculas em maiusculas
40 '
50 INPUT "Entre com um texto";A$
60 FOR I = 1 TO LEN(A$)
70    X$ = MID$(A$,I,1)
80    IF (X$=)"a") AND (X$(="z") THEN X$ = CHR$(ASC(X$) - 32)
90    MID$(A$,I,1) = X$
100 NEXT I
110 PRINT A$
120 GOTO 50
```

Exercício 8

```
10 '
20 '   Este programa imprime a tabela ASCII
30 '   dos codigos 32 ate 126
40 '
50 INPUT "Voce quer a saida tambem na impressora (S/N)";OP$
60 FOR X = 32 TO 126
70    PRINT X;"= "; CHR$(X);
80    IF OP$ = "s" OR OP$ = "S" THEN LPRINT X;"= "; CHR$(X);
90 NEXT X
```

Exercício 9

```
10      '       Programa : EXTENSO/BAS
20      '       Funcao   : Escrever por extenso um valor introduzido
30      '       Autor    : MARCO ISHIMURA - CPD/EAESP/FGV
40      '       Data     : Outubro/84
50      '       Baseado na versao em COBOL de RUBENS PRATES
60 E$="MILHAOMILHOESCRUZEIROSCENTAVOSCENTOS"
70 A$="    UM  DOIS  TRESQUATRO CINCO  SEIS  SETE  OITO  NOVEDEZ     ONZE   "
80 B$="  DOZE    TREZE QUATORZE   QUINZEDEZESSEISDEZESSETE  DEZOITO DEZENOVE "
90 C$="VINTETRINTA  QUAR CINQU  SESS   SET   OIT   NOVENTACEM CENTO   DUZ  "
100 D$="TREZQUATRO QUINH  SEIS  SETE  OITO  NOV"
110 I$=A$+B$+C$+D$
120 DIM NUM(11)
130 INPUT"FORNECA O VALOR A SER CONVERTIDO";V$
140 VA$=RIGHT$(STRING$(11,"0")+V$,11)
150 FOR I=1 TO 11  : NUM(I)=VAL(MID$(VA$,I,1)) : NEXT I
160 G1X=VAL(LEFT$(VA$,3)):G2X=VAL(MID$(VA$,4,3))
170 G3X=VAL(MID$(VA$,7,3)):G4X=VAL(RIGHT$(VA$,2))
180 '
190 '      verifica as combinacoes existentes
200 '
210 IF G1X >0 THEN ICOND=ICOND+8
220 IF G2X >0 THEN ICOND=ICOND+4
230 IF G3X >0 THEN ICOND=ICOND+2
240 IF G4X >0 THEN ICOND=ICOND+1
250 '
260 '      rotina de processamento
270 '
280 ON ICOND GOTO 320,360,400,440,480,520,560,600,640,680,720,760,800,840,880
290 '
```

```
300 '      centavos
310 '
320 GOSUB 930:GOTO 1820
330 '
340 '      cruzeiros
350 '
360 GOSUB 990:GOTO 1820
370 '
380 '      cruzeiros e centavos
390 '
400 GOSUB 930:GOSUB 1650:GOSUB 990:GOTO 1820
410 '
420 '      mil
430 '
440 GOSUB 1690:GOSUB 1050:GOTO 1820
450 '
460 '      mil e centavos
470 '
480 GOSUB 930:GOSUB 1650:GOSUB 1690:GOSUB 1050:GOTO 1820
490 '
500 '      mil e cruzeiros
510 '
520 GOSUB 990:GOSUB 1740:GOSUB 1050:GOTO 1820
530 '
540 '      mil,cruzeiros e centavos
550 '
560 GOSUB 930:GOSUB 1650:GOSUB 990:GOSUB 1740:GOSUB 1050:GOTO 1820
570 '
580 '      milhao
590 '
600 GOSUB 1690:GOSUB 1780:GOSUB 1100:GOTO 1820
610 '
620 '      milhao e centavos
630 '
640 GOSUB 930:GOSUB 1650:GOSUB 1690:GOSUB 1780:GOSUB 1100:GOTO 1820
650 '
660 '      milhao e cruzeiros
670 '
680 GOSUB 990:GOSUB 1740:GOSUB 1100:GOTO 1820
690 '
700 '      milhao,cruzeiros e centavos
710 '
720 GOSUB 930:GOSUB 1650:GOSUB 990:GOSUB 1740:GOSUB 1100:GOTO 1820
730 '
740 '      milhao e mil
750 '
760 GOSUB 1690:GOSUB 1050:GOSUB 1740:GOSUB 1100:GOTO 1820
770 '
780 '      milhao,mil e centavos
790 '
800 GOSUB 930:GOSUB 1650:GOSUB 1050:GOSUB 1740:GOSUB 1100:GOTO 1820
810 '
820 '      milhao,mil e cruzeiros
830 '
840 GOSUB 990:GOSUB 1740:GOSUB 1050:GOSUB 1740:GOSUB 1100:GOTO 1820
850 '
860 '      milhao,mil,cruzeiros e centavos
870
880 GOSUB 930:GOSUB 1650:GOSUB 990:GOSUB 1740:GOSUB 1050:GOSUB 1740
890 GOSUB 1100:GOTO 1820
900 '
910 '      processa centavos
920 '
930 IBASE2=10:IBASE3=11:ITAB=23
```

```
940 IF G4X=1 THEN FTAB=29 ELSE FTAB=30
950 GOSUB 1550:GOSUB 1160:RETURN
960 '
970 '      processa cruzeiros
980 '
990 IBASE1=7:IBASE2=8:IBASE3=9:ITAB=14
1000 IF G3X=1 THEN FTAB=21ELSE FTAB=22
1010 GOSUB 1550:GOSUB 1340:RETURN
1020 '
1030 '      processa mil
1040 '
1050 IBASE1=4:IBASE2=5:IBASE3=6:ITAB=7:FTAB=9
1060 GOSUB 1550:GOSUB 1340:RETURN
1070 '
1080 '      processa milhao
1090 '
1100 IBASE1=1:IBASE2=2:IBASE3=3
1110 IF G1X=1 THEN ITAB=1:FTAB=6 ELSE ITAB=7:FTAB=13
1120 GOSUB 1550:GOSUB 1340:RETURN
1130 '
1140 '      processa dezena
1150 '
1160 ON NUM(IBASE2) GOTO 1200,1260,1260,1260,1260,1260,1260,1260,1260
1170 IF NUM(IBASE3)=0  THEN RETURN
1180 ITAB=6*(NUM(IBASE3)-1)+1:FTAB=ITAB+5:GOSUB 1460:RETURN
1190 '
1200 '      dezena = 1
1210 '
1220 IF NUM(IBASE3) <> 0 THEN 1240
1230 ITAB=112:FTAB=114:GOSUB 1460:RETURN
1240 ITAB=58+9*(NUM(IBASE3)-1):FTAB=ITAB+8:GOSUB 1460:RETURN
1250 '
```

```
1260 '      dezena > 1
1270 '
1280 IF NUM(IBASE3) =0 THEN 1300
1290 ITAB=6*(NUM(IBASE3)-1)+1:FTAB=ITAB+5:GOSUB 1460:GOSUB 1650
1300 IF NUM(IBASE2) <=3 THEN 1320
1310 ITAB=187:FTAB=190:GOSUB 1460
1320 ITAB=139+6*(NUM(IBASE2)-2):FTAB=ITAB+5:GOSUB 1460:RETURN
1330 '
1340 '      processa centena
1350 '
1360 GOSUB 1160:IF NUM(IBASE1)=0 THEN RETURN
1370 IF NUM(IBASE1)=1 AND NUM(IBASE2)=0 AND NUM(IBASE3)=6 THEN 1380 ELSE 1390
1380 FTAB=193:ITAB=191:GOSUB 1460:RETURN
1390 IF NUM(IBASE2) <> 0 OR NUM(IBASE3) <> 0 THEN GOSUB 1650
1400 ON NUM(IBASE1) GOTO 1410,1420,1420,1430,1420,1430,1430,1430,1430
1410 ITAB=195:FTAB=199:GOSUB 1460:RETURN
1420 ITAB=32:FTAB=36:GOSUB 1550:GOTO 1440
1430 ITAB=31:FTAB=36:GOSUB 1550
1440 ITAB=200+6*(NUM(IBASE1)-2):FTAB=ITAB+5:GOSUB 1460:RETURN
1450 '
1460 '      move da tabela I$
1470 '
1480 IF ITAB=242 THEN SAIDA$="E"+SAIDA$
1490 SAIDA$=MID$(I$,ITAB,(FTAB-ITAB)+1)+SAIDA$
1500 IF ITAB=19 OR ITAB=145 OR ITAB=104 OR ITAB=112 THEN SAIDA$=STRING$(2," ")+SAIDA$
1510 IF ITAB=55 OR ITAB=191 OR ITAB=195 OR ITAB=212 THEN SAIDA$=STRING$(2," ")+SAIDA$
1520 IF LEFT$(SAIDA$,2)="  " THEN SAIDA$=RIGHT$(SAIDA$,LEN(SAIDA$)-1) ELSE RETURN
1530 GOTO 1520
1540 '
1550 '      move da tabela E$
1560 '
1570 SAIDA$=MID$(E$,ITAB,(FTAB-ITAB)+1)+SAIDA$
```

```
1580 IF ITAB=31 OR ITAB=32  GOTO 1600
1590 SAIDA$=STRING$(2," ")+SAIDA$
1600 IF LEFT$(SAIDA$,2)="  " THEN SAIDA$=RIGHT$(SAIDA$,LEN(SAIDA$)-1) ELSE RETURN
1610 GOTO 1600
1620 '
1630 '      coloca "E"
1640 '
1650 SAIDA$=" E"+SAIDA$:RETURN
1660 '
1670 '      coloca "CRUZEIROS"
1680 '
1690 ITAB=14:FTAB=22:GOSUB 1550:RETURN
1700 '
1710 '
1720 '      coloca virgula ","
1730 '
1740 SAIDA$=","+SAIDA$:RETURN
1750 '
1760 '      coloca "DE"
1770 '
1780 SAIDA$=" DE"+SAIDA$:RETURN
1790 '
1800 '      rotina de fim
1810 '
1820 PRINT SAIDA$
1930 LPRINT:LPRINT V$ :LPRINT: LPRINT SAIDA$
1940 END
1950 SAVE"B:EXTENSO.BAS",A
```

Exercício 10

```
10 '
20 '      Este programa identifica se e letra ou numero
30 '      um caracter introduzido via teclado
40 '
50 PRINT "Entre com um caracter"
60 X$ = INKEY$
70 IF X$ = "" THEN GOTO 60
80 IF X$ => "0" AND X$ <= "9" THEN GOTO 130
90 IF X$ => "A" AND X$ <= "Z" THEN 140
100 IF X$ => "a" AND X$ <= "z" THEN 140
110 PRINT "O caracter ";X$;
120 PRINT " nao e numerico, tampouco alfabetico": GOTO 50
130 PRINT "O caracter ";X$;" e numerico": GOTO 50
140 PRINT "O caracter ";X$;" e alfabetico":GOTO 50
```

Exercício 11

```
10 '
20 '     Este programa transforma ddmmaa  em  dd/mm/aa
30 '
40 INPUT "Entre com a data no formato ddmmaa";DT$
50 PRINT LEFT$(DT$,2);"/";MID$(DT$,3,2);"/";RIGHT$(DT$,2)
60 GOTO 40
```

Exercício 13

```
10 '
20 '     Este programa transforma ddmmaa em dd mmm aa
30 '
40 DIM TMES$(12)
50 DATA " Jan "," Fev "," Mar "," Abr "," Mai "," Jun "
60 DATA " Jul "," Ago "," Set "," Out "," Nov "," Dez "
70 FOR I%=1 TO 12
80    READ TMES$(I%)
90 NEXT I%
100 INPUT "entre com a data no formato ddmmaa";DT$
110 PRINT LEFT$(DT$,2);TMES$(VAL(MID$(DT$,3,2)));
120 PRINT RIGHT$(DT$,2)
130 GOTO 100
```


Exercício 14

```
10 '
20 '  Este programa converte DDMMAA em sss,dia,mmm
30 '  sss = Dia da Semana
40 DIM DS$(7),MA$(12)
50 DATA "Seg","Ter","Quar","Qui"
60 DATA "Sex";"Sab","Dom"
70 DATA "Jan","Fev","Mar","Abr","Mai"
80 DATA "Jun","Jul","Ago","Set"
90 DATA "Out","Nov","Dez"
100 FOR I%=1 TO 7
110     READ DS$(I%)
120 NEXT I%
130 FOR I%=1 TO 12
140     READ MA$(I%)
150 NEXT I%
160 INPUT "ENTRE COM A DATA (DDMMAA) ";DT$
170 D$=LEFT$(DT$,2):DIA=VAL(D$)
180 M$=MID$(DT$,3,2):MES=VAL(M$)
190 A$=RIGHT$(DT$,2):ANO=VAL(A$)
200 GOSUB 270
210 PRINT "O DIA DA SEMANA E' ";DS$(DSEM)
220 PRINT "O MES E' ";MA$(Mi)
230 INPUT "DIGITE (S) PARA NOVA DATA ";S$
240 IF S$="S" THEN 160  ELSE 250
250 END
260 '
270 '  Rotina para calculo do dia da semana
280 '
290 Mi=MES
300 SCL=19
310 IF MES < 3 THEN MES=MES+12:ANO=ANO-1
320 MES=MES-2
330 W=INT(((13*MES)-1)/5)
340 X=INT(ANO/4)
350 Y=INT(SCL/4)
360 Z=W+X+Y+DIA+ANO - (2*SCL)
370 DSEM=Z-(INT(Z/7)*7)
380 IF INT(DSEM)=0 THEN DSEM=7 ELSE 390
390 RETURN
```

Exercício 15

```
10 '
20 '    Este programa verifica a validade de datas
30 '
40 DIM TMES(12)
50 DATA 31,28,31,30,31,30,31,31,30,31,30,31
60 FOR I% = 1 TO 12
70    READ TMES(I%)
80 NEXT I%
90 INPUT "Entre com a data no formato ddmmaa";DT$
100 IF LEN(DT$) <> 6 THEN 220
110 FOR I% = 1 TO 6
120    IF MID$(DT$,I%,1) > "9" THEN 220
130    IF MID$(DT$,I%,1) < "0" THEN 220
140 NEXT I%
150 MES% = VAL(MID$(DT$,3,2))
160 IF MES% > 12 OR MES% < 1 THEN 220
170 ANO% = VAL(MID$(DT$,5,2))
180 IF ANO% MOD 4 = 0 THEN TMES(2) = 29 ELSE TMES(2) = 28
190 DIA% = VAL(LEFT$(DT$,2))
200 IF DIA% > TMES(MES%) THEN 220
210 PRINT DT$;" e uma data valida": GOTO 90
220 PRINT DT$;" e uma data invalida": GOTO 90
```

Exercício 17

```
10 '
20 '    Este programa transforma um numero arabico
30 '    em algarismos romanos
40 '
50   INPUT "Entre com o numero desejado";N
60   IF N => 1000 THEN PRINT "M";: N = N - 1000: GOTO 60
70   IF N =>  500 THEN PRINT "D";: N = N -  500: GOTO 70
80   IF N =>  100 THEN PRINT "C";: N = N -  100: GOTO 80
90   IF N =>   50 THEN PRINT "L";: N = N -   50: GOTO 90
100  IF N =>   10 THEN PRINT "X";: N = N -   10: GOTO 100
110  IF N =>    5 THEN PRINT "V";: N = N -    5: GOTO 110
120  IF N =>    1 THEN PRINT "I";: N = N -    1: GOTO 120
130  PRINT
140  GOTO 50
```


Exercício 18

```
10 '
20 '   Este programa imprime apenas o pre-nome
30 '   de um nome completo
40 '
50 PN$ = ""
60 INPUT "Entre com o nome completo";N$
70 FOR I = 1 TO LEN(N$)
80    IF MID$(N$,I,1) = " " THEN 110
90    PN$ = PN$ + MID$(N$,I,1)
100 NEXT I
110 PRINT PN$
120 GOTO 50
```

Exercício 19

```
10 '
20 '   Este programa imprime um nome alinhando-o a direita
30 '
40 INPUT "Entre com um nome";N$
50 PRINT TAB(40-LEN(N$)) N$
60 GOTO 40
```

Exercício 20

```
10 '
20 '   Este programa imprime as iniciais de um nome
30 '
40 INPUT "Entre com o nome completo";N$
50 INI$ = LEFT$(N$,1)
60 FOR I = 2 TO LEN(N$)
70    IF MID$(N$,I,1) () " " THEN 90
80    INI$ = INI$ + MID$(N$,I+1,1)
90 NEXT I
100 PRINT INI$
110 GOTO 40
```

Exercício 21

```
10 '
20 '   Este programa transforma um numero binario
30 '   em decimal
40 '
50 VD=0
60 INPUT "Entre com o numero em binario"; A$
70 P=0
80 FOR I = LEN(A$) TO 1 STEP -1
90    VD = VD +(2^P) * VAL(MID$(A$,I,1))
100   P=P+1
110 NEXT I
120 PRINT "O valor correspondente em decimal e' "; VD
130 GOTO 50
```

Exercício 25

```
10 '
20 '   Este programa verifica se um string e' palindromo
30 '
40 INPUT X$
50 FOR I%=LEN(X$) TO 1 STEP -1
60     Y$=Y$+MID$(X$,I%,1)
70 NEXT I%
80 IF X$=Y$ THEN PRINT "PALINDROMO" : END
90 PRINT "NAO E PALINDROMO":END
```


CAPÍTULO 10

Exercício 1

```
10 '
20 '  Este programa verifica se dois arquivos sao iguais
30 '  Obs: Possui apenas finalidade demonstrativa , nao
40 '       servindo como programa de uso geral
50 '
60 INPUT "ENTRE COM O NOME DO PRIMEIRO ARQUIVO ";NA1$
70 INPUT "ENTRE COM O NOME DO SEGUNDO  ARQUIVO ";NA2$
80 OPEN "I",#1,NA1$
90 OPEN "I",#2,NA2$
100 IF EOF(1) THEN CHFIM=CHFIM+1
110 IF EOF(2) THEN CHFIM=CHFIM+2
120 ON CHFIM GOTO 170  ,170  ,160
130 LINE INPUT#1,RG1$
140 LINE INPUT#2,RG2$
150 IF RG1$()RG2$ THEN 170  ELSE 100
160 PRINT "OS ARQUIVOS SAO IGUAIS ":END
170 PRINT "OS ÀRQUIVOS SAO DIFERENTES ":END
```

CAPÍTULO 11

Exercício 1

```
10 '
20 '   Este programa inclui ou pesquisa uma ORTN
30 '
40 OPEN "R",#1,"ORTN.DAT"
50 FIELD#1,4 AS VRTN$
60 INPUT "(C)onsulta  ou  (I)nclusao";OPCAO$
70 IF OPCAO$="I" THEN 100
80 IF OPCAO$="C" THEN 160
90 GOTO 60
100 INPUT "ENTRE COM O MES E ANO";MES,ANO
110 INPUT "ENTRE COM O VALOR DA ORTN";VLR
120 REG%=MES-9+12*(ANO-64)
130 LSET VRTN$=MKS$(VLR)
140 PUT#1,REG%
150 GOTO 60
160 INPUT "ENTRE COM O MES E ANO";MES,ANO
170 REG%=MES-9+12*(ANO-64)
180 GET#1,REG%
190 PRINT "O VALOR DA ORTN E' ";CVS(VRTN$)
200 GOTO 60
```

CAPÍTULO 12

Exercício 1

Programa 1

```
10 '
20 '  Este programa le os dados da matriz A
30 '
40 INPUT "Entre com o numero de elementos";N
50 DIM A(N)
60 COMMON A(),N
70 FOR I=1 TO N
80     INPUT A(I)
90 NEXT I
100 INPUT "Digite C para continuar ";OP$
110 IF OP$<>"C" THEN 100
120 CHAIN "PROG2.BAS"
```

Programa 2

```
10 '
20 '  Este programa classifica em ordem crescente
30 '
40 COMMON A(),N
50 CHV=0
60 FOR I=1 TO N-1
70     IF A(I) <= A(I+1) THEN 120
80     CHV=1
90     WK=A(I)
100    A(I)=A(I+1)
110    A(I+1)=WK
120 NEXT I
130 IF CHV=1 THEN 50
140 FOR I=1 TO N
150    PRINT A(I);
160 NEXT I
170 PRINT
180 INPUT "Digite C para continuar";OP$
190 IF OP$ <> "C" THEN 180
200 CHAIN "PROG3.BAS",A
```

Programa 3

```
10 '
20 '  Este programa classifica em ordem
```



```
30 '  decrescente a matriz A
40 '
50 COMMON A(),N
60 CHV=0
70 FOR I=1 TO N-1
80     IF A(I) => A(I+1) THEN 130
90     CHV=1
100    WK=A(I)
110    A(I)=A(I+1)
120    A(I+1)=WK
130 NEXT I
140 IF CHV=1 THEN 60
150 FOR I=1 TO N
160    PRINT A(I);
170 NEXT I
180 PRINT
190 INPUT "Digite C para continuar";OP$
200 IF OP$ <> "C" THEN 190
210 CHAIN "PROG4.BAS"
```

Programa 4

```
10 '
20 '  Este programa calcula a media aritmetica
30 '
40 FOR I=1 TO N
50     SOMA=SOMA + A(I)
60 NEXT I
70 MED=SOMA/N
80 PRINT "Media = ";MED
90 INPUT "Digite C para continuar";OP$
100 IF OP$ <> "C"THEN 90
110 CHAIN "PROG1.BAS"
```

Exercício 2

Programa 1

```
10 '
20 '  Este programa le os dados da matriz A
30 '
40 INPUT "Entre com o numero de elementos";N
50 DIM A(N)
60 COMMON A(),N
70 FOR I=1 TO N
80     INPUT A(I)
90 NEXT I
```

```
100 INPUT "Digite C para continuar ";OP$
110 IF OP$<>"C" THEN 100
120 CHAIN "PROG2X.BAS"
```

Programa 2

```
10 '
20 '  Este programa calcula o desvio padrao
30 '
40 COMMON A(),N
50 FOR I=1 TO N
60     SOMA=SOMA+A(I)
70 NEXT I
80 MED=SOMA/N
90 SOMA=0
100 FOR I=1 TO N
110    SOMA=SOMA+(A(I)-MED)^2
120 NEXT I
130 DP=SQR(SOMA/19)
140 PRINT "Desvio padrao = ";DP
150 INPUT "Digite C para continuar ";OP$
160 IF OP$<> "C" THEN 150
170 CHAIN "PROG3X.BAS"
```

Programa 3

```
10 '
20 '  Este programa calcula a mediana
30 '
40 IF N/2 <> INT(N/2) THEN 110
50 '
60 '  Numero de elementos e par
70 '
80 MEDIANA=(A(N/2)+A((N/2)+1))/2
90 GOTO 140
100 '
110 '  Numero de elementos e impar
120 '
130 MEDIANA=A((N+1)/2)
140 PRINT "Mediana = ";MEDIANA
150 INPUT "Digite C para continuar";OP$
160 IF OP$ <> "C" THEN 150
170 CHAIN "PROG1X.BAS"
```

GLOSSÁRIO

ALEATÓRIO Sujeito ao acaso.

ALFANUMÉRICO Caracteres alfanuméricos, numéricos e símbolos especiais combinados, que são reconhecidos pelo computador.

ALGORITMO Seqüência de procedimentos a serem efetuados para se obter um resultado num número finito de etapas.

ARQUIVO Conjunto organizado de dados armazenados e relacionados entre si.

ARQUIVO SEQÜENCIAL Arquivo no qual os dados só se tornam disponíveis após a leitura do dado imediatamente anterior.

ARQUIVO RANDÔMICO Arquivo no qual todos os dados estão disponíveis a qualquer momento e em qualquer ordem.

ASCII American Standard Code for Information Interchange.
Código padrão americano para a representação de caracteres alfabéticos, numéricos e especiais num computador e intercâmbio de informações entre diferentes equipamentos.

ASSEMBLER Linguagem de programação formada por declarações simbólicas mnemônicas, similar a linguagem de máquina.

BACKUP Cópia de segurança.

BACKSPACE Retrocesso.

BASIC Beginner's All-purpose Symbolic Instruction Code. Uma das muitas linguagens de programação. Originalmente criada para ensinar programação.

BIBLIOTECA Conjunto de sub-rotinas freqüentemente usadas.

BIT Menor unidade de informação. Qualquer dos numerais binários 0 ou 1.

BUFFER Memória intermediária usada para compatibilizar velocidades de transferência de dados entre os componentes de um sistema.

BYTE Conjunto de 8 bits.

CARRIAGE RETURN Retorno de carro.

CLEAR SCREEN Limpar o video.

COMANDO Ordem dada pelo usuário que causa a execução de alguma operação pré-definida.

COMENTÁRIO Observações introduzidas no programa para torná-lo mais facilmente entendível.

COMPILADOR Programa tradutor que converte um programa fonte em um programa em linguagem de máquina.

CONCATENAÇÃO União de strings ou arquivos para formar um novo.

CP/M Control Program for Microcomputers.
Programa de Controle para Microcomputadores. Sistema operacional para micros mais utilizado no mundo.

CRIPTOGRAFIA Arte de escrever em código.

CURSOR Caracter luminoso que indica no video a posição de entrada do próximo caracter.

DECLARAÇÃO Ordem para o computador dizendo o que é para ser feito. O mesmo que instrução.

DEFAULT Valor assumido pelo computador quando não fornecído pelo usuário.

DEPURAÇÃO Localizar e eliminar os erros do programa.

DIAGRAMA DE BLOCOS Representação simbólica dos passos a serem seguidos em um programa. O mesmo que fluxograma.

DICIONÁRIO DE VARIÁVEIS Relação das variáveis utilizadas no programa com seus respectivos significados.

DISQUETE Pequeno disco flexível de poliester dentro de um envelope plástico. Existem três tamanhos básicos: 3, 5.25 e 8 polegadas.

DIRETÓRIO Índice dos arquivos do disquete.

DRIVE Compartimento onde é inserido o disquete.

EDIÇÃO Processo de modificar ou criar programas e arquivos.

ENDEREÇO Uma localização na memória onde um byte é armazenado.

EOF Indicação de fim de arquivo. (End of file).

EXTENSÃO Complemento ao nome de arquivo.

FLUXOGRAMA Representação simbólica dos passos a serem seguidos em um programa. O mesmo que diagrama de blocos.

FUNÇÃO DE USUÁRIO Função definida pelo usuário, através da declaração DEF FN.

HEXADECIMAL. Sistema de numeração na qual a base é 16.

HOBISTA Pessoa que desenvolve uma atividade nas horas de lazer, sem compromisso profissional.

INSTRUÇÃO Ordem para o computador dizendo o que é para ser feito. O mesmo que declaração.

INTERATIVIDADE Capacidade de conversação nos dois sentidos entre o usuário e o computador.

INTERPRETADOR Programa que converte uma instrução em linguagem de alto nível em linguagem de máquina e a executa.

K 1024 bytes.

LAÇO Seqüência de instruções que é executada repetidamente até que ocorra uma condição. O mesmo que loop.

LAÇO ANINHADO Um laço dentro de outro laço.

LINE FEED Operação que move o cursor para a próxima linha.

LINGUAGEM DE ALTO NÍVEL Linguagem em que cada instrução corresponde a várias instruções em linguagem de máquina.

LINGUAGEM DE MÁQUINA Linguagem usada diretamente pelo microcomputador.

LINKEDITOR Programa que carrega e faz as ligações entre arquivos relocáveis, módulos de biblioteca e rotinas em Assembler necessários para criar e executar um arquivo em linguagem de máquina.

MATRIZ Conjunto de dados organizados segundo uma seqüência determinada.

MBASIC Interpretador BASIC desenvolvido pela MICROSOFT.

MEMÓRIA Parte do computador que armazena informações.

MENU A apresentação de um conjunto de opções a serem escolhidas, tipicamente exibidas no video.

MICROPROCESSADOR Circuito integrado que contém os principais elementos da unidade central de processamento.

MICROSOFT Empresa (software house) americana que desenvolveu uma versão muito utilizada do BASIC.

NULO String sem conteúdo.

OCTAL Sistema de numeração na qual a base é 8.

OTIMIZAR Melhorar o desempenho.

OVERFLOW Condição causada por operações matemáticas cujo resultado excede a capacidade do computador.

OVERLAY Técnica de usar seguidamente as mesmas áreas da memória em estágios diferentes de um programa.

PALAVRAS RESERVADAS Instruções, funções ou operadores de uso exclusivo da linguagem BASIC.

PROGRAMA Seqüência de instruções escritas numa determinada linguagem, que descreve o que o computador deve executar.

PROGRAMA FONTE Programa escrito em linguagem de alto nível.
O mesmo que código fonte ou fonte.

PROGRAMA OBJETO Programa resultante da compilação.

PROGRAMAÇÃO MODULAR Técnica para projetar programas como um conjunto de unidades inter-relacionadas (módulos) que podem mais tarde ser agrupados para formar um programa completo.

REGISTRO Conjunto de campos que contém dados.

ROTINA Conjunto completo de instruções que executa uma parte específica do programa.

SCROLL Movimentação do texto na tela para liberar espaço para mais uma linha de texto.

SISTEMA OPERACIONAL Programa especial que controla e coordena todas as operações de um computador.

SOFTWARE Programas, rotinas, sistemas operacionais, procedimentos e documentação relativa a operação de um sistema de computação.

STRING Seqüência de letras, números ou outro caracter.

UTILITÁRIO Programa que facilita a operação do computador.

Impresso na
ERCA Editora e Gráfica Ltda.
Rua Silva Vale, 870 - Cavalcante
Rio de Janeiro - RJ

DATA CENTER – Coloca a sua disposição, sua experiência didática e instalações nos seguintes endereços:

São Paulo:

- R. Cardeal Arcoverde, 2.903 – Pinheiros – São Paulo
  Tel.: (011) 211-2048
- R. Comendador Cantinho, 101 – Penha – São Paulo
  Tel.: (011) 941-4823
- Av. Cruzeiro do Sul, 2.611 – Santana – São Paulo
  Tel.: (011) 299-0327
- R. Desembargador Elizeu Guilherme, 101 – Loja B – Paraíso – São Paulo
  Tel.: (011) 251-2904
- R. General Glicério, 412 – Centro – Santo André
  Tel.: (011) 449-2689
- R. Mal. Deodoro, 561/1º andar – Centro – São Bernardo do Campo
  Tel.: (011) 452-3222

Rio de Janeiro:

- Av. Churchill, 94/4º andar – Centro – Rio de Janeiro
  Tel.: (021) 240-2375
- R. Dagmar da Fonseca, 191 – Madureira – Rio de Janeiro
  Tel.: (021) 350-5163
- R. Maria Adelaide de Carvalho, 51 – Centro – Nova Iguaçu
- R. Maestro Felicio Toledo, 489 – Centro – Niterói
  Tel.: (021) 719-6764

Paraná:

- R. Barão do Rio Branco, 344 – Centro – Curitiba
  Tel.: (041) 225-1728
- R. Saldanha Marinho, 636 – Centro – Curitiba
  Tel.: (041) 233-6561
- R. Brasil, 742 – Centro – Londrina
  Tel.: (0432) 223-4828

Minas Gerais:

- R. Guajajaras, 536 – Centro – Belo Horizonte
  Tel.: (031) 226-0797

Pernambuco:

- Av. Dantas Barreto, 507/4º andar – Centro – Recife
  Tel.: (081) 224-9620

Ceará:

- R. Barão do Rio Branco, 750 – Centro – Fortaleza
  Tel.: (085) 231-0537
- R. Castro Silva, 121 – Centro – Fortaleza

Bahia:

- R. da Graça, 149 – Graça – Salvador
  Tel.: (071) 237-6392
- Av. D. João VI, 7 – Brotas – Salvador
  Tel.: (071) 233-1949

Distrito Federal:

- SCRN – Quadra 502 – Bloco B, 59/1º andar – Brasília
  Tel.: (061) 226-1843

Rio Grande do Norte:

- Av. Rio Branco, 767 – Centro – Natal
  Tel.: (084) 222-5729

Alagoas:

- R. Pedro Monteiro, 226 – Centro – Maceió
  Tel.: (082) 223-5556

Espírito Santo:

- R. General Osório, 127/1º andar – Centro – Vitória
  Tel.: (027) 222-5614