O objectivo deste guia é, antes de mais, fornecer uma informação essencial sobre a linguagem BASIC, a mais generalizada tanto em sistemas de microcomputadores como em sistemas de maiores dimensões. Ilustrado com numerosos exemplos e fluxogramas, explica de uma forma perfeitamente acessível ao principiante o que é a linguagem BASIC e quais os passos necessários para a resolução dos problemas de programação.

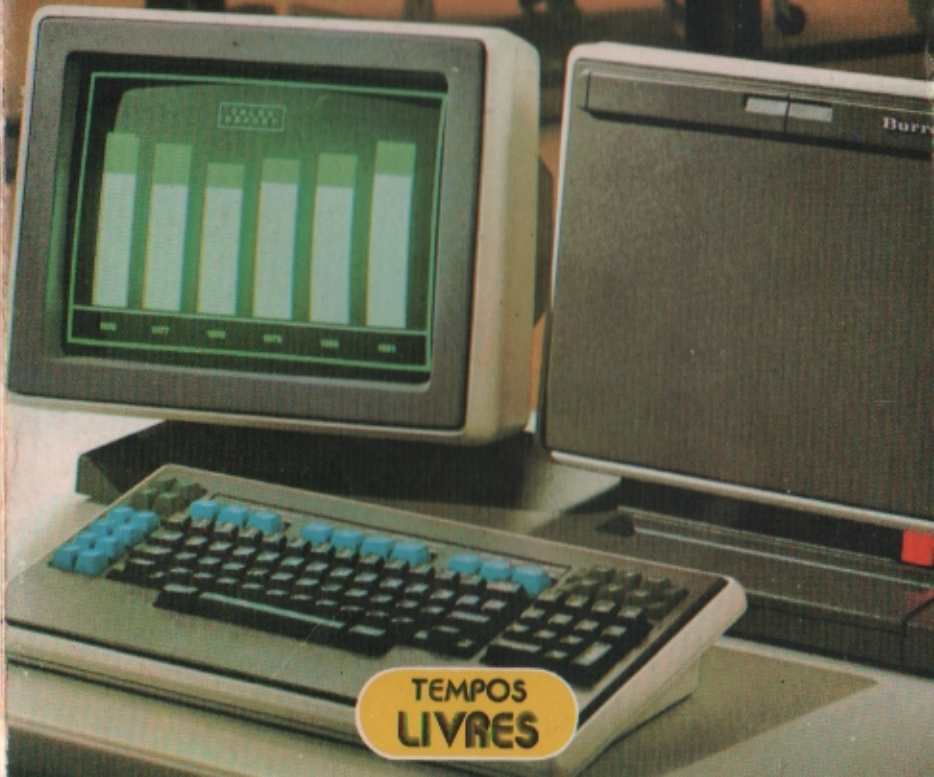
EDITORIAL PRESENÇA / MARTINS FONTES

Guia Prático de Basic

33

Roger Hunt

GUIA PRÁTICO DE BASIC



TEMPOS LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Petar Trifunovitch* e *Sava Vukovitch*
4. ABC do Bridge, *Pierre Jais* e *H. Lahana*
5. Guia Prático de Fotografia, *W. D. Emanuel*
6. ABC do Judo, *E. J. Harrison*
7. Como Fazer Cinema, *Paul Petzold*
8. Bridge Moderno *Pierre Jais* e *H. Lahana*
9. Fotografia — Técnicas e Truques, *Edwin Smith*
10. ABC dos Estilos – da Arquitectura ao Mobiliário, *A. Aussel*
11. Fotografia — Técnicas e Truques, *Edwin Smith*
12. A Pesca Submarina, *António Ribera*
13. Teoria dos Finais de Partida, *Yuri Averbach*
14. Aprenda Rádio, *B. Fighiera*
15. Guia do Cão, *Louise Laliberté-Robert* e *Jean-Pierre Robert*
16. ABC do Aquário, *Anthony Evans*
17. Iniciação à Electricidade e Electrónica, *Fernand Huré*
18. Os Transístores, *Fernand Huré*
19. Karaté, I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricone*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté, II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *Curt Af Enehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez — Teoria do Meio Jogo I, *Bondarevsky*
35. Manual do Super 8, I, *Myron A. Matzkin*
36. ABC da Criação de Periquitos, *Cyril H. Rogers*
37. O Livro dos Gatos, *Barbel Gerber* e *Horst Bielfeld*
38. Manual do Super 8, II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. Witham Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Cocker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabian Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*
51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires* e *C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelhos, *P. Surdeau* e *R. Henaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen* e *Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Böelle* e *B. Doyen*
63. Enxerto de Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *L. A. Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*
67. Doces e Conservas de Frutas, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Roy Genders*
70. Calendário do Horticultor, *Luis Alsina Grau*
71. Jogos Electrónicos, *F. Rayer*
72. Cultivo de Cogumelos e Trufas, *A. Rigau*
73. Aprenda Televisão a Cores, *G. J. King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Pode de Árvores e Arbustos, *R. Genders*
76. Como Treinar o Seu Cão, *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, *Heinrich Stöckle*
78. A Criação de Caracóis, *Matías Josa*
79. Rádio – Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébault*
81. Iniciação à Jardinagem, *Noel Clarasó*
82. A Congelação dos Alimentos, *S. Lapointe*
83. Windsurf – Prancha à Vela, *E. Prade*
84. Raças de Cães, *O. Hasselfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia – Embalsamamento de Pássaros e Mamíferos, *Harry Hjortaa*
89. Jogging – Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *S. Richmond*
91. Jogos T.V., *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie Prémont* e *Nicole Philippi*
95. O Minitrampolim na Escola, *Sonja Helmer Jensen* e *Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, *B. Fighiera*
97. O Cultivo do Tomate, *Louis N. Flawn*
98. Pilhas Solares, *F. Juster*
99. Criação Doméstica de Coelhos, *C. F. Snow*
100. Iniciação ao Futebol, *W. Männle* e *H. Arnold*
101. Horóscopos Chineses, *G. Haddenbach*
102. Guia Prático de Marcenaria, *C. H. Hayward*
103. Andebol, *Fritz* e *Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perus, Pintadas e Codornizes, *J. Sauze*
106. Crepes, Doces e Salgados, *F. Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *Rick Abbot* e *Mike Baker*
110. Futebol – Técnica e Táctica, *Kurt Laval*
111. A Vaca Leiteira, *C. T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*
113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marie A. Rank*
115. O Cubo para quem já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, a Torre e o Barril do Diabo, *M. Mrowka* e *W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung-Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hanns-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hill*
121. Construção de um Alimentador de Corrente, *Waldemar Baitinger*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *Christa-Maria* e *Richard Kerler*
126. A Pesca e os seus Segredos, *Armand Deschamps*
127. O Osciloscópio, *R. Rateau*
128. Guia Prático da Banda do Cidadão, *J. M. Normand*
129. Sumos e Batidos, *Manfred Donderski*
130. Introdução à Programação de Microcomputadores, *Peter C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*
133. Guia Prático de Basic, *Roger Hunt*
134. Introdução à Electrónica Digital, *Ian Sinclair*
135. ABC do Vídeo, *David K. Matthewson*

136. Fotografia em Movimento, *Don Morley*
137. Guia de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *Sidney F. Ray*
139. Guia Moderno da Canaricultura, *Manuel Gonçalves*
140. Minielectrónica para Amadores, *Heinz Richter*
141. ABC da Programação de Computadores, *John Shelley*
142. Tarot — O Futuro Pelas Cartas, *Edwin J. Nigg*
143. ABC da Equitação, *Dorothy Johnson*
144. Como Programar o seu ZX 81, *Patrick Gueulle*
145. 100 Avarias TV e a Maneira Prática de as Detectar, *P. Duranton*
146. ABC da Horticultura, *Louis Giordano*
147. Basic Para Microcomputadores, *A. P. Stephenson*
148. Como Programar o Seu ZX Spectrum, *Tim Hartnell* e *Dilwyn Jones*
149. Iniciação aos Motores Diesel, *David S. Maclean*
150. 60 jogos para o ZX Spectrum, *David Harwod*
151. As Linhas da Mão, *Rose Hubert*
152. Cozinha Italiana, *Rotrand Degner*
153. Manual do ZX Spectrum, *Simpson* e *Terrel*
154. Z80 Assembler para o ZX Spectrum – Iniciação ao Código de Máquina, *João Paulo Fragoso*
155. Aeróbica, *H. Schulz*
156. ABC do Atletismo, *Denis Watts*
157. 26 Programas Basic para Microcomputadores, *Derrick Daines*
158. Aprenda Pascal no seu Microcomputador, *Jeremy Ruston*
159. Guia Moderno da Suinicultura, *Colin Whittemore*
160. O Bar em Sua Casa – 888 Cocktails, *Aladar von Wesendonk*
161. Código de Máquina Para Principiantes, *James Walsh*
162. Código de Máquina Para Programadores Avançados, *Paul Holmes*
163. ABC da Fruticultura, *Henri Gosselin*
164. ABC da Canoagem, *Alan Byde*
165. Guia de Fortran, *Philip Ridler*
166. Manual da Secretária, *Philippa Ramage*
167. ABC das Antenas, *Gordon J. King*
168. Programar Aventuras no Seu Computador, *Andrew Nelson*
169. Guia do Sinclair QL, *Boris Allan*
170. Novas Aventuras no Seu ZX Spectrum, Peter Shaw e *James Mortleman*
171. O Computador no Escritório, *John Shelley*
172. Sobremesas, *Fred Timber*
173. Rádio do Circuito Oscilante ao Receptor de Ondas Curtas, *Richard Zierl*
174. Xadrez: ABC das Aberturas, *V. N. Panore*
175. A Construção de Pequenos Transformadores, *M. Dourian* e *F. Juster*
176. Guia de Pascal, *David Watt*
177. O ZX Spectrum na Educação, *Tim Hastnel, C. Johnson* e *D. Valentine*
178. Iniciação ao Snooker, *John Pulman*
179. Circuitos com Diacs, Tiristores e Triacs, *Fritz Bergtold*

ROGER HUNT

GUIA PRÁTICO DE BASIC

2.ª edição

EDITORIAL PRESENÇA • LIVRARIA MARTINS FONTES
PORTUGAL BRASIL

Título original
POCKET GUIDE TO BASIC

Tradução de João Torres
Fotografia da capa gentilmente cedida
pela Burroughs Corporation

EDITORIAL PRESENÇA, LDA.
Rua Augusto Gil, 35-A
1000 LISBOA

Como utilizar este guia prático

O objectivo deste guia é fornecer uma introdução à programação em BASIC. Assume-se que o leitor não tem experiência em trabalhar com computadores. O guia começa com um exemplo elaborado para dar alguma indicação sobre a estrutura da linguagem antes de serem abordadas em pormenor as suas características. Inclui-se um fluxograma para mostrar claramente a sequência de passos que conduzem a uma solução e de estabelecer alguma coisa sobre o nível de raciocínio necessário à preparação de problemas para serem resolvidos num computador.

Seguidamente são fornecidas as definições das partes que compõem a linguagem de programação BASIC, como forma de introdução à linguagem, e para dar maior significado às descrições que se seguem dos vários comandos BASIC. Não se espera que esta secção seja inteiramente compreendida ou apreciada numa primeira leitura. À medida que for avançando no texto, ir-se-á familiarizando com as definições e terminologia.

Os comandos BASIC ou palavras-chave que constituem o vocabulário da linguagem são por sua vez introduzidos, acompanhados de numerosos exemplos ilustrativos. Os exemplos dados destinam-se essencialmente a demonstrar determinados aspectos particulares mais do que a resolver problemas. Frequentemente, fornece-se um pequeno exemplo antes de ser formalmente introduzida uma nova característica com o objectivo de encorajar o leitor a ter em conta a utilidade dessa característica, antes de examinar as suas regras gramaticais.

A ordem pela qual as palavras-chave e as declarações BASIC são introduzidas não é alfabética nem aleatória. As declarações foram agrupadas de acordo com a sua finalidade e introduzidas pela ordem seguinte:

LET, PRINT e END: para permitirem a escrita de programas simples que possam ser testados de imediato.

REM: para dar legibilidade aos programas.

READ, DATA, RESTORE e INPUT: para atribuição de dados.

GO TO, IF... THEN, FOR-NEXT, STOP, ON... GO TO, GO SUB e RETURN: para controlarem o fluxo do programa.

DIM, OPTION BASE: para permitirem o tratamento mais fácil de conjuntos de dados pela utilização de quadros.

Funções, RANDOMIZE, TAB e DEF: algumas características especiais que permitem levar a cabo tarefas de programação úteis.

Declarações de ficheiros: para permitirem salvaguardar e recuperar dados em ficheiros externos ao programa.

MAT: para permitir a manipulação de matrizes. Note-se que estes comandos não existem em todas as implementações de BASIC.

O guia termina com um certo número de exemplos, incluídos sem mais explicações, para ilustrar ainda tudo o que antes foi dito.

Introdução

Uma linguagem chamada BASIC

BASIC é uma linguagem de programação, isto é, uma linguagem ou conjunto de regras destinadas a permitir os programadores exprimirem problemas de forma a poderem ser resolvidos pela utilização de um computador. A palavra BASIC é um acrónimo para *Beginners All-purpose Symbolic Instruction Code,* ou seja, Código simbólico de instruções para todos os fins para principiantes. A linguagem é estruturalmente simples e possui um vocabulário reduzido, sendo por isso ideal para iniciar os utilizadores de sistemas de computadores. É, além disso, suficientemente poderosa para satisfazer um amplo leque de aplicações. A utilização de notação matemática normalizada torna-se adequada à resolução de problemas de tipo matemático, assim como a facilidade de manipulação de caracteres demonstra que é uma linguagem igualmente bem adequada ao tratamento de informação textual. A existência generalizada de BASIC tanto em sistemas de microcomputadores como em sistemas de maiores dimensões é um facto adicional em seu favor.

Existem numerosas versões ou implementações de BASIC assim como muitas diferenças entre os vários dialectos da linguagem. Este guia baseia-se na definição normalizada ECMA-55, geralmente conhecida por "BASIC Mínimo". É de esperar que quase tudo, se não mesmo tudo, o que é dito neste guia seja consistente com qualquer versão da linguagem que venha a utilizar. No pior dos casos, teria de desaprender muito pouco.

Um programa-exemplo

Problema: Determinar o número de estudantes que passaram num exame, contando igualmente o número de chumbos. Os dados fornecidos são uma nota limite para passa-

gem no exame e uma lista contendo um número não especificado de notas de alunos.

A arte da programação consiste em se ser capaz de desmontar o problema que vai ser submetido ao computador de uma forma suficientemente precisa para se obter a solução. Lembre-se que tudo o que desejar que o computador faça deve ser previamente concebido e expresso sob a forma de uma sequência de acontecimentos. Todos os acontecimentos ou passos necessários à resolução devem encontrar-se presentes, e escritos na ordem correcta. Durante a aprendizagem, os problemas que resolverá são geralmente tão simples que normalmente nunca utilizaria um computador para os resolver. São directos e exigem pouco ou nenhum planeamento. Contudo, quanto mais complexo for o problema mais importante será que esta parte do exercício seja bem feita.

O método de fazer um fluxograma do problema é muitas vezes utilizado pelos programadores como auxiliar na fase de planeamento. O fluxograma apresentado na fig. 1 demonstra como se pode determinar o número de alunos que passam ou chumbam no exame.

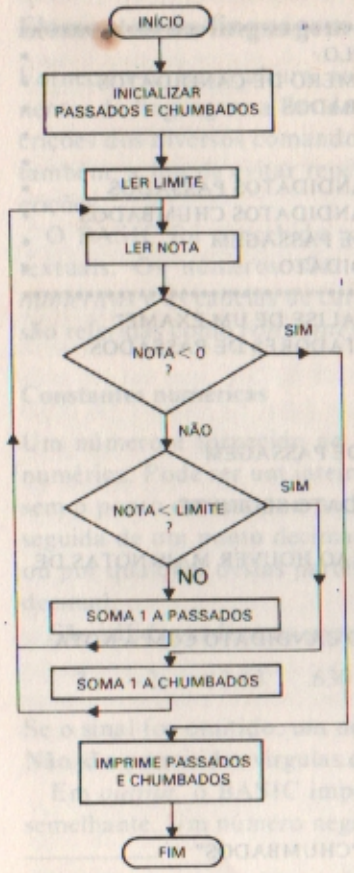


Fig. 1

Notas

1. Uma acção representa-se geralmente dentro de um bloco rectangular.
2. Quando se faz uma pergunta, ou seja, quando é preciso tomar uma decisão, utiliza-se um bloco losangonal de onde saem duas linhas, uma para SIM e outra para NÃO. É costume escrever as palavras SIM e NÃO, ou as letras S e N, junto das linhas. Qualquer dos cantos do losango pode ser utilizado para o SIM.
3. O início e o fim do programa indicam-se dentro de símbolos de forma especial.
4. Utilizam-se setas para indicar o sentido do fluxo do programa.
5. Um fluxograma ajuda a expor a lógica de uma forma simples e a desenvolver uma solução passo a passo. É considerado como mais simples de compreender à primeira vista que uma descrição narrativa. Um fluxograma fornece um modelo por onde se podem fazer passar dados de teste a fim de verificar a correcção da lógica utilizada.
6. Repare-se que, no nosso exemplo, PASSADOS e CHUMBADOS são colocados a zero antes de se iniciar a sequência repetitiva. Eles são nomes simbólicos utilizados para representar somas acumuladas devendo, portanto, encontrar-se inicialmente vazios antes que lhes seja somado alguma coisa.

Eis a nossa solução para o problema dado, escrita sob a forma de um programa em BASIC. O *output* do programa é fornecido após a listagem do programa.

Programa

```
010 REM ************************************************
020 REM * PROGRAMA EXEMPLO                              *
030 REM * DETERMINAR O NUMERO DE CANDIDATOS             *
035 REM * PASSADOS E CHUMBADOS                          *
040 REM * NUM EXAME                                     *
050 REM *                                               *
060 REM * P — NUMERO DE CANDIDATOS PASSADOS             *
070 REM * C — NUMERO DE CANDIDATOS CHUMBADOS            *
080 REM * X — NOTA LIMITE DE PASSAGEM                   *
090 REM * N — NOTA DO CANDIDATO                         *
100 REM ************************************************
110 PRINT "PROGRAMA DE ANALISE DE UM EXAME"
120 REM  INICIALIZAR OS CONTADORES DE PASSADOS
125 REM  E CHUMBADOS
130 LET P = 0
140 LET C = 0
150 REM  LER A NOTA LIMITE DE PASSAGEM
160 READ X
170 REM  LER NOTA DO CANDIDATO SEGUINTE
180 READ N
190 REM  SAIDA DO CICLO SE NAO HOUVER MAIS NOTAS DE
195 REM  CANDIDATOS
200 IF N < O THEN 280
210 REM  COMPARA A NOTA DO CANDIDATO COM A NOTA
215 REM  LIMITE
220 IF N < X THEN 250
230 P = P + 1
240 GO TO 180
250 C = C + 1
260 GO TO 180
270 REM  FIM DO CICLO
280 PRINT P; "PASSADOS E"; C; "CHUMBADOS"
290 DATA 10
300 DATA 12, 16, 8, 14, 13, 9
310 DATA 4, 11, 13, 15, 7
320 DATA -1
330 END
```


Output

```
PROGRAMA DE ANALISE DE UM EXAME
 7 PASSADOS E 4 CHUMBADOS
```

Elementos da linguagem

Fornecem-se, nesta altura, as definições das partes componentes da linguagem a fim de dar maior significado às descrições dos diversos comandos de BASIC que se seguem e, também, a fim de evitar repetições desnecessárias nas descrições.

O BASIC foi concebido para tratar dados numéricos e textuais. Os números são conhecidos como *constantes numéricas* e as cadeias de caracteres que formam um texto são referidas como *constantes de cadeia*.

Constantes numéricas

Um número é fornecido ao BASIC como uma constante numérica. Pode ser um inteiro, ou seja, um número escrito sem o ponto decimal, ou pode consistir numa parte inteira seguida de um ponto decimal e de uma parte fraccionária, ou por qualquer destas partes isoladamente com um ponto decimal.

São válidas as formas seguintes:

7 -2. 12.43 .630 -0.07305 +2106418

Se o sinal for omitido, um número é considerado positivo. Não são permitidas vírgulas dentro de um número.[1]

Em *output*, o BASIC imprime[2] números de uma forma semelhante. Um número negativo é precedido por um sinal

[1] Nos países anglo-saxónicos são utilizadas vírgulas, em vez de pontos, para assinalar as posições dos milhares, milhões, etc. Por sua vez, é utilizado um ponto decimal em vez da vírgula decimal (*N. T.*).

[2] Ao longo do livro e o livro adoptou-se o verbo *imprimir* quer o *output* se faça para uma impressora quer para um terminal vídeo (*N. T.*).

menos e um número positivo é precedido por um espaço, isto é, um espaço em vez do sinal mais. O BASIC também imprime automaticamente números maiores ou menores que a dimensão especificada na implementação através de uma representação que utiliza um expoente. A notação utilizada, onde E representa a base 10, parece-se com a notação científica. Por exemplo:

3.50000E+9	em vez de	3500000000
9.00000E-6	"	.000009
-1.92000E-5	"	-0.0000192

Os números podem igualmente ser introduzidos como *input* utilizando a forma exponencial.

Os valores máximo e mínimo que podem ser tratados, assim como a precisão dos dígitos guardados em memória, dependem da implementação de BASIC que vai ser utilizada. No BASIC *Mínimo*, este leque de valores vai desde 1E-38 a 1E+38 e uma precisão não inferior a 6 dígitos decimais.

Constantes de cadeia

Uma cadeia é fornecida ao BASIC como uma sequência de caracteres dentro de aspas. Qualquer carácter de impressão pode surgir numa cadeia de caracteres com excepção das próprias aspas, dado que as aspas fecham automaticamente a cadeia. Algumas implementações permitem a utilização de plicas dentro de uma cadeia. Note-se que, numa cadeia, um espaço é um carácter significativo.

São exemplos de constantes de cadeia:

"OUTROSSIM, JOAQUIM"
"1 DE OUTUBRO DE 1980"
" " (uma cadeia vazia ou nula)

Notar que embora uma cadeia possa conter caracteres

numéricos não podem ser feitas operações aritméticas com dados guardados em memória como cadeias.

O BASIC permite ainda definir uma cadeia sem a utilização de aspas numa declaração DATA e como resposta digitada face a um pedido de INPUT de dados de cadeia.

Variáveis numéricas

Variáveis são nomes ou identificadores utilizados para referenciar posições de memória do computador. Pode-se atribuir um valor a uma variável numérica e seguidamente modificá-lo quantas vezes se desejar. Uma variável simples é identificada por um único carácter alfabético ou por um carácter alfabético seguido por um único dígito. São exemplos de nomes numéricos:

D P X1 N7

Notar que, em algumas implementações, as variáveis numéricas são colocadas a zero antes da execução do programa, de forma que uma variável a que não seja atribuído um valor antes de ser utilizada é automaticamente equivalente a zero. Dado que esse tipo de inicialização não se encontra normalizado em todos os sistemas, recomendamos fortemente que se atribua sempre um valor às variáveis numéricas antes de serem utilizadas. Notar ainda que algumas implementações permitem a utilização de nomes com duas letras.

Variáveis de cadeia

As variáveis de cadeia são utilizadas para identificar posições de memória que contêm cadeias de caracteres. Uma variável de cadeia simples identifica-se por dois caracteres: um carácter alfabético seguido por cifrão ($). Exemplos de variáveis de cadeia:

A$ Q$ C$ Y$

O número de caracteres que é permitido numa cadeia associada a uma variável depende da implementação de BASIC. O *Mínimo* define um comprimento máximo de pelo menos dezoito caracteres.

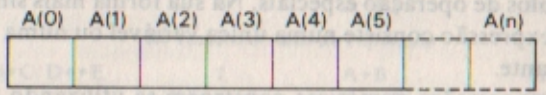
Lembre-se que os caracteres são representados internamente por um código numérico, sendo cada carácter identificado por um valor único. O carácter "A" possui o menor valor dos caracteres alfabéticos e "Z" o maior valor. Ao comparar o valor de cadeias, o BASIC compara os caracteres correspondentes de cada cadeia. Por exemplo: se a cadeia associada a X$ tem o valor "GATO" e a Y$ tem o valor "RATO", então o valor de X$ é menor que o de Y$ pois o valor de G é menor que o de R. A ordenação de informação em caracteres em BASIC é, pois, bastante simples.

Variáveis de quadro

Os programas de computador necessitam muitas vezes de tratar colecções de dados relacionados entre si. Torna-se conveniente ter a possibilidade de utilizar apenas um nome para identificar a colecção em vez de nomes diferentes para cada um dos elementos da colecção. Uma variável de quadro (*array variable*) é uma colecção de variáveis com o mesmo nome, em que a identificação de cada elemento individual do quadro é feita utilizando um subscrito, ou índice, que actua como um número de referência. As convenções para a nomeação de variáveis de quadro são as mesmas utilizadas para identificar variáveis numéricas simples (únicas) e variáveis de cadeia simples. No entanto, o nome de uma variável numérica de quadro apenas pode possuir uma letra. O subscrito é colocado entre parêntesis imediatamente a seguir ao nome da variável. A presença do subscrito permite distinguir uma variável de quadro de uma variável simples. Exemplos de variáveis de quadro subscritas:

A(5) D(N) B$(3) L$(K-1)

Note-se que um subscrito se pode exprimir sob a forma de uma variável, e que A(5) se pode referir à sexta posição do conjunto de posições associadas à variável numérica A, quando o quadro começa em A(0).



Em algumas implementações, a primeira posição é sempre referenciada por A(1), não possuindo A(0) qualquer significado. Noutras, é possível indicar se se quer que o subscrito de uma variável comece em zero ou um. O limite inferior de um subscrito será discutido mais amplamente quando se descrever a declaração DIM.

Note-se, que, no mesmo programa, não se pode utilizar o mesmo nome para identificar simultaneamente uma variável simples e uma variável de quadro.

A linguagem BASIC permite que o tamanho de uma variável de quadro, ou seja, o número de posições de memória que lhe estão associadas, seja declarado através de uma declaração DIM. O limite superior permitido depende da versão de BASIC. Se esta declaração não for incluída será automaticamente atribuído ao subscrito de uma variável de quadro o alcance de 0 a 10 ou de 1 a 10, conforme as implementações.

O BASIC pode normalmente tratar variáveis de quadro com duas dimensões (matrizes). Um elemento particular de uma variável bidimensional é identificado por um par de subscritos separados por uma vírgula.

E(3,4) R$(20,15)

O primeiro subscrito especifica a linha, o segundo especi-

fica a coluna. Portanto, a variável numérica E possui 3 linhas e 4 colunas, ou seja, 12 elementos (3 × 4).

Expressões numéricas

Uma expressão fornece um meio de calcular um valor e é formada pela ligação de variáveis e constantes através de símbolos de operação especiais. Na sua forma mais simples, uma expressão consiste numa única variável ou numa única constante.

As expressões numéricas constroem-se utilizando os seguintes operadores aritméticos:

+ adição	* multiplicação
– subtracção	/ divisão
** exponenciação (isto é, elevar a uma potência)	

Note-se que alguns sistemas utilizam outros símbolos para a exponenciação, por exemplo, ↑ ou ∧ (acento circunflexo).

Numa expressão mista, isto é, onde ocorra mais do que um operador aritmético, o cálculo é executado segundo a sequência matemática normal. A exponenciação é tratada primeiro; seguem-se-lhe a multiplicação e a divisão, segundo a ordem em que apareçam na expressão, começando da esquerda para a direita. A adição e a subtracção são tratadas no fim, segundo a sua ordem de aparição da esquerda para a direita.

Podem utilizar-se os parêntesis para alterar a sequência normal de operações, passando a ser executado primeiro o que se encontrar dentro dos parêntesis. Na Tabela 1 mostram-se exemplos da avaliação aritmética.

Tabela 1

Expressão numérica	*Ordem*	*Operação*	*Resultado*
A+B*C/D**E	1	D**E	R1
	2	B*C	R2
	3	R2/R1	R3
	4	A+R3	R4
	R4 contém o valor da expressão		
(A+B)*C/D**E	1	A+B	T1
	2	D**E	T2
	3	T1*C	T3
	4	T3/T2	T4
	T4 contém o valor da expressão		

Dois operadores aritméticos não podem ser colocados um ao lado do outro como em X/–Y. A construção correcta será X/(–Y). Por outro lado, a presença de um operador não é assumida implicitamente: A(B+1) deve ser escrito A*(B+1).

Expressões de cadeia

Uma expressão de cadeia limita-se a uma cadeia única sob a forma de uma variável de cadeia ou de uma constante de cadeia, tal como foram anteriormente definidas.

Algumas versões de BASIC permitem juntar duas cadeias através de uma operação conhecida como concatenação de cadeias, sendo utilizado o sinal mais (+) como operador de concatenação, por exemplo, X$+Y$.

Expressões relacionais

Uma expressão relacional consiste em duas expressões numéricas, ou duas expressões de cadeia, separadas por um

operador relacional. A linguagem BASIC reconhece seis operadores de relação diferentes:

Operador	*Significado*
=	igual a
<>	diferente de
<	menor que
<=	menor ou igual a
>	maior que
>=	maior ou igual a

Uma expressão relacional só é utilizada no interior da estrutura de controlo IF... THEN (ver mais adiante).

Algumas versões de BASIC permitem a utilização de operadores lógicos a fim de combinar duas ou mais expressões relacionais numa expressão relacional composta. Quando existam, os operadores lógicos são: AND (e), OR (ou) e NOT (negação).

Estrutura de um programa

Um programa BASIC consiste numa sequência de instruções escritas como uma série de declarações BASIC. Cada declaração ou comando é escrita numa única linha e cada linha começa com um número de linha. O número de linha funciona como um rótulo *(label)* e é utilizado para indicar a ordem segundo a qual as declarações irão ser assembladas para execução. Uma declaração pode também ser recuperada de memória, a fim de ser modificada ou apagada, através de uma simples referência ao número de linha apropriado.

As declarações do programa são executadas segundo a ordem dos números de linha até a sequência ser alterada por um comando interno ao programa. A execução continua até ser atingida a declaração END ou ser encontrado um STOP. A declaração END é sempre a última declaração, possuindo portanto o número de linha mais alto do programa.

Estrutura das declarações

Uma declaração inicia-se por um número de linha único destinado a posicionar a declaração na sequência do programa. A parte de instrução de uma declaração começa com uma palavra-chave de BASIC que especifica qual a instrução a ser executada. Cada declaração deve caber numa linha, não podendo portanto exceder 72 caracteres de comprimento. Só é permitida uma declaração por linha. O número de linha deve ser um inteiro de 1 a 9999. Note-se que algumas versões permitem o limite superior de 99999 para o número de linha.

As diferentes implementações de BASIC variam no que respeita a possibilidade da existência ou não de *espaços* no interior de declarações. Como regra geral não se deve permitir a ocorrência de espaços no início de uma linha, ou no seio de números de linha ou palavras reservadas. Contudo, podem e devem introduzir-se espaços dentro da estrutura geral das instruções pois eles facilitam a leitura dos programas. Tudo o que contribua para facilitar a leitura de programas deve ser adoptado.

Números de linha

Tal como se disse anteriormente, os números de linha são inteiros de 1 a 9999. Ao escrever um programa, não numere as instruções em sequência imediata. Utilize saltos de numeração que permitam a inserção posterior de instruções adicionais. Lembre-se de que é difícil escrever um programa correcto à primeira vez, ficando sempre sujeito a alterações e adições. Nos exemplos fornecidos neste livro, utilizou-se geralmente um incremento de 10 nos números de linha.

Comandos BASIC

A linguagem BASIC será agora formalmente apresentada através da explicação de cada uma das palavras de comando e da descrição das regras gramaticais que regem a sua utilização. Incluem-se também exemplos de programas para ilustrar os comandos e a utilização das declarações.

Cada programa-exemplo consiste numa listagem de todas as declarações BASIC que constituem o programa, segundo a ordem dos números de declaração. O *output* produzido pela execução do programa encontra-se escrito, o mais possível, da mesma forma que aquele que seria obtido num periférico de computador. A listagem do programa encontra-se escrita em caracteres maiúsculos tal como o *output* do programa. Sempre que um dado de *input* é fornecido pelo utilizador durante a execução do programa, esse dado de *input* encontra-se escrito em minúsculas. Imediatamente a seguir ao *output* fornece-se igualmente um comentário destinado a chamar a atenção para determinados pontos.

Os variados *comandos de sistema* necessários para fazer coisas tais como traduzir o programa com a respectiva versão de BASIC, criar o ficheiro de programa, obter uma listagem ou instruir o computador para executar o programa, não são incluídos. Estes comandos são específicos de cada sistema computador. Será no entanto necessário lembrar-se que, para além dos comandos BASIC que constituem o seu programa, é preciso comunicar com o computador, o que é feito através de comandos de sistema especiais. Terá igualmente de se familiarizar com o teclado do dispositivo de *input* que utilizar. Também ele terá os seus próprios comandos especiais representados por teclas específicas; por exemplo, uma tecla de "RETURN" que será utilizada no fim de cada linha de *input* não só para se posicionar no início da próxima linha como também para transmitir a linha de *input* para o computador.

EXEMPLO 1: Introduzindo as declarações PRINT e END

Programa

```
10 PRINT "A MINHA PRIMEIRA LINHA DE OUTPUT"
20 PRINT 3 + 4
30 PRINT "3 + 4"
40 PRINT 2 -9/3
50 END
```

Output

```
A MINHA PRIMEIRA LINHA DE OUTPUT   Produzido pela declaração 10
 7                                                            20
 3 + 4                                                        30
-1                                                            40
```

Comentário

PRINT fornece uma informação de *output*. Uma declaração PRINT pode ser utilizada para realizar aritmética.

EXEMPLO 2: Introduzindo LET

Programa

```
10 LET A = 6
20 LET B = 2.5
30 LET T = A * B
40 LET E$ = "RICARDO JOSE"
50 PRINT E$
60 PRINT T
70 PRINT "FIM DO OUTPUT"
80 END
```

Output

```
RICARDO JOSE
 15
FIM DO OUTPUT
```

Comentário

LET atribui valores a variáveis. Uma instrução para imprimir (PRINT) uma variável fornece como *output* a informa-

ção guardada na posição de memória associada a essa variável.

Note-se que nos exemplos anteriores, e nos que se seguem, deixaram-se espaços antes e depois dos sinais de igual e dos operadores aritméticos a fim de aumentar a facilidade de leitura do programa. A sua utilização é, como se disse anteriormente, opcional.

Declaração LET

A declaração LET é utilizada para atribuir um valor a uma variável durante a execução de um programa. Tem a seguinte forma:

n LET *v* = expressão numérica ou
n LET *v*$ = expressão de cadeia

onde *n* é o número da declaração e *v* a variável. A variável pode ser subscrita. A expressão numérica pode ter qualquer grau de complexidade desde que o seu cálculo conduza a um único valor atribuível à variável especificada. Exemplos:

LET A = 5	Atribui o valor 5 à variável numérica A
LET B$ = "EXEMPLO"	Atribui a cadeia de caracteres entre aspas à variável de cadeia B$.
LET D = P/Q + 0.5	A expressão numérica é avaliada utilizando os valores guardados nas posições de memória de P e Q, de acordo com as operações aritméticas indicadas. O resultado é atribuído a D.
LET F$(2) = "FEV"	Atribui a cadeia FEV à posição do vector F$ associada com o subscrito 2.

LET X = P5	Atribui a X o valor actualmente guardado em P5.
LET N = N + 1	Recupera o valor actual de N e soma-lhe 1. O novo valor de N é então atribuído a N, substituindo em memória o anterior valor.

Note-se que o símbolo '=' se pode traduzir como *'é substituído por'* numa declaração de atribuição.

Em algumas versões de BASIC, a palavra LET é opcional. Poder-se-á então escrever A = 5 em vez de LET A = 5.

Declaração PRINT

A declaração PRINT destina-se a transmitir informação do computador para um dispositivo externo, normalmente um terminal com impressora ou vídeo. A declaração na sua forma mais simples é:

n PRINT *x*

onde *x* é o *item* de impressão, quer seja uma expressão numérica ou de cadeia.

PRINT B	Imprime o valor guardado na posição de memória associada à variável B.
PRINT C * 9/5 + 32	Imprime o valor da expressão, utilizando o actual valor de C para fins de cálculo.
PRINT "RESULTADO"	Imprime a cadeia de caracteres entre aspas.
PRINT D$	Imprime a cadeia guardada na posição de memória associada à variável D$.
PRINT 565	Imprime a constante numérica 565.

Um pedido de impressão (PRINT) de uma variável provoca a recuperação para *output* do conteúdo da posição de memória associada a essa variável; quando se trata de uma constante, os caracteres especificados são preparados para *output* exactamente segundo a mesma sequência. Quando o *item* de impressão tem a forma de uma expressão numérica, a expressão é previamente calculada de forma a fornecer um único valor que é então preparado para *output*. Cada declaração de PRINT dá origem a uma nova linha de *output*, excepto quando a declaração PRINT anterior terminar por uma vírgula ou ponto-e-vírgula.

A declaração PRINT pode ainda tomar a forma seguinte:

n PRINT x_1 *ps* x_2 *ps*... x_i

onde *x* é uma lista de *items* de impressão, *ps* é um separador de impressão (vírgula ou ponto-e-vírgula) e *i* é o número de *items* da lista de impressão.

Uma linha de impressão é dividida num número de zonas de impressão definidas pela implementação. Na maioria das versões existem 5 zonas de igual largura, excepto possivelmente a última zona.

Em resumo, *items* separados por vírgulas são impressos em zonas diferentes enquanto *items* separados por ponto-e-vírgula são impressos consecutivamente. A vírgula implica "deixar de imprimir na zona em que se encontra e continuar na próxima zona"; o ponto-e-vírgula implica "esquecer a existência de zonas e continuar no ponto em se encontra".

A cadeia de *output* de uma expressão numérica consiste num espaço inicial se o seu valor for positivo, ou de um sinal menos inicial se o valor for negativo, seguido por uma representação do número numa forma inteira, decimal ou exponencial. Esta representação é sempre seguida de um espaço final. Portanto, mesmo quando se utiliza o ponto-e-vírgula como separador, os resultados numéricos são, de facto, sempre separados pelo menos por um espaço.

A cadeia de *output* de uma expressão de cadeia é idêntica à cadeia de caracteres que constitui a expressão de cadeia e pode ser mais comprida que uma zona de impressão. No caso de ser maior, ela continua pela zona de impressão seguinte. A utilização da vírgula como separador indica apenas que o próximo *item* de impressão começa na próxima zona de impressão disponível.

Quando na mesma declaração PRINT se declaram mais *items* do que aqueles que podem ser impressos numa linha, a impressão continua geralmente na linha seguinte. Pode-se utilizar uma vírgula no fim de uma declaração PRINT a fim de assegurar que o *output* gerado pelo próximo PRINT começa na próxima zona de impressão disponível (não necessariamente na linha seguinte). De forma semelhante, um ponto-e-vírgula no fim de uma declaração PRINT determina que a próxima cadeia de *output* continua na mesma linha caso ainda exista espaço.

A única forma de, em BASIC, deixar uma linha em branco e saltar para a linha seguinte é emitir um comando PRINT vazio, isto é, sem quaisquer *items* a imprimir. É vulgar encontrar em programas tais instruções destinadas a imprimir "nada".

A linguagem BASIC permite, no entanto, que o *output* comece numa coluna determinada da linha de impressão. Isto consegue-se utilizando a função TAB (ver mais adiante).

O exemplo seguinte ilustra vários pontos relacionados com o uso da vírgula e do ponto-e-vírgula para separar campos de *output*.

EXEMPLO 3: Mais alguma coisa sobre o comando PRINT

Programa

```
010 LET A = 6
020 LET B = 4
030 LET C = 20
```

```
040 LET D = A * B/C
050 PRINT A, B, C, D, B - C
060 PRINT A; B; C; D
070 PRINT
080 PRINT A, A, A, A, A, A, A
090 PRINT
100 PRINT B * C, "NOT ON A NEW LINE",
110 PRINT (B * C) ** A
120 LET X$ = "THREE BLIND MICE,"
130 LET Y$ = X$
140 PRINT
150 PRINT
160 PRINT Y$; X$
170 END
```

Output

```
                                                    Output da
                                                    declaração
6             4           20          1.2          -16      50
6  4  20  1.2                                              60
                                                           70
6             6           6           6            6       80
6             6
                                                           90
80          NOT ON A NEW LINE      2.62144E+11             100/110
                                                           140
                                                           150
THREE BLIND MICE, THREE BLIND MICE,                        160
```

Comentário

Os *outputs* das declarações 50 e 60 mostram a diferença entre o espacejamento obtido num *output* numérico utilizando vírgulas e o ponto-e-vírgula. As declarações 70, 90, 140 e 150 produzem linhas em branco. O *output* da declaração 80 estende-se para uma segunda linha, dado existirem mais *items* de impressão que zonas de impressão. A vírgula no fim da declaração 100 provoca a impressão na mesma linha do próximo *item* especificado (na 110). O ponto-e-vírgula na declaração 160 faz com que as duas cadeias sejam impressas sem intervalo.

Declaração END

A declaração END destina-se a indicar o fim de um programa BASIC. Ela conclui a sequência de declarações que constituem o programa e termina a execução do programa ao ser encontrada. Tem a seguinte forma:

n END

onde *n* é o mais alto número de linha utilizado no programa.

Declaração REM

REM é uma declaração não-executável destinada a permitir a inclusão de anotações no programa. Como esta declaração é ignorada em tempo de execução, o controlo do programa é passado para a primeira declaração executável que se lhe segue. Permite ao programador colocar comentários e notas explicativas na estrutura do programa a fim de o tornar mais legível e compreensivo. Tem a seguinte forma:

n REM *comentário*

onde o comprimento do comentário se encontra apenas limitado pelo comprimento da linha. Comentários mais longos ou comentários que sejam mais facilmente legíveis quando separados podem ser codificados utilizando uma sucessão de declarações REM, por exemplo:

```
060 REM * P -- NUMERO DE CANDIDATOS PASSADOS          *
070 REM * C -- NUMERO DE CANDIDATOS CHUMBADOS         *
080 REM * X -- NOTA LIMITE DE PASSAGEM               *
090 REM * N -- NOTA DO CANDIDATO                     *
100 REM ************************************************
110 PRINT "PROGRAMA DE ANALISE DE UM EXAME"
120 REM INICIALIZAR OS CONTADORES DE PASSADOS
125 REM E CHUMBADOS
130 LET P = 0
```

```
140 LET F = 0
150 REM   LER A NOTA LIMITE DE PASSAGEM
```

EXEMPLO 4: Introduzindo READ e DATA

Programa

```
10 REM   LER UM ITEM DE DADOS E IMPRIMI-LO A FIM DE
20 REM   VERIFICAR SE FOI BEM LIDO E CORRECTAMENTE
25 REM   ATRIBUIDO
30 READ N
40 DATA 4.5
50 PRINT "N =";N
60 END
```

Output

```
N = 4.5
```

Comentário

O valor 4.5 é fornecido ao programa através da declaração DATA em 40. O valor é atribuído a N pela declaração READ em 30. A linha de *output* é produzida pela declaração PRINT em 50.

Declaração READ

A declaração READ é utilizada para a atribuição de valores a variáveis a partir de uma sequência de dados criada por uma ou mais declarações DATA. Tem a forma seguinte:

n READ $v_1, v_2, ..., v_i$

onde cada *v* é uma variável numérica ou de cadeia, simples ou subscrita. Quando se especifica mais do que uma variável numa declaração READ, utilizam-se vírgulas para as separar.

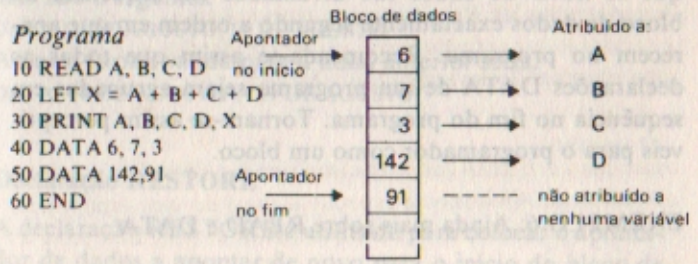
Ao ser executada uma declaração READ, são atribuídos valores às variáveis listadas retirados de um "bloco de dados" que foi preenchido com dados a partir de uma ou mais declarações DATA. O primeiro *item* de dados do

bloco é atribuído à primeira variável, o segundo à segunda variável e assim sucessivamente. A sequência de dados é mantida por um apontador que aponta para o primeiro *item* no início da execução do programa. À medida que cada *item* de dados é lido e atribuído, o apontador é avançado para o *item* seguinte.

Existe em BASIC um comando RESTORE que permite voltar a colocar o apontador no início do bloco de dados. Torna-se assim possível ler a sequência de dados mais do que uma vez.

Note-se que a constante atribuída a uma variável deve ser do mesmo tipo que a variável. Só se pode atribuir uma cadeia de caracteres a uma variável de cadeia e uma constante numérica a uma variável numérica. Volta-se aqui a frisar este princípio básico pois é fácil cometer este erro num READ. Qualquer tentativa de atribuir dados de tipo errado durante a execução do programa é rejeitada como erro fatal.

EXEMPLO 5: Algo mais sobre READ e DATA



Output

```
6          7          3          142        158
```

Declaração DATA

A declaração DATA é utilizada para juntar dados àquilo que pode ser considerado como um "bloco de dados" interno ao programa. Os dados passam então a encontrar-se disponíveis para atribuição, durante a execução do programa, a variáveis especificadas em declarações READ. A declaração DATA tem a seguinte forma:

n DATA $d_1, d_2, ..., d_n$

onde cada *d* é um *item* de dados individual, sob a forma de uma constante numérica ou de cadeia. *Items* de dados sucessivos são separados por vírgulas.

Os *items* de dados são colocados no bloco em tempo de compilação. Durante a execução do programa, uma declaração DATA é ignorada tal como qualquer outra declaração não-executável, por exemplo, a declaração REM, e o controlo passa para a declaração executável imediatamente a seguir.

Dado que a declaração DATA não toma parte durante a execução do programa, pode ser escrita em qualquer lugar da sequência de instruções. É importante, contudo, notar que os *items* de dados são arrumados em sequência no bloco de dados exactamente segundo a ordem em que aparecem no programa. Recomenda-se assim que todas as declarações DATA de um programa sejam agrupadas em sequência no fim do programa. Tornam-se assim perceptíveis para o programador como um bloco.

EXEMPLO 6: Ainda mais sobre READ e DATA

Programa

```
010 REM  ESTE PROGRAMA LE ALGUNS DADOS
020 REM  FAZ ALGUMAS CONTAS
030 REM  E IMPRIME O RESULTADO
040 READ X, A$, M$, C
```



```
050 PRINT "TOTAL "; M$; X; "POLEGADAS"
060 LET C = C + X
070 PRINT
080 PRINT A$; "ACUMULADA "; "TOTAL"; C; "POL."
090 DATA 1.96, "PRECIPITACAO ", "MENSAL"
100 DATA 10.07
110 END
```

Output

```
TOTAL MENSAL   1.96 POLEGADAS

PRECIPITACAO ACUMULADA TOTAL   12.03 POL.
```

Comentário

Para se obterem intervalos ao imprimir cadeias utilizando o ponto-e-vírgula é necessário incluir os espaços nas cadeias.

Um *item* de cadeia de caracteres não necessita de se encontrar entre aspas numa declaração DATA. Uma cadeia nessas condições não poderá, no entanto, começar por um espaço, um dígito, um sinal mais ou menos, um ponto ou uma vírgula, nem poderá incluir espaços brancos no final. A declaração seguinte é válida:

```
DATA SEGUNDA, TERCA, QUARTA
```

mas não a seguinte:

```
DATA 18 DE AGOSTO, 19 DE AGOSTO
```

A única forma válida para a linha anterior seria:

```
DATA "18 DE AGOSTO", "19 DE AGOSTO"
```

Declaração RESTORE

A declaração RESTORE é utilizada para colocar o apontador de dados a apontar de novo para o início do bloco de dados. Isto permite que o próximo READ a ser encontrado volte a atribuir os dados desde o início da sequência. Tem a seguinte forma:

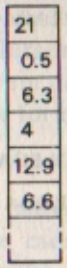
n RESTORE

Por exemplo:

Segmento de programa

```
010 READ A
020 READ B, X
030 RESTORE
040 READ M1, M2
•
•
•
200 DATA 21
210 DATA 0.5, 6.3,4
220 DATA 12.9, 6.6
```

Bloco de dados



Atribuições:

A = 21 M1 = 21
B = 0.5 M2 = 0.5
X = 6.3

Note-se que qualquer tentativa para ler um *item* de dados que não exista é um erro fatal. Esta situação ocorrerá se, durante a execução, for encontrado um comando READ numa altura em que o apontador se encontrar posicionado à frente do último *item* de dados do bloco, isto é, apontar para uma posição vazia. Em algumas implementações existe uma opção NODATA que permite testar a existência de dados no bloco. No entanto, não constitui uma opção normalizada da linguagem.

EXEMPLO 7: Introduzindo INPUT

Programa

```
10 REM   ACEITAR UM ITEM DE DADOS COMO INPUT
15 REM   E IMPRIMI-LO
20 REM   PARA VERIFICAR SE FOI ACEITE CORRECTAMENTE
10 INPUT N
40 PRINT "N ="; N
50 END
```

Output

```
? 4.5
N = 4.5
```


Comentário

A primeira linha do *output* representa o pedido feito pelo sistema para a introdução de dados, seguido do valor introduzido pelo programador em resposta. A segunda linha constitui o verdadeiro *output* do programa e é gerada pela declaração 40.

Declaração INPUT

A declaração INPUT permite a atribuição de valores de dados a variáveis por acção do próprio utilizador do programa durante a sua execução. É a única declaração da linguagem BASIC que exige interacção durante a fase de execução. A declaração INPUT tem a seguinte forma:

n INPUT $v_1, v_2, \ldots, v_i$

onde cada v é uma variável numérica ou de cadeia, simples ou subscrita. Sempre que se especifica mais do que uma variável numa declaração INPUT, devem utilizar-se vírgulas para as separar.

Ao executar uma declaração INPUT, um programa que opere em modo interactivo faz uma pausa e espera que sejam atribuídos valores às variáveis especificadas. O BASIC interroga o utilizador com um pedido de *input* geralmente sob a forma de um ponto de interrogação apresentado no dispositivo de *vídeo* do terminal. Notar que um programa *não pode* continuar a execução até serem fornecidos os dados pedidos.

Ao introduzir dados em resposta a um pedido de INPUT, deve-se separar cada *item* por um vírgula. Algumas versões permitem a utilização do espaço como separador de *items* de dados numéricos. Podem introduzir-se cadeias de caracteres sem aspas desde que não comecem por um dígito, um sinal mais ou menos, um ponto ou uma vírgula.

Quaisquer dados introduzidos devem possuir o mesmo tipo do das variáveis a que vão ser atribuídos. Caso se

introduza o tipo de dados errado, ou dados a menos ou a mais, normalmente o sistema assinala o erro e pede novamente os dados até serem introduzidos correctamente.

É fácil um utilizador esquecer quais os dados que devem ser introduzidos, quer se trate de um programa muito grande quer de um programa pouco utilizado. Recomenda-se, por isso, que, antes de surgir no écran o pedido de *input* do sistema, o próprio programa emita um comentário que esclareça a resposta que deve ser fornecida. Não há nada mais frustrante do que não nos lembrarmos da resposta ao ponto de interrogação do sistema!

EXEMPLO 8: PRINT em associação com INPUT

Programa

```
010 REM   PROGRAMA PARA IMPRIMIR A DATA E
020 REM   A TEMPERATURA MAXIMA REGISTADA
030 PRINT "DIA DA SEMANA"
040 INPUT D$
050 PRINT "DATA"
060 INPUT M$
070 PRINT "TEMP MAX";
080 INPUT T
090 PRINT
100 PRINT
110 PRINT "TEMP MAX DE"; D$; M$
120 PRINT "FOI"; T; "GRAUS CENTIGRADOS"
130 END
```

Output

```
DIA DE SEMANA
? quinta-feira
DATA
? 7 de outubro
TEMP MAX? 16

TEMP MAX DE QUINTA-FEIRA 7 DE OUTUBRO
FOI DE 16 GRAUS CENTIGRADOS
```


Comentário

A cadeia de caracteres "quinta-feira" na segunda linha do *output* são os dados introduzidos pelo utilizador em resposta ao pedido de *input* na linha 40 do programa. A cadeia "7 de Outubro" na quarta linha do *output* é a resposta à declaração 60 e, finalmente, o número 16 é a resposta à declaração 80. As declarações 30, 50 e 70 são comentários que dão significado aos pedidos de *input* do sistema. O ponto de interrogação gerado pelo sistema segue-se imediatamente à cadeia TEMP MAX no *input* devido à utilização do ponto-e-vírgula no final da declaração 70. Na quarta linha de *output* a resposta encontra-se entre aspas pois a cadeia de caracteres começava por um dígito.

Algumas versões de BASIC permitem a seguinte construção, que elimina a necessidade das declarações PRINT com anotações:

```
INPUT "DIA DA SEMANA"; D$
```

EXEMPLO 9: Introduzindo GO TO e IF... THEN

Programa

```
010 REM   SOMAR OS PRIMEIROS 100 NUMEROS INTEIROS
020 REM   N : PROXIMO NUMERO S : SOMA ACUMULADA
030 S = 0
040 N = 0
050 N = N + 1
060 S = S + N
070 IF N = 100 THEN 90
080 GO TO 50
090 PRINT "SOMA DOS PRIMEIROS 100 INTEIROS"
100 PRINT "IGUAL A"; S
110 END
```

Output

```
SOMA DOS PRIMEIROS 100 INTEIROS
IGUAL A 5050
```

Comentário

O controlo passa da declaração 70 para a 90 só quando o valor de N é realmente igual a 100. Até que isso se verifique, cada vez que for executada a declaração 70, o controlo passa em sequência normal para a declaração 80. A estrutura IF... THEN é utilizada para escapar ao ciclo constituído pela declaração GO TO.

Notar que neste exemplo o fluxo de controlo do programa poderia ser conseguido substituindo as declarações 70 e 80 pela declaração única:

```
070 IF N< 100 THEN 50
```

Declaração GO TO

A declaração GO TO produz uma transferência incondicional de controlo para o número de linha especificado, alterando assim, portanto, a sequência de instruções definidas pelos números das declarações. Tem a seguinte forma:

n GO TO *nl*

onde *nl* é o número de linha da declaração para onde deve ser passado o controlo.

GO TO 50 resulta num salto para a declaração com o número 50, continuando a execução a partir desse ponto e não no número de linha que se segue imediatamente ao da declaração GO TO.

A declaração GO TO pode provocar um salto para uma linha com um número mais alto ou mais baixo que o seu, isto é, pode transferir o controlo para a frente ou para trás.

Deve ter-se um certo cuidado ao utilizar saltos incondicionais pois é muito fácil estabelecer-se involuntariamente um ciclo para o qual não existe saída ou, por exemplo, saltar um grupo de declarações que acabam por nunca vir a ser executadas. Pode-se escapar a um ciclo "sem fim" interrompendo o programa quando ele se está a executar num

ambiente interactivo. No entanto, isso "mata" ou termina o programa no ponto em que a interrupção é feita, qualquer que ele seja. Notar que o método de interrupção é uma característica do sistema em que o programa está a ser executado e não da linguagem BASIC.

Declaração IF... THEN

A declaração IF... THEN estabelece uma transferência de controlo sujeita a condições especificadas na própria declaração. Tem a forma:

n IF *(expressão relacional)* THEN *nl*

onde *nl* é o número de linha da declaração para onde o controlo deve ser passado caso a expressão relacional seja verdadeira (TRUE).

Uma expressão relacional consiste em duas expressões numéricas ou duas expressões de cadeia ligadas por um operador de relação. Ao ser executada a declaração IF... THEN, os valores das expressões são comparados de acordo com o operador de relação especificado. Quando o resultado da comparação for verdadeiro (TRUE) o controlo é transferido para o número de linha especificado depois da palavra THEN. Quando a comparação não for verdadeira, ou falsa (FALSE), a execução do programa continua na declaração que vem imediatamente a seguir ao IF... THEN. Os operadores de relação permitidos são:

= igual a	<> diferente de
< menor que	<= menor ou igual a
> maior que	>= maior ou igual a

Na avaliação de uma expressão relacional numérica, pode ser necessário calcular primeiro os valores das duas expressões numéricas antes de a comparação especificada pelo operador de relação se poder fazer. Uma condição ou é

satisfeita ou não é. Uma expressão ou é verdadeira (TRUE) ou falsa (FALSA). Por exemplo, tomemos a seguinte expressão:

IF (X + 1) ** 2 >= Y * 3 THEN *nl*

quando X = 1 e Y = 4 a expressão é FALSA pois 4 não é >= 12

= 2	= 3	VERDADEIRA	9 é =9
= 3	= 2	VERDADEIRA	16 é >6
= 4	= 1	VERDADEIRA	25 é >3

Na avaliação de uma expressão de cadeia, as duas expressões de cadeia são comparadas carácter a carácter, do primeiro ao último carácter. Recorde-se que, internamente, cada carácter é representado por um código numérico único. O carácter A tem o menor valor dos caracteres alfabéticos e Z o maior. Para que duas cadeias sejam iguais, é necessário que tenham exactamente o mesmo comprimento e contenham os mesmos caracteres exactamente na mesma ordem.

Em cadeias de igual comprimento, o primeiro par de caracteres correspondentes desiguais determina qual é a maior. Por exemplo,

PEDRO é maior que PEDRA,

pois "O" é maior que "A".

Dado que o primeiro par de caracteres correspondentes desiguais determina qual das cadeias é maior, a mais curta das cadeias pode ser a maior das duas. Por exemplo,

ABRIR é maior que ABERTO.

Quando duas cadeias são de comprimentos diferentes e os caracteres comparáveis são iguais, a cadeia mais longa é a maior. Por exemplo,

MARTELO é maior que MARTE.

Notar que os espaços dentro de uma cadeia são significativos. O espaço é tratado como um carácter e possui o seu próprio código de representação interna (mais alto que qualquer dos caracteres alfabéticos). Por exemplo,

"VASCO " é maior que "VASCO".

Algumas versões de BASIC permitem a utilização de operadores lógicos a fim de combinar duas ou mais expressões numa única expressão relacional composta. Os operadores permitidos são AND ("e"), OR ("ou") e NOT (negação).

Serão dados mais exemplos de IF... THEN quando da introdução da declaração DIM e variáveis de quadro.

EXEMPLO 10: Introduzindo FOR——NEXT

Programa

```
010 REM  PROGRAMA — POTENCIAS DE 3
020 FOR N = 1 TO 12
030 PRINT "3 ELEVADO A"; N; " = "; 3 ** N
040 NEXT N
050 END
```

Output

```
3 ELEVADO A 1 = 3
3 ELEVADO A 2 = 9
3 ELEVADO A 3 = 27
3 ELEVADO A 4 = 81
3 ELEVADO A 5 = 243
3 ELEVADO A 6 = 729
3 ELEVADO A 7 = 2187
3 ELEVADO A 8 = 6561
3 ELEVADO A 9 = 19683
3 ELEVADO A 10 = 59049
3 ELEVADO A 11 = 177147
3 ELEVADO A 12 = 531441
```

Comentário

A declaração PRINT é a única declaração dentro do ciclo. Ela é executada 12 vezes produzindo 12 linhas de *output*.

Declarações FOR——NEXT

FOR e NEXT são um par de declarações utilizadas para estabelecer e controlar ciclos. FOR inicia o ciclo e especifica quantas vezes o ciclo vai ser executado. NEXT indica o fim da sequência de declarações dentro do ciclo e devolve o controlo à primeira declaração a seguir a FOR, depois de ter incrementado o contador de ciclos. As declarações do ciclo são todas as declarações que se encontram entre FOR e NEXT. A forma da declaração FOR é a seguinte:

n FOR *vn* = en_1 TO en_2 STEP en_3

onde *vn* é uma variável numérica simples, conhecida como variável de controlo do ciclo ou contador de ciclos. Esta mesma variável de controlo é utilizada na declaração NEXT.

en_1, en_2 e en_3 são expressões numéricas representando respectivamente o valor inicial da variável de controlo, o máximo ou valor limite, e a quantidade de que vai ser incrementada a variável de controlo em cada execução do ciclo. A especificação do valor do incremento (STEP) é opcional. Na sua ausência, o valor de defeito do incremento é STEP 1.

As expressões são, onde necessário, calculadas durante a execução a fim de ser encontrado o valor inicial, o valor limite e o valor do incremento. O valor inicial é atribuído à variável de controlo e comparado com o limite para se determinar se se encontra dentro do alcance especificado, isto é, se é menor ou igual ao limite quando o incremento é positivo ou se é maior ou igual ao limite quando o incremento é negativo. No caso de a variável de controlo estar dentro do alcance, são executadas em sequência as declarações do ciclo até ser atingida a declaração NEXT. Esta declaração tem a seguinte forma:

n NEXT *vn*

onde *vn* é a mesma variável numérica simples de controlo definida na declaração FOR.

Ao ser executada a declaração NEXT, a variável de controlo é incrementada da quantidade indicada em STEP e é feito um teste ao seu valor comparando-o com o valor limite. Se se encontrar dentro do seu alcance, o controlo passa para a primeira declaração executável que se segue à declaração FOR. Este ciclo continua a executar-se enquanto a variável de controlo se encontrar dentro do alcance definido. Assim que um novo incremento colocar a variável de controlo fora do alcance, o controlo passa para a primeira declaração executável a seguir ao NEXT.

É permitido um STEP ou incremento negativo, devendo neste caso o valor inicial da variável de controlo ser maior que o valor limite. Por outro lado, os valores inicial e limite e o valor do incremento tanto podem ser números reais como inteiros. A declaração:

FOR J = 0.1 TO 10 STEP 0.1

é sintaticamente correcta e estabelece um ciclo que é executado 100 vezes até ser satisfeito.
A declaração:

FOR I = 5.5 TO — 2.0 STEP — 0.5

estabelece um ciclo que é executado 16 vezes antes de ser satisfeito.

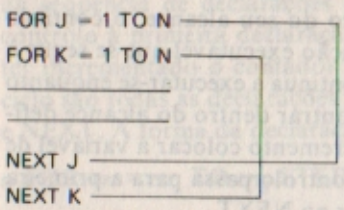
Um ciclo FOR pode ser colocado dentro de outro ciclo FOR, desde que não se sobreponham. O número de ciclos que podem ser colocados dentro de outros depende da implementação de BASIC. Exemplo:

```
S = 0
FOR J = 1 TO N
FOR K = 1 TO N
S = S + A(J,K)
NEXT K
NEXT J
```

É uma construção válida porque o ciclo controlado pela variável K se encontra totalmente dentro do ciclo da variável J.

Utilizando os contadores de ciclos como subscritos ou índices, conseguiu-se recuperar e adicionar os valores da matriz (n × n) a fim de obter a soma de todos os valores da matriz.

Exemplo do *uso incorrecto* de ciclos FOR:



É uma construção inválida dado que os ciclos J e K se sobrepõem.

Ao utilizar a estrutura FOR——NEXT é necessário ter cuidado para não se saltar para dentro dum ciclo a partir do exterior, por exemplo, através de uma declaração GO TO. Se a declaração FOR não tiver sido executada, por se lhe ter saltado por cima, a variável de controlo da declaração FOR não tem qualquer valor. É igualmente necessário evitar alterar o valor da variável de controlo dentro do ciclo. Finalmente, não se deve utilizar a mesma variável de controlo para um ciclo que se encontra dentro de outro ciclo.

EXEMPLO 11: Introduzindo STOP

Programa

```
010 PRINT "ESCREVA A SUA IDADE"
020 INPUT A
030 IF A >30 THEN 60
040 PRINT "AINDA TEM A VIDA A SUA FRENTE"
050 STOP
060 IF A >55 THEN 90
070 PRINT "A VIDA PASSA..."
080 STOP
090 PRINT "A REFORMA APROXIMA-SE"
100 END
```


Output

```
ESCREVA A SUA IDADE
? 44
A VIDA PASSA...
```

Comentário

Existem vários pontos de paragem neste programa. Ao ser recebido o valor 44 como *input* na declaração 20, a execução não vai além da declaração 80.

Declaração STOP

A declaração STOP faz parar a execução de um programa. Só é necessária se se desejar parar a execução de um programa antes de ser atingida a declaração END. A declaração STOP tem a seguinte forma:

n STOP

Durante a execução, o programa termina neste ponto.

A declaração STOP pode aparecer em mais do que um ponto de um programa, mas, se um programa atingir naturalmente a declaração END, não há qualquer necessidade de se utilizar esta declaração.

EXEMPLO 12: Paragem através de um pedido de *input*

Programa

```
010 REM   CONVERTER LIBRAS-PESO EM QUILOS
020 PRINT   "INTRODUZA O PESO EM LIBRAS"
025 PRINT   "(VALOR NEGATIVO PARA PARAR)";
030 INPUT P
040 IF P<0 THEN 100
050 REM   1 KG = 2.2 LBS
060 LET K = P / 2.2
070 PRINT K; "KG"
080 PRINT
090 GO TO 20
100 END
```

Output

```
INTRODUZA O PESO EM LIBRAS
(VALOR NEGATIVO PARA PARAR)? 11
5. KG

INTRODUZA O PESO EM LIBRAS
(VALOR NEGATIVO PARA PARAR)? 152.
69.0909 KG

INTRODUZA O PESO EM LIBRAS
(VALOR NEGATIVO PARA PARAR)? -1
```

Comentário

O programa continua a executar o mesmo ciclo até ser introduzido um valor negativo em resposta à declaração 30. Cada vez que é executada a declaração 40 é feito um teste ao valor do *input* para verificar se é negativo. Quando isso acontecer, o controlo salta para a declaração END em 100 e a execução termina. A construção de uma saída deste tipo é muito vulgar em programação. Notar que um sistema de operação pode possuir meios próprios de terminar a execução de um programa através de um comando como, por exemplo, *"stop"* ou *"quit"*. No entanto, isto *não é* BASIC, nem uma solução de programação aceitável.

Declaração ON... GO TO

A estrutura ON... GO TO oferece uma transferência condicional que permite fazer passar o controlo para uma de diferentes linhas. Tem a seguinte forma:

n ON *en* GO TO nl_1, nl_2, ..., nl_i

onde *en* é uma expressão numérica e os nl_i são números de linha para onde o controlo é passado conforme o valor da expressão numérica.

Durante a execução, a expressão numérica é calculada. Se o seu resultado for decimal, é arredondado de forma a ser obtido um inteiro. Quando o seu valor é 1, o controlo é

transferido para nl_1; quando é 2, o controlo é transferido para nl_2; e quando é *i*, para nl_i. Por exemplo, ao ser executada a declaração

ON X GO TO 300, 330, 180, 400

o controlo passa para 300 quando X = 1
330 quando X = 2
180 quando X = 3
400 quando X = 4

O valor da expressão numérica deve encontrar-se entre 1 e *i*. No exemplo anterior o valor X deve encontrar-se entre 1 e 4.

O número de saltos condicionais que se podem incluir numa estrutura ON... GO TO encontra-se apenas limitado pela quantidade de números de linha que é possível escrever numa única linha. Na realidade, encontra-se limitado pelo número relativamente pequeno de saltos que é geralmente necessário construir em determinado ponto de um programa.

Sub-rotinas

É muito vulgar na escrita de programas querer executar a mesma sequência de declarações mais do que uma vez em diferentes etapas da execução do programa. Para evitar a repetição de código em cada ponto do programa em que tal sequência é necessária, pode construir-se com essas declarações um subprograma sub-rotina que será chamado sempre que necessário. A utilização da estrutura sub-rotina significa que uma secção de programa utilizada com muita frequência só tem de ser codificada uma única vez. A linguagem BASIC utiliza a declaração GO SUB para efectuar a transferência de controlo da parte principal do programa para o subprograma e a declaração RETURN para devolver o controlo ao programa principal.

Declaração GO SUB

A declaração GO SUB assegura a transferência de controlo para uma sub-rotina definida pelo utilizador. Tem a seguinte forma:

n GO SUB *nl* (é igualmente aceitável escrever GOSUB)

onde *nl* é o número de linha da primeira declaração da sub-rotina.

Ao ser executada a declaração GO SUB, o controlo passa para a primeira declaração da sub-rotina e a execução prossegue a partir desse ponto até ser encontrada uma declaração RETURN.

EXEMPLO 13: Introduzindo GO SUB e RETURN

Programa

```
010 REM  ESTE PROGRAMA DEMONSTRA A UTILIZACAO
015 REM  DAS DECLARACOES GO SUB E RETURN
020 REM  COMO FORMA DE TRANSFERIR O CONTRÓLO
025 REM  NUM PROGRAMA. NAO PODE SER CONSIDERADO
030 REM  UM EXEMPLO REALISTA DE QUANDO DEVERA
040 REM  SER UTILIZADA UMA SUB-ROTINA
050 PRINT "DEMONSTRACAO DE UMA SUB-ROTINA"
060 LET X = 5
070 GO SUB 120
080 PRINT X, Y
090 GO SUB 120
100 PRINT X, Y
110 STOP
120 X = X ** 2
130 Y = X  10
140 RETURN
150 END
```

Output

```
DEMONSTRACAO DE UMA SUBROTINA
 25            2.5
 625           62.5
```


Comentário

A sub-rotina consiste nas declarações 120-130 e é executada duas vezes. Da primeira vez a devolução do controlo é feita para a declaração 80 e da segunda para a declaração 100. Note-se que pode ser feita mais do que uma referência à mesma sub-rotina dentro do programa principal. O controlo é sempre devolvido para a declaração do programa principal que se segue à declaração GO SUB que houvera accionado o salto para a sub-rotina.

Uma sub-rotina pode ser chamada de outra sub-rotina. Pode, portanto, aparecer uma declaração GO SUB dentro de uma sequência GO SUB... RETURN. Uma declaração GO SUB pode igualmente ser colocada dentro de um ciclo FOR... NEXT sem afectar negativamente a execução do ciclo.

Declaração RETURN

A declaração RETURN assegura a devolução do controlo ao programa principal ou à sub-rotina que a chamou. RETURN deve ser sempre a última declaração de uma sub-rotina. Tem a seguinte forma:

n RETURN

Ao ser executada a declaração RETURN, o controlo é dirigido para a primeira declaração executável do programa principal que imediatamente se segue à declaração GO SUB que fez executar a sub-rotina. Repare-se que não é necessário declarar à frente do comando RETURN o número de linha que indica o ponto do programa para onde deve ser passado o controlo. Este ponto depende da declaração GO SUB especifica que referencia a sub-rotina.

Se a lógica o exigir, uma sub-rotina pode conter mais do que uma declaração RETURN. A única exigência de construção é que a última declaração da sub-rotina seja um RETURN.

Quadros: a declaração DIM

A declaração DIM é utilizada para definir o tamanho de variáveis de quadro (*array variables*) de forma a serem reservadas posições de memória para sua utilização durante a execução do programa. A linguagem BASIC está preparada para tratar variáveis bidimensionais, ou matrizes, assim como variáveis unidimensionais, ou vectores. A declaração DIM tem a seguinte forma:

n DIM v_1 *(s),* v_2 *(s), ...,* v_i *(s)*

onde *v* é uma variável numérica ou de cadeia que é declarada como variável de quadro pela sua inclusão na lista da declaração DIM, e *s* é uma constante numérica, entre parêntesis, que define o tamanho desse quadro. No caso de um quadro bidimensional é necessário declarar dois subscritos (separados por uma vírgula), isto é, $v\ (s_1, s_2)$, onde s_1 é o número de linhas e s_2 o número de colunas da matriz.

Exemplo de uma declaração DIM:

DIM A(25), B$(100), X(15,20)

Note-se que, quando se declara mais do que uma variável num DIM, estas deverão ser separadas por vírgulas.

Pode-se utilizar num programa uma variável de quadro sem uma declaração DIM explícita. No entanto, para um quadro não declarado, a linguagem BASIC impõe para o subscrito um limite superior de 10. O facto de uma variável possuir subscritos é suficiente para o BASIC a reconhecer como uma variável de quadro, apesar de não se encontrar declarada como tal numa declaração DIM.

O alcance de um subscrito para um quadro declarado numa declaração DIM vai de zero até ao número especificado no DIM para essa variável. Para um quadro não-declarado, o alcance vai de 0 a 10, isto é, 11 posições de memória disponíveis.

EXEMPLO 14: Introduzindo DIM

Programa

```
010 REM  PROGRAMA PARA DETERMINAR O ELEMENTO
015 REM  COM VALOR MINIMO DE UM VECTOR
020 REM  COM 20 ELEMENTOS
030 REM  ================================
040 REM  DECLARAR A DIMENSAO DO VECTOR
050 DIM N(20)
060 REM  CICLO PARA ATRIBUIR VALORES AOS ELEMENTOS
065 REM  DO VECTOR
070 FOR K = 1 TO 20
080 READ N(K)
090 NEXT K
100 REM  FIM DO CICLO
110 REM  RECUPERAR O VALOR GUARDADO NA PRIMEIRA
115 REM  POSICAO
120 REM  CONSERVANDO-O TEMPORARIAMENTE COMO
125 REM  VALOR MINIMO
130 X= N(1)
140 REM  CICLO PARA TESTAR OS VALORES SUCESSIVOS
145 REM  E DESCOBRIR UM
150 REM  NOVO VALOR MINIMO SUBSTITUINDO
155 REM  O ANTERIOR EM X
160 FOR I = 2 TO 20
170 IF X < N(I) THEN 190
180 LET X = N(I)
190 NEXT I
200 REM  FIM DO CICLO
210 PRINT "O MENOR ELEMENTO DO VECTOR ="; X
220 DATA 176, 21, 274, 186, 99, 11, 29, 101, 75
230 DATA 33, 127, 71, 90, 30, 211, 13, 28, 61, 138
240 END
```

Output

```
O MENOR ELEMENTO DO VECTOR = 11
```

Existe uma opção da linguagem BASIC que permite alterar o valor mínimo dos subscritos de variáveis dimensionadas para 1. É a declaração OPTION BASE que tem a seguinte forma:

n OPTION BASE *i*

onde *i* pode ter o valor 1 ou 0.

Num programa que não inclua a declaração OPTION BASE o limite inferior dos subscritos é, por defeito, zero. Quando se pretender uma base de 1 deve colocar-se a declaração OPTION BASE antes da declaração DIM, onde se faz a declaração dos quadros, ou antes do primeiro desses quadros quando não são declarados por DIM.

Em algumas implementações de BASIC passa-se exactamente o contrário. O limite inferior por defeito é 1, havendo a necessidade de se declarar uma base de 0 através de uma declaração OPTION BASE. Notar que, normalmente, a base não pode ser invertida dentro de um programa, devendo a declaração OPTION BASE ser utilizada apenas uma única vez no mesmo programa.

A necessidade de utilizar OPTION BASE é extremamente rara. É o subscrito que controla a utilização de um quadro e é o programador que define o subscrito. No exemplo seguinte, o facto de o limite inferior poder ser zero, por defeito, não tem qualquer consequência.

```
10 DIM X(20)
20 FOR K = 1 TO 20
30 INPUT X(K)
40 NEXT K
```

A variável de controlo para o ciclo FOR determina que o primeiro valor de *input* é atribuído a X(1), o segundo a X(2), etc. X(0) não é utilizado e, sob o ponto de vista do programa, é como se não existisse.

Existem variações na forma como as diferentes implementações de BASIC manipulam a base de quadros. Em algumas, como foi referido anteriormente, a base, ou limite inferior, é 1 por defeito. Podem igualmente existir outras variações menores no que diz respeito ao formato e à

maneira de utilizar esta opção. Essas variações devem ser procuradas na documentação da versão de BASIC que estiver a utilizar, se vierem a ser necessárias.

Apresenta-se seguidamente um esquema conceptual de um quadro bidimensional, matriz, mostrando os subscritos que referenciam cada posição.

1,1	1,2	1,3	1,4	1,5	1,6
2,1	2,2	2,3	2,4	2,5	2,6
3,1	3,2	3,3	3,4	3,5	3,6
4,1	4,2	4,3	4,4	4,5	4,6

Segmento de programa

```
70 FOR J = 1 TO 4
80 FOR K = 1 TO 6
90 READ A(J,K)
100 NEXT K
110 NEXT J
•
•
•
300 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
310 DATA 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
320 DATA 21, 22, 23, 24
330 END
```

O ciclo controlado por K (ciclo interior) é executado 6 vezes em cada passo do ciclo controlado por J (ciclo exterior). Portanto, são lidos 6 items de dados para cada linha da matriz.

As declarações DATA das linhas 300 a 320 colocam os dados em posições sucessivas do bloco de dados como se indica abaixo.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

17	18	19	20	21	22	23	24		

A declaração READ na linha 90 atribui valores à matriz A durante a execução do programa da seguinte forma:

K colunas

J linhas						
	1	2	3	4	5	6
	7	8	9	10	11	12
	13	14	15	16	17	18
	19	20	21	22	23	24

EXEMPLO 15: Um programa de ordenação alfabética para demonstrar a utilização de DIM, FOR——NEXT e IF ... THEN

Programa

```
010 REM   PROGRAMA DE ORDENACAO ALFABETICA
015 REM   ATE 100 NOMES
020 REM   ************************************************
030 REM   DECLARAR A DIMENSAO DO VECTOR E
035 REM   INICIALIZAR A ZERO O CONTADOR
040 DIM N$(101)
050 LET K = 0
060 PRINT  "ESCREVA OS NOMES A ORDENAR"
070 PRINT  "TERMINANDO PELA PALAVRA ULTIMO"
080 PRINT
090 REM   ESTABELECER O CICLO QUE VAI PREENCHER AS
095 REM   POSICOES DO VECTOR COM OS DADOS DE INPUT
100 FOR   M = 1 TO 101
110 INPUT N$(M)
120 REM   TESTAR O FIM DOS DADOS
130 IF N$(M) = "ULTIMO" THEN 200
140 REM   ADICIONAR 1 AO CONTADOR DE NOMES
150 K = K + 1
160 NEXT M
170 REM   FIM DO CICLO
180 REM   ************************************************
190 REM   ESTABELECER CICLOS ENCADEADOS PARA
195 REM   A ORDENACAO
200 FOR   M = 1 TO K -1
```



```
210 FOR   J = M + 1 TO K
220 REM   COMPARAR NOMES TROCANDO DE POSICOES
225 REM   QUANDO E ENCONTRADO
230 REM   UM NOME COM UM VALOR MENOR
240 IF N$(M) < = N$(J) THEN 280
250 LET X$ = N$(M)
260 LET N$(M) = N$(J)
270 LET N$(J) = X$
280 NEXT J
290 REM   FIM DO CICLO INTERIOR
300 NEXT M
310 REM   FIM DO CICLO EXTERIOR
320 PRINT
330 PRINT
340 PRINT "LISTA DE NOMES ORDENADA"
350 PRINT
360 REM   ESTABELECER O CICLO DE IMPRESSAO
365 REM   DO CONTEUDO DOS
370 REM   ELEMENTOS DO VECTOR AGORA ORDENADOS
375 REM   ALFABETICAMENTE
380 FOR J = 1 TO K
390 PRINT N$(J)
400 NEXT J
410 REM   FIM DO CICLO
420 END
```

Output

```
ESCREVA OS NOMES A ORDENAR
TERMINANDO PELA PALAVRA ULTIMO

? pedro
? sara
? victória
? joana
? simão
? helena
? ricardo
? último
```

LISTA DE NOMES ORDENADA

HELENA
JOANA
PEDRO
RICARDO
SARA
SIMAO
VICTORIA

Comentário

Repare-se que são declaradas 101 posições porque se torna necessária uma posição extra no fim da lista de nomes destinada a conter a palavra "último". Devem examinar-se cuidadosamente as declarações 250 a 270 onde se efectua a troca de nomes entre posições de memória, sem que se perca qualquer nome. Esta troca é tão fundamental como a sequência completa de ordenação em numerosos programas de computador.

Após a execução do ciclo FOR ... NEXT (declarações 100—160) o vector N$ contém

N$(1)	N$(2)	N$(3)	N$(4)	N$(5)	N$(6)	N$(7)	N$(8)
Pedro	Sara	Victória	Joana	Simão	Helena	Ricardo	Ultimo

A sequência de troca de posições, abaixo indicada, é executada pela primeira vez quando M=1 e J=4 (para J=1,2 e 3 salta-se por cima da sequência porque a condição em 240 é satisfeita)

```
250 LET X$ = N$(M)
260 LET N$(M) = N$(J)
270 LET N$(J) = X$
```

Conteúdo de X$ depois da execução de 250 (para M=1 e J=4)

X$
PEDRO

Vector N$ depois da execução de 260

N$(1)	N$(2)	N$(3)	N$(4)	N$(5)	N$(6)	N$(7)	N$(8)	X$
JOANA	Sara	Victória	Joana	Simão	Helena	Ricardo	Último	Pedro

Vector N$ depois da execução de 270

Joana	Sara	Victória	PEDRO	Simão	Helena	Ricardo	Último	Pedro

PEDRO e JOANA trocaram de lugares com sucesso ajudados pela posição auxiliar de memória X$.

Funções

Uma função é um procedimento definido que pode ser utilizado repetidamente a partir de um programa referenciando-o simplesmente pelo seu nome. A linguagem BASIC possui um determinado número de funções de biblioteca *standard*, predefinidas, para o cálculo de algumas das funções numéricas mais comuns (por exemplo, a determinação da raiz quadrada de um número). A linguagem permite igualmente que os programadores definam as suas próprias funções para seu uso pessoal pela utilização da declaração DEF (ver mais adiante).

Quando uma função é referenciada e executada é devolvido ao programa um único valor. Pode ser feita referência a uma função em qualquer ponto onde seja igualmente permitida uma expressão numérica. Pode encontrar-se isolada ou fazer parte de uma expressão.

Funções de biblioteca standard

Acede-se a estas funções especificando o seu nome seguido pelo chamado argumento da função entre parêntesis. Um argumento pode apresentar-se sob a forma de uma expressão numérica qualquer. Ao ser calculada, tal expressão produz um único valor que é então utilizado pela função como dados. A expressão numérica que constitui argumento de uma função pode incluir referências a outras funções.

Nas definições que apresentamos a seguir, X é o argumento da função. Lembre-se que este pode ser uma expressão numérica, qualquer que seja o seu grau de complexidade.

Funções trigonométricas

ATN(X) Devolve o arco-tangente de X em radianos, isto é, o ângulo cuja tangente é X, entre $-\pi/2$ a $+\pi/2$.
COS(X) Co-seno de X, com X em radianos.
SIN(X) Seno de X, com X em radianos.
TAN(X) Tangente de X, com X em radianos.

Funções exponenciais

EXP(X) Exponencial de X, isto é, o valor da base dos logaritmos naturais (e=2.71828) elevado à potência X.
LOG(X) Logaritmo natural de X. X deve ser maior que 0.
SQR(X) Raiz quadrada de X. X deve ser maior que 0.

Funções aritméticas

ABS(X) Valor absoluto de X, isto é, o valor de X se este for positivo ou o seu simétrico se for negativo.
INT(X) O maior inteiro que não seja maior que X.
SGN(X) Determina o sinal de X, devolvendo o valor 1 se $X > 0$, o valor 0 se $X = 0$ e o valor -1 se $X < 0$.

EXEMPLO 16: Introduzindo uma selecção de funções de biblioteca.

Programa

```
010 LET X = 12
020 LET Y = -5.85
030 LET A = SQR(X)
```



```
040 LET B = EXP(X)
050 LET C = LOG(X)
060 LET D = SGN(Y)
070 LET E = INT(Y + X)
080 PRINT "A RAIZ QUADRADA DE";X;"=";A
090 PRINT "O VALOR DE E ELEVADO A";X;"=";B
100 PRINT "O LOGARITMO DE";X;"=";C
110 PRINT
120 PRINT "O VALOR DEVOLVIDO COM BASE";
125 PRINT "NO SINAL DE Y =";D
130 PRINT "O MAIOR INTEIRO QUE NAO E MAIOR";
135 PRINT "QUE Y + X =";E
140 END
```

Output

```
A RAIZ QUADRADA DE 12 = 3.4641
O VALOR DE E ELEVADO A 12 = 162755.
O LOGARITMO DE 12 = 2.4891

O VALOR DEVOLVIDO COM BASE NO SINAL DE Y = -1
O MAIOR INTEIRO QUE NAO E MAIOR QUE Y + X = 6
```

Comentário

A possibilidade de aceder a uma biblioteca de funções poupa muito tempo e esforço ao programador. Se fosse necessário incluir todo o código para determinar a raiz quadrada de um número, a exponencial, etc., o programa tornar-se-ia não só muito mais longo como igualmente muito mais complicado.

Função utilitária

RND Selecciona o chamado pseudo-número aleatório (*random number*) seguinte de uma sequência predefinida de pseudo-números aleatórios própria da implementação. Estes números encontram-se entre os valores 0 e 1. Pode-se obter um ponto de partida imprevisível nesta sequência de números utilizando a declaração RANDOMIZE antes de ser feita referência à função RND.

Note-se que a definição mínima de BASIC não impõe qualquer argumento para a função RND. Contudo, algumas versões empregam um argumento para assegurar um maior controlo sobre o ponto de partida da sequência.

EXEMPLO 17: Introduzindo a função RND

Programa

```
10 PRINT "SIMULACAO DO LANCAMENTO DE UM DADO"
20 PRINT
30 REM  ESTABELECIMENTO DE UM CICLO PARA SIMULAR
35 REM  20 LANCAMENTOS
40 FOR  K = 1 TO 20
50 LET  A = INT(RND * 6) + 1
60 PRINT A;
70 NEXT K
80 END
```

Output

```
SIMULACAO DO LANCAMENTO DE UM DADO

2 3 6 1 4 4 4 3 4 6 3 2 4 2 2 4 6 6 6 4
```

Comentário

Examinemos a declaração 50. Recorde-se que um número extraído da sequência de números aleatórios pela função RND pertence ao intervalo entre 0 e 1. Multiplicando o número aleatório por 6 e truncando o resultado para fornecer um número inteiro, através da função INT, o número passa a ser 0, 1, 2, 3, 4 ou 5. Finalmente, adicionando 1 obtém-se 1, 2, 3, 4, 5 ou 6 que representam as seis faces de um dado.

Os números aleatórios são de extrema utilidade como dados de teste em alguns problemas. Podem ser igualmente úteis como forma de repetir a mesma sequência de números para ver o que acontece quando se faz uma modificação a um programa. Uma segunda passagem do programa forneceria os mesmos 20 números.

Declaração RANDOMIZE

A declaração RANDOMIZE modifica o ponto de partida da sequência de pseudo-números aleatórios que é acedida pela função RND. A forma como o faz é imprevisível de forma que um programa que contenha a declaração RANDOMIZE e uma referência à função RND obtém números diferentes cada vez que o programa é executado. A declaração RANDOMIZE tem a forma:

n RANDOMIZE (a forma RANDOM é igualmente aceitável)

Ao ser executada, é gerado um novo ponto de partida para a sequência de números aleatórios.

Adicionando RANDOMIZE ao Exemplo 17, a sequência de 20 lançamentos do dado será diferente cada vez que o programa é executado.

EXEMPLO 17A: RANDOMIZE antes de RDN

Programa

```
10 PRINT "SIMULALACAO DO LANCAMENTO DE UM DADO"
20 PRINT
25 RANDOMIZE
30 REM  ESTABELECIMENTO DE UM CICLO PARA SIMULAR
35 REM  20 LANCAMENTOS
40 FOR  K = 1 TO 20
50 LET  A = INT(RND * 6) + 1
60 PRINT A;
70 NEXT K
80 END
```

Output (primeira execução)

```
SIMULACAO DO LANCAMENTO DE UM DADO

1 3 6 2 5 6 5 3 6 4 3 5 2 5 2 2 1 6 5 1
```

Output (segunda execução)

```
SIMULACAO DO LANCAMENTO DE UM DADO

2 1 6 4 5 4 5 4 2 5 3 6 4 1 3 1 4 5 5 3
```

Note-se que a declaração RANDOMIZE não existe em todas as implementações de BASIC, embora se encontre definida como uma característica do BASIC Mínimo. Se não se encontra disponível é porque a sua função é conseguida através da existência de um argumento na função RND, isto é, RND(X) pode, com X menor que zero, gerar uma sequência diferente cada vez que for utilizada.

Função TAB

TAB é uma função especial que permite o controlo exacto da posição de impressão. Só pode ser utilizada dentro de uma declaração PRINT. TAB(X) move a posição de impressão para a coluna cujo número é obtido pelo cálculo de X arredondado para o inteiro mais próximo. O valor de X deve ser não-negativo.

Normalmente há 72 posições de impressão numeradas de 0 a 71. TAB(8) move a posição de impressão para o número 8, que é na realidade a nona (9) posição da esquerda para a direita, como se ilustra no exemplo seguinte.

EXEMPLO 18: Introduzindo TAB

Programa

```
010 PRINT "  ABCDEFGHIJKLMNOPQRSTUVWXYZ"
020 PRINT "  0 1 2 3 4 5 6 7 8 9 0 1 2  3 4 5 6 7 8 9 0 1 2  3 4 5"
030 LET X=8
040 PRINT TAB(X);"H";TAB(25);"Z"
050 END
```

Output

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0 1 2 3 4 5 6 7 8 9 0 1 2  3 4 5 6 7 8 9 0 1 2  3 4 5
        H                         Z
```


Comentário

A implementação de BASIC que vier a utilizar pode utilizar números desde 1 a 72. Experimente com este exemplo para determinar exactamente o significado de TAB(25). Não será possível vir a ter exacto controlo sobre a impressão até conhecer exactamente o funcionamento de TAB.

Utiliza-se um ponto-e-vírgula como separador a seguir ao chamamento de TAB de forma a que a impressão do próximo item se faça na posição especificada.

A função TAB dá ao programador a capacidade de controlar exactamente a posição em que ficará o *output* numa linha. Isto é particularmente útil quando se deseja apresentar os resultados sob a forma de um gráfico.

EXEMPLO 19: Fazendo gráficos com TAB

Programa

```
10 REM   UTILIZACAO DA FUNCAO TAB PARA FAZER
20 REM   O GRAFICO DE UMA CURVA
30 FOR   K = O TO 2 * 3.14159 STEP 0.75
40 PRINT TAB(20 * (1 + SIN(K)));"*"
50 NEXT K
60 END
```

Output

```
                    *
                *
                        *
                    *
                *
            *
        *
    *
            *
```

Comentário

Depois de calculado o seno de K, é-lhe adicionado 1. O resultado é multiplicado por 20 e arredondado para o inteiro mais próximo pela função TAB.

Este valor arredondado determina a posição de impressão da constante "*". Cada vez que o ciclo FOR é executado, o asterisco é impresso numa posição diferente e numa linha diferente.

Funções definidas pelo utilizador

O BASIC permite a definição, pelo utilizador, de funções numa única linha pela utilização da declaração DEF. Uma vez definida, o BASIC calcula o valor da função sempre que o nome da função apareça no programa.

Declaração DEF

A declaração DEF é utilizada tanto para fornecer o nome de uma função, como a definição completa da função. Tem a seguinte forma:

n DEF FNα (p_1, p_2, ..., p_i) = *e*

onde FNα é o nome da função, p_1, ..., p_i é a lista de argumentos mudos para a definição da função, sob a forma de uma ou mais variáveis numéricas simples, e *e* é a definição matemática da função sob a forma de uma expressão numérica.

As duas primeiras letras de um nome de função devem sempre consistir nas maiúsculas FN. A terceira letra, que distingue o nome de uma função do de outra, pode ser qualquer letra do alfabeto.

As variáveis colocadas na lista de argumentos mudos *p* são as variáveis que aparecem na expressão e que necessitam de possuir valores no momento da execução do programa em que é referenciada a função, de forma a que a função possa ser calculada. São apelidados de argumentos mudos porque quando a função é calculada os valores desses argumentos são os especificados através dos argumentos na chamada da função e não na sua definição. Os argumentos mudos na definição fornecem simplesmente uma maneira de passagem desses valores.

EXEMPLO 20: Introduzindo a declaração DEF

Programa

```
010 REM   EXEMPLO DE COMO CRIAR UMA FUNCAO
020 REM   E DE COMO FUNCIONAM OS ARGUMENTOS
030 REM
040 REM   FUNCAO PARA CONVERTER DOLARES EM LIBRAS
050 REM   FND(X) = X/2.4065
060 REM   FUNCAO PARA CONVERTER IENES EM LIBRAS
070 DEF   FNY(X) = X/524
080 REM   TESTE DAS FUNCOES
090 LET   C = 100
100 LET   D = FND(C)
110 LET   Y = FNY(C)
120 PRINT C;"DOLARES =";D;"LIBRAS"
130 PRINT
140 PRINT C;"IENES =";Y;"LIBRAS"
150 END
```

Output

```
100 DOLARES = 41.5541 LIBRAS
100 IENES = .19084 LIBRAS
```

Comentário

A função para converter dólares em libras é definida na declaração 50 e a função para converter iénes em libras na declaração 70. As funções são depois chamadas, nas linhas 100 e 110, com um valor fornecido no argumento C (por isso, por vezes chamado argumento real). O valor C, ou seja 100, é passado para cada uma das funções através do argumento mudo X em cada uma das definições. Assim, para o cálculo de cada função X = C = 100.

A definição de uma função deve surgir no programa antes de ser feita qualquer referência à função, isto é, o número de linha da declaração DEF deve ser inferior ao

número de linha da declaração em que surgir a primeira referência à função.

A expressão numérica que forma a definição matemática pode conter outras variáveis para além das que aparecem na lista de argumentos mudos. A fim de o ilustrar, o Exemplo 20 foi modificado e reproduz-se mais abaixo. As variáveis R1 e R2 são introduzidas nas definições das funções em substituição das constantes 2.4065 e 524. Os valores de R1 e R2 são atribuídos através de uma declaração INPUT.

EXEMPLO 20A: Funções definidas pelo utilizador

Programa

```
010 REM   EXEMPLO DE COMO CRIAR UMA FUNCAO
020 REM   E DE COMO FUNCIONAM OS ARGUMENTOS
030 REM
040 REM   FUNCAO PARA CONVERTER DOLARES EM LIBRAS
050 DEF   FND(X)=X/R1
060 REM   FUNCAO PARA CONVERTER IENES EM LIBRAS
070 DEF   FNY(X)=X/R2
080 REM   TESTE DAS FUNCOES
082 PRINT "INTRODUZA COTACOES DO DOLAR E DO IENE"
085 INPUT R1, R2
087 PRINT
090 LET   C=100
100 LET   D=FND(C)
110 LET   Y=FNY(C)
120 PRINT C;"DOLARES=";D;"LIBRAS"
130 PRINT
140 PRINT C;"IENES=";Y;"LIBRAS"
150 END
```

Output

```
INTRODUZA COTACOES DO DOLAR E DO IENE
? 2.4065,524
100 DOLARES = 41.5541 LIBRAS
100 IENES = .19084 LIBRAS
```


Comentário

Neste exemplo as cotações das moedas são introduzidas por teclado no momento em que o programa é executado, sendo atribuídas a R1 e R2 através da declaração INPUT na linha 85. Note-se que a variável R1 ocorre na definição da função FND na linha 50 e R2 na definição da função FNY na linha 70. Quando a função FND é chamada (linha 100) o valor de R1 encontra-se disponível para o cálculo da função apesar de R1 não ser um argumento da função; o mesmo se passa para o valor de R2 em FNY na linha 110.

Uma definição de função pode fazer referência a outras funções definidas pelo utilizador ou a quaisquer das funções de biblioteca *standard*.

EXEMPLO 21: Outra função definida pelo utilizador

Programa

```
010 REM   FUNCAO PARA ARREDONDAR VALORES
020 REM   PARA DUAS CASAS DECIMAIS
030 DEF   FNA(Y) = INT(100 * Y + 0.5) / 100
040 FOR   K = 1 TO 3
050 PRINT "N =";
060 INPUT N
070 PRINT "VALOR ARREDONDADO =";FNA(N)
080 NEXT X
090 END
```

Output

```
N = ? 17.32715
VALOR ARREDONDADO = 17.33
N = ? 99.0941
VALOR ARREDONDADO  = 99.09
N = ? 45.555
VALOR ARREDONDADO = 45.56
```

Note-se que algumas implementações de BASIC permitem a construção de funções multilineais utilizando a declaração DEF para definir o nome da função e FNEND para assina-

lar o fim da definição da função. As linhas entre DEF e FNEND representam a definição matemática da função. Algumas implementações permitem ainda a definição pelo utilizador de funções de cadeia.

Ficheiros

Quando os resultados de um programa são necessários como dados de *input* para outro, é útil ter a possibilidade de deixar no sistema uma cópia da informação pronta a ser utilizada de novo quando necessária. A informação retida desta maneira é armazenada num suporte permanente, tal como um disco magnético, e o conjunto de informação passa a ser chamado *ficheiro de dados*.

Os mecanismos para salvaguardar e recuperar ficheiros de suportes permanentes são essencialmente características próprias do sistema computador que se está a utilizar, e será necessário aprender alguns comandos especiais do sistema. De uma forma geral, dá-se a conhecer um ficheiro ao sistema atribuindo-lhe um *nome de ficheiro*. O sistema conserva um catálogo dos nomes de todos os ficheiros permanentes tornando-se portanto possível recuperar qualquer ficheiro através do seu nome.

As diferentes versões de BASIC incluem normalmente um certo número de comandos que permitem a criação de ficheiros de dados externos e a sua utilização posterior dentro de programas BASIC. As versões diferem também na forma como é feito o tratamento de ficheiros e o seu controlo. As características que a seguir se explicam nesta secção existirão quase com certeza na sua versão, embora possam variar os pormenores das especificações.

Um comando WRITE *ficheiro* coloca dados num ficheiro externo.
Um comando READ *ficheiro* lê dados de um ficheiro

externo onde foram previamente colocados por WRITE *ficheiro*.
Um comando PRINT *ficheiro* coloca dados num ficheiro externo.
Um comando INPUT *ficheiro* lê dados de um ficheiro externo onde foram colocados por PRINT *ficheiro*.
WRITE e PRINT colocam, ambos, dados em ficheiros embora com formatos diferentes.
O comando READ só pode compreender o formato WRITE e INPUT o formato PRINT.

Antes de se poder escrever ou ler de um ficheiro, é necessária uma declaração FILE para relacionar um número de ficheiro com um nome de ficheiro em particular. O número de ficheiro é geralmente chamado *ordinal* dum ficheiro. Qualquer WRITE, READ, PRINT ou INPUT utilizado num programa é "ligado" ao ficheiro externo apropriado utilizando o ordinal desse ficheiro. Note-se que o ordinal 1 é utilizado para o primeiro ficheiro a ser referenciado, 2 para o segundo, etc. O limite do número de ficheiros que podem ser utilizados num programa é uma característica própria da implementação que está a ser utilizada.

Como se verá nos exemplos que se seguem, o carácter especial # é utilizado no formato de todas as declarações de BASIC para tratamento de ficheiros.

EXEMPLO 22: Introduzindo FILE # e WRITE #

Programa

```
10 FILE # 1 = "TESTEA"
20 FOR K = 1 TO 3
30 INPUT N
40 T = N * 2.5
50 WRITE # 1, N, T
60 NEXT K
70 END
```

Output

```
? 40
? 2
? 27
```

Comentário

A declaração 10 especifica o nome do ficheiro que deve estar disponível para utilização durante a execução do programa. Declara igualmente o ordinal que actuará como referência dentro do programa ao ficheiro externo. "TESTEA" é o nome pelo qual o ficheiro é conhecido para o sistema e, note-se, que aparece uma única vez no programa. O ordinal, contudo, aparece sempre que há necessidade de aceder ao ficheiro externo; neste caso, na declaração 50 para assegurar que os valores associados a N e T são colocados no ficheiro externo. Repare-se que os únicos valores que aparecem na saída são os introduzidos por teclado pelo programador durante a execução repetida da declaração 30. O verdadeiro *output* não é visível dado que os resultados foram copiados para um ficheiro. Como prova de como foram realmente copiados, são chamados pelo próximo programa-exemplo.

EXEMPLO 23: Introduzindo READ # e NODATA

Programa

```
10 FILE # 1 = "TESTEA"
20 READ # 1, A, B
30 PRINT A, B
40 NODATA # 1, 60
50 GO TO 20
60 END
```

Output

```
 40          100
 2           5
 27          67.5
```


Comentário

A declaração 10 especifica o nome do ficheiro externo que vai ser utilizado no programa. A declaração 20 lê valores encontrados no ficheiro associado ao ordinal 1, isto é, TESTEA que fora criado no Exemplo 22, e guarda-os em A e B. Estes valores são depois impressos pela declaração 30. Repare-se na utilização da declaração NODATA #. Podem existir implementações com a declaração RESTORE # ou RESET # para reposicionar o apontador no início do ficheiro.

Note-se que as regras que se aplicam no formato das declarações READ, PRINT e INPUT vulgares também se aplicam, em geral, às declarações correspondentes para tratamento de ficheiros, isto é, as variáveis especificadas devem corresponder ao tipo de dados a que se referem.

Um ficheiro de dados pode igualmente ser preparado como uma actividade totalmente externa a qualquer programa pela introdução dos dados por teclado. Para tal, seria necessário um comando de sistema para dar um nome ao ficheiro e os dados seriam simplesmente digitados numa linha de cada vez. Alguns sistemas exigem que cada linha comece com um número de linha, mas, mesmo quando não for este o caso, é muitas vezes útil incluir números de linha como forma de ordenar o ficheiro. O exemplo seguinte assume serem necessários números de linha:

novo, endereço (comando de sistema especial, próprio do sistema utilizado para passar este exemplo)

```
10 SRA. D. E. WARNER
20 VIVENDA SOL
30 P. SRA. DA ROCHA
40 ARMACAO DE PERA
50 ALGARVE
60 FIM ENDERECO
```

EXEMPLO 24: Leitura de dados de um ficheiro, preparado externamente, por intermédio de INPUT

Programa

```
010 FILE # 1 = "ENDERECO"
020 LET   L$ = "FIM ENDERECO"
030 FOR   J = 1 TO 8
040 INPUT # 1, N, A$(J)
050 IF   A$(J) = L$ THEN 70
060 NEXT J
070 FOR   K = 1 TO J - 1
080 PRINT A$(K)
090 NEXT K
100 END
```

Output

```
SRA. D. E. WARNER
VIVENDA SOL
P. SRA. DA ROCHA
ARMACAO DE PERA
ALGARVE
```

Comentário

A declaração 40 aceita dados de *input* do ficheiro externo associado com o ordinal 1, neste caso o ficheiro de nome ENDERECO criado anteriormente por introdução directa dos dados por teclado. Note-se que os números de linha do ficheiro de dados são lidos e armazenados de cada vez em N. A variável N não é utilizada em mais nenhum lugar no programa. A sua finalidade é garantir que os dados válidos em cada linha sejam recuperados e guardados no quadro A$. Note-se igualmente que, neste exemplo, o teste de fim de dados no ficheiro é realizado utilizando um *item* de dados do próprio ficheiro (a linha 60 do ficheiro).

Em conclusão, deve-se salientar de novo que o tratamento de ficheiros depende do sistema que se está a utilizar e da implementação de BASIC disponível. Os exemplos aqui incluídos sugerem as potencialidades de tratamento de ficheiros do BASIC. Não foram incluídos os comandos próprios do sistema para salvaguardar e referenciar ficheiros.

Resumo de declarações matriciais

Muitas versões de BASIC possuem capacidades incorporadas para levar a cabo determinadas manipulações de matrizes. As capacidades disponíveis podem ser algumas ou todas as que são abordadas neste resumo.

MAT READ	Preenche matrizes por ordem de linha com dados obtidos do bloco de dados fornecido por declarações DATA.
MAT INPUT	Permite preencher matrizes uma linha de cada vez por introdução de dados através de terminal durante a execução do programa.
MAT PRINT	Imprime matrizes por ordem de linha.

EXEMPLO 25: Introduzindo MAT READ e MAT PRINT

Programa

```
10 DIM X(3,3)
20 MAT READ X
30 MAT PRINT X
40 DATA 5, 10, 15, 20, 25
50 DATA 30, 35, 40, 45
60 END
```

Output

```
 5            10           15

 20           25           30

 35           40           45
```

Comentário

A declaração 20 é suficiente para atribuir os nove *items* de dados ao quadro bidimensional ou matriz X, e a declaração 30 imprime o conteúdo de toda a matriz. Repare-se que é necessário especificar as dimensões da matriz antes de se

poder utilizar a declaração MAT, como é feito na linha 10. Note-se igualmente que a declaração MAT PRINT faz sair as suas colunas como se o separador de impressão fosse uma vírgula, e que é deixada uma linha em branco entre as linhas da matriz.

EXEMPLO 26: Introduzindo MAT INPUT

Programa

```
10 DIM  A(2,3)
20 PRINT "INTRODUZA OS ELEMENTOS DA MATRIZ";
25 PRINT "LINHA A LINHA"
30 MAT  INPUT A
40 PRINT
50 MAT  PRINT A;
60 END
```

Output

```
INTRODUZA OS ELEMENTOS DA MATRIZ LINHA A LINHA
? 8,3,7
? 6,1,6

 8  3  7

 6  1  6
```

Comentário

O programa faz uma pausa na execução aguardando a introdução dos dados de input na declaração 30. A cada pedido de dados é introduzida, pelo teclado, uma linha completa como especificado na declaração DIM. Notar que o ponto-e-vírgula no fim da linha 50 faz com que a matriz seja impressa sem espacejamentos.

Podem executar-se várias operações aritméticas sobre matrizes uma vez que elas se encontrem devidamente preenchidas com dados.

MAT *Atribuição*	MAT A = B
MAT *Adição*	MAT C = A + B

MAT *Subtracção*	MAT C = A – B
MAT *Multiplicação escalar*	MAT C = (*n*) * A
MAT *Multiplicação*	MAT D = E * F

Nos exemplos anteriores, as matrizes A, B e C devem sempre possuir dimensões de quadro idênticas. No exemplo de multiplicação de matrizes, o número de colunas da matriz E deve ser igual ao número de linhas de F; o número de linhas de D deve ser igual ao número de linhas de E, e o número de colunas de D deve ser igual ao número de colunas de F. Na multiplicação escalar, *n* pode ser qualquer expressão aritmética.

Existe igualmente um certo número de funções MATriciais especiais.

MAT IDN	Gera uma matriz identidade consistindo em 1's ao longo da diagonal principal e 0's nos restantes elementos.
MAT CON	Gera uma matriz consistindo apenas de 1's.
MAT ZER	Gera uma matriz consistindo apenas de 0's.
MAT TRN	Transpõe uma matriz ao longo da diagonal principal.
MAT INV	Inverte uma matriz quadrada.

EXEMPLO 27: Miscelânea MATricial

Programa

```
010 REM  DEFINIR A DIMENSAO DE MATRIZES
020 DIM  A(3,3), X(3,3), Y(3,3), D(3,3), E(3,3)
030 REM  LER DADOS DE UM BLOCO DE DADOS
040 MAT  READ X, Y
050 PRINT "MATRIZ X"
060 MAT  PRINT X
070 PRINT "MATRIZ Y"
080 MAT  PRINT Y
090 REM  SOMAR MATRIZES
100 MAT  A = X + Y
110 REM  TRANSPOR UMA MATRIZ
```

```
120 MAT   D = TRN(Y)
130 REM   MULTIPLICACAO ESCALAR
140 MAT   E = (3) * X
150 PRINT "MATRIZ A = X + Y"
160 MAT   PRINT A
170 PRINT "MATRIZ D = MATRIZ Y TRANSPOSTA"
180 MAT   PRINT D
190 PRINT "MATRIZ E = 3 * MATRIZ X"
200 MAT   PRINT E;
210 REM   DADOS PARA O BLOCO DE DADOS
220 DATA 1, 5, 7, 2, 8, 3, 9, 6, 4
230 DATA 60, 20, 70, 40, 50, 10, 90, 30, 80
240 END
```

Output

```
MATRIZ X
 1          5          7

 2          8          3

 9          6          4

MATRIZ Y
 60         20         70

 40         50         10

 90         30         80

MATRIZ A = X + Y
 61         25         77

 42         58         13

 99         36         84

MATRIZ D = MATRIZ Y TRANSPOSTA
 60         40         90

 20         50         30

 70         10         80
```



```
MATRIZ E = 3 * MATRIZ X
 3          15         21

 6          24         9

 27         18         12
```

Outros exemplos

Problema Determinar a variância de um conjunto de números, num máximo de 50.

Programa

```
010 REM   PROGRAMA CALCULO DA VARIANCIA
015 REM   DIMENSIONAR O QUADRO DE VALORES
020 REM   LER O NUMERO DE DADOS
030 DIM   A(50)
040 READ N
050 REM   INICIALIZAR SOMATORIOS
060 LET   S1 = 0
070 LET   S2 = 0
080 REM   CICLO PARA LER E SOMAR VALORES
090 FOR   J = 1 TO N
100 READ A(J)
110 LET   S1 = S1 + A(J)
120 NEXT J
130 REM   CALCULAR O VALOR MEDIO
140 LET   M = S1 / N
150 REM   CICLO PARA SOMAR OS VALORES ABSOLUTOS
160 REM   DOS DESVIOS DE CADA VALOR EM RELACAO
170 REM   AO VALOR MEDIO JA CALCULADO
180 FOR   K = 1 TO N
190 LET   S2 = S2 + ABS(A(K) — M)
200 NEXT K
210 REM   CALCULAR A VARIANCIA
220 LET   D = S2 / N
230 PRINT "A VARIANCIA =";D
240 PRINT "E O VALOR MEDIO =";M
250 DATA 11
260 DATA 56, 75, 41, 80, 67, 30, 17, 48, 58, 83, 24
270 END
```

Output

```
A VARIANCIA = 18.7603
E O VALOR MEDIO = 52.6364
```

Problema Dada uma série de artigos de "stock" com os respectivos preços de venda e custos líquidos, elabore uma tabela para apresentar estes dados e indicar o lucro realizado com a venda de cada elemento, assim como o lucro expresso sob a forma de percentagem.

Programa

```
010 REM  PROGRAMA INVENTARIO DE "STOCK"
020 PRINT TAB(25);"TABELA DE INVENTARIO"
030 PRINT
040 PRINT "ARTIGO";TAB(12);"P. VENDA";
045 PRINT TAB(25);"CUSTO LIQ.";TAB(40);"LUCRO";
050 PRINT TAB(51);"PERCENT."
060 LET  T$ = "STOP"
070 READ N$
080 IF  N$ = T$ THEN 999
090 READ X, Y
100 LET  P = X — Y
110 LET  C = P / X * 100
120 PRINT N$, X, Y, P, C
130 GO TO 70
140 DATA "CANETA", 0.22, 0.16
150 DATA "LAPIS", 0.12, 0.085
160 DATA "TINTA", 0.45, 0.33
170 DATA "REGUA", 0.27, 0.19
180 DATA "AGRAFADOR", 2.35, 1.54
190 DATA "STOP"
999 END
```

Output

TABELA DE INVENTARIO

ARTIGO	P. VENDA	CUSTO LIQ.	LUCRO	
CANETA	.22	.16	.06	27.2727
LAPIS	.12	.085	.035	29.1667
TINTA	.45	.33	.12	26.6667
REGUA	.27	.19	.08	29.6296
AGRAFADOR	2.35	1.54	.81	34.4681

Problema Dados um capital inicial, o período de depósito em anos e uma taxa de juro que não varia durante esse período, calcular o capital acumulado, ano a ano. Esboce uma solução que permita introduzir os dados de uma forma interactiva.

Programa

```
010 PRINT "CAPITAL INICIAL"
015 PRINT "(VALOR NEGATIVO PARA PARAR)"
020 INPUT P
030 IF P < 0 THEN 190
040 PRINT "TAXA DE JURO"
050 INPUT R
060 PRINT "NUMERO DE ANOS"
070 INPUT Y
080 PRINT
090 PRINT
100 PRINT "TABELA DE CAPITAL ACUMULADO"
110 PRINT P;"ESCUDOS";"A";R;"POR CENTO"
120 PRINT "ANO", "CAPITAL ACUM."
130 FOR  K = 1 TO Y
140 LET  A = P * (1 + R / 100) ** K
150 PRINT K, A
160 NEXT  K
170 PRINT
180 GO TO 10
190 END
```

Output

```
CAPITAL INICIAL
(VALOR NEGATIVO PARA PARAR)
? 2000
TAXA DE JURO
? 11.5
NUMERO DE ANOS
? 5

TABELA DE CAPITAL ACUMULADO
 2000 ESCUDOS A 11.5 POR CENTO
ANO          CAPITAL ACUM.
 1            2230.
 2            2486.45
```

```
 3          2772.39
 4          3091.22
 5          3446.71

CAPITAL INICIAL
(VALOR NEGATIVO PARA PARAR)
? 750
TAXA DE JURO
? 17
NUMERO DE ANOS
? 3

TABELA DE CAPITAL ACUMULADO
  750 ESCUDOS A 17 POR CENTO
ANO          CAPITAL ACUM.
 1             877.5
 2             1026.67
 3             1201.21

CAPITAL INICIAL
(VALOR NEGATIVO PARA PARAR)
? — 1
```

Problema Preparar uma tabela de conversão de libras esterlinas para coroas, florins, francos e marcos.

Programa

```
010 REM   PROGRAMA DE CONVERSAO DE MOEDA
020 PRINT "TABELAS DE CONVERSAO DE MOEDA"
030 PRINT "**********************************************"
040 PRINT
050 REM   INTRODUCAO DA DATA
060 PRINT "DATA";
070 INPUT D$
080 REM   K: COROAS   G: FLORINS
090 REM   F: FRANCOS   M: MARCOS
100 PRINT "TAXAS DE CAMBIO"
105 PRINT "COROAS, FLORINS, FRANCOS, MARCOS"
110 INPUT K, G, F, M
115 PRINT "**********************************************"
120 PRINT "BASEADA NAS TAXAS DE CAMBIO VALIDAS EM"
```



```
125 PRINT D$
130 PRINT
140 PRINT "LIBRAS", "COROAS", "FLORINS",
145 PRINT "FRANCOS", "MARCOS"
150 PRINT
160 REM   ESTABELECER PARAMETROS PARA PASSAR
165 REM   PARA A ROTINA DE CONVERSAO
170 REM   S: VALOR INICIAL E: VALOR FINAL
180 REM   P: VALOR DO INCREMENTO
190 LET   S = 1
200 LET   E = 9
210 LET   P = 1
220 GO SUB 500
230 REM   REESTABELECER OS PARAMETROS
240 LET   S = 10
250 LET   E = 100
260 LET   P = 10
270 GO SUB 500
275 GO TO 999
280 REM   CICLO PARA CALCULAR VALORES
290 REM   E IMPRIMI-LOS
500 FOR   J = S TO E STEP P
510 LET   C1 = J * K
520 LET   C2 = J * G
530 LET   C3 = J * F
540 LET   C4 = J * M
550 PRINT J, C1, C2, C3, C4
560 NEXT  J
570 RETURN
999 END
```

Output

```
TABELAS DE CONVERSAO DE MOEDA
**********************************************************

DATA? "16 DE OUTUBRO DE 1980"
TAXAS DE CAMBIO
COROAS, FLORINS, FRANCOS, MARCOS
?13.42,4.72,10.02,4.35
**********************************************************
BASEADA NAS TAXAS DE CAMBIO VALIDAS EM
16 DE OUTUBRO DE 1980
```

LIBRAS	COROAS	FLORINS	FRANCOS	MARCOS
1	13.42	4.72	10.02	4.35
2	26.84	9.44	20.04	8.7
3	40.26	14.16	30.06	13.05
4	53.68	18.88	40.08	17.4
5	67.1	23.6	50.1	21.75
6	80.52	28.32	60.12	26.1
7	93.94	33.04	70.14	30.45
8	107.36	37.76	80.16	34.8
9	120.78	42.48	90.18	39.15
10	134.2	47.2	100.2	43.5
20	268.4	94.4	200.4	87.
30	402.6	141.6	300.6	130.5
40	536.8	188.8	400.8	174.
50	671.	236.	501.	217.5
60	805.2	283.2	601.2	261.
70	939.4	330.4	701.4	304.5
80	1073.6	377.6	801.6	348.
90	1207.8	424.8	901.8	391.5
100	1342.	472.	1002.	435.

ÍNDICE

Este livro acabou de se imprimir
em 1985
para a
EDITORIAL PRESENÇA, LDA.
na
Empresa Gráfica Feirense, Lda.
Vila da Feira