JEREMY RUSTON

APRENDA PASCAL

no seu microcomputador

TEMPOS LIVRES

A. MARQUES
84

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Peter 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 I, *Edwin Smith*
10. ABC dos Estilos, da Arquitectura ao Mobiliário, *A. Aussel*
11. Fotografia — Técnicas e Truques II, *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 da Criação de Canários, *Curt Af Anehjelm*
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, *Fabián 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 Person*
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 das Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *Luis Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*

67. Doces e Conservas de Fruta, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Roy Genders*
70. Calendário do Horticultor, *Luís Alsina Grau*
71. Jogos Electrónicos, *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, *Alejo Rigau*
73. Aprenda Televisão a Cores, *Gordon J. King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Poda de Árvores e Arbustos, *Roy 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, *Suzanne Lapointe*
83. Windsurf — Prancha à Vela, *Ernstfried Prade*
84. Raças de Cães, *O. Hasseldfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia, Embalsamamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging — Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos TV, *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie de 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. A Criação Doméstica de Coelhos, *C. F. Snow*
100. Iniciação ao Futebol, *Wieland Männle* e *Heinz Arnold*
101. Horóscopos Chineses, *Georg Haddenbach*
102. Guia Prático de Marcenaria, *Charles H. Hayward*
103. Andebol, *Fritz* e *Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perus, Pintadas e Codornizes, *Jérome Sauze*
106. Crepes — Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott* e *M. Baker*
110. Futebol — Técnica e Táctica, *Kurt Lavall*
111. A Vaca Leiteira, *Colin T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*
113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marieanne Ränk*
115. O Cubo Para Quem Já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, *M. Mrowka e W. J. Weber*
117. Gansos e Patos, *Marie Mourithe*
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, *Goger Hunt*
134. Introdução à Electrónica Digital, *Ian R. Sinclair*
135. ABC do Vídeo, *David Matthewson*
136. Fotografia em Movimento, *Don Morley*
137. Guia Prático de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *Sidney F. Ray*
139. Guia Prático 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, *Pratick 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 Harwood*
151. As Linhas da Mão, *Rosè Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *Simpson* e *Terrell*
154. Z 80 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*

JEREMY RUSTON

APRENDA PASCAL NO SEU MICROCOMPUTADOR

EDITORIAL PRESENÇA

Título original
LEARN PASCAL ON YOUR BASIC MICRO

Tradução de Conceição Jardim e Eduardo Nogueira
Capa de António Marques

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

I

INTRODUÇÃO AO PASCAL

A escolha da linguagem de programação de um computador é, na melhor das hipóteses, confusa. Depois de se ter escolhido uma linguagem (como o Pascal), surge um segundo problema: o de aprender essa linguagem. Este livro é portanto concebido para servir dois propósitos: ajudá-lo a decidir se o Pascal é ou não uma linguagem adaptável às suas necessidades, e ensinar-lhe a programação em Pascal se assim acontecer.

Para utilizar bem este livro é necessário possuir um computador, utilizando o pseudo-compilador Pascal fornecido no final da obra ou um compilador Pascal profissional. No primeiro caso, o leitor estará apenas a utilizar uma parte do Pascal.

O livro encontra-se portanto dividido em duas partes. A primeira interessa a ambos os tipos de leitores, enquanto que a segunda trata apenas daquelas características do Pascal não suportadas pelo compilador fornecido neste livro, só podendo ser usadas com eficácia por um leitor que disponha de um "verdadeiro" compilador de Pascal. No momento em que atingir a divisão entre estas duas partes, o leitor já estará em condições de decidir se vale ou não a pena adquirir uma versão comercial de Pascal para o seu computador, se não possuir ainda uma. Nestas condições, se verificar que não gosta de programar em Pascal, não se verá forçado a pagar o compilador antes de o descobrir.

Se tenciona utilizar este livro com o meu compilador (incluído no final da obra), deve antes do mais ler o apêndice que o descreve. Este apêndice dir-lhe-á qual dos compiladores listados deverá usar, e descreve o seu modo de funcionamento. Ao longo da primeira parte do livro todos os comentários dirigidos exclusivamente ao leitor que usa o meu compilador estarão encerrados em parêntesis rectos.

Se está a utilizar um compilador comercial, é difícil fornecer-lhe aqui quaisquer indicações precisas sobre o modo de o usar, dado que cada um funciona a seu modo. O manual que acompanha o compilador dar-lhe-á as informações necessárias. Um problema dos compiladores comerciais consiste em os seus autores tenderem a ampliar o Pascal. Estas extensões surgem na forma de novas ordens e modos "melhorados" de fazer as mesmas coisas. Na vasta maioria dos casos, estas extensões não interferem no Pascal "standard" em que este livro se baseia. Algumas dessas extensões são aliás praticamente universais, sendo mencionadas nos locais apropriados deste livro.

A sequência habitual de escrita e execução de um programa Pascal assemelha-se à seguinte:

1) É necessário introduzir o programa no computador. Os compiladores de Pascal raramente lhe permitem fazê-lo; o leitor ver-se-á forçado a adquirir um programa separado, designado por "editor" (programa de montagem) para este efeito. Um "editor" é um programa de uso geral que lhe permite criar documentos através do teclado (quer se trate de cartas para a sua avó ou de programas Pascal), montá-los e finalmente gravá-los em memórias periféricas. Os programas em causa são normalmente bastante sofisticados, permitindo deslocar blocos inteiros de programa, ou procurar palavras específicas numa fracção de segundo. Um exemplo do "editor" é o WordMaster (para os computadores CP/M).

2) Deve-se dar ordem ao compilador para compilar o programa. Isto é feito muitas vezes deixando o programa de montagem e invocando directamente o compilador através do teclado. O compilador tirará então o seu programa da memória periférica e convertê-lo-á num código interno, que em seguida é igualmente enviado para o suporte magnético.

3) O programa compilado pode em seguida ser colocado em execução escrevendo algumas ordens no teclado.

É imediatamente óbvio a partir das considerações feitas que no caso de se cometer um erro num programa Pascal é necessário voltar ao programa de montagem para corrigi-lo e repetir em seguida todo o processo. Isto faz perder bastante tempo em relação ao necessário para escrever e corrigir programas BASIC. O único modo de superar este problema consiste em garantir que o seu programa não contém nenhum erro!

O Pascal foi concebido deste modo porque nos anos 1969/70 os computadores eram tão caros que o único modo eficaz de escrever programas consistia em fazê-lo longe do computador e introduzi-los neste apenas depois de terem sido estudados no papel. O simples facto de o Pascal ter sobrevivido a este laborioso processo é já um testemunho da elegância e utilidade desta linguagem...

Todos os programas Pascal, tanto os simples como os complexos, possuem a mesma forma genérica:

PROGRAM EXEMPLO(ENTRADA,SAÍDA)
—
—
—
BEGIN
—
—

—

END.

A primeira secção de traços é a parte de inicialização do programa, que indica ao computador tudo o que deve conhecer para começar a executar o programa, isto é, o conjunto de instruções que aparecem depois na segunda secção de traços.

As palavras impressas em letra negra são designadas "palavras reservadas". Estão reservadas para fins especiais, não sendo possível usá-las para qualquer outra coisa. A maior parte das outras palavras que ocorrem em programas Pascal são chamadas "identificadores".

A linha:

PROGRAM EXEMPLO(ENTRADA,SAÍDA);

é chamada "título do programa". Fornece ao computador várias informações preliminares de que este necessita para poder iniciar a execução do programa — como o seu nome. Neste exemplo, o nome de programa é "Exemplo". O leitor pode dar aos seus programas qualquer nome, desde que este:

a) Não seja uma palavra reservada (estas variam de compilador para compilador, pelo que deverá verificar no manual do compilador quais as palavras a evitar);
b) Comece por uma letra;
c) O resto dos caracteres seja uma combinação de letras e algarismos.

É uma boa ideia dar aos programas os nomes usados como nome de ficheiro para os guardar na disquette ou noutro suporte magnético.

O nome do programa é um exemplo de identificador. Mesmo que o computador não se importe com o nome que você

dá ao seu programa, é lógico usar nomes que dêem uma ideia do objectivo do programa. Vejamos alguns exemplos:

PROGRAM IMPOSTO(ENTRADA,SAÍDA);
PROGRAM XADREZ(ENTRADA,SAÍDA);
PROGRAMA RAIZQUAD(ENTRADA,SAÍDA);

Como o caracter "espaço" não é uma letra do alfabeto, o nome não o pode conter. Isto pode tornar difíceis de ler nomes como o último apresentado acima. Por esta razão muitas versões de Pascal permitem o uso do caracter 'sublinhado' (_) nos identificadores. Este carácter é então usado em vez do espaço para facilitar a leitura dos identificadores. O último exemplo dado poderia portanto ser escrito:

PROGRAM RAIZ_QUAD(ENTRADA,SAÍDA);

Os nomes seguintes são todos incorrectos porque o identificador contém sempre alguma incorrecção:

PROGRAM 2HORÁRIO(ENTRADA,SAÍDA);
(começa por um algarismo)
PROGRAM TOM&JERRY(ENTRADA,SAÍDA);
(contém um caracter não aceite)
PROGRAM LED ZEPPELIN(ENTRADA,SAÍDA);
(contém um espaço)

Em teoria não existe qualquer restrição quanto ao comprimento dos identificadores; mas a maior parte das versões de Pascal apenas consideram os primeiros oito caracteres de cada identificador como significativos. Assim, o identificador ANAMARIADOSSANTOS será tratado como igual a ANAMARIADASILVA. Os nomes compridos são normalmente pouco prá-

ticos para escrever e recordar, pelo que convirá usar versões abreviadas. Por exemplo, a maior parte das pessoas tenderá a chamar a um programa ANALCTS em vez de ANÁLISE-DECUSTOS.

Depois do nome encontra-se um par de parêntesis. Para efeitos deste livro, estes parêntesis devem conter sempre as palavras 'ENTRADA,SAÍDA'.

Este final do título do programa é marcado por um ponto e vírgula. O Pascal utiliza o ponto e vírgula do mesmo modo que se usa em português o ponto final, para indical o final de uma frase. Existem regras muito rigorosas para o emprego do ponto e vírgula nos programas Pascal. Serão indicadas mais tarde.

Voltando ao nosso modelo, o leitor recorda que existem nele duas secções preenchidas com traços. A primeira destas é designada por *secção de declarações,* e a segunda por *corpo do programa.*

A secção de declarações fornece ao computador as informações preliminares de que necessita para poder executar o programa. Por exemplo, é necessário indicar nesta secção todos os nomes das variáveis usadas no programa.

O corpo do programa contém as instruções que formam o programa.

Estamos agora em posição de examinar um verdadeiro programa Pascal. Talvez o leitor queira introduzi-lo no computador [os utilizadores do meu compilador devem omitir a primeira linha deste e de todos os outros programas que se seguirem, dado que o meu compilador não necessita do nome do programa].

```
PROGRAM DIZEROLÁ(ENTRADA,SAÍDA);
BEGIN
    WRITE ('Olá')
END
```

Note que este programa não inclui uma parte das instruções. O corpo do programa é constituído por uma única instrução:

```
WRITE ('OLÁ')
```

Todas as instruções presentes no corpo de um programa Pascal consistem num identificador, o par de parêntesis. Este par de parêntesis contém os dados sobre os quais deverá actuar a instrução. Em alguns casos raros, as instruções não actuam sobre quaisquer dados, sendo portanto omitidos os parêntesis.

No caso da instrução WRITE, o conteúdo dos parêntesis é impresso no visor da televisão. O modo como isto acontece depende dos dados indicados. Quaisquer caracteres encerrados nos parêntesis serão simplesmente impressos no visor tal como estão.

A instrução WRITE deste exemplo escreve portanto a palavra 'Olá' no visor de televisão. É importante notar que as aspas não são impressas, dado que servem apenas para delimitar a cadeia de caracteres, e não fazem parte desta. Se se deseja incluir aspas numa instrução WRITE, é portanto necessário recorrer às aspas duplas ('').

O leitor terá talvez notado que a palavra WRITE (escrever) não é uma palavra reservada (não está em letra maiúscula) ao contrário de PRINT em Basic (à qual é equivalente), que é uma palavra reservada. A razão disto só se tornará evidente na última parte deste livro, mas por agora basta-nos saber que WRITE é um identificador "standard".

Depois de o programa estar em execução podemos querer alterá-lo de modo a imprimir um texto diferente.

O programa que se segue aplica alguns dos pontos referidos:

```
PROGRAM CIRCUNFERÊNCIA(ENTRADA,SAÍDA);
```

```
    CONST PI=3.1415926;
    VAR RAIO,CIRCUN:REAL;

BEGIN {Bloco principal do programa}

    WRITE ('INDIQUE RAIO DO CÍRCULO');
    READLN (RAIO);
    CIRCUN:=RAIO*2*PI;
    WRITELN ('O CÍRCULO TEM UM PERÍMETRO DE
    ',CIRCUN)

END.
```

As linhas em branco servem para tornar o programa mais legível marcando as divisões entre diferentes secções. Podem ser omitidas quando se escreve o programa porque não afectam a execução deste, mas não é prudente sacrificar a legibilidade dos programas apenas para poupar algum trabalho na entrada das instruções.

A secção de declarações do programa anterior é constituída por duas partes: a declaração CONST e a declaração VAR. A declaração CONST cria um identificador designando PI com o "valor" 3,1415926. Este identificador de constante pode então ser usado em vez de qualquer número nos cálculos do programa. A declaração CONST assume a forma de uma palavra, CONST, seguida de um identificador, um sinal de igual e o valor do identificador. O final de cada declaração é marcado por outro ponto e vírgula. Se este ponto e vírgula é seguido de outro identificador é criada uma nova constante; no caso contrário é seguida de uma palavra reservada (VAR neste exemplo). Tudo isto pode parecer complicado, mas o leitor verificará que faz sentido em uso.

A parte VAR tem uma sintaxe semelhante, excepto que é

possível declarar mais do que um identificador ao mesmo tempo separando-os com vírgulas. A maior parte dos outros programas que o leitor encontrará em seguida ilustrarão isto. Os identificadores declarados na secção VAR são variáveis e não constantes. As variáveis são identificadores que podem receber valores diferentes durante a execução do programa. Podem também ser tratados como constantes nos cálculos. As memórias das máquinas de calcular de bolso actuam um pouco como variáveis.

A diferença entre as partes VAR e CONST consiste em que o valor atribuído a uma constante não pode alterar-se no programa, mas as variáveis podem ser modificadas à nossa vontade.

Existem várias boas razões para usar constantes em programas em vez de variáveis normais, sempre que possível. Se se escreve um programa para realizar cálculos complexos, e imprimir em seguida os resultados numa impressora, deseja-se garantir que os resultados não sejam impressos sobre as perfurações que dividem as folhas. Nestas condições pretende-se que o programa imprima um certo número de linhas em branco quando a impressora se aproxima da parte inferior da folha, de modo a colocar a posição de impressão no início da folha seguinte. Neste caso é lógico declarar uma constante chamada "DimPag" com o valor de 66 (o número normal de linhas por página impressa no Reino Unido) de tal modo que no caso de o número de linhas em cada página dever ser alterado por qualquer razão, ser apenas necessário alterar a parta de declaração de constantes em vez de estudar todo o programa à procura do número literal que corresponde à quantidade de linhas por página. Em resumo, as constantes podem tornar mais fácil a manutenção de programas. Uma outra razão é o aumento de legibilidade. Em vez de ter o número 3,1415926 em vários pontos do programa, é prático usar uma constante que o designa.

BEGIN (início) marca o princípio do programa principal. A primeira linha do programa principal é uma instrução

WRITE. Neste exemplo, a instrução avisa o utilizador do programa de que deve indicar o raio do círculo.

A instrução READLN na linha seguinte aceita um número do teclado e guarda-o na variável RAIO. Deste ponto em diante, a variável RAIO pode ser tratada exactamente como um número, e sempre que ocorre um cálculo o computador utilizará em vez dela o valor que lhe corresponde. A única forma de alterar o número armazenado em RAIO consiste em usar o identificador noutra instrução READLN, ou atribuir-lhe um valor diferente numa instrução de atribuição.

A instrução seguinte é uma instrução de atribuição. Pode-se reconhecer uma instrução de atribuição pela presença do sinal dois pontos/igual, :=. Quando o computador encontra uma instrução deste tipo, realiza a soma à direita do símbolo e atribui o seu resultado à variável que se encontra à esquerda dele.

A soma da direita é designada pelo nome "expressão". Uma expressão Pascal pode ser uma constante, uma variável, ou uma combinação de ambos ligados por separadores. Os operadores são os sinais de soma, subtracção, multiplicação e divisão. Neste exemplo só é usado o operador da multiplicação (*). O resultado da expressão é o perímetro de uma circunferência com o raio RAIO. Assim, de acordo com a ideia já exposta de uso de identificadores descritivos, o resultado é guardado na variável CIRCUN.

Existem algumas diferenças entre a "aritmética Pascal" e a matemática das máquinas de calcular. Em primeiro lugar, o Pascal utiliza símbolos não habituais para designar as quatro operações. Como se mencionou acima, é usado o asterisco* para a multiplicação; e para a divisão usa-se um traço oblíquo (/). Os sinais + e − (mais e menos) são usados do modo habitual. A segunda diferença reside no modo como as expressões são avaliadas. Se se "escreve" 2+2*8 em algumas máquinas de calcular, o resultado será 32. O Pascal avaliará em 18 a expres-

são em causa. Isto deve-se ao facto de o Pascal realizar todas as multiplicações e divisões antes das somas e subtracções. Assim, no exemplo anterior avalia 2*8, e em seguida soma 2 ao resultado. Pode-se alterar a ordem pela qual as expressões são avaliadas utilizando parêntesis. Se se pede ao computador que avalie (2+2)*8, a resposta será de facto 32.

Ao contrário do Basic, o valor de uma variável antes de surgir numa instrução READLN ou de atribuição é imprevisível. Assim, todas as variáveis devem ser cuidadosamente inicializadas antes de serem empregues numa instrução WRITE ou no segundo membro de uma instrução de atribuição.

A instrução de atribuição em Pascal é um pouco diferente da usada em Basic. Nesta linguagem, escreve-se normalmente 'A=23', omitindo a palavra LET. Podemos perguntar-nos qual a razão de o Pascal empregar o símbolo dois pontos antes do sinal de igual. A verdade é que o homem que inventou o Pascal era um purista e um optimista. Era um purista porque descobriu que o sinal de igual usado neste contexto é mal usado. Num programa Basic pode ocorrer uma frase do tipo T=T+1, mas T Nunca será certamente igual a T+1, o que pode tornar-se confuso para um principiante. Tentou portanto remediar a situação decretanto o uso dos dois pontos além do sinal de igual. O seu principal objectivo é portanto auxiliar o principiante. Esta atitude foi bastante optimista, porque não compreendeu que quase todas as pessoas que aprendem a programar computadores fazem-no num computador que utiliza Basic.

A última linha do programa é um novo tipo de instrução de saída, a instrução WRITELN. A diferença entre WRITELN e WRITE consiste em que a primeira desloca o ponto de impressão para uma nova linha depois de imprimir o que se pretende.

A instrução
WRITELN;

desloca o ponto de impressão da instrução de saída seguinte para uma nova linha, ou seja, se a deslocação de saída anterior foi uma WRITE passa à linha seguinte e se foi uma WRITELN imprime uma linha em branco. Pode ser portanto usada para aumentar o espaçamento entre as saídas no visor, melhorando a legibilidade.

[No meu compilador é necessário usar WRITELN (' ');, que imprime simplesmente um espaço e passa à linha nova. Por esta razão, onde apropriado, usarei sempre esta forma na primeira secção do livro].

Esta instrução WRITELN demonstra que é igualmente possível imprimir números. Quando descobre que é necessário imprimir um identificador, a instrução não escreve a palavra ''RAIO''; determina o valor desta variável e imprime-o. Como é necessário imprimir dois elementos diversos, estes devem estar separados por dois pontos na instrução WRITELN.

A palavra reservada END define o final da declaração WRITELN, não sendo portanto necessário o ponto e vírgula. Define igualmente o final do programa, como já discutimos anteriormente.

As palavras encerradas em chavetas depois da palavra reservada BEGIN são comentários. Como tais são ignorados pelo computador, mas podem ajudar a explicar o programa a outros que o leiam, ou ao próprio programador. Os programas mais complicados deste livro estão bastante comentados, porque é importante que o leitor os compreenda [o meu compilador não suporta comentários]. Se o seu computador não possui chavetas no teclado, pode usar '(*' para iniciar um comentário e '*)' para o terminar. Qualquer que seja o delimitador de comentário que utilize, deve ser o mesmo no início e no final deste. A expressão '{início de programa *)' não é portanto válida.

Podemos agora ampliar o nosso exemplo dando-lhe a forma geral de um programa Pascal. A nova listagem será:

```
PROGRAM PROGNOME(INPUT,OUTPUT);
  CONST
  —
  —
  —
  VAR
  —
  —
  —
  BEGIN
  —
  —
  —
  END.
```

Se passar os olhos pelo livro, verificará que todos os programas incorporam algumas ou todas estas partes. Note que as secções VAR e CONST que existem no programa anterior são opcionais, mas que as outras não o são.

Talvez o leitor tenha notado que tenho inserido espaços no início de algumas das linhas de cada programa. Estes espaços em branco servem para fazer sobressair as diferentes secções. À medida que a forma geral do programa se tornar mais complexa, o leitor apreciará a vantagem deste método. Algumas versões do Basic permite proceder do mesmo modo para ciclos FOR-NEXT, com o mesmo objectivo.

O programa que se segue ilustra outras características do Pascal:

```
PROGRAM SALÁRIOS(INPUT,OUTPUT);
CONST
    SALHORA=3.5;
VAR
```

```
    HORAS:REAL;
    DIAS:REAL;
BEGIN
    WRITE ('INDIQUE NÚMERO DE HORAS DE TRA-
           BALHO POR DIA, E NÚMERO DE DIAS');
    READLN (HORAS,DIAS);
    WRITE ('SALÁRIO SEMANAL: ',SALHORA*
           HORAS*DIAS);
END.
```

Este exemplo é trivial, mas ilustra bem alguns aspectos dos programas Pascal. Na secção VAR, apresenta-se pela primeira vez mais de uma declaração. A instrução READLN tem neste caso dois argumentos. Como o leitor pode verificar, se se pretende indicar mais de uma variável, os diferentes nomes destas são separados por vírgulas. O mesmo acontece com todas as instruções Pascal que possam ter mais de um argumento.

II

TIPOS SIMPLES DE DADOS E INSTRUÇÃO IF

A palavra REAL foi usada em declaração VAR do último capítulo sem ser explicada.

Na secção VAR, a posição da palavra REAL pode ser ocupada por quatro palavras diferentes. Cada uma delas indica características diferentes da variável assim criada. Cada um destes diferentes "tipos" será explicado agora, juntamente com as suas características próprias.

O TIPO REAL

Uma variável declarada como sendo do tipo REAL é definida como sendo um número real — um número que pode incluir uma vírgula decimal. Números como 2,5 são números reais, assim como 4,0. Os números reais podem ser somados, subtraídos, multiplicados e divididos de acordo com as regras do último capítulo.

Os números reais são os mais semelhantes àqueles que usamos na nossa vida quotidiana. No entanto, em resultado do Pascal ser essencialmente concebido para uso como linguagem científica e educacional, recorre a um formato no caso de certos números reais que é estranho à maioria dos leitores. Este formato é conhecido pelo nome de notação "científica" ou "exponencial".

Os números escritos em notação exponencial possuem duas partes: a mantissa e o expoente. Estas duas partes são separadas pela letra "E". Assim, o formato de um número exponencial é <mantissa> E <expoente>. A mantissa é um número normal, entre 1 e 10. Se o número possui menos de 10 algarismo, os ausentes são substituídos por zeros (ou seja, 2,3 passa a 2,30000000). Quando se escreve um número no teclado em formato exponencial, pode-se omitir estes zeos. A mantissa é seguida pela letra E, e depois pelo expoente. Este último indica a potência de 10 que deve ser multiplicada pela mantissa para obter o número em causa. Por exemplo, 2,300000000E4 é igual a 2,3 vezes 10 à quarta potência, ou seja 23000. Não é necessário indicar os números deste formato, mas o Pascal imprime os números deste modo, o que nem sempre é útil. O utilizador dispõe da instrumentação necessária para definir o formato usado. Um número real numa instrução WRITE ou WRITELN pode ser seguido por dois números (ou variáveis) que indicam a "largura de campo" onde o número deve ser impresso e o número de casas decimais a imprimir. Por exemplo, WRITELN (PI:20:6) imprime (considerando que PI terá sido convenientemente definida) PI justificado à direita (ou seja com zeros à esquerda) de tal modo que a distância entre o lado esquerdo do visor e o último algarismo do número é 20 posições. A saída é limitada a 6 casas decimais. Esta característica é bastante útil nas aplicações em que a máquina está a ser usada por um operador não muito brilhante e em que o computador tenderia a imprimir valores com um rigor desnecessário, informando por exemplo que a taxa de conversão do dólar em libras é de 2,3453463474357257E2.

[Esta característica não é suportada pelo meu compilador. Quando vir os símbolos de formatação numa instrução WRITE, é geralmente mais seguro omiti-los completamente.]

A aritmética real nem sempre é exacta. Se o computador

imprimir 10 casas decimais, dará a resposta 123456789+ +0,000000000001 sob a forma 123456789, que só é aproximadamente correcta.

O TIPO INTEIRO

As variáveis inteiras assemelham-se às variáveis reais, mas não podem conter números com vírgula decimal. Assim, todas as variáveis inteiras podem ser expressas como variáveis reais acrescentando uma vírgula decimal e um zero à direita desta. Os exemplos seguintes ilustram esta diferença:

Números reais:
2346,34574568+E89
−23,4
−3,0
4,2345
−2,235
Números inteiros:
32146
−23
23
643
266
−3

Tendo em conta o facto de todos os inteiros poderem ser expressos como números reais (por exemplo −3,0 em vez de −3), o leitor pode não compreender porque razão o Pascal utiliza números inteiros. As razões são as seguintes: velocidade e rigor.

O aumento de velocidade permitido pelos inteiros decorre do facto de estes números poderem ser armazenados interna-

mente no computador em duas partes separadas, enquanto que o real necessita de 6 partes. É muito mais rápido processar essas duas partes em vez das seis, pelo que a aritmética de números inteiros é executada mais rapidamente.

A razão pela qual as variáveis reais podem não ser rigorosas está relacionada com o facto de certas fracções não poderem ser expressas exactamente em binário, do mesmo modo que 1/3 não pode ser convertido em número decimal sem perder algum rigor.

De qualquer modo, o leitor deve tentar usar número inteiros — porque dá menos trabalho escrevê-los...

As regras da aritmética inteira são levemente diferentes das usadas na aritmética real. Usam-se os mesmo quatro símbolos das operações; mas o operador de divisão, '/', dá um resultado real em vez de inteiro.Para obter um resultado inteiro numa divisão, é necessário usar os operadores DIV e MOD. X DIV Y dá a parte inteira do resultado da divisão de X por Y, e X MOD Y dá o resto da divisão de X por Y.

O conjunto dos inteiros é limitado. O maior número inteiro que o Pascal pode utilizar é indicado pela constante MAXINT. Todos os compiladores Pascal definem esta constante [excepto o meu]. O menor inteiro que pode ser usado é dado por −MAXINT. Em geral MAXINT é 32767. A gama de números utilizáveis é seriamente limitada em comparação com as variáveis reais, mas muitos programas de carácter não-científico podem utilizar variáveis contidas neste domínio (muitos compiladores permitem recorrer a um conjunto especial de inteiros designados por «inteiros compridos» numa gama mais extensa — à custa da velocidade de execução e do gasto de memória. Deve verificar no manual do seu compilador se dispõe desta extensão).

As variáveis inteiras são declaradas do mesmo modo que as variáveis reais, sendo no entanto a palavra INTEGER (Inteiro) usada em vez de REAL na declaração VAR.

Os inteiros presentes nas instruções WRITE podem ser formatados de modo semelhante às variáveis reais, não sendo no entanto possível especificar o número de casas decimais (porque não existem casas decimais nos inteiros). É possível especificar a largura do campo. Basta isto para assegurar que a impressão seja realizada do modo mais apropriado a cada caso.

O TIPO CHAR

As variáveis tipo CHAR (Caracter) são um pouco diferentes das reais e inteiras porque não contêm um número, e sim uma letra ou símbolo únicos. O programa que se segue dá-nos um exemplo do uso das variáveis CHAR:

```
PROGRAM DEMO(INPUT,OUTPUT);
VAR
    A,B,C,:CHAR;
BEGIN
    WRITE ('INDIQUE AS SUAS INICIAIS
    (APENAS TRÊS)');
    READLN (A,B,C,);
    WRITE (C,B,A)
END.
```

Este programa permite-lhe indicar as iniciais do seu nome, imprimindo-as depois por ordem inversa.

Não é difícil compreender o programa; a instrução READLN actua quando aplicada a variáveis de tipo CHAR de um modo um pouco diferente do quando aplicada a variáveis numéricas.

A instrução READLN com um argumento de tipo CHAR espera que seja indicado um caracter — mas não é necessário carregar em «return» depois de escrever o caracter. Assim, no

programa anterior é possível indicar todas as iniciais sem as separar usando «return». O meu compilador, no entanto, substitui READLN por INPUT e utiliza variáveis de cadeias em vez de variáveis de tipo CHAR, pelo que o programa anterior necessita de «return» depois de cada caracter. Na realidade, a instrução READLN é apenas aproximadamente igual à instrução INPUT do Basic quando o argumento da instrução é numérico. Quando o argumento é do tipo CHAR, comporta-se de um modo mais semelhante às instruções INKEY$ ou GET do Basic.

Isto poderia ser considerado uma desvantagem do meu compilador, mas como as potencialidades de processamento de caracteres do Pascal são extremamente limitadas quando comparadas com as do Basic, não ocorrem muitos problemas na primeira parte do livro relacionados com variáveis de tipo CHAR.

[A versão do meu compilador para o ZX Spectrum só permite que as variáveis de tipo CHAR possuam um nome com um caracter. Os programas deste livro utilizarão em geral nomes mais compridos, pelo que os possuidores do Spectrum terão de renomear as varáveis de tipo CHAR].

O TIPO BOOLEANO

Este último tipo de variável é ainda mais limitado quanto ao número de valores diferentes que pode assumir em relação aos tipos anteriores. As variáveis reais podem assumir vários milhões de valores, as inteiras vários milhares e as de tipo CHAR várias centenas. As variáveis de tipo booleano só podem assumir dois valores diferentes!

Os dois valores em causa são VERDADEIRO e FALSO. Podem-se aplicar quatro operadores às variáveis booleanas: AND, OR, EOR (por vezes designado EXOR) e NOT. Os resultados destes operadores podem ser observados na tabela seguinte:

Verdadeiro	AND	Verdadeiro	=	Verdadeiro
Verdadeiro	AND	Falso	=	Falso
Falso	AND	Verdadeiro	=	Falso
Falso	AND	Falso	=	Falso
Verdadeiro	OR	Verdadeiro	=	Verdadeiro
Verdadeiro	OR	Falso	=	Verdadeiro
Falso	OR	Verdadeiro	=	Verdadeiro
Falso	OR	Falso	=	Falso
Verdadeiro	EOR	Verdadeiro	=	Falso
Verdadeiro	EOR	Falso	=	Verdadeiro
Falso	EOR	Verdadeiro	=	Verdadeiro
Falso	EOR	Falso	=	Falso
	NOT	Verdadeiro	=	Falso
	NOT	Falso	=	Verdadeiro

Isto significa que é válida uma expressão como "Esfomeado AND Pobre". As vantagens que isto nos traz só se tornam óbvias depois de conhecermos a instrução IF.

As variáveis booleanas são declaradas e atribuídas do modo habitual. A única coisa anormal quanto a elas é que não consentem o uso de instruções READLN ou WRITE.

O leitor recorda-se certamente de que a secção CONST de um programa não especifica o tipo de constantes declarado. Como existem quatro tipos de variáveis, talvez o leitor pergunte a si mesmo porque razão é necessário declarar explicitamente os tipos de variáveis e não os tipos das constantes. A razão disto é que o computador pode sempre determinar o tipo de uma constante examinando simplesmente o seu valor (por exemplo, as constantes de tipo CHAR são marcadas por uma única plica). Como uma variável de uma secção VAR não possui qualquer

valor, é necessário indicar à máquina inicialmente qual o seu tipo.

A declaração CONST também lhe permite declarar constantes de tipo CHAR de mais de um caracter.

A INSTRUÇÃO "IF"

A instrução IF altera a execução de um programa em função de certos acontecimentos. Esta declaração pode assumir duas formas:

IF condição **THEN** instrução

e:

IF condição **THEN** instrução 1
ELSE instrução 2

No primeiro caso, a instrução que se encontra à frente da palavra reservada THEN só é executada se a condição que se segue a IF for considerada Verdadeira. No segundo caso tudo se passa da mesma maneira, mas se a condição for Falsa será executada a instrução que se segue a ELSE.

Uma expressão booleana pode consistir de duas expressões numéricas ou CHAR normais, ligadas por "=" (igual), "< >" (diferente de), ">=" (maior do que ou igual a), "<=" (menor do que ou igual a), ">" (maior do que) ou "<" (menor do que). É possível ligar duas ou mais expressões numa mesma instrução IF usando os ligadores AND, OR e EOR. Um outro tipo de expressão booleana é uma expressão formada por variáveis booleanas, ligadas por ligadores booleanos. Nestas condições devem-se usar normalmente parêntesis, como se ilustra em alguns dos programas deste capítulo, a fim de aumentar a legibilidade.

Se deseja utilizar mais do que uma instrução em qualquer

das partes das construções anteriores designadas por "instruções", terá de usar a construção BEGIN-END para limitar as instruções. O programa que se segue exemplifica isto e alguns outros problemas que estamos a discutir:

```
PROGRAM DECISÃO(INPUT,OUTPUT);
VAR
    A,B:REAL;
BEGIN
    WRITE ('INDIQUE RAIZ QUADRADA DE 400');
    READLN (A)
    IF A=20 THEN WRITELN ('CORRECTO')
ELSE WRITELN ('ERRADO');
    WRITE ('QUAL É A SUA IDADE?');
    READLN (B)
    IF (B>16) AND (A<>20) THEN
    BEGIN
    WRITELN ('ISSO NÃO É MUITO BOM PARA');
    WRITELN ('ALGUÉM COM ',B,' ANOS DE IDADE.')
    END
    ELSE
    BEGIN
    WRITELN («COMO TEM APENAS «,B,» ANOS DE
    IDADE);
    WRITELN («NÃO ESTÁ MUITO MAL.»)
    END
END.
```

[O meu compilador não é muito brilhante a descobrir onde termina uma condição de uma instrução IF e começa a palavra THEN. Se a condição termina num identificador, toda ela deve ser inserida entre parêntesis. É portanto necessário escrever IF (A=B) THEN, em vez de IF A=B THEN.

Por outro lado, o meu compilador insiste no uso da construção BEGIN-END em instruções IF, mesmo que seja usada apenas uma instrução. Todos os programas da primeira parte do livro respeitam esta limitação].

Os aspectos a notar aqui são que, como anteriormente, não existe um ponto e vírgula antes de qualquer das instruções END.

Se o leitor recebesse o programa anterior sem quaisquer pontos e vírgulas, teria provavelmente alguma dificuldade em colocá-los nos locais apropriados da listagem. Existem no entanto algumas regras simples que governam o uso dos pontos e vírgulas.

Todos os compiladores Pascal actuam sobre os programas que devem compilar examinando um caracter de cada vez. Por exemplo, o meu compilador chama continuamente uma rotina na linha 1770, que determina o caracter seguinte do programa. Como este é o único contacto que o compilador tem com o programa, não reconhece a existência de cortes entre linhas, apenas reconhecendo uma cadeia contínua de caracteres. O ponto e vírgula permite ao compilador descobrir onde se encontra o final das instruções, sem as limitações da "linha nova" empregue pelo Basic. A vantagem deste método é que se torna possível escrever programas onde uma instrução seja mais comprida do que uma linha única do visor.

Para garantir que os pontos e vírgulas sejam colocados nos locais apropriados, é necessário definir rigidamente o que entendemos por um "programas Pascal".

Um programa Pascal consiste em:

1) A palavra **PROGRAM**;
2) Um identificador (o nome do programa);
3) Um parêntesis esquerdo;
4) As palavras "INPUT,OUTPUT";
5) Um parêntesis direito;
6) Um ponto e vírgula;

7) Um bloco;
8) Um ponto final.

Um bloco é:
1) Uma declaração **CONST** opcional;
2) Uma declaração **VAR** opcional;
3) A palavra **BEGIN**;
4) Uma ou mais instruções, separadas por pontos e vírgulas;
5) A palavra **END**.

Note-se que as instruções são *separadas* por pontos e vírgulas, não terminadas por elas. É por isso que não colocamos um ponto e vírgula antes da instrução END final.

A instrução pode ser:
1) Uma atribuição;
2) Uma chamada de procedimento;
3) A palavra **BEGIN**, seguida por uma ou mais instruções separadas por pontos e vírgulas e pela palavra END;
4) A palavra **IF**, uma expressão, a palavra **THEN**, uma instrução, e opcionalmente a palavra **ELSE** e uma última instrução.

Existem outros tipos de instruções, que serão referidos no momento adequado.

A chamada de procedimento ("Procedure") é uma das instruções WRITE, WRITELN ou READLN.

Se o leitor estudou bem a explicação anterior, aplicando-a ao exemplo dado, deve compreender o modo de usar dos pontos e vírgulas.

Vale a pena sublinhar que uma instrução pode ser "vazia". Isto significa que não contém quaisquer caracteres, ou apenas espaços e comentários [o meu compilador não aceitará instruções vazias]. Tendo em conta que são aceitáveis instruções nulas, torna-se possível escrever programas como o seguinte:

```
PROGRAM VAZIO (INPUT,OUTPUT);
BEGIN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
END.
```

constituído apenas por instruções vazias. Talvez o leitor tenha notado que este programa contraria a regra básica de não inclusão de um ponto e vírgula antes de uma instrução **END**...

III

FUNÇÕES BÁSICAS E CICLO FOR

Uma função é um processo que altera um número de um modo qualquer. Nestas condições, a função "raiz quadrada" transforma um número na sua raiz quadrada. Em Pascal existem muitas destas funções pré-definidas, e é fácil definir novas funções, como veremos.

As funções são identificadas por identificadores previamente definidos. Por exemplo, a palavra SQRT significa "raiz quadrada". As funções ocorrem sempre antes de um par de parêntesis contendo o argumento da função, com a instrução WRITE. Ao contrário desta, no entanto, uma função só pode surgir em expressões. Isto deve-se ao facto de todas as funções produzirem um valor, e se uma função fosse chamada como parte de um programa do mesmo modo que a instrução WRITE não haveria maneira de utilizar o resultado. No final deste capítulo o leitor encontrará alguns programas de exemplo usando funções.

O conteúdo dos parêntesis associados a uma função pode ser qualquer coisa desde uma nova expressão até um número.

Vamos apresentar uma lista completa de funções "standard". Alguns compiladores possuem mais funções, mas nenhum deve possuir menos do que estas.

Nenhum dos nomes de funções é uma palavra reservada.

ABS dá o valor absoluto do seu argumento. O valor absoluto de um número é esse mesmo número sem o sinal menos ("−"). O valor absoluto de −3 é portanto 3, tal como o valor absoluto do próprio número 3. ABS aceita um argumento de tipo real ou inteiro, e o resultado é sempre do mesmo tipo que o argumento. ABS é usada principalmente em programas matemáticos e científicos. Os matemáticos consideram que esta função fornece o módulo de um número, mas não deve ser confundida com o operador MOD.

ARCTAN (por vezes designada por ATAN ou ATN). Esta função dá o arco cuja tangente é o seu argumento. Isto significa que se A for a tangente de PI radianos, a atribuição B = ARCTAN(A) dará um valor B igual a PI. ARCTAN pode utilizar um argumento real ou inteiro, mas o seu resultado é sempre real [o meu compilador utiliza a forma ATN desta função].

CHR converte um número num caracter. Todos os caracteres disponíveis no seu computador têm um código numérico equivalente (muitas vezes uma variante do código ASCII "standard"). A função CHR permite-nos converter o código em causa no próprio caracter. Esta função deve ter um argumento inteiro, e produz um resultado de tipo CHAR.

COS dá o co-seno do seu argumento, que pode ser de tipo inteiro ou real. O argumento é sempre avaliado em radianos. O resultado é sempre de tipo real.

EXP dá a constante "e" elevada à potência do seu argumento. Pode igualmente ser considerada como o logaritmo anti-natural de um número. O argumento pode ser do tipo real ou inteiro, mas o resultado é sempre de tipo real.

LN dá ao logaritmo natural ou neperiano do seu argumento. O argumento deve ser de tipo real ou inteiro, e o resultado é sempre de tipo real.

ODD dá o resultado booleano Verdadeiro se o argumento

for um número ímpar, e Falso no caso contráro. O argumento deve ser de tipo inteiro. Um outro modo de definir esta função consiste em considerar que é capaz de nos indicar se um número é ou não divisível por dois.

ORD é o oposto de CHR, convertendo um caracter no código que lhe corresponde. O seu argumento deve ser do tipo CHAR, e o seu resultado é sempre um inteiro.

PRED dá o inteiro que é inferior numa unidade ao seu argumento, ou o caracter cujo código é inferior numa unidade ao do argumento [o meu compilador suporta apenas PRED's de inteiros]. Na maior parte dos computadores PRED('B')='A', mas não é prudente partir do princípio de que assim acontece sem primeiro consultar a documentação do compilador.

ROUND fornece o inteiro mais próximo de um argumento real.

SIN produz o seno do seu argumento (que deve ser em radianos), de tipo real ou inteiro. O resultado é sempre de tipo real.

SQR (não confundir com a SQR do Basic) dá o quadrado do seu argumento. Este pode ser de tipo real ou inteiro. O resultado é sempre do mesmo tipo do argumento.

SQRT dá a raiz quadrada do seu argumento. Este pode ser de tipo real ou inteiro, mas o resultado é sempre de tipo real.

SUCC é o oposto de PRED — dá o "sucessor" do seu argumento. Tanto o argumento como o resultado serão de tipo CHAR ou inteiro.

TRUNC assemelha-se a ROUND, excepto no facto de, em vez de arredondar o argumento, dar como resultado o inteiro igual ou imediatamente inferior ao número original. Nestas condições, ROUND arredonda para o inteiro mais próximo e TRUNC ignora a parte do número real que se encontra a seguir à vírgula decimal, arredondando sempre para menos. 1,7 será arredondado para 2, mas "truncado" para 1.

O Pascal "standard" não possui uma função TAN (cálculo de tangente). É no entanto possível obtê-la recorrendo à expressão SIN(X)/COS(X). Como se indicou acima, todas as funções trigonométricas trabalham apenas em radianos. Para converter de graus para radianos divide-se por 180 e multiplica-se po PI. Para converter radianos em graus, divide-se por PI e multiplica-se por 180.

[O meu compilador só suporta as funções SUCC, PRED, ROUND, SQR e ODD se o computador onde está a ser utilizado suporta funções definidas pelo utilizador. Isto deve-se ao facto de estas serem as únicas funções «standard» do Pascal sem equivalente Basic. Insere as definições destas funções nas primeiras cinco linhas do programa Basic].

☆☆☆☆☆☆

Se pedissem ao leitor que escrevesse um programa para imprimir todos os números inteiros entre 1 e 10, o único modo de o poder fazer utilizando os conhecimentos adquiridos até agora seria:

```
PROGRAM NÚMEROS(INPUT,OUTPUT);
BEGIN
    WRITELN (1);
    WRITELN (2);
    WRITELN (3);
    WRITELN (4);
    WRITELN (5);
    WRITELN (6);
    WRITELN (7);
    WRITELN (8);
    WRITELN (9);
    WRITELN (10);
END.
```

Não se sinta obrigado a introduzir este programa na máquina...

Este programa dá com efeito os resultados pretendidos, mas que diria o leitor se fosse necessário escrever todos os inteiros entre 1 e 100?

Existe um modo muito simples de repetir uma parte de um programa um dado número de vezes. Usa-se para tal a instrução FOR, com o seguinte formato:

FOR identificador: + início **TO** fim
DO instrução

Tal como acontece na instrução IF, se for necessário usar mais do que uma instrução na parte final deve-se recorrer à construção BEGIN-END.

O identificador é uma variável de tipo inteiro.

O ciclo FOR indica ao computador que deve executar a instrução que se segue a DO (''fazer''), tomando a variável de comando deste ciclo todos os valores entre ''início'' e ''fim''. Assim, se início for igual a 1 e fim igual a 10, o ciclo será executado dez vezes.

Se por alguma razão se deseja que ''início'' seja maior do que ''fim'', de tal modo que a variável seja decrementada de cada vez que o ciclo é executado, deve substituir TO por DOWNTO no exemplo anterior. Se utilizar DOWNTO, deve assegurar que os dois valores são razoáveis — isto é, que o primeiro é maior do que o segundo.

[O meu compilador insiste em usar a construção BEGIN--END, mesmo quando não é estritamente necessária].

Este programa imprime o conjunto de caracteres do seu computador — se utilizar o código ASCII.

PROGRAM CONJCAR(INPUT,OUTPUT)

```
VAR
    CH:INTEGER;
BEGIN
    FOR CH:=32 TO 126 DO
    WRITE (CHR(CH))
END.
```

O programa anterior utiliza apenas uma única instrução a seguir à palavra DO, pelo que não é necessária a construção BEGIN-END.

[No meu compilador, o programa em causa transforma-se do seguinte modo:

```
PROGRAM CONJCAR(INPUT,OUTPUT);
VAR
    CH:INTEGER;
BEGIN
    FOR CH:=32 TO 126 DO
        BEGIN
            WRITE (CHR(CH))
        END
END.
```

Note a colocação de pontos e vírgulas no programa].

Os valores de início e fim do ciclo FOR podem ser expressões ou números literais.

Agora que introduzimos a instrução FOR, torna-se possível apresentar programas de exemplo um pouco menos triviais.

O programa que se segue permite ao leitor indicar dez números e imprimir em seguida a sua média:

```
PROGRAM MÉDIA(INPUT,OUTPUT);
VAR
```

```
    NUM,TOT:REAL;
    N:INTEGER;

BEGIN

    TOT:=0

    FOR N:=1 TO 10 DO
    BEGIN
        WRITE ('INDIQUE NÚMERO ',N);
        READLN (NUM);
        TOT:=TOT+NUM
    END;

    WRITELN ('');
    WRITELN ('A MÉDIA É ',TOT/10)

END.
```

O uso de espaços iniciais e linhas em branco torna este programa muito mais fácil de ler do que no caso contrário — compare-o com o mesmo programa escrito de outro modo:

```
PROGRAM MÉDIA(INPUT,OUTPUT);
VAR
NUM,TOT:REAL;
N:INTEGER;
BEGIN
Tot:=0
FOR N:=1 TO 10 DO
BEGIN
WRITE ('INDIQUE NÚMERO ',N);
READLN (NUM);
```

```
TOT: = TOT + NUM
END;
WRITELN ('');
WRITELN ('A MÉDIA É ',TOT/10)
END.
```

Quando o programa é escrito deste modo é muito mais difícil observar onde terminam e começam os blocos BEGIN--END, e toda a estrutura do programa apresenta-se um tanto confusa. Em programas mais extensos torna-se ainda mais importante usar a barra de espaços do teclado.

A execução do programa não deveria causar quaisquer problemas.

Depois da execução de um ciclo FOR, a variável de comando ou indexação (que se encontra a seguir à palavra FOR) não deve ser usada sem que lhe seja atribuído novo valor. Isto deve-se ao facto de o seu valor ser indeterminado ao abandonar o ciclo. O seu valor não pode por outro lado ser alterado enquanto o ciclo está a ser executado.

Um problema dos ciclos FOR é que não é possível sair do ciclo sem passar por todos os valores atribuídos à variável de comando na instrução FOR. O modo habitual de resolver este problema consiste em usar uma «flag» booleana especial que decide se o código que forma o ciclo deve ou não ser executado:

```
FLAG: = VERDADEIRO;

FOR CICLO: = 10 TO 1000 DO

BEGIN

IF FLAG THEN
BEGIN
```

```
—
—
—
IF CONDIÇÃO THEN FLAG:=FALSO

END { IF }

END { FOR }
```

O programa só executará o conteúdo do ciclo se a FLAG for VERDADEIRA. Nestas condições, se a FLAG for passada a FALSA quando se deseja abandonar o ciclo, quaisquer execuções restantes serão omitidas.

Esta solução é provavelmente a mais elegante (o que deve ser característico dos programas), mas é ainda preferível codificar o programa de tal modo que se evite a ocorrência desta situação.

A variável de indexação é sempre incrementada ou decrementada de um; se se deseja utilizar algum outro valor, é necessário recorrer a subterfúgios. Por exemplo, um programa que imprima qualquer letra do alfabeto pode ser escrito do seguinte modo (considerando uma vez mais que se estão a usar códigos ASCII):

```
PROGRAM OUTRO(INPUT,OUTPUT);
VAR
    ÍNDICE:INTEIRO;
BEGIN
    FOR ÍNDICE:=1 TO 13 DO
        BEGIN
            WRITE (CHR(ÍNDICE*2+63))
        END
END.
```

A expressão peculiar existente entre parêntesis na instrução WRITE serve para ter em conta o código ASCII da letra "A", 65.

Alternativamente, poderíamos recorrer a uma solução um pouco mais esotérica deste problema:

```
PROGRAM OUTUBRO(INPUT,OUTPUT);
VAR
    ÍNDICE:INTEIRO;
BEGIN
    FOR ÍNDICE:=1 TO 26 DO
    BEGIN
    IF ODD(ÍNDICE) THEN
        BEGIN
            WRITE (CHR(ÍNDICE+64))
        END {IF}
    END {FOR}
END.
```

Este programa é excessivamente complicado devido à necessidade de usar a construção BEGIN-END ao usar o meu compilador, se bem que seja de facto quase tão simples como a primeira variante.

Os comentários que se seguem às instruções END no programa anterior facilitam a delimitação dos blocos BEGIN--END no programa. É um bom hábito recorrer a comentários como estes.

Se o leitor seguiu o programa até aqui, talvez se veja um tanto desarmado com o programa que se segue:

```
PROGRAM FAIXAS(INPUT,OUTPUT);
VAR
LINHA,ESTRELA,FAIXA:INTEGER;
```

```
BEGIN
FOR LINHA:=1 TO 6 DO
BEGIN
FOR ESTRELA:=1 TO 10 DO
BEGIN
WRITE ('*')
END;
FOR FAIXA:=1 TO 20 DO
BEGIN
WRITE ('=')
END;
WRITELN ('')
END;
FOR LINHA:=1 TO 5 DO
BEGIN
FOR LINHA:=1 TO 30 DO
BEGIN
WRITE ('=')
END;
WRITELN ('')
END
END.
```

Este programa imprime de uma forma estilizada estrelas e linhas.

O programa que acabamos de apresentar constitui um bom exemplo de ciclos contidos noutros. É fácil compreender a construção destes ciclos, mas é igualmente fácil cometer um erro nessa construção, terminando um ciclo interior depois de terminar o exterior, em particular no caso de programas complexos. Comentar todas as instruções END ajuda a não confundir os ciclos.

Vamos apresentar em seguida um programa mais simples usando estes ciclos. O programa imprime tabelas de multiplica-

ção entre 1 e 12 (não utilize uma impressora com este programa, porque gastará muito papel...).

```
PROGRAM TABELAS(INPUT,OUTPUT);
VAR
    A,B:INTEGER;
BEGIN
    FOR A:=1 TO 12 DO
        BEGIN
            FOR B:=1 TO 12 DO
        BEGIN
            WRITELN (A,' VEZES ',B.' SÃO ',A*B)
        END {CICLO B}
    END {CICLO A}
END.
```

O leitor talvez queira alterar este programa a fim de imprimir as tabelas desejadas, mas incluindo uma instrução READLN.

O ciclo FOR é obviamente muito útil, mas nem sempre constitui o melhor modo de repetir uma secção de código um dado número de vezes, como veremos no capítulo seguinte.

IV

CICLOS REPEAT E WHILE

O único modo de repetir uma parte do programa consiste em usar o ciclo FOR. No entanto, esta nem sempre é a melhor maneira de se obter o que se pretende.

Se o leitor necessitasse de escrever uma programa capaz de calcular a média de uma lista de números, podia escrevê-lo do modo ilustrado no capítulo anterior, usando o ciclo FOR. Mas que aconteceria no caso de o utilizador querer determinar a média de 20 números e não dos 10 previstos? O único modo de resolver o problema consiste em alterar a listagem do programa cada vez que este é executado; é também possível especificar quantos números devem ser considerados antes de lhes dar entrada na máquina, como no exemplo seguinte:

```
PROGRAM MEDIA(INPUT,OUTPUT);

VAR
    NUM,VAL,IND,TOT:INTEGER

BEGIN

WRITE ('QUANTOS NUMEROS?');
    READLN (NUM);
```

```
TOT:=0

FOR IND:=1 TO NUM DO

    BEGIN
    WRITE ('INDIQUE VALOR',IND);
    READLN (VAL);
    TOT:=TOT+VAL
END {DO CICLO}

WRITELN ('');
WRITELN ('A MEDIA E',TOT/NUM)

END.
```

Este programa satisfaz o nosso objectivo, mas tem ainda uma utilidade limitada dado que o utilizador é obrigado a saber de avanço a quantidade de números a indicar. Isto nem sempre é prático. Necessitamos de um programa de outro tipo:

```
PROGRAM MELHOR(INPUT,OUTPUT);

VAR
    TOT,VAL,NUM:INTEGER

BEGIN

TOT:=0;
    NUM: =1;

    REPEAT

        WRITELN ('INDIQUEVALOR',NUM);
```

```
        READLN (VAL);
        TOT: = TOT + VAL;
        NUM: = NUM + 1

    UNTIL VAL = 0;

        WRITELN ('');
        WRITELN ('A MEDIA E',TOT/NUM)

    END.
```

Esta versão do programa continuará a aceitar números até ser escrito um valor de 0, imprimindo em seguida a média dos números indicados anteriormente.

O ciclo REPEAT-UNTIL usado no programa executa as instruções entre a palavra REPEAT e a palavra UNTIL, até a condição definida depois da palavra UNTIL ser verificada (ser Verdadeira). É no entanto necessário ter em conta alguns aspectos de sintaxe. Se no interior do ciclo se encontra mais do que uma instrução, estas devem ser separadas por pontos e vírgulas e, tal como no caso da construção BEGIN-END, não é necessário qualquer ponto e vírgula antes da palavra UNTIL. A condição que se segue à palavra UNTIL deve terminar em ponto e vírgula, a menos que a ela se siga outra instrução UNTIL ou a palavra END.

[O meu compilador obriga a colocar a condição entre parêntesis quando termina por um identificador e é seguida de qualquer caracter alfanumérico, tal como nas condições da declaração IF].

A beleza do ciclo REPEAT-UNTIL consiste em não ser necessário saber previamente quantas vezes deverá ser executado quando escrevemos o programa, nem aliás quando o computador começa a executá-lo.

A forma geral do ciclo REPEAT é:

—
—
—
REPEAT
—
<instruções Pascal>
—
UNTIL expressão booleana
—
—
—

O diagrama anterior menciona "instruções Pascal" em vez da simples "instrução Pascal" que encontramos nas instruções IF e FOR. É por isto que a construção BEGIN-END não é necessária neste caso — o final do ciclo está suficientemente bem definido sem necessidade de qualquer outra marca além da instrução UNTIL.

Um uso muito vulgar da instrução REPEAT consiste em repetir todo o programa. Por exemplo, se se escreve um programa para calcular raízes de uma equação quadrática, é preferível fechar o programa num ciclo REPEAT, que apenas termina quando o utilizador abandona o programa. A razão disto é que permite ao utilizador repetir a parte de cálculos do programa sem ter de passar pela execução de todo ele, o que pode ser bastante aborrecido em algumas combinações máquina/compilador.

O ciclo REPEAT-UNTIL pode ser codificado em Basic do seguinte modo:

—
—
—

```
100 REM REPEAT
—
—
—
150 ^&^%^&^&@$#(*@!%&&#!^
—
—
—
200 IF NOT (condição) THEN GOTO 100
—
—
—
```

Se o leitor está habituado ao Basic, compreende facilmente que quando o programa anterior é executado a linha 150 é sempre executada, quer a condição da linha 200 (a instrução UNTIL) seja ou não verdadeira. Isto pode provocar problemas em programas onde a entrada num ciclo, sem verificar a condição, executa instruções que não o deviam ser. O modo mais simples de ultrapassar isto consiste em colocar todo o ciclo REPEAT-UNTIL numa instrução IF:

```
IF condição THEN

    BEGIN

        REPEAT

        instruções
        UNTIL NOT (condição)

    END
```

Se este segmento é executado, o programa não começa a executar as instruções no ciclo REPEAT-UNTIL no caso de a condição que se segue à parte UNTIL do ciclo ser Verdadeira. Esta técnica pode ser um pouco confusa, pelo que foi desenvolvido um outro tipo de ciclo:

WHILE condição **DO** declaração

Como é habitual, se for necessário mais do que uma instrução na parte «declaração», deve-se recorrer à construção BEGIN-END [e, também como é habitual, o meu compilador necessita desta construção em quaisquer circunstâncias, mesmo quando só se encontra uma instrução a seguir à palavra DO].

No caso do ciclo WHILE ("enquanto"), a instrução (ou instruções) que se segue à palavra DO só são executadas enquanto a condição for Verdadeira. Assim, se o computador encontra uma instrução WHILE cuja condição seja falsa, ignora o resto do ciclo. Isto permite-nos escrever segmentos de código como "WHILE ERRO DO EMENDA". Se o ciclo Repeat fosse usado nestas circunstâncias em vez do ciclo While, o programa tentaria "emendar" o erro apesar de este não existir.

Os principiantes têm normalmente alguma dificuldade quanto a decidir qual o tipo de ciclo a usar numa dada situação, devido à sua aparente semelhança. O modo mais simples de decidir o ciclo a usar consiste em escrever em português a instrução pretendida, tendo em conta que WHILE significa "enquanto" e UNTIL significa "até". Uma das duas formas de descrever o problema parecerá errada, escolhendo-se portanto a outra. Considerando o exemplo anterior, os dois modos de descrever o problema em linguagem comum serão:

1) ENQUANTO o sistema não funciona correctamente, tentar emendá-lo.

2) Tentar emendar o sistema ATÉ não haver erro.

A segunda estas descrições só será aceitável se o sistema não funcionar convenientemente quando o ciclo começa a ser executado, pelo que se torna óbvio que a solução adequada à primeira.

V

PROGRAMAS

CURVAS SINUSOIDAIS MÓVEIS

Este programa desenha uma sinusoidal em movimento no visor.

```
PROGRAM SENO(INPUT,OUTPUT);

CONST
    LARGURA = 50;

VAR
    CENTRO,FACTOR,ANGULO,ESPACOS,CICLO:
                                    INTEGER;

BEGIN

    CENTRO: = LARGURA DIV 2;
    FACTOR: = Centro-2;

    REPEAT

        FOR ANGULO: = 0 TO 43 DO
            BEGIN
```

```
                ESPACOS:=TRUNC(SIN(ANGULO/7)*
                                FACTOR)+CENTRO;

                FOR CICLO:=1 TO ESPACOS DO
                    BEGIN
                    WRITE (' ');  {um espaco}
                    END;
                WRITELN ('+');
                END

        UNTIL FALSO

    END.
```

Para usar o programa no seu computador deve indicar a largura em caracteres do seu visor na secção CONST.

O programa imprime uma sequência contínua de sinais '+' precedidos por um certo número de espaços. Este número de espaços é proporcional a uma curva sinusoidal.

O programa funcionará no meu compilador sem quaisquer problemas.

Note a função TRUNC. É necessário usá-la de modo a evitar a atribuição do resultado real de Seno à variável ESPAÇOS, o que provocaria um erro, dado que ESPAÇOS é uma variável inteira.

CONVERSÃO DE TEMPERATURAS

Este programa permite converter uma temperatura em graus Celsius na temperatura equivalente em Fahrenheit.

```
PROGRAM CONV(INPUT,OUTPUT);
VAR
```

```
    FAR,CENT:REAL;

BEGIN

    WRITE ('Centígrados?');
    READLN (Cent);
    FAR:=1.8*CENT+32;
    WRITELN ('Fahrenheit=',FAR:8.2)

END.
```

[Os utilizadores do meu compilador deverão omitir os valores de formatação numérica na instrução Writeln].

EQUAÇÕES QUADRÁTICAS

Este programa simples resolverá uma equação da forma "a*x*x+b*x+c=0", considerando que todas as raízes são reais.

```
PROGRAM QUAD(INPUT,OUTPUT);

VAR
    A,B,C:REAL;

BEGIN

WRITE ('Indique a,b e c');
READLN (A,B,C);

WRITELN ('As raizes sao: ',
          (-B+SQRT(B*B-4*A*C)) / (2*A), 'e',
          (-B-SQRT(B*B-4*A*C)) / (2*A) )
```

END.

Talvez o leitor deseje alterar o programa inserindo-o num ciclo REPEAT, permitindo a resolução de mais do que uma equação em cada execução.

VI

"PROCEDURES" DEFINIDAS PELO UTILIZADOR

Em qualquer linguagem de programação, a mecânica de um programa pode opôr-se ao raciocínio por trás do programa e do algoritmo (o algoritmo é o método usado para resolver algum problema; por exemplo, no final do capítulo sobre funções foi dado um algoritmo para a conversão de radianos em graus).

Isto pode acontecer de um modo dramático em programas compridos escritos em Basic. A listagem do meu compilador é dada no fim deste livro; se a consultar agora, necessitará de alguns dias para compreender como funciona, porque o algoritmo foi escondido pela codificação.

A solução para este problema consiste em pôr de parte a codificação do programa e indicar apenas o algoritmo ao computador.

O Pascal é a única linguagem que se aproxima desta solução ideal.

Permite-nos resolver um problema de um modo bastante semelhante ao que seria utilizado por um ser humano.

Se lhe fosse pedido que comprasse um artigo qualquer, o leitor dividiria subconscientemente essa tarefa em elementos mais pequenos. Poderia exprimi-la do seguinte modo:

1. Arranjar dinheiro
2. Ir à loja

3. Escolher o artigo
4. Pagá-lo
5. Voltar para casa.

Isto é lógico e fácil de compreender.

Se lhe fosse pedido em seguida que codificasse um programa de comando de um robot capaz de executar a mesma tarefa, o código seria muito menos óbvio:

1. Verificar se as pilhas do robot estão bem carregadas;
2. Enviar um sinal às pernas, instruindo-as no sentido de se deslocarem em direcção à porta;
3. Abrir a porta;
4. Passar através da porta;
5. Fechar a porta;
6. Instruir as pernas para caminharem até à loja.

E mesmo isto já é uma simplificação.

Em Pascal seria possível codificar o mesmo programa do seguinte modo:

```
BEGIN

    OBTER_DINHEIRO;
    IR_A_LOJA;
    ESCOLHER_ARTIGO;
    PAGAR_ARTIGO;
    VOLTAR_A_CASA

END.
```

O programa é agora imediatamente compreensível, mesmo para alguém que nunca tenha visto um computador.

O programa anterior é escrito definindo alguns novos "procedimentos" ("procedures") com nomes como "OBTER_DINHEIRO". Estas "procedures" podem ser usadas nas mesmas circunstâncias que WRITE e READ.

Por sua vez, a definição de cada "procedure" pode ser uma lista de outras "procedures". Por exemplo, "VOLTAR_A_CASA" pode consistir no seguinte:

```
SAIR_DA_LOJA
CAMINHAR_NA_RUA
ABRIR_PORTA_DA_CASA
ENTRAR_EM_CASA
FECHAR_A_PORTA.
```

O aspecto aqui mais importante é que o uso de "procedures" definidas pelo utilizador permite separar as tarefas que constituem o programa e executá-las independentemente. Um efeito lateral bastante útil disto é verificar em separado todas as procedures antes de as combinar num mesmo programa.

DEFINIÇÃO DE PROCEDURES

Existem diversos tipos diferentes de procedures definidas pelo utilizador. Por agora trataremos apenas do tipo mais simples.

As procedures podem possuir argumentos, do mesmo modo que a instrução WRITE, ou podem ser chamadas por si sós, sem qualquer argumento, como nos exemplos anteriores.

Os dois tipos são diferenciados pela presença ou ausência de um par de parêntesis depois do nome da procedure. Note que não pode invocar uma procedure que envolva argumentos sem definir estes, e vice-versa.

As definições de procedures fazem parte da secção de declarações de um programa. Aparecem entre a secção VAR de

um programa e o BEGIN principal. Cada definição de uma procedure deve terminar por um ponto e vírgula.

O caso geral de uma definição de procedure é o seguinte:

PROCEDURE <identificador>;

CONST
—
—
VAR
—
—
BEGIN
—
—
END;

A palavra PROCEDURE alerta o compilador sobre a presença de uma definição de procedure, sendo nessas condições análoga a VAR e CONST.

O <identificador> é o nome do procedimento. O nome deve satisfazer as regras normais, indicadas no capítulo um ao descrever o nome do programa.

As secções CONST e VAR de uma procedure podem ser omitidas. Definem simplesmente quaisquer variáveis e constantes necessárias à procedure.

A palavra BEGIN marca o início das instruções que constituem a procedure. Estas instruções são terminadas pela palavra END e um ponto e vírgula.

Uma procedure pode aceder variáveis definidas no programa principal (sendo por isso que as definições de procedures ocorrem após as definições de variáveis e constantes). Parece portanto desnecessário permite-lhes que definam variáveis e constantes

próprias. A razão do uso destas é que depois de a procedure ter sido executada, as variáveis declaradas na procedure são anuladas, libertando a memória do computador para outras coisas.

Nestas condições, o uso de variáveis "locais" é útil na conservação da memória (para ser exacto, a razão mais importante da sua existência é que permitem o recurso mais fácil à procedure; este recurso será tratado mais adiante).

Por exemplo, vejamos uma procedure que limpa o visor (a maior parte dos visores permitem a recepção de um código específico que limpa a imagem automaticamente. O método usado nesta procedure funcionará para qualquer terminal/VDU ("Video Display Unit"), sendo mais útil para trabalho simples).

```
PROCEDURE CLS;

VAR
CICLO:INTEGER;

BEGIN

    FOR CICLO:=1 TO 50 DO WRITELN

END;
```

Esta procedure funciona imprimindo 50 linhas em branco. Uma procedure real deste tipo enviaria igualmente o cursor para o canto superior esquerdo do visor, algo que não é possível programar sem conhecer o terminal usado.

Para executar uma procedure, escreva simplesmente CLS no programa. Quando o compilador encontra o nome de uma procedure, salta para o bloco de código que a define. Quando a procedure está terminada, o computador volta ao programa original no ponto onde o deixou.

Existem obviamente fortes semelhanças entre as procedures Pascal e as subrotinas Basic. Muitos programadores Basic só utilizam subrotinas para blocos de código que devem ser executados mais do que uma vez no mesmo programa. O tratamento de uma procedure Pascal do mesmo modo equivale apenas a desperdiçar um recurso valioso.

É perfeitamente correcto chamar uma procedure de dentro de outra. Veremos o efeito disto mais adiante.

Vários compiladores Pascal permitem-nos escrever certas procedures como rotinas de biblioteca; isto significa que podem ser incluídas noutros programas sem ser necessário re-escrevê-las. Por exemplo, seria muito útil dispor-se de uma procedure capaz de ordenar diferentes elementos. Poderia ser declarada como uma procedure de biblioteca e usada em todos os programas.

Uma propriedade bastante úitil das procedures quando compradas aos blocos de código que incorporam é que estes podem ser examinados permitindo uma resposta fácil à pergunta "Como funciona?". Mas a pergunta "Que faz?" requer já um exame mais detalhado. Se se substituir o código original por uma chamada de procedure, bastará olhar para o identificador para ter uma resposta imediata à pergunta.

No caso de programas cuja manutenção ou alteração deve ser realizada por outras pessoas esta característica é bastante útil.

É fácil modificar a procedure CLS de modo a imprimir um número variável de linhas em branco:

```
PROCEDURE BRANCO;

VAR
    CICLO:INTEGER;

BEGIN
```

```
    FOR CICLO:=1 TO LINHAS DO WRITELN

END;
```

Se esta procedure for incorporada num programa, a variável LINHAS deve ser associada ao número de linhas em branco a imprimir. Isto pode não ser conveniente no caso de uma rotina de biblioteca, pois pressupõe que o programa que chama a rotina não utiliza uma variável chamada LINHAS para outro fim.

A solução deste problema reside em definir LINHAS como um parâmetro da procedure. A seguir indica-se o modo de definir uma tal procedure:

```
PROCEDURE BRANCO(LINHAS:INTEGER);

VAR
    CICLO:INTEGER;

BEGIN

    FOR CICLO:=1 TO LINHAS DO WRITELN

END;
```

É então possível chamar a procedure com qualquer das seguintes instruções:

```
BRANCO(10)
BRANCO(olá)
BRANCO(3+A)
```

Sempre que o compilador encontra uma chamada de procedure incluindo parâmetros, copia o valor contido nos parêntesis

para a variável declarada no título da procedure, comparando simultaneamente o tipo do argumento da chamada de procedure com o especificado no título desta.

É importante notar que se fosse realizada qualquer alteração no valor de LINHAS no corpo da procedure, esta alteração não seria reflectida no valor da variável usada para chamar a procedure. Isto é óbvio quando se considera que a procedure pode ser chamada com uma expressão em vez de uma variável. Esta expressão pode ser um número, e seria impossível alterar o valor do número de modo a reflectir qualquer alteração em LINHAS.

Tudo isto é muito interessante, mas cai por terra se quisermos escrever uma procedure que faça alguma coisa como trocar os valores de duas variáveis. Talvez o leitor pense que a procedure seguinte realiza este truque, mas se a experimentar verificará que tal não acontece:

```
PROCEDURE TROCA(A,B:INTEGER);

VAR
    TEMP:INTEGER;
        BEGIN
            TEMP:=A;
            A:=B;
            B:=TEMP

END;
```

A razão do fracasso desta procedure é que apesar de todas as variáveis estarem convenientemente ajustadas, as alterações não se se reflectem nas variáveis usadas para chamar a rotina (como se referiu acima, esta procedure podia até ser chamada sem usar variáveis). A procedure que se segue realizará porém a tarefa correctamente:

```
PROCEDURE TROCA(VAR A,B:INTEGER);

VAR
    TEMP:INTEGER;

BEGIN

    TEMP:=A;
    A:=B;
    B:=TEMP

END;
```

A única diferença entre esta procedure e a anterior reside na presença da palavra VAR nos parêntesis depois do indentificador da procedure.

Quando todas as variáveis de um título de procedure estão precedidas pela palavra VAR, as variáveis transformam-se em "pseudónimos" das usadas para chamar o programa. Significa isto que quaisquer alterações realizadas nas variáveis interiores da procedure serão reflectidas nas variáveis usadas na instrução de chamada.

Pode-se portanto utilizar as variáveis delaradas deste modo como se fossem as variáveis originalmente usadas para chamar a procedure.

A desvantagem deste método é que não é possível usar expressões para invocar a procedure — só servirão variáveis do tipo adequado.

É possível misturar os diferentes tipos de argumento de uma procedure separando as diferentes partes com pontos e vírgulas:

```
PROCEDURE TANGENTE(ARG:REAL;QUAD:INTEGER);
PROCEDURE TITULO(A:CHAR;VAR HE:REAL);
```

Assim, ao contrário das subrotinas Basic, as procedures de Pascal podem actuar um pouco como funções na medida em que podem dispor de um argumento e produzir um resultado. O problema do uso de procedures deste modo consiste em não possibilitar a inclusão de chamadas de subrotinas em expressões. Nestas condições, o Pascal permite ao utilizador defenir tanto funções como procedures. Este tópico será explicado no capítulo que se segue.

VII

FUNÇÕES DEFINIDAS PELO UTILIZADOR

As funções são declaradas de modo praticamente semelhante às procedures:

FUNCTION <nome><parâmetros>:<tipo>;

A parte <nome> declara o nome da função do modo habitual. Deve satisfazer as regras habituais dos identificadores para ser aceite.

A parte <parâmetros> é idêntica à das procedures. Tal como nestas, a parte parâmetros pode ser omitida no caso de a função não utilizar quaisquer argumentos.

A parte <tipo> declara o tipo do resultado da função. Esta nunca deve ser omitida.

O resto da instrução da função é idêntico à estrutura de uma definição de procedure.

Para terminar a definição de uma função deve-se declarar um resultado. Isto é feito com uma linha da forma:

<nome>:=<resultado>

onde <nome> é o nome da função e <resultado> é o resultado da função, que deve ser do tipo correcto, como é especificado no título da definição.

Apresentamos agora um exemplo de função simples:

```
FUNCTION SIMPLES:INTEGER;

BEGIN
    SIMPLES: = 34;
END;
```

A função não utiliza quaisquer argumentos, e produz sempre o valor 34. Bastarão algumas pequenas alterações ao programa para dar-lhe o valor da constante PI, obtendo-se um modo lento e subtil de introduzir este valor em programas.

A lista anterior de funções apresenta uma omissão notável do ponto de vista dos programadores habituados ao Basic — o Pascal não suporta uma função de números aleatórios.

Como o leitor sabe, o gerador de números aleatórios de um computador não é de facto muito aleatório, produzindo apenas números pseudo-aleatórios.

Os números pseudo-aleatórios são números que pertencem a uma longa sequência que no entanto se repete a si mesma. É impossível conseguir números verdadeiramente aleatórios, mas para o que for necessário estudemos um gerador de números "aleatórios" em Pascal:

```
FUNCTION RAND(VAR FONTE:
                INTEGER):REAL;

BEGIN

    RAND: = FONTE/65535;
    FONTE: = (25173 * FONTE + 13849)MOD 65536

END;
```

Para utilizar esta função é necessário escolher um valor inicial a partir do qual é gerada a sequência de números. Normalmente o programa que usa o gerador de números aleatórios pedirá ao utilizador que indique um número de minutos ou algum outro acontecimento pseudo-aleatório.

É sempre necessário chamar RAND com esta Fonte como argumento. A função produzirá um número entre (e possivelmente incluindo) zero e um. É fácil alterar este valor de modo a ser sempre inteiro, para simular por exemplo o lançamento de um dado.

A função tal como se encontra pode ultrapassar as potencialidades de um computador incapaz de suportar inteiros de 32 bits.

A necessidade de indicação de uma fonte pode ser eliminada através de algum esforço de programação.

O principal uso de uma função deste tipo é em jogos que necessitam de um elemento aleatório. Estes jogos obrigam muitas vezes o utilizador a ler páginas de instruções antes de o jogo começar. O final das instruções é normalmente assinalado carregando na tecla de retorno ("Enter" ou "Return"). Basta portanto escolher um número base igual a 10, por exemplo, executando em seguida um ciclo WHILE que é executado enquanto a máquina espera pelo accionamento da tecla de retorno. Este ciclo pode aliás incrementar o número-fonte. Como o tempo necessário para ler uma página de instruções pode variar bastante, a fonte usada será neste caso pseudo-aleatória.

As funções definidas pelo utilizador têm um uso limitado para o leitor dado o seu conhecimento reduzido de Pascal, dado que muitas tarefas requerem a definição de mais do que um valor. Veremos mais tarde como é possível vencer esta dificuldade.

Em situações a usar como a que se segue encontramos uma forma simples de utilizar funções:

```
REPEAT
    JOGAR_JOGO
UNTIL JOGO_TERMINADO
```

JOGO_TERMINADO pode então ser uma função produzindo um valor booleano, Verdadeiro ou Falso, conforme o jogo já terminou de facto ou não.

O capítulo que se segue será dedicado à apresentação de mais programas de exemplo, que nos darão mais algumas ideias sobre o uso de funções.

VIII

PROGRAMAS

MÉTODO NEWTON-RAPHSON

Este programa realiza um dos processos básicos da análise numérica.

Pode resolver a maior parte das equações do tipo:

$f(x) = 0$

determinando "x" quando "f(x)" é igual a zero.

Escolhendo cuidadosamente f(x), torna-se portanto possível resolver a maior parte das equações.

A equação f(x) é guardada na definição de funções do programa seguinte, e pode ser alterada se se desejar.

A equação base é:

$f(x) = x * x - 2$

Se f(x) for zero, "x" deverá ser o quadrado de dois, pelo que o programa na sua forma actual determina raízes quadradas. Podem substituir-se outros números pelo "2".

```
PROGRAM RAPHSON(INPUT,OUTPUT);
```

```
    VAR
X,S,INICIO,ERRO,T,B:REAL;

FUNCTION F(X:REAL):REAL;
    BEGIN
        F: = X * X - 2
        END;

    BEGIN

        WRITE ('Indique ponto de partida: ');
        READLN (INICIO);
        WRITE ('Indique erro maximo: ');
        READLN (ERRO);

        S: = INICIO;
        X: = S;

        WHILE ABS (F(X)) > = ERRO DO
            BEGIN
            T: = F(X);
            X: = X + 0.00001;
            B: =(F(X)-T)/0.00001;
            S: = S - T/B;
            X: = S;
            WRITELN (X)
            END

    WRITELN ('A solução e ',X)

END.
```

Quando se faz executar este programa, a máquina pede-nos

dois números. O primeiro é o ponto a partir do qual o programa determina a raíz. A indicação de "1" produz normalmente bons resultados.

A máquina pede-nos em seguida que indiquemos o erro máximo admissível. Em resposta a esta pergunta devemos indicar um valor bastante reduzido, de cerca de 0,0000001. Se mais tarde necessitarmos de uma resposta com um rigor ainda superior bastará diminuir o erro máximo de um factor de 10.

IX

TIPOS DE DADOS DEFINIDOS PELO UTILIZADOR

As propriedades das variáveis declaradas como pertencendo a qualquer dos tipos discutidos nos capítulos anteriores são determinadas pela versão de Pascal em uso. É possível definir os nossos próprios tipos de variáveis, que podem ser concebidas de modo a adaptarem-se melhor a um problema concreto do que os tipos "standard".

Os novos tipos de variáveis são declarados numa parte de declaraçao de tipos que é colocada entre as partes de declaração de constantes (CONS) e variáveis (VAR) de um programa.

As declarações de tipo podem igualmente ocorrer na parte de declaração de uma procedure ou função, permitindo-nos usar tipos locais de dados.

A declaração de um tipo escalar é simplesmente uma lista dos valores que uma variável deste tipo pode assumir. A instrução

```
TYPE RESPOSTA = (SIM,NAO);
```

indica que uma variável do tipo RESPOSTA só pode assumir odois valores, SIM e NAO, e nenhum outro. Este novo tipo pode ser usado nas partes VAR do modo habitual:

```
VAR A:RESPOSTA;
```

É possível combinar as duas declarações:

```
VAR A:(SIM,NAO);
```

Esta segunda forma é normalmente evitada, dado que não envolve a palavra RESPOSTA, que é bastante útil para indicar o objectivo do tipo criado.

Outros excemplos de tipos escalares:

```
TYPE

SAUDACAO = (OLA,BOM_DIA);
GRUPO = (LED_ZEPPELIN,STEPPENWOLF);
CANCAO = (KASHMIR,BLACK_DOG);
```

Um valor não pode pertencer a mais do que um tipo. Portanto, a instrução seguinte é ilegal:

```
TYPE

FELINO = (LEAO,TIGRE,GATO,LINCE);
DOMESTICO = (CAO,GATO,PEIXE — DOURADO);
```

Os nomes dos valores listados na instrução de um tipo escalar são constantes desse tipo. Podemos portanto escrever:

```
FERIADO: = DOMINGO;
RELACAO: = PRIMO;
```

Não são permitidas atribuições mistas. Isto significa que só se podem atribuir os valores do tipo FELINO a uma variável desse tipo.

Não é possível realizar operações matemáticas como a de

soma ou subtracção envolvendo variáveis de um tipo escalar, mas pode-se fazer testes com os operadores relacionais(>, <, etc). Assim, DOMINGO <SEGUNDA_FEIRA.

As funções PRED e SUCC podem ser aplicadas a variáveis de um tipo escalar. Nestas condições, PRED(DOMINGO) = = SABADO e SUCC(DOMINGO) = SEGUNDA_FEIRA, considerando um tipo escalar dos dias da semana.

O SUCC do último elemento da lista não se encontra definido, tal como o PRED do primeiro elemento de uma lista.

A função ORD dá-nos a posição do seu argumento na lista. No entanto, ORD(DOMINGO) é 0, dado que a contagem começa por zero.

Se bem que se possam usar escalares na maior parte das situações em que se utilizam variáveis normais, não podem ser usados em instruções READ ou WRITE.

Pode-se apenas WRITE o valor ORD de uma variável escalar.

☆☆☆☆

Um tipo "subdomínio" é definido por duas constantes, que podem ser elas próprias constantes escalares. Por exemplo:

```
TYPE

INDICE = 1..20;
```

Isto declara um inteiro (dado que as constantes usadas na instrução são inteiras) cujo domínio é limitado. As duas constantes usadas devem ser, tal como neste caso, do mesmo tipo e diferentes, tendo a primeira um menor valor ORDinal.

Note-se que não são permitidos subdomínios de Real.

O tipo escalar associado a um subdomínio é o tipo de

constantes usado para o declarar.

As variáveis de subdomínio são declaradas do modo habitual nas secções VAR:

```
VAR

CICLO:INDICE;
```

Tal como nas instruções escalares, as instruções de TYPE e VAR podem ser combinadas numa secção VAR, mas isto nem sempre é muito prudente.

Qualquer operador que possa ser usado com uma variável de um dado tipo particular pode também ser usado num subdomínio desse tipo.

Nestas condições, uma variável de tipo INDICE comporta-se exactamente como uma variável inteira normal, excepto no que se refere às restrições que os seus valores podem assumir.

READ e WRITE (e os seus derivados) só podem ser usados em subdomínios do tipo INTEGER e CHAR.

Um uso subtil destes subdomínios em algumas aplicações é a verificação de domínios. Por exemplo, se quisermos garantir que um dado valor indicado pelo utilizador de um programa seja um inteiro numa dada gama, necessitamos apenas de definir um tipo subdomínio, permitindo ao utilizador atribuir o valor directamente à variável em causa. Se o valor indicado não se encontrar dentro dos limites especificados, resultará um erro.

A declaração IF é útil para escolher uma de duas possibilidades em função do valor de uma expressão booleana. A instrução CASE (caso) é uma declaração IF mais generalizada; permite escolher uma de diferentes acções em função do valor de uma expressão escalar ou de subdomínio.

Por exemplo, suponhamos que existia um espectáculo de circo com os seguintes preços:

Crianças: – 50$00
Estudantes: – 100$00
Adultos: – 250$00

A instrução CASE pode então ser usada para calcular o bilhete:

```
TYPE

TIPOPESSOA = (CRIANCA,ESTUDANTE,ADULTO);

VAR

PRECO:INTEGER;

PESSOA:TIPOPESSOA;

BEGIN

—
—
—

CASE PESSOA OF

CRIANCA:
    PRECO: = 50;

ESTUDANTE:
    PRECO: = 100;

ADULTO:
    PRECO: = 250
```

```
END;

WRITELN ('O preco e ',PRECO)

END.
```

A variável Pessoa na instrução CASE é designada "selector de caso". Os diferentes valores possíveis do selector de caso surgem na instrução CASE como etiquetas de caso. A seguir a cada etiqueta encontra-se uma instrução. A instrução de CASE permite escrever programas como o anterior de uma forma mais óbvia do que recorrendo à instrução IF.

Se for necessária mais do que uma instrução após uma etiqueta de caso é necessário usar a construção BEGIN-END.

É possível escrever instruções CASE como a seguinte, com mais do que uma etiqueta de caso para cada instrução (note que qualquer elemento só pode ocorrer num etiqueta de caso):

```
CASE COD OF

DOMINGO:;

SEGUNDAFEIRA,TERCAFEIRA: ----------;

END;
```

Uma das anteriores etiquetas de caso aponta para uma instrução vazia. Isto é perfeitamente legal, podendo ser útil para evitar codificar toda a instrução CASE noutra instrução IF.

X

QUADROS

Muitos programas envolvem a manipulação de grandes quantidades. Por exemplo, um problema típico consistiria em fornecer ao computador as notas concedidas a 345 estudantes num exame, deixando depois à máquina a tarefa de determinar o valor de passagem de tal modo que houvesse um aproveitamento de 60% dos estudantes.

A parte crucial do problema consiste em processar 345 dados diferentes. A única maneira de guardar esta quantidade de informação no computador (tendo em conta o que aprendemos até agora) consiste em declarar 345 variáveis do tipo inteiro, cada uma delas com uma nota.

Se for possível fazer uma instrução VAR apropriada, o problema ficará em princípio resolvido. Ocorre no entanto um problema quando se torna necessário introduzir as notas no computador. Seria necessário recorrer a 345 instruções READLN!

O tipo «quadro» (*array*) do Pascal é uma colecção de qualquer número de variáveis do mesmo tipo, todas referenciadas pelo mesmo nome.

Por exemplo, um quadro chamado NOTAS poderia ser usada para guardar as notas dos estudantes no problema anterior. Para indicar ao computador qual das 345 notas individuais estamos a considerar, utiliza-se um índice. Este é formado por um par de parêntesis rectos contendo um número entre 1 e 345.

NOTAS [3] seria a nota do terceiro estudante, NOTAS [45] a do quadragésimo quinto, etc. Todos estes valores são elementos de um mesmo quadro designado por NOTAS.

As variáveis são declaradas na secão VAR de um programa do seguinte modo:

```
VAR

NOTAS:ARRAY [1..345] OF INTEGER;
```

A secção VAR deve indicar ao computador o nome do quadro e os limites do seu índice.

Depois de um quadro ter sido declarado, os seus elementos separados podem ser tratados como variáveis simples do tipo inteiro:

```
NOTAS [0]:=324-JOAO;
NOTAS [345]:=1-NOTAS [2];
```

Este não é um exemplo importante do uso de quadros. Os quadros são particularmente úteis quando combinados em ciclos FOR, dado que se torna então possível ler todas as 345 notas com apenas as seguintes instruções:

```
FOR CICLO: = 1 TO 345 DO
    BEGIN
        WRITE ('Indique nota do estudante ',CICLO);
        READLN (NOTAS [CICLO])
END;
```

Poder-se-ia usar uma codificação semelhante para determinar a média dos valores e dar saída às notas.

Um programa completo deste tipo deveria utilizar constan-

tes. Por exemplo, é provável que, no caso de ser usado o mesmo programa para o exame seguinte, o número de estudantes se tenha alterado.

Sem uma linha do tipo:

```
CONTS

ESTUDANTES = 345;
```

tornar-se-ia aborrecido fazer as modificações necessárias.

Ao contrário do Basic, o Pascal permite-nos definir quadros completamente iguais a outros recorrendo a uma linha do tipo:

```
NOTAS: = ANTIGASNOTAS;
```

(considerando que ambas as variáveis são quadros com as mesmas dimensões).

Os quadros também podem ser usados como parâmetros de ''procedures'' e funções.

Nestas condições, os parêntesis rectos só são necessários no caso de o programa referir um elemento particular de um quadro.

APÊNDICE I

USO DO COMPILADOR

O compilador apresentado neste apêndice é bastante sofisticado e funciona bem naquilo que se pretende que faça. É importante ter em conta que não substitui de modo algum um compilador comercial, mas que é perfeito para aprendizagem dos aspectos básicos da sintaxe Pascal.

O compilador é apresentado sob a forma de três listagens Basic, uma em Basic Microsoft, comum a muitas máquinas, outra na forma popular da Basic Sinclair (para o ZX Spectrum) e outra para o Microcomputador BBC.

Todas as versões funcionam do mesmo modo.

O programa Pascal é escrito em instruções DATA localizadas entre as linhas 500 e 999. A ordem RUN 1000 ou GO TO 1000 converterá este programa Pascal num programa Basic, que é renumerado de modo a começar na linha 10.

Nas versões aqui reproduzidas, o programa Basic proveniente de um programa Pascal é simplesmente impresso, sem qualquer tentativa de o fazer executar. Isto deve-se ao facto de o método a utilizar para tal ser diferente nas diversas máquinas.

Os possuidores do ZX Spectrum terão de escrever novamente o programa Basic para o poderem executar.

Os Microcomputadores BBC podem enviar o programa Basic para fita magnética executando uma ordem *SPOOL antes de fazer executar o compilador. Pode-se então recorrer a uma

simples ordem *EXEC para fazer reentrar o programa na memória.

Os utilizadores de Basic Microsoft terão em geral de conceber rotinas próprias para reenviar o código traduzido para a máquina.

A descrição que se segue baseia-se na versão Basic Microsoft:

O compilador será executado em qualquer computador que suporte o seguinte:

Nomes de variáveis com dois caracteres;
Linhas de várias instruções;
Quadros de cadeias alfanuméricas;
Quadros numéricos;
As seguintes palavras-chave:
READ
DATA
REM
GOSUB
DIM
END
RESTORE
PRINT
IF...THEN...
GOTO
FOR...=...TO...
NEXT...
RETURN
LEN
MID$
AND
OR
NOT
DEF FN...

CHR$
LEFT$
RIGHT$
INT
ABS
ATN
COS
EXP
LN
ASC
SQR

Cada compilador tem cerca de 24 K, incluindo os meus copiosos comentários em REM.

Se decidir omitir estas declarações REM quando teclar o programa, tenha o cuidado de assegurar que as instruções REM referenciadas por GOTOs e GOSUBs são deixadas, mesmo que se mantenham apenas como REMs vazias.

O programa funciona examinando o programa Pascal caracter a caracter e formando um programa Basic equivalente.

Este processo não é de modo algum difícil, consistindo a parte mais importante em garantir a execução de verificações da sintaxe completas. Assim, mesmo tendo em conta que o compilador não é o método mais eficaz de executar programas Pascal, permite sem dúvida ao leitor familiarizar-se com a sintaxe Pascal antes de adquirir um compilador comercial.

O compilador para o ZX Spectrum é idêntico de muitos pontos de vista à versão Microsoft, tendo porém em conta as restrições do Basic Sinclair quanto ao tratamento de cadeias.

A versão BBC está mais próxima da Microsoft original (escrita num Osborne I com compilador Basic), se bem que seja um tanto melhorada.

A versão de Pascal suportada pelo compilador é:

AND
BEGIN
CONST
DIV – ver MOD
DO
DOWNTO
ELSE
END
FOR
IF
MOD – só permitida se a sua versão do Basic suporta MOD
NOT
OR
REPEAT
THEN
TO
UNTIL
VAR

TRUE
FALSE

INTEGER
BOOLEAN
REAL
CHAR

ABS
ATN
CHR
COS
EXP
LN

ODD
ORD
PRED
ROUND
SIN
SQR
SQRT
SUCC
TRUNC

APÊNDICE II

O COMPILADOR BASIC MICROSOFT

```
 500 DATA CONST
 510 DATA SAUD='OLÁ';
 520 DATA BEGIN
 530 DATA IF SAUD='OLÁ' THEN
 540 DATA BEGIN
 550 DATA WRITE ('SIM')
 560 DATA END
 610 DATA END.
1000 REM ------------------------------------------
1010 REM ------------------------------------------
1020 REM ----------COMPILADOR ----------
1030 REM ------------------------------------------
1040 REM ----------PASCAL----------
1050 REM ------------------------------------------
1060 REM ------------------------------------------
1070 REM
1080 REM        Por Jeremy Ruston
1090 REM        Copyright (C) 1982
1100 REM        VERSÃO MICROSOFT BASIC
1110 REM ------------------------------------------
1120 REM
1130 REM >>>>> Obter Pascal
1140 GOSUB 1310
```

```
1150 REM >>>>> Inicializar
1160 DIM BA$ (LE*2+8), VA$(20), TV(20), CO$(20),
     TC(20), SS$(50)
1170 SP=50
1180 LP=1
1190 CP=1
1200 PV=1
1210 PC=1
1220 JA$=""
1230 JB$=""
1240 KA$=""
1250 MA$=""
1260 REM >>>>> Compilá-lo
1270 GOSUB 5170
1280 REM >>>>> Saída Basic
1290 GOSUB 1500
1300 END
1310 REM ----------
1320 REM Obter Pascal
1330 REM ----------
1340 RESTORE
1350 PRINT "PASCAL:":PRINT
1360 LE=1
1370 READ A$
1380 PRINT A$
1390 IF A$="END." THEN GOTO 1420
1400 LE=LE+1
1410 GOTO 1370
1420 LE=LE+1
1430 DIM PA$(LE)
1440 RESTORE
1450 FOR T=1 TO LE-1
1460 READ PA$(T)
```

```
1470 NEXT T
1480 PA$(LE)="....."
1490 RETURN
1500 REM ------------------------------------------
1510 REM SAÍDA BASIC
1520 REM ------------------------------------------
1530 PRINT:PRINT "BASIC:":PRINT
1540 FOR T=1 TO LB-1
1550 PRINT T=10;" ";BA$(T)
1560 NEXT T
1570 RETURN
1580 REM ------------------------------------------
1590 REM INCREMENTAR INDICADORES PASCAL
1600 REM ------------------------------------------
1610 CP=CP+1
1620 IF CP>LEN(PA$(LP)) THEN CP=1:LP=LP+1
1630 RETURN
1640 REM ------------------------------------------
1650 REM DECREMENTAR INDICADORES PASCAL
1660 REM ------------------------------------------
1670 CP=CP-1
1680 IF CP=0 THEN LP=LP-1:CP=LEN(PA$(LP))
1690 RETURN
1700 REM ------------------------------------------
1710 REM OBTER CARACTER PASCAL SEGUINTE
1720 REM E INCREMENTAR INDICADORES
1730 REM ------------------------------------------
1740 A$=MID$(PA$(LP),CP,1)
1750 GOSUB 1580:REM >>>>> INCREMENTAR
     INDICADORES PASCAL
1760 RETURN
1770 REM ------------------------------------------
1780 REM OBTER CARACTER PASCAL SEGUINTE,
```

```
1790 REM IGNORANDO ESPAÇOS, E
1800 REM INCREMENTAR INDICADORES
1810 REM ------------------------------------------
1820 GOSUB 1700:REM >>>>> OBTER CARACTER
     PASCAL SEGUINTE
1830 IF A$=" " THEN GOTO 1820
1840 RETURN
1850 REM ------------------------------------------
1860 REM ERRO
1870 REM ------------------------------------------
1880 PRINT:PRINT:PRINT
1890 IF LP<>1 THEN PRINT PA$(LP-1)
1900 PRINT PA$(LP)
1910 IF LP<>LE THEN PRINT PA$(LP+1)
1920 PRINT ">>>>>> ERRO - ";A$;CHR$(7)
1930 END
1940 REM ------------------------------------------
1950 REM PROTEGER INDICADORES PASCAL
1960 REM ------------------------------------------
1970 LG=LP
1980 CG=CP
1990 RETURN
2000 REM ------------------------------------------
2010 REM RESTORE INDICADORES PASCAL
2020 REM ------------------------------------------
2030 LP=LG
2040 CP=CG
2050 RETURN
2060 REM ------------------------------------------
2070 REM OBTER ALGARISMO
2080 REM ------------------------------------------
2090 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
```

```
2100 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2110 IF A$>="0" AND A$<="9" THEN RETURN
2120 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2130 A$=" "
2140 RETURN
2150 REM ------------------------------------
2160 REM OBTER LETRA
2170 REM ------------------------------------
2180 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
2190 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2200 IF A$>="A" AND A$<="Z" THEN RETURN
2210 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2220 A$=" "
2230 RETURN
2240 REM ------------------------------------
2250 REM OBTER INTEIRO SEM SINAL
2260 REM ------------------------------------
2270 AA$=" "
2280 GOSUB 2060:REM >>>>> OBTER ALGARISMO
2290 IF A$=" " THEN A$=AA$:RETURN
2300 AA$=AA$+A$
2310 GOTO 2280
2320 REM ------------------------------------
2330 REM OBTER IDENTIFICADOR
2340 REM ------------------------------------
2350 BA$=" "
2360 GOSUB 2150:REM >>>>> OBTER LETRA
2370 IF A$=" " THEN RETURN
```

```
2380 BA$=BA$+A$
2390 GOSUB 2060:REM >>>>> OBTER ALGARISMO
2400 BB$=A$
2410 BA$=BA$+A$
2420 GOSUB 2150:REM >>>>> OBTER LETRA
2430 IF A$=" " AND BB$=" " THEN
     A$=BA$:RETURN
2440 GOTO 2380
2450 REM ------------------------------------
2460 REM OBTER NÚMERO SEM SINAL
2470 REM ------------------------------------
2480 CB$=" "
2490 GOSUB 2240:REM >>>>> OBTER INTEIRO
     SEM SINAL
2500 IF A$=" " THEN RETURN
2510 CB$=CB$+A$
2520 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
2530 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2540 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2550 IF A$<>"." THEN GOTO 2620
2560 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2570 CB$=CB$+A$
2580 GOSUB 2240:REM >>>>> OBTER INTEIRO
     SEM SINAL
2590 IF A$=" " THEN RETURN
2600 CB$=CB$+A$
2610 GOSUB 1940
2620 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
```

```
2630 IF A$<>"E" THEN A$=CB$:GOSUB
     2000:RETURN
2640 CB$=CB$+A$
2650 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
2660 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2670 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2680 IF A$="+" OR A$="-" THEN CB$=CB$+
     A$:GOSUB 1770:REM >>>>> OBTER
     CARACTER SEGUINTE <> DE ESPAÇO
2690 GOSUB 2240:REM >>>>> OBTER INTEIRO
     SEM SINAL
2700 IF A$=" " THEN RETURN
2710 A$=CB$+A$
2720 RETURN
2730 REM ------------------------------------
2740 REM OBTER CADEIA
2750 REM ------------------------------------
2760 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
2770 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2780 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2790 IF A$<>"'" THEN A$=" ":RETURN
2800 DB$=CHR$(34)
2810 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2820 GOSUB 1700:REM >>>>> OBTER CARACTER
     PASCAL SEGUINTE
2830 DB$=DB$+A$
```

```
2840 IF A$<>"'" THEN GOTO 2820
2850 A$=LEFT$(DB$,LEN(DB$)-1)+CHR$(34)
2860 RETURN
2870 REM ------------------------------------
2880 REM OBTER CONSTANTE SEM SINAL
2890 REM ------------------------------------
2900 CD=CP:LD=LP
2910 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
2920 IF A$=" " THEN CP=CD:LP=LD:GOTO 3000
2930 FL=0
2940 FOR ET=1 TO PC-1
2950 IF CO$(ET)=A$ THEN FL=ET
2960 NEXT ET
2970 IF FL=0 THEN
     A$=" ":CP=CD:LP=LD:RETURN
2980 IF TC(FL)=4 THEN A$=A$+"$"
2990 RETURN
3000 GOSUB 2450:REM >>>>> OBTER NÚMERO
     SEM SINAL
3010 IF A$<>" " THEN RETURN
3020 GOSUB 2730:REM >>>>> OBTER CADEIA
3030 IF A$<>" " THEN RETURN
3040 RETURN
3050 REM ------------------------------------
3060 REM OBTER CONSTANTE
3070 REM ------------------------------------
3080 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
3090 IF A$=" " THEN GOTO 3170
3100 FL=0
3110 FOR FT=1 TO PC=1
3120 IF CO$(FT)=A$ THEN FL=FT
```

```
3130 NEXT FT
3140 IF FL=0 THEN A$=" ":RETURN
3150 IF TC(FL)=4 THEN A$=A$+"$"
3160 RETURN
3170 B$=" "
3180 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3190 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3200 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3210 IF A$="+" OR A$="-" THEN B$=A$:GOSUB
     1770:REM >>>>> OBTER CARACTER
     SEGUINTE < DE ESPAÇO
3220 GOSUB 2450:REM >>>>> OBTER NÚMERO
     SEM SINAL
3230 IF A$<>" " THEN A$=B$+A$:RETURN
3240 GOSUB 2730:REM >>>>> OBTER CADEIA
3250 IF A$<>" " THEN A$=B$+A$:RETURN
3260 A$="MAU IDENTIFICADOR DE CONSTANTE"
3270 GOTO 1850
3280 REM ------------------------------------
3290 REM OBTER VARIÁVEL
3300 REM ------------------------------------
3310 LD=LP:CD=CP
3320 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
3330 IF A$=" " THEN LP=LD:CP=CD:RETURN
3340 FL=0
3350 FOR GT=1 TO PV-1
3360 IF VA$(GT)=A$ THEN FL=GT
3370 NEXT GT
3380 IF TV(FL)=4 THEN A$=A$+"$"
```

```
3390 IF FL<>0 THEN RETURN
3400 A$=" "
3410 LP=LD:CP=CD
3420 RETURN
3430 REM ------------------------------------------
3440 REM PUSH JA$,JB$,KA$,MA$
3450 REM ------------------------------------------
3460 SS$(SP)=JA$
3470 SS$(SP-1)=JB$
3480 SS$(SP-2)=KA$
3490 SS$(SP-3)=MA$
3500 SP=SP-4
3510 RETURN
3520 REM ------------------------------------------
3530 REM PUSH PARA O STACK
3540 REM ------------------------------------------
3550 SS$(SP)=A$
3560 SP=SP-1
3570 RETURN
3580 REM ------------------------------------------
3590 REM PULL JA$,JB$,KA$,MA$
3600 REM ------------------------------------------
3610 MA$=SS$(SP+1)
3620 KA$=SS$(SP+2)
3630 JB$=SS$(SP+3)
3640 JA$=SS$(SP+4)
3650 SP=SP+4
3660 RETURN
3670 REM ------------------------------------------
3680 REM PULL DO STACK
3690 REM ------------------------------------------
3700 SP=SP+1
3710 A$=SS$(SP)
```

```
3720 RETURN
3730 REM ------------------------------------------
3740 REM OBTER FACTOR
3750 REM ------------------------------------------
3760 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3770 GB$=" "
3780 FOR GT=1 TO 3
3790 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3800 GB$=GB$+A$
3810 NEXT GT
3820 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3830 IF GB$<>"NOT" THEN GB$=" "
3840 GOSUB 2870:REM >>>>> OBTER
     CONSTANTE SEM SINAL
3850 IF A$<>" " THEN A$=GB$+A$:RETURN
3860 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
3870 IF A$<>" " THEN A$=GB$+A$:RETURN
3880 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3890 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3900 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3910 IF A$<>"(" THEN GOTO 4040
3920 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3930 A$=GB$+"("
3940 GOSUB 3520:REM >>>>> PUSH
3950 GOSUB 3430:REM >>>>> PUSH CADEIAS
3960 GOSUB 4970:REM >>>>> OBTER EXPRESSÃO
```

```
3970 GOSUB 3580:REM >>>>> PULL CADEIAS
3980 GB$=A$
3990 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4000 IF A$<>")" THEN A$="FALTA
     PARENTESIS":GOTO 1850
4010 GOSUB 3670:REM >>>>> PULL
4020 A$=A$+GB$+")"
4030 RETURN
4040 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
4050 GC$=GB$
4060 GB$=" "
4070 FOR GT=1 TO 5
4080 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4090 GB$=GB$+A$
4100 NEXT GT
4110 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4120 GN=0
4130 IF LEFT$(GB$,3)="ABS" THEN
     GN=3:A$="ABS"
4140 IF LEFT$(GB$,3)="ATN" THEN
     GN=3:A$="ATN"
4150 IF LEFT$(GB$,3)="CHR" THEN
     GN=3:A$="CHR"
4160 IF LEFT$(GB$,3)="COS" THEN
     GN=3:A$="COS"
4170 IF LEFT$(GB$,3)="EXP" THEN
     GN=3:A$="EXP"
4180 IF LEFT$(GB$,3)="LN" THEN
     GN=2:A$="LN"
```

```
4190 IF LEFT$(GB$,3)="ODD" THEN
    GN=3:A$="FNOD"
4200 IF LEFT$(GB$,3)="ORD" THEN
    GN=3:A$="ASC"
        4210 IF LEFT$(GB$,4)="PRED" THEN
            GN=4:A$="FNPR"
4220 IF LEFT$(GB$,5)="ROUND" THEN
    GN=5:A$="FNRO"
4230 IF LEFT$(GB$,3)="SIN" THEN
    GN=3:A$="SIN"
4240 IF LEFT$(GB$,4)="SQRT" THEN
    GN=4:A$="SQR"
4250 IF LEFT$(GB$,3)="SQR" THEN
    GN=3:A$="FNSQ"
4260 IF LEFT$(GB$,4)="SUCC" THEN
    GN=4:A$="FNSU"
4270 IF LEFT$(GB$,5)="TRUNC" THEN
    GN=5:A$="INT"
4280 IF GN=0 THEN A$="MÁ CHAMADA DE
    FUNÇÃO"GOTO 1850
4290 GC$=GC$+A$
4300 FOR GT=1 TO GN
4310 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4320 NEXT GT
4330 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4340 IF A$<>"(" THEN A$="FALTA PARENTESIS
    PARA CHAMADA DE FUNÇÃO":GOTO 1850
4350 GC$=GC$+A$
4360 A$=GC$
4370 GOSUB 3520:REM >>>>> PUSH
4380 GOSUB 3430:REM >>>>> PUSH CADEIAS
```

```
4390 GOSUB 4970:REM >>>>> OBTER EXPRESSÃO
4400 GOSUB 3580:REM >>>>> PULL CADEIAS
4410 GC$=A$
4420 GOSUB 3670:REM >>>>> PULL
4430 GC$=A$+GC$
4440 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4450 IF A$<>")" THEN A$="FALTA PARENTESIS
     PARA FECHAR FUNÇÃO":GOTO 1850
4460 A$=GC$+")"
4470 RETURN
4480 REM ------------------------------------
4490 REM OBTER TERMO
4500 REM ------------------------------------
4510 GOSUB 3730:REM >>>>> OBTER FACTOR
4520 IF A$="" THEN A$="FACTOR EM FALTA":
     GOTO 1850
4530 JA$=A$
4540 JB$=" "
4550 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
4560 FOR JT=1 TO 3
4570 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4580 JB$=JB$+A$
4590 NEXT JT
4600 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4610 IF JB$<>"DIV" AND JB$<>"MOD" AND
     JB$<>"AND" AND LEFT$(JB$,1)<>"*" AND
     LEFT$(JB$,1)<>"/" THEN A$=JA$:RETURN
4620 JA=1
4630 IF JB$="AND" OR JB$="DIV" OR
```

```
    JB$="MOD" THEN JA=3
4640 FOR T=1 TO JA
4650 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4660 NEXT T
4670 GOSUB 3730:REM >>>>> OBTER FACTOR
4680 IF A$="" THEN A$="FACTOR EM FALTA":
    GOTO 1850
4690 JA$=JA$+LEFT$(JB$,JA)+A$
4700 GOTO 4540
4710 REM ------------------------------------
4720 REM EXPRESSÃO SIMPLES
4730 REM ------------------------------------
4740 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
4750 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4760 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
4770 KA$=" "
4780 IF A$="+" OR A$="-" THEN
    KA$=A$:GOSUB 1770:REM >>>>> OBTER
    CARACTER SEGUINTE <> DE ESPAÇO
4790 GOSUB 4480:REM >>>>> OBTER TERMO
4800 IF A$="" THEN A$="TERMO EM FALTA":
    GOTO 1850
4810 KA$=KA$+A$
4820 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
4830 KB$=" "
4840 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4850 KB$=KB$+A$
```

```
4860 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4870 KB$=KB$+A$
4880 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4890 IF LEFT$(KB$,1) <> "+" AND
     LEFT$(KB$,1) <> "-" AND KB$ <> "OR"
     THEN A$=KA$:RETURN
4900 KA$=KA$+LEFT$(KB$,1)
4910 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4920 IF KB$="OR" THEN KA$=KA$+"R":GOSUB
     1770:REM >>>>> OBTER 'R'
4930 GOSUB 4480:REM >>>>> OBTER TERMO
4940 KA$=KA$+A$
4950 IF A$=" " THEN A$=KA$:RETURN
4960 GOTO 4820
4970 REM ------------------------------------
4980 REM OBTER EXPRESSÃO
4990 REM ------------------------------------
5000 GOSUB 4710:REM >>>>> OBTER
     EXPRESSÃO SIMPLES
5010 MA$=A$
5020 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5030 MB$=" "
5040 FOR MT=1 TO 2
5050 GOSUB  1770:REM >>>>> OBTER
     CARACTER SEGUINTE <> DE ESPAÇO
5060 MB$=MB$+A$
5070 NEXT MT
5080 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
```

```
5090 MC$=LEFT$(MB$,1)
5100 IF MC$ <> " < " AND MC$ <> " = " AND
     MC$ <> " > " THEN A$=MA$:RETURN
5110 IF MB$=" <> " OR MB$=" < = " OR
     MB$=" > = " THEN GOSUB 1770:GOSUB
     1770:A$=MB$:GOTO 5130
5120 GOSUB 1770:A$=MC$
5130 MA$=MA$+A$
5140 GOSUB 4710:REM >>>>> OBTER
     EXPRESSÃO SIMPLES
5150 A$=MA$+A$
5160 RETURN
5170 REM ----------
5180 REM ----------
5190 REM ----------
5200 REM ----------
5210 REM ==========
5220 REM ==========
5230 REM ==== PROGRAMA PRINCIPAL ====
5240 REM ==========
5250 REM ==========
5260 REM ----------
5270 REM ----------
5280 REM ----------
5290 REM ----------
5300 BA$(1)="DEF FNOD(X)=((X-INT(X/2)*2)=1)"
5310 BA$(2)="DEF FNPR(X)=X-1
5320 BA$(3)="DEF FNSU(X)=X+1"
5330 BA$(4)=DEF FNRO(X)=INT(X+0.5)"
5340 BA$(5)="DEF FNSQ(X)=X*X"
5350 LB=6
5360 GOSUB 5400:REM >>>>> SECÇÃO CONST
5370 GOSUB 5790:REM >>>>> SECÇÃO VAR
```

```
5380 GOSUB 6230:REM >>>>> BLOCO
5390 RETURN
5400 REM ------------------------------------------
5410 REM SECÇÃO CONST
5420 REM ------------------------------------------
5430  CO$(1)="TRUE":CO$(2)=FALSE":PC=3
5440 BA$(LB)="REM ------ CONST ----------"
5450 BA$(LB+1)="TRUE=-1"
5460 BA$(LB+2)="FALSE=0"
5470 LB=LB+3
5480 MD$=" "
5490 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5500 FOR MT=1 TO 5
5510 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5520 MD$=MD$+A$
5530 NEXT MT
5540 IF MD$ <> "CONST" THEN GOSUB
     2000:RETURN
5550 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR SEGUINTE
5560 IF A$="" THEN A$="IDENTIFICADOR DE
     CONSTANTE EM FALTA":GOTO 1850
5570 CO$(PC)=A$
5580 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5590 IF A$<>"=" THEN A$="FALTA SINAL DE
     IGUAL":GOTO 1850
5600 GOSUB 3050:REM >>>>> OBTER
     CONSTANTE
5610 IF A$="" THEN A$=" FALTA CONSTANTE":
     GOTO 1850
```

```
5620 IF LEFT$(A$,1)=CHR$(34) THEN TC(PC)=4
5630 BA$(LB)=CO$(PC)
5640 IF TC(PC)=4 THEN BA$(LB)=BA$(LB)+"$"
5650 BA$(LB)=BA$(LB)+"="+A$
5660 LB=LB+1
5670 PC=PC+1
5680 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5690 IF A$<>";" THEN A$="FALTA PONTO E
     VÍRGULA":GOTO 1850
5700 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5710 MD$=" "
5720 FOT MT=1 TO 5
5730 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5740 MD$=MD$+A$
5750 NEXT MT
5760 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
5770 IF MD$="BEGIN" OR LEFT$(MD$,3)="VAR"
     THEN RETURN
5780 GOTO 5550
5790 REM ------------------------------------
5800 REM SECÇÃO VAR
5810 REM ------------------------------------
5820 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5830 ME$=" "
5840 FOR T=1 TO 3
5850 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5860 ME$=ME$+A$
```

```
5870 NEXT T
5880 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
5890 IF ME$ <> "VAR" THEN RETURN
5900 GOSUB 1770:GOSUB 1770:GOSUB 1770
5910 A$="END"
5920 GOSUB 3520:REM >>>>> PUSH
5930 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
5940 IF A$="" THEN A$="FALTA
     IDENTIFICADOR":GOTO 1850
5950 GOSUB 3520:REM >>>>> PUSH
5960 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5970 IF A$=":" THEN GOTO 6000
5980 IF A$<>"," THEN A$="FALTA VÍRGULA NA
     SECÇÃO VAR":GOTO 1850
5990 GOTO 5930
6000 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
6010 IF A$<>"REAL" AND A$<>"INTEGER" AND
     A$<>"CHAR" AND A$<>"BOOLEAN" THEN
     A$="TIPO ILEGAL":GOTO 1850
6020 IF A$="REAL" THEN MM=1
6030 IF A$="INTEGER" THEN MM=2
6040 IF A$="BOOLEAN" THEN MM=3
6050 IF A$="CHAR" THEN MM=4
6060 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6070 IF A$<>";" THEN A$="FALTA PONTO E
     VÍRGULA":GOTO 1850
6080 GOSUB 3670:REM >>>>> PULL
6090 IF A$="END" THEN GOTO 6140
```

```
6100 VA$(PV)=A$
6110 TV(PV)=MM
6120 PV=PV+1
6130 GOTO 6080
6140 ME$=" "
6150 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
6160 FOR T=1 TO 5
6170 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6180 ME$=ME$+$
6190 NEXT T
6200 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
6210 IF ME$="BEGIN" THEN RETURN
6220 GOTO 5910
6230 REM
6240 REM BLOCO
6250 REM ----------
6260 BA$(LB)-"REM ---- BLOCK ----"
6270 LB=LB+1
6280 NA$=" "
6290 EN=0
6300 FOR T=1 TO 5
6310 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6320 NA$=NA$+A$
6330 NEXT T
6340 IF NA$<>"BEGIN" THEN A$="FALTA
     BEGIN":GOTO 1850
6350 GOSUB 7090:REM >>>>> OBTER
     DECLARAÇÃO
6360 IF EN=0 THEN GOTO 6350
```

```
6370 REM ------------------------------------------
6380 REM SEGUNDA PASSAGEM
6390 REM ------------------------------------------
6400 FOR T=1 TO LB-1
6410 A$=BA$(T)
6420 IF A$="REM ----END----" THEN
     GOSUB 6460
6430 IF A$="REM ----UNTIL----" THEN
     GOSUB 6970
6440 NEXT T
6450 RETURN
6460 LI=T-1
6470 BE=0
6480 IF BA$(LI)="REM ---BEGIN---" AND
     BE=0 THEN GOTO 6540
6490 IF BA$(LI)="REM ---BEGIN---" THEN
     BE=BE+1
6500 IF BA$(LI)="REM ---END""" THEN
     BE=BE-1
6510 LI=LI-1
6520 IF LI=0 THEN PRINT "BEGIN/END NÃO
     APARELHADOS":END
6530 GOTO 6480
6540 IF BA$(LI-2)="REM ---IF---" THEN
     GOTO 6750
6550 IF BA$(LI-2)="REM ---WHILE---"
     THEN GOTO 6700
6560 IF BA$(LI-2)="REM ---FOR---" THEN
     GOTO 6600
6570 BA$(LI)="REM BEGIN"
6580 BA$(T)='REM END"
6590 RETURN
6600 BA$(LI)="REM BEGIN"
```

```
6610 BA$(LI-2)="REM FOR"
6620 H$=" "
6630 H=5
6640 H$=H$+MID$(BA$(LI-1),H,1)
6650 H=H+1
6660 IF MID$(BA$(LI-1),H,1)="=" THEN
     GOTO 6680
6670 GOTO 6640
6680 BA$(T)="NEXT "+H$+":REM END"
6690 RETURN
6700 BA$(LI)="REM BEGIN"
6710 BA$(LI-2)="REM WHILE"
6720 BA$(T)="GOTO "+STR$((LI-2)*10)+":
     REM END
6730 BA$(LI-1)=BA$(LI-1)+STR$((T+1)*10)
6740 RETURN
6750 EL=0
6760 IF BA$(T+1)="REM ---ELSE---" THEN
     EL=1
6770 BA$(LI-1)="REM THEN"
6780 BA$(LI-3)=BA$(LI-3)+STR$((T+1)*10)
6790 BA$(LI-2)="REM IF"
6800 BA$(LI-4)=BA$(LI-4)+STR$((LI-1)*10)
6810 BA$(LI)="REM BEGIN"
6820 BA$(T)="REM END"
6830 IF EL=0 THEN RETURN
6840 BA$(T+1)="REM ELSE"
6850 BA$(T+2)="REM BEGIN"
6860 EN=0
6870 LL=T+3
6880 IF BA$(LL)="REM ---END---" AND
     EN=0 THEN GOTO 6940
6890 IF BA$(LL)="REM ---END---" THEN
```

```
    EN=EN+1
6900 IF BA$(LL)="REM ---BEGIN---" THEN
    EN=EN-1
6910 LL=1LL+1
6920 IF LL >= LB THEN PRINT "SECÇÃO ELSE
    ESTÁ MAL":END
6930 GOTO 6880
6940 BA$(T)="GOTO "+STR$(LL*10)+":
    REM END"
6950 BA$(LL)="REM END"
6960 RETURN
6970 LI=T-1
6980 RE=0
6990 IF BA$(LI)="REM ---REPEAT---" AND
    RE=0 THEN GOTO 7050
7000 IF BA$(LI)="REM ---REPEAT---" THEN
    RE=RE+1
7010 IF BA$(LI)="REM ---UNTIL---" THEN
    RE=RE-1
7020 LI=LI-1
7030 IF LI=0 THEN PRINT "REPEAT/UNTIL NÃO
    EMPARELHADOS":END
7040 GOTO 6990
7050 BA$(T+1)=BA$(T+1)+STR$(LI*10)
7060 BA$(LI)="REM REPEAT"
7070 BA$(T)="REM UNTIL"
7080 RETURN
7090 REM ------------------------------------
7100 REM OBTER DECLARAÇÃO
7110 REM ------------------------------------
7120 NA$=" "
7130 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
```

```
7140 FOR T=1 TO 6
7150 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7160 NA$=NA$+A$
7170 NEXT T
7180 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
7190 IF LEFT$(NA$,5)="WRITE" THEN GOTO 7480
7200 IF NA$="READLN" THEN GOTO 7780
7210 IF LEFT$(NA$,5)="BEGIN" THEN GOTO 8000
7220 IF LEFT$(NA$,3)="END" THEN GOTO 8090
7230 IF LEFT$(NA$,2)="IF" THEN GOTO 8260
7240 IF LEFT$(NA$,4)="THEN" THEN GOTO 8460
7250 IF LEFT$(NA$,4)="ELSE" THEN GOTO 8550
7260 IF NA$="REPEAT" THEN GOTO 8640
7270 IF LEFT$(NA$,5)="UNTIL" THEN GOTO 8730
7280 IF LEFT$(NA$,5)="WHILE" THEN GOTO 8860
7290 IF LEFT$(NA$,3)="FOR" THEN GOTO 9040
7300 REM ----------
7310 SEM ATRIBUIÇÃO
7320 REM ----------
7330 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
7340 IF A$="" THEN A$="VARIÁVEL
     INCORRECTA":GOTO 1850
7350 BA$(LB)=A$+"="
7360 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7370 IF A$<>":" THEN A$="FALTAM DOIS
     PONTOS NA ATRIBUIÇÃO":GOTO 1850
7380 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7390 IF A$<>"=" THEN A$="FALTA SINAL DE
     IGUAL NA DECLARAÇÃO DE
```

```
      ATRIBUIÇÃO'': GOTO 1850
7400 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
7410 IF A$="" THEN A$="AUSÊNCIA DE
     EXPRESSÃO DE ATRIBUIÇÃO'':GOTO 1850
7420 BA$(LB)=BA$(LB)+A$
7430 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7440 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7450 IF A$<>";" THEN GOSUB 2000:REM >>>>>
     PROTEGER INDICADORES PASCAL
7460 LB=LB+1
7470 RETURN
7480 REM ------------------------------------
7490 REM WRITE/WRITELN
7500 REM ------------------------------------
7510 FOR T=1 TO 5
7520 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7530 NEXT T
7540 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7550 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7570 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7580 NA$=NA$+A$
7590 IF NA$<>"LN" THEN NA$="":GOSUB 2000:
     REM >>>>> PROTEGER INDICADORES
     PASCAL
7600 BA$(LB)="PRINT "
7610 GOSUB 1770:REM >>>>> OBTER CARACTER
```

```
      SEGUINTE <> DE ESPAÇO
7620 IF A$<>"(" THEN A$="FALTA PARENTESIS
     EM WRITE/LN)":GOTO 1850
7630 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7640 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7650 GOSUB 2000:REM >>>>> PROTEGER
     INDICADORES PASCAL
7660 IF A$=")" THEN GOSUB 1770:GOTO 7720
7670 GOSUB 4970:REM >>>>> AVALIADOR DA
     EXPRESSÃO
7680 BA$(LB)=BA$(LB)+A$
7690 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7700 IF A$=",," THEN BA$(LB)=BA$(LB)+A$:GOTO
     7630
7710 IF A$<>")" THEN A$="SEPARADOR ILEGAL
     EM INSTRUÇÃO WRITE/LN":GOTO 1850
7720 IF NA$<>"LN" THEN
     BA$(LB)=BA$(LB)+";"
7730 LB=LB+1
7740 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7750 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7760 IF A$<>";" THEN GOSUB 2000:REM >>>>>
     PROTEGER INDICADORES PASCAL
7770 RETURN
7780 REM ------------------------------------
7790 REM READLN
7800 REM ------------------------------------
7810 FOR T=1 TO 6
```

```
7820 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7830 NEXT T
7840 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7850 IF A$<>"(" THEN A$="FALTA PARENTESIS
     DE INSTRUÇÃO READLN":GOTO 1850
7860 BA$(LB)="INPUT"
7870 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
7880 IF A$="" THEN A$="FALTA VARIÁVEL":
     GOTO 1850
7890 BA$(LB)=BA$(LB)+A$
7900 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7910 IF A$=")" THEN GOTO 7950
7920 IF A$<>"," THEN A$="FALTA VÍRGULA EM
     INSTRUÇÃO READLN":GOTO 1850
7930 BA$(LB)=BA$(LB)+","
7940 GOTO 7870
7950 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7960 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7970 IF A$<>";" THEN GOSUB 2000:REM >>>>>
     PROTEGER INDICADORES PASCAL
7980 LB=LB+1
7990 RETURN
8000 REM ------------------------------------
8010 REM BEGIN
8020 REM ------------------------------------
8030 BA$(LB)=REM ---BEGIN---"
8040 LB=LB+1
8050 FOR T=1 TO 5
```

```
8060 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8070 NEXT T
8080 RETURN
8090 REM ------------------------------------
8100 REM END
8110 REM ------------------------------------
8120 FOR T=1 TO 3
8130 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8140 NEXT T
8150 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
8160 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPACO
8170 GOSUB 2000:REM >>>>> PROTEGER
     INDICADORES PASCAL
8180 IF A$ <> "." THEN GOTO 8230
8190 EN=1
8200 BA$(LB)="END"
8210 LB=LB+1
8220 RETURN
8230 BA$(LB)="REM ---END---"
8240 LB=LB+1
8250 RETURN
8260 REM ------------------------------------
8270 REM IF
8280 REM ------------------------------------
8290 GOSUB 1770:GOSUB 1770:REM >>>>> OBTER
     CARACTER SEGUINTE <> DE ESPAÇO
8300 GOSUB 4970
8310 IF A$="" THEN A$="AUSÊNCIA DE
     CONDIÇÃO EM INSTRUÇÃO IF":GOTO 1850
```

```
8320 BA$(LB)="IF "+A$+" THEN "
8330 LB=LB+1
8340 BA$(LB+1)="REM ---IF---"
8350 BA$(LB)="IF NOT("+A$+") THEN "
8360 LB=LB+2
8370 ME$=" "
8380 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
8390 FOR T=1 TO 4
8400 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8410 ME$=ME$+A$
8420 NEXT T
8430 IF ME$<>"THEN" THEN A$="FALTA THEN
     NA INSTRUÇÃO IF":GOTO 1850
8440 GOSUB 2000:REM >>>>> PROTEGER
      INDICADORES PASCAL
8450 RETURN
8460 REM ------------------------------------
8470 REM THEN
8480 REM ------------------------------------
8490 FOR T=1 TO 4
8500 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8510 NEXT T
8520 BA$(LB)="REM ---THEN---"
8530 LB=LB+1
8540 RETURN
8550 REM ------------------------------------
8560 REM ELSE
8570 REM ------------------------------------
8580 FOR T=1 TO 4
8590 GOSUB 1770:REM >>>>> OBTER CARACTER
```

```
    SEGUINTE <> DE ESPAÇO
8600 NEXT T
8610 BA$(LB)="REM ---ELSE---"
8620 LB=LB+1
8630 RETURN
8640 REM ------------------------------------
8560 REM REPEAT
8660 REM ------------------------------------
8670 FOR T=1 TO 6
8680 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
8790 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
8690 NEXT T
8700 BA$(LB)="REM ---REPEAT---"
8710 LB=LB+1
8720 RETURN
8730 REM ------------------------------------
8740 REM UNTIL
8750 REM ------------------------------------
8760 BA$(LB)="REM ---UNTIL"
8770 LB=LB+1
8780 FOR T=1 TO 5
8790 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
8800 NEXT T
8810 GOSUB 4970:REM >>>>> OBTER
    EXPRESSÃO
8820 IF A$="" THEN A$="FALTA EXPRESSÃO EM
    INSTRUÇÃO UNTIL":GOTO 1850
8830 BA$(LB)="IF NOT("+A$+") THEN GOTO "
8840 LB=LB+1
8850 RETURN
```

```
8860 REM ------------------------------------
8870 REM WHILE
8880 REM ------------------------------------
8890 BA$(LB)="REM ---WHILE---"
8900 LB=LB+1
8910 FOR T=1 TO 5
8920 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8930 NEXT T
8940 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
8950 IF A$="" THEN A$="FALTA EXPRESSÃO EM
     INSTRUÇÃO WHILE":GOTO 1850
8960 BA$(LB)="IF NOT("+A$+") THEN GOTO "
8970 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8980 MM$=A$
8990 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9000 MM$=MM$+A$
9010 IF MM$<>"DO" THEN A$="FALTA DO EM
     INSTRUÇÃO WHILE":GOTO 1850
9020 LB=LB+1
9030 RETURN
9040 REM ------------------------------------
9050 REM FOR
9060 REM ------------------------------------
9070 BA$(LB)="REM ---FOR---"
9080 LB=LB+1
9090 FOR T=1 TO 3
9100 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9110 NEXT T
```

```
9120 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
9130 IF A$="" OR TV(FL)=4 THEN
     A$="INCORRECTO COMO VARIÁVEL DE
     COMANDO DE CICLO":GOTO 1850
9140 BA$(LB)="FOR "+A$+"="
9150 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9160 IF A$<>":" THEN A$=" FALTAM DOIS
     PONTOS EM INSTRUÇÃO FOR":GOTO 1850
9170 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9180 IF A$<>"=" THEN A$=FALTA SINAL DE
     IGUAL EM INSTRUÇÃO FOR":GOTO 1850
9190 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
9200 IF A$="" THEN A$="MAU VALOR INICIAL
     PARA CICLO":GOTO 1850
9210 BA$(LB)=BA$(LB)+A$+" TO "
9220 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
9230 ME$=" "
9240 FOR T=1 TO 6
9250 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9260 ME$=ME$+A$
9270 NEXT T
9280 MM=0
9290 IF LEFT$(ME$,2)="TO" THEN
MM=2:ME$="TO"
9300 IF ME$="DOWNTO" THEN MM=6
9310 IF MM=0 THEN A$="FALTA TO/DOWNTO EM
     INSTRUÇÃO FOR":GOTO 1850
9320 GOSUB 2000:REM >>>>> RESTORE
```

```
    INDICADORES PASCAL
9330 FOR T=1 TO MM
9340 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
9350 NEXT T
9360 GOSUB 4970:REM >>>>> OBTER
    EXPRESSÃO
9370 BA$(LB)=BA$(LB)+A$
9380 IF A$="" THEN A$="FALTA EXPRESSÃO EM
    INSTRUÇÃO FOR":GOTO 1850
9390 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
9400 MM$=A$
9410 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
9420 MM$=MM$+A$
9430 IF MM$<>"DO" THEN A$=" FALTA DO EM
     INSTRUÇÃO FOR":GOTO 1850
9440 IF ME$="DOWNTO" THEN
    BA$(LB)=BA$(LB)=BA$(LB)+" STEP -1"
9450 LB=LB+1
9460 RETURN

>RUN
PASCAL:
CONST
SAUD='OLÁ';
BEGIN
IF SAUD='OLÁ' THEN
BEGIN
WRITE ('SIM')
END
END.
```

BASIC:

```
 10 DEF FNOD(X)=((X-INT(X/2)*2)=1)
 20 DEF FNPR(X)=X-1
 30 DEF FNSU(X)=X+1
 40 DEF FNRO(X)=INT (X+0.5)
 50 DEF FNSQ(X)=X*X
 60 REM ----CONST----
 70 TRUE=-1
 80 FALSE=0
 90 SAUD$="OLÁ";
100 REM ----BLOCK----
110 IF SAUD$="OLÁ" THEN 140
120 IF NOT (SAUD$="OLÁ") THEN 180
130 REM IF
140 REM THEN
150 REM BEGIN
160 PRINT "SIM";
170 REM END
180 END
```

APÊNDICE III

O COMPILADOR PARA MICROCOMPUTADOR BBC

```
>LIST
 500 DATA CONST
 510 DATA SAUD='OLÁ';
 520 DATA BEGIN
 530 DATA IF SAUD='OLÁ' THEN
 540 DATA BEGIN
 550 DATA WRITE ('SIM')
 560 DATA END
 610 DATA END.
1000 REM
1010 REM
1020 REM ----------COMPILADOR----------
1030 REM  ------------------------------
1040 REM ----------PASCAL----------
1050 REM  ------------------------------
1060 REM  ------------------------------
1070 REM
1080 REM       Por Jeremy Ruston
1090 REM       Copyright (C) 1982
1100 REM       VERSÃO BBC
1110 REM  ------------------------------
1120 REM
1130 REM >>>>> Obter Pascal
```

```
1140 GOSUB 1310
1150 REM >>>>> Inicializar
1160 DIM BA$ (LE*2+8), VA$(20), TV(20), CO$(20),
     TC(20), SS$(50)
1170 SP=50
1180 LP=1
1190 CP=1
1200 PV=1
1210 PC=1
1220 JA$=""
1230 JB$=""
1240 KA$=""
1250 MA$=""
1260 REM >>>>> Compilá-lo
1270 GOSUB 5170
1280 REM >>>>> Saída Basic
1290 GOSUB 1500
1300 END
1310 REM ----------
1320 REM OBTER PASCAL
1330 REM ----------
1340 RESTORE
1350 PRINT "PASCAL:":PRINT
1360 LE=1
1370 READ A$
1380 PRINT A$
1390 IF A$="END." THEN GOTO 1420
1400 LE=LE+1
1410 GOTO 1370
1420 LE=LE+1
1430 DIM PA$(LE)
1440 RESTORE
1450 FOR T=1 TO LE-1
```

```
1460 READ PA$(T)
1470 NEXT T
1480 PA$(LE)="....................."
1490 RETURN
1500 REM ------------------------------------------
1510 REM SAÍDA BASIC
1520 REM ------------------------------------------
1530 PRINT:PRINT "BASIC:":PRINT
1540 FOR T=1 TO LB-1
1550 PRINT T*10;" "BA$(T)
1560 NEXT T
1570 RETURN
1580 REM ------------------------------------------
1590 REM INCREMENTAR INDICADORES PASCAL
1600 REM ------------------------------------------
1610 CP=CP+1
1620 IF CP>LEN(PA$(LP)) THEN CP=1:LP=LP+1
1630 RETURN
1640 REM ------------------------------------------
1650 REM DECREMENTAR INDICADORES PASCAL
1660 REM ------------------------------------------
1670 CP=CP-1
1680 IF CP=0 THEN LP=LP-1:CP=LEN(PA$(LP))
1690 RETURN
1700 REM ------------------------------------------
1710 REM OBTER CARACTER PASCAL SEGUINTE
1720 REM E INCREMENTAR INDICADORES
1730 REM ------------------------------------------
1740 A$=MID$(PA$(LP),CP,1)
1750 GOSUB 1580:REM >>>>> INCREMENTAR
     INDICADORES PASCAL
1760 RETURN
1770 REM ------------------------------------------
```

```
1780 REM OBTER CARACTER PASCAL SEGUINTE,
1790 REM IGNORANDO ESPAÇOS, E
1800 REM INCREMENTAR INDICADORES
1810 REM ------------------------------------------
1820 GOSUB 1700:REM >>>>> OBTER CARACTER
     PASCAL SEGUINTE
1830 IF A$=" " THEN GOTO 1820
1840 RETURN
1850 REM ------------------------------------------
1860 REM ERRO
1870 REM ------------------------------------------
1880 PRINT:PRINT:PRINT
1890 IF LP<>1 THEN PRINT PA$(LP-1)
1900 PRINT PA$(LP)
1910 IF LP<>LE THEN PRINT PA$(LP+1)
1920 PRINT ">>>>>> ERRO - ";A$;CHR$(7)
1930 END
1940 REM ------------------------------------------
1950 REM PROTEGER INDICADORES PASCAL
1960 REM ------------------------------------------
1970 LG=LP
1980 CG=CP
1990 RETURN
2000 REM ------------------------------------------
2010 REM RESTORE INDICADORES PASCAL
2020 REM ------------------------------------------
2030 LP=LG
2040 CP=CG
2050 RETURN
2060 REM ------------------------------------------
2070 REM OBTER ALGARISMO
2080 REM ------------------------------------------
2090 GOSUB 1940:REM >>>>> PROTEGER
```

```
    INDICADORES PASCAL
2100 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
2110 IF A$>="0" AND A$<="9" THEN RETURN
2120 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
2130 A$=" "
2140 RETURN
2150 REM ------------------------------------
2160 REM OBTER LETRA
2170 REM ------------------------------------
2180 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
2190 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
2200 IF A$>="A" AND A$<="Z" THEN RETURN
2210 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
2220 A$=" "
2230 RETURN
2240 REM ------------------------------------
2250 REM OBTER INTEIRO SEM SINAL
2260 REM ------------------------------------
2270 AA$=" "
2280 GOSUB 2060:REM >>>>> OBTER ALGARISMO
2290 IF A$=" " THEN A$=AA$:RETURN
2300 AA$=AA$+A$
2310 GOTO 2280
2320 REM ------------------------------------
2330 REM OBTER IDENTIFICADOR
2340 REM ------------------------------------
2350 BA$=" "
2360 GOSUB 2150:REM >>>>> OBTER LETRA
```

```
2370 IF A$=" " THEN RETURN
2380 BA$=BA$+A$
2390 GOSUB 2060:REM >>>>> OBTER ALGARISMO
2400 BB$=A$
2410 BA$=BA$+A$
2420 GOSUB 2150:REM >>>>> OBTER LETRA
2430 IF A$=" " AND BB$=" " THEN
     A$=BA$:RETURN
2440 GOTO 2380
2450 REM ------------------------------------
2460 REM OBTER NÚMERO SEM SINAL
2470 REM ------------------------------------
2480 CB$=" "
2490 GOSUB 2240:REM >>>>> OBTER INTEIRO
     SEM SINAL
2500 IF A$=" " THEN RETURN
2510 CB$=CB$+A$
2520 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
2530 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2540 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
2550 IF A$<>"." THEN GOTO 2620
2560 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
2570 CB$=CB$+A$
2580 GOSUB 2240:REM >>>>> OBTER INTEIRO
     SEM SINAL
2590 IF A$=" " THEN RETURN
2600 CB$=CB$+A$
2610 GOSUB 1940
2620 GOSUB 1770:REM >>>>> OBTER CARACTER
```

```
    SEGUINTE <> DE ESPAÇO
2630 IF A$<>"E" THEN A$=CB$:GOSUB
    2000:RETURN
2640 CB$=CB$+A$
2650 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
2660 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
2670 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
2680 IF A$+" OR A$="-" THEN CB$=CB$+
    A$:GOSUB 1770:REM >>>>> OBTER
    CARACTER SEGUINTE <> DE ESPAÇO
2690 GOSUB 2240:REM >>>>> OBTER INTEIRO
    SEM SINAL
2700 IF A$=" " THEN RETURN
2710 A$=CB$+A$
2720 RETURN
2730 REM ------------------------------------
2740 REM OBTER CADEIA
2750 REM ------------------------------------
2760 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
2770 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
2780 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
2790 IF A$<>",," THEN A$=" ":RETURN
2800 DB$=CHR$(34)
2810 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
2820 GOSUB 1770:REM >>>>> OBTER CARACTER
    PASCAL SEGUINTE
```

```
2830 DB$=DB$+A$
2840 IF A$<>", " THEN GOTO 2820
2850 A$=LEFT$(DB$,LEN(DB$)-1)+CHR$(34)
2860 RETURN
2870 REM ------------------------------------
2880 REM OBTER CONSTANTE SEM SINAL
2890 REM ------------------------------------
2900 CD=CP:LD=LP
2910 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
2920 IF A$=" " THEN CP=CD:LP=LD:GOTO 3000
2930 FL=0
2940 FOR ET=1 TO PC-1
2950 IF CO$(ET)=A$ THEN FL=ET
2960 NEXT ET
2970 IF FL=0 THEN
     A$=" ":CP=CD:LP=LD:RETURN
2980 IF TC(FL)=4 THEN A$=A$+"$"
2990 RETURN
3000 GOSUB 2450:REM >>>>> OBTER NÚMERO
     SEM SINAL
3010 IF A$<>" " THEN RETURN
3020 GOSUB 2730:REM >>>>> OBTER CADEIA
3030 IF A$<>" " THEN RETURN
3040 RETURN
3050 REM ------------------------------------
3060 REM OBTER CONSTANTE
3070 REM ------------------------------------
3080 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
3090 IF A$=" " THEN GOTO 3170
3100 FL=0
3110 FOR FT=1 TO PC=1
```

```
3120 IF CO$(FT)=A$ THEN FL=FT
3130 NEXT FT
3140 IF FL=0 THEN A$=" ":RETURN
3150 IF TC(FL)=4 THEN A$=A$+"$"
3160 RETURN
3170 B$=" "
3180 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3190 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3200 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3210 IF A$="+" OR A$="-" THEN B$=A$:GOSUB
     1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3220 GOSUB 2450:REM >>>>> OBTER NÚMERO
     SEM SINAL
3230 IF A$<>" " THEN A$=B$+A$:RETURN
3240 GOSUB 2730:REM >>>>> OBTER CADEIA
3250 IF A$<>" " THEN A$=B$+A$:RETURN
3260 A$="MAU IDENTIFICADOR DE CONSTANTE"
3270 GOTO 1850
3280 REM ------------------------------------
3290 REM OBTER VARIÁVEL
3300 REM ------------------------------------
3310 LD=LP:CD=CP
3320 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
3330 IF A$=" " THEN LP=LD:CP=CD:RETURN
3340 FL=0
3350 FOR GT=1 TO PV-1
3360 IF VA$(GT)=A$ THEN FL=GT
3370 NEXT GT
```

```
3380 IF TV(FL)=4 THEN A$=A$+"$"
3390 IF FL<>0 THEN RETURN
3400 A$=" "
3410 LP=LD:CP=CD
3420 RETURN
3430 REM ------------------------------------------
3440 REM PUSH JA$,JB$,KA$,MA$
3450 REM ------------------------------------------
3460 SS$(SP)=JA$
3470 SS$(SP-1)=JB$
3480 SS$(SP-2)=KA$
3490 SS$(SP-3)=MA$
3500 SP=SP-4
3510 RETURN
3520 REM ------------------------------------------
3530 REM PUSH PARA O STACK
3540 REM ------------------------------------------
3550 SS$(SP)=A$
3560 SP=SP-1
3570 RETURN
3580 REM ------------------------------------------
3590 REM PULL JA$,JB$,KA$,MA$
3600 REM ------------------------------------------
3610 MA$=SS$(SP+1)
3620 KA$=SS$(SP+2)
3630 JB$=SS$(SP+3)
3640 JA$=SS$(SP+4)
3650 SP=SP+4
3660 RETURN
3670 REM ------------------------------------------
3680 REM PULL DO STACK
3690 REM ------------------------------------------
3700 SP=SP+1
```

```
3710 A$=SS$(SP)
3720 RETURN
3730 REM ------------------------------------------
3740 REM OBTER FACTOR
3750 REM ------------------------------------------
3760 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3770 GB$=" "
3780 FOR GT=1 TO 3
3790 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3800 GB$=GB$+A$
3810 NEXT GT
3820 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3830 IF GB$<>"NOT" THEN GB$=" "
3840 GOSUB 2870:REM >>>>> OBTER
     CONSTANTE SEM SINAL
3850 IF A$<>" " THEN A$=GB$+A$:RETURN
3860 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
3870 IF A$<>" " THEN A$=GB$+A$:RETURN
3880 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
3890 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3900 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
3910 IF A$<>"(" THEN GOTO 4040
3920 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
3930 A$=GB$+"("
3940 GOSUB 3520:REM >>>>> PUSH
3950 GOSUB 3430:REM >>>>> PUSH CADEIAS
```

```
3960 GOSUB 4970:REM >>>>> OBTER EXPRESSÃO
3970 GOSUB 3580:REM >>>>> PULL CADEIAS
3980 GB$=A$
3990 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4000 IF A$<>")" THEN A$=FALTA
     PARENTESIS":GOTO 1850
4010 GOSUB 3670:REM >>>>> PULL
4020 A$=A$+GB$+")"
4030 RETURN
4040 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
4050 GC$=GB$
4060 GB$=" "
4070 FOR GT=1 TO 5
4080 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4090 GB$=GB$+A$
4100 NEXT GT
4110 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4120 GN=0
4130 IF LEFT$(GB$,3)="ABS" THEN
     GN=3:A$="ABS"
4140 IF LEFT$(GB$,3)="ATN" THEN
     GN=3:A$="ATN"
4150 IF LEFT$(GB$,3)="CHR" THEN
     GN=3:A$="CHR"
4160 IF LEFT$(GB$,3)="COS" THEN
     GN=3:A$="COS"
4170 IF LEFT$(GB$,3)="EXP" THEN
     GN=3:A$="EXP"
4180 IF LEFT$(GB$,3)="LN" THEN
```

```
    GN-2:A$3-"LN"
4190 IF LEFT$(GB$,3)="ODD" THEN
    GN=3:A$="FNOD"
4200 IF LEFT$(GB$,3)="ORD" THEN
    GN=3:A$="ASC"
4210 IF LEFT$(GB$,4)="PRED" THEN
    GN=4:A$="FNPR"
4220 IF LEFT$(GB$,5)="ROUND" THEN
    GN=5:A$="FNRO"
4230 IF LEFT$(GB$,3)="SIN" THEN
    GN=3:A$="SIN"
4240 IF LEFT$(GB$,4)="SQRT" THEN
    GN=4:A$="SQR"
4250 IF LEFT$(GB$,3)="SQR" THEN
    GN=3:A$="FNSQ"
4260 IF LEFT$(GB$,4)="SUCC" THEN
    GN=4:A$="FNSU"
4270 IF LEFT$(GB$,5)="TRUNC" THEN
    GN=5:A$="INT"
4280 IF GN=0 THEN A$="MÁ CHAMADA DE
    FUNÇÃO"GOTO 1850
4290 GC$=GC$+A$
4300 FOR GT=1 TO GN
4310 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4320 NEXT GT
4330 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4340 IF A$<>"(" THEN A$="FALTA PARENTESIS
    PARA CHAMADA DE FUNÇÃO":GOTO 1850
4350 GC$=GC$+A$
4360 A$=GC$
4370 GOSUB 3520:REM >>>>> PUSH
```

```
4380 GOSUB 3430:REM >>>>> PUSH CADEIAS
4390 GOSUB 4970:REM >>>>> OBTER EXPRESSÃO
4400 GOSUB 3580:REM >>>>> PULL CADEIAS
4410 GC$=A$
4420 GOSUB 3670:REM >>>>> PULL
4430 GC$=A$+GC$
4440 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4450 IF A$<>")" THEN A$="FALTA PARENTESIS
     PARA FECHAR FUNÇÃO":GOTO 1850
4460 A$=GC$+")"
4470 RETURN
4480 REM ------------------------------------
4490 REM OBTER TERMO
4500 REM ------------------------------------
4510 GOSUB 3730:REM >>>>> OBTER FACTOR
4520 IF A$="" THEN A$="FACTOR EM FALTA":
     GOTO 1850
4530 JA$=A$
4540 JB$=" "
4550 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
4560 FOR JT=1 TO 3
4570 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4580 JB$=JB$+A$
4590 NEXT JT
4600 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4610 IF JB$<>"DIV" AND JB$<>"MOD" AND
     JB$<>"AND" AND LEFT$(JB$,1)<>"*" AND
     LEFT$(JB$,1)<>"/" THEN A$=JA$:RETURN
4620 JA=1
```

```
4630 IF JB$="AND" OR JB$="DIV" OR
    JB$="MOD" THEN JA=3
4640 FOR T=1 TO JA
4650 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4660 NEXT T
4670 GOSUB 3730:REM >>>>> OBTER FACTOR
4680 IF A$="" THEN A$="FACTOR EM FALTA":
    GOTO 1850
4690 JA$=JA$+LEFT$(JB$,JA)+A$
4700 GOTO 4540
4710 REM ------------------------------------
4720 REM EXPRESSÃO SIMPLES
4730 REM ------------------------------------
4740 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
4750 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
4760 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
4770 KA$=" "
4780 IF A$="+" OR A$="-" THEN
    KA$=A$:GOSUB 1770:REM >>>>> OBTER
    CARACTER SEGUINTE <> DE ESPAÇO
4790 GOSUB 4480:REM >>>>> OBTER TERMO
4800 IF A$="" THEN A$="TERMO EM FALTA":
    GOTO 1850
4810 KA$=KA$+A$
4820 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
4830 KB$=" "
4840 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
```

```
4850 KB$=KB$+A$
4860 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4870 KB$=KB$+A$
4880 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
4890 IF LEFT$(KB$,1) <> "+" AND
     LEFT$(KB$,1) <> "-" AND KB$ <> "OR"
     THEN A$=KA$:RETURN
4900 KA$=KA$+LEFT$(KB$,1)
4910 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
4920 IF KB$="OR" THEN KA$=+"R":GOSUB
     1770:REM >>>>> OBTER 'R'
4930 GOSUB 4480:REM >>>>> OBTER TERMO
4940 KA$=KA$+A$
4950 IF A$=" " THEN A$=KA$:RETURN
4960 GOTO 4820
4970 REM ------------------------------------
4980 REM OBTER EXPRESSÃO
4990 REM ------------------------------------
5000 GOSUB 4710:REM >>>>> OBTER
     EXPRESSÃO SIMPLES
5010 MA$=A$
5020 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5030 MB$=" "
5040 FOR MT=1 TO 2
5050 GOSUB  1770:REM >>>>> OBTER
     CARACTER SEGUINTE <> DE ESPAÇO
5060 MB$=MB$+A$
5070 NEXT MT
5080 GOSUB 2000:REM >>>>> RESTORE
```

```
    INDICADORES PASCAL
5090 MC$=LEFT$(MB$,1)
5100 IF MC$ <> " < " AND MC$ <> " = " AND
    MC$ <> " > " THEN A$=MA$:RETURN
5110 IF MB$=" <> " OR MB$=" < = " OR
    MB$=" > = " THEN GOSUB 1770:GOSUB
    1770:A$=MB$:GOTO 5130
5120 GOSUB 1770:A$=MC$
5130 MA$=MA$+A$
5140 GOSUB 4710:REM >>>>> OBTER
    EXPRESSÃO SIMPLES
5150 A$=MA$+A$
5160 RETURN
5170 REM ----------
5180 REM ----------
5190 REM ----------
5200 REM ----------
5210 REM ==========
5220 REM ==========
5230 REM ==== PROGRAMA PRINCIPAL ====
5240 REM ==========
5250 REM ==========
5260 REM ----------
5270 REM ----------
5280 REM ----------
5290 REM ----------
5300 BA$(1)="DEF FNOD(X)=((X-INT(X/2)*2)=1)"
5310 BA$(2)="DEF FNPR(X)=X-1
5320 BA$(3)="DEF FNSU(X)=X+1"
5330 BA$(4)=DEF FNRO(X)=INT(X+0.5)"
5340 BA$(5)="DEF FNSQ(X)=X*X"
5350 LB=6
5360 GOSUB 5400:REM >>>>> SECÇÃO CONST
```

```
5370 GOSUB 5790:REM >>>>> SECÇÃO VAR
5380 GOSUB 6230:REM >>>>> BLOCO
5390 RETURN
5400 REM ————————————————————————
5410 REM SECÇÃO CONST
5420 REM ————————————————————————
5430  CO$(1)="TRUE":CO$(2)=FALSE":PC=3
5440 BA$(LB)="REM ——— CONST ———————
5450 BA$(LB+1)="TRUE=-1"
5460 BA$(LB+2)="FALSE=0"
5470 LB=LB+3
5480 MD$=" "
5490 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
5500 FOR MT=1 TO 5
5510 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5520 MD$=MD$+A$
5530 NEXT MT
5540 IF MD$ <> "CONST" THEN GOSUB
     2000:RETURN
5550 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR SEGUINTE
5560 IF A$="" THEN A$="IDENTIFICADOR DE
     CONSTANTE EM FALTA":GOTO 1850
5570 CO$(PC)=A$
5580 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5590 IF A$<>"=" THEN A$="FALTA SINAL DE
     IGUAL":GOTO 1850
5600 GOSUB 3050:REM >>>>> OBTER
     CONSTANTE
5610 IF A$="" THEN A$=" FALTA CONSTANTE":
```

```
    GOTO 1850
5620 IF"LEFT$(A$,1)=CHR$(34) THEN TC(PC)=4
5630 BA$(LB)=CO$(PC)
5640 IF TC(PC)=4 THEN BA$(LB)=BA$(LB)+"$"
5650 BA$(LB)=BA$(LB)+"="+A$
5660 LB=LB+1
5670 PC=PC+1
5680 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
5690 IF A$<>";" THEN A$="FALTA PONTO E
    VÍRGULA":GOTO 1850
5700 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
5710 MD$=" "
5720 FOT MT=1 TO 5
5730 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
5740 MD$=MD$+A$
5750 NEXT MT
5760 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
5770 IF MD$="BEGIN" OR LEFT$(MD$,3)="VAR"
    THEN RETURN
5780 GOTO 5550
5790 REM ----------
5800 REM SECÇÃO VAR
5810 REM ----------
5820 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
5830 ME$=" "
5840 FOR T=1 TO 3
5850 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
```

```
5860 ME$=ME$+A$
5870 NEXT T
5880 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
5890 IF ME$ <> "VAR" THEN RETURN
5900 GOSUB 1770:GOSUB 1770:GOSUB 1770
5910 A$="END"
5920 GOSUB 3520:REM >>>>> PUSH
5930 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
5940 IF A$="" THEN A$="FALTA
     IDENTIFICADOR":GOTO 1850
5950 GOSUB 3520:REM >>>>> PUSH
5960 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
5970 IF A$=":" THEN GOTO 6000
5980 IF A$<>"," THEN A$="FALTA VÍRGULA NA
     SECÇÃO VAR":GOTO 1850
5990 GOTO 5930
6000 GOSUB 2320:REM >>>>> OBTER
     IDENTIFICADOR
6010 IF A$<>"REAL" AND A$<>"INTEGER" AND
     A$<>"CHAR" AND A$<>"BOOLEAN" THEN
     A$="TIPO ILEGAL":GOTO 1850
6020 IF A$="REAL" THEN MM=1
6030 IF A$="INTEGER" THEN MM=2
6040 IF A$="BOOLEAN" THEN MM=3
6050 IF A$="CHAR" THEN MM=4
6060 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6070 IF A$<>";" THEN A$="FALTA PONTO E
     VÍRGULA":GOTO 1850
6080 GOSUB 3670:REM >>>>> PULL
```

```
6090 IF A$="END" THEN GOTO 6140
6100 VA$(PV)=A$
6110 TV(PV)=MM
6120 PV=PV+1
6130 GOTO 6080
6140 ME$=" "
6150 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
6160 FOR T=1 TO 5
6170 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6180 ME$=ME$+$
6190 NEXT T
6200 GOSUB 2000:REM >>>>> RESTORE
     INDICADORES PASCAL
6210 IF ME$="BEGIN" THEN RETURN
6220 GOTO 5910
6230 REM
6240 REM BLOCO
6250 REM ----------
6260 BA$(LB)-"REM ---- BLOCK ----"
6270 LB=LB+1
6280 NA$=" "
6290 EN=0
6300 FOR T=1 TO 5
6310 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
6320 NA$=NA$+A$
6330 NEXT T
6340 IF NA$<>"BEGIN" THEN A$="FALTA
     BEGIN":GOTO 1850
6350 GOSUB 7090:REM >>>>> OBTER
     DECLARAÇÃO
```

```
6360 IF EN=0 THEN GOTO 6350
6370 REM ------------------------------------------
6380 REM SEGUNDA PASSAGEM
6390 REM ------------------------------------------
6400 FOR T=1 TO LB-1
6410 A$=BA$(T)
6420 IF A$="REM ----END----" THEN
     GOSUB 6460
6430 IF A$="REM ----UNTIL----" THEN
     GOSUB 6970
6440 NEXT T
6450 RETURN
6460 LI=T-1
6470 BE=0
6480 IF BA$(LI)="REM ---BEGIN---" AND
     BE=0 THEN GOTO 6540
6490 IF BA$(LI)="REM ---BEGIN---" THEN
     BE=BE+1
6500 IF BA$(LI)="REM ---END"""" THEN
     BE=BE-1
6510 LI=LI-1
6520 IF LI=0 THEN PRINT "BEGIN/END NÃO
     EMPARELHADOS":END
6530 GOTO 6480
6540 IF BA$(LI-2)="REM ---IF---" THEN
     GOTO 6750
6550 IF BA$(LI-2)="REM ---WHILE---"
     THEN GOTO 6700
6560 IF BA$(LI-2)="REM ---FOR---" THEN
     GOTO 6600
6570 BA$(LI)="REM BEGIN"
6580 BA$(T)='REM END"
6590 RETURN
```

```
6600 BA$(LI)="REM BEGIN"
6610 BA$(LI-2)="REM FOR"
6620 H$=" "
6630 H=5
6640 H$=H$+MID$(BA$(LI-1),H,1)
6650 H=H+1
6660 IF MID$(BA$(LI-1),H,1)="=" THEN GOTO
     6680
6670 GOTO 6640
6680 BA$(T)="NEXT "+H+":REM END"
6690 RETURN
6700 BA$(LI)="REM BEGIN"
6710 BA$(LI-2)="REM WHILE"
6720 BA$(T)="GOTO "STR$(LI-2)*10)+":REM
     END
6730 BA$(LI-1)=BA$(LI-1)+STR$((T+1)*10)
6740 RETURN
6750 EL=0
6760 IF BA$(T+1)="REM ---ELSE---" THEN
     EL=1
6770 BA$(LI-1)="REM THEN"
6780 BA$(LI-3)=BA$(LI-3)+STR$((T+1)*10)
6790 BA$(LI-2)="REM IF"
6800 BA$(LI-4)=BA$(LI-4)+STR$((LI-1)*10)
6810 BA$(LI)="REM BEGIN"
6820 BA$(T)="REM END"
6830 IF EL=0 THEN RETURN
6840 BA$(T+1)="REM ELSE"
6850 BA$(T+2)="REM BEGIN"
6860 EN=0
6870 LL=T+3
6880 IF BA$(LL)="REM ---END---" AND
     EN=0 THEN GOTO 6940
```

```
6890 IF BA$(LL)="REM ---END---" THEN
     EN=EN+1
6900 IF BA$(LL)="REM ---BEGIN---" THEN
     EN=EN-1
6910 LL=LL+1
6920 IF LL>=LB THEN PRINT "SECÇÃO ELSE ESTÁ
     MAL":END
6930 GOTO 6880
6940 BA$(T)="GOTO "+STR$(LL*10)+"REM
     END"
6950 BA$(LL)="REM END"
6960 RETURN
6970 LI=T-1
6980 RE=0
6990 IF BA$(LI)="REM ---REPEAT---" AND
     RE=0 THEN GOTO 7050
7000 IF BA$(LI)="REM ---REPEAT---" THEN
     RE=RE+1
7010 IF BA$(LI)="REM ---UNTIL---" THEN
     RE=RE-1
7020 LI=LI-1
7030 IF LI=0 THEN PRINT "REPEAT/UNTIL NÃO
     EMPARELHADOS":END
7040 GOTO 6990
7050 BA$(T+1)=BA$(T+1)+STR$(LI*10)
7060 BA$(LI)="REM REPEAT"
7070 BA$(T)="REM UNTIL"
7080 RETURN
7090 REM ------------------------------------
7100 REM OBTER DECLARAÇÃO
7110 REM ------------------------------------
7120 NA$=" "
7130 GOSUB 1940:REM >>>>> PROTEGER
```

```
    INDICADORES PASCAL
7140 FOR T=1 TO 6
7150 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7160 NA$=NA$+A$
7170 NEXT T
7180 GOSUB 2000:REM >>>>> RESTORE
    INDICADORES PASCAL
7190 IF LEFT$(NA$,5)="WRITE" THEN GOTO 7480
7200 IF NA$="READLN" THEN GOTO 7780
7210 IF LEFT$(NA$,5)="BEGIN" THEN GOTO 8000
7220 IF LEFT$(NA$,3)="END" THEN GOTO 8090
7230 IF LEFT$(NA$,2)="IF" THEN GOTO 8260
7240 IF LEFT$(NA$,4)="THEN" THEN GOTO 8460
7250 IF LEFT$(NA$,4)="ELSE" THEN GOTO 8550
7260 IF NA$="REPEAT" THEN GOTO 8640
7270 IF LEFT$(NA$,5)="UNTIL" THEN GOTO 8730
7280 IF LEFT$(NA$,5)="WHILE" THEN GOTO 8860
7290 IF LEFT$(NA$,3)="FOR" THEN GOTO 9040
7300 REM ————————————————————
7310 REM ATRIBUIÇÃO
7320 REM ————————————————————
7330 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
7340 IF A$="" THEN A$="VARIÁVEL
    INCORRECTA":GOTO 1850
7350 BA$(LB)=A$+"="
7360 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7370 IF A$<>":" THEN A$="FALTAM DOIS
    PONTOS NA ATRIBUIÇÃO":GOTO 1850
7380 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7390 IF A$<>"=" THEN A$="FALTA SINAL DE
```

```
    IGUAL NA DECLARAÇÃO DE ATRIBUIÇÃO'':
    GOTO 1850
7400 GOSUB 4970:REM >>>>> OBTER
    EXPRESSÃO
7410 IF A$="" THEN A$="AUSÊNCIA DE
    EXPRESSÃO DE ATRIBUIÇÃO'':GOTO 1850
7420 BA$(LB)=BA$(LB)+A$
7430 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
7440 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7450 IF A$<>";" THEN GOSUB 2000:REM >>>>>
    PROTEGER INDICADORES PASCAL
7460 LB=LB+1
7470 RETURN
7480 REM ----------
7490 REM WRITE/WRITELN
7500 REM ----------
7510 FOR T= TO 5
7520 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7530 NEXT T
7540 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
7550 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7560 NA$=A$
7570 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
7580 NA$=NA$+A$
7590 IF NA$<>"LN" THEN NA$="":GOSUB 2000:
    REM >>>>> PROTEGER INDICADORES
    PASCAL
```

```
7600 BA$(LB)="PRINT "
7610 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7620 IF A$<>"(" THEN A$="FALTA PARENTESIS
     EM WRITE/LN)":GOTO 1850
7630 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7640 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7650 GOSUB 2000:REM >>>>> PROTEGER
     INDICADORES PASCAL
7660 IF A$=")" THEN GOSUB 1770:GOTO 7720
7670 GOSUB 4970:REM >>>>> AVALIADOR DA
     EXPRESSÃO
7680 BA$(LB)=BA$(LB)+A$
7690 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7700 IF A$=",," THEN BA$(LB)=BA$(LB)+A$:GOTO
     7630
7710 IF A$<>")" THEN A$="SEPARADOR ILEGAL
     EM INSTRUÇÃO WRITE/LN":GOTO 1850
7720 IF NA$<>"LN" THEN BA$(LB)=BA$(LB)+";"
7730 LB=LB+1
7740 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7750 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7760 IF A$<>";" THEN GOSUB 2000:REM >>>>>
     PROTEGER INDICADORES PASCAL
7770 RETURN
7780 REM ----------
7790 REM READLN
7800 REM ----------
```

```
7810 FOR T=1 TO 6
7820 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7830 NEXT T
7840 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7850 IF A$<>"(" THEN A$="FALTA PARENTESIS
     DE INSTRUÇÃO READLN":GOTO 1850
7860 BA$(LB)="INPUT"
7870 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
7880 IF A$="" THEN A$="FALTA VARIÁVEL":
     GOTO 1850
7890 BA$(LB)=BA$(LB)+A$
7900 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7910 IF A$=")" THEN GOTO 7950
7920 IF A$<>"," THEN A$="FALTA VÍRGULA EM
     INSTRUÇÃO READLN":GOTO 1850
7930 BA$(LB)=BA$(LB)+","
7940 GOTO 7870
7950 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
7960 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
7970 IF A$<>";" THEN GOSUB 2000:REM >>>>>
     PROTEGER INDICADORES PASCAL
7980 LB=LB+1
7990 RETURN
8000 REM ------------------------------------
8010 REM BEGIN
8020 REM ------------------------------------
8030 BA$(LB)=REM ---BEGIN---"
8040 LB=LB+1
```

```
8050 FOR T=1 TO 5
8060 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8070 NEXT T
8080 RETURN
8090 REM ------------------------------------
8100 REM END
8110 REM ------------------------------------
8120 FOR T=1 TO 3
8130 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8140 NEXT T
8150 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
8160 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPACO
8170 GOSUB 2000:REM >>>>> PROTEGER
     INDICADORES PASCAL
8180 IF A$ <> "." THEN GOTO 8230
8190 EN=1
8200 BA$(LB)="END"
8210 LB=LB+1
8220 RETURN
8230 BA$(LB)="REM ---END---"
8240 LB=LB+1
8250 RETURN
8260 REM ------------------------------------
8270 REM IF
8280 REM ------------------------------------
8290 GOSUB 1770:GOSUB 1770:REM >>>>> OBTER
     CARACTER SEGUINTE <> DE ESPAÇO
8300 GOSUB 4970
8310 IF A$="" THEN A$="AUSÊNCIA DE
```

```
    CONDIÇÃO EM INSTRUÇÃO IF":GOTO 1850
8320 BA$(LB)="IF "+A$+" THEN "
8330 LB=LB+1
8340 BA$(LB+1)="REM ---IF---"
8350 BA$(LB)="IF NOT("+A$+") THEN "
8360 LB=LB+2
8370 ME$=" "
8380 GOSUB 1940:REM >>>>> PROTEGER
    INDICADORES PASCAL
8390 FOR T=1 TO 4
8400 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8410 ME$=ME$+A$
8420 NEXT T
8430 IF ME$<>"THEN" THEN A$="FALTA THEN
    NA INSTRUÇÃO IF":GOTO 1850
8440 GOSUB 2000:REM >>>>> PROTEGER
    INDICADORES PASCAL
8450 RETURN
8460 REM ----------
8470 REM THEN
8480 REM ----------
8490 FOR T=1 TO 4
8500 GOSUB 1770:REM >>>>> OBTER CARACTER
    SEGUINTE <> DE ESPAÇO
8510 NEXT T
8520 BA$(LB)="REM ---THEN---"
8530 LB=LB+1
8540 RETURN
8550 REM ----------
8560 REM ELSE
8570 REM ----------
8580 FOR T=1 TO 4
```

```
8590 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8600 NEXT T
8610 BA$(LB)="REM ---ELSE---"
8620 LB=LB+1
8630 RETURN
8640 REM ------------------------------------
8560 REM REPEAT
8660 REM ------------------------------------
8670 FOR T=1 TO 6
8680 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8690 NEXT T
8700 BA$(LB)="REM ---REPEAT---"
8710 LB=LB+1
8720 RETURN
8730 REM ------------------------------------
8740 REM UNTIL
8750 REM ------------------------------------
8760 BA$(LB)="REM ---UNTIL"
8770 LB=LB+1
8780 FOR T=1 TO 5
8790 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8800 NEXT T
8810 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
8820 IF A$="" THEN A$="FALTA EXPRESSÃO EM
     INSTRUÇÃO UNTIL":GOTO 1850
8830 BA$(LB)="IF NOT("+A$+") THEN GOTO "
8840 LB=LB+1
8850 RETURN
8860 REM ------------------------------------
```

```
8870 REM WHILE
8880 REM ------------------------------------------
8890 BA$(LB)="REM ---WHILE---"
8900 LB=LB+1
8910 FOR T=1 TO 5
8920 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8930 NEXT T
8940 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
8950 IF A$="" THEN A$="FALTA EXPRESSÃO EM
     INSTRUÇÃO WHILE":GOTO 1850
8960 BA$(LB)="IF NOT("+A$+") THEN GOTO "
8970 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
8980 MM$=A$
8990 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9000 MM$=MM$+A$
9010 IF MM$<>"DO" THEN A$="FALTA DO EM
     INSTRUÇÃO WHILE":GOTO 1850
9020 LB=LB+1
9030 RETURN
9040 REM ------------------------------------------
9050 REM FOR
9060 REM ------------------------------------------
9070 BA$(LB)="REM ---FOR---"
9080 LB=LB+1
9090 FOR T=1 TO 3
9100 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9110 NEXT T
9120 GOSUB 3280:REM >>>>> OBTER VARIÁVEL
```

```
9130 IF A$="" OR TV(FL)=4 THEN A$=
     "INCORRECTO COMO VARIÁVEL DE
     COMANDO DE CICLO":GOTO 1850
9140 BA$(LB)="FOR "+A$+"="
9150 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9160 IF A$<>":" THEN A$=" FALTAM DOIS
     PONTOS EM INSTRUÇÃO FOR":GOTO 1850
9170 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9180 IF A$<>"=" THEN A$=FALTA SINAL DE
     IGUAL EM INSTRUÇÃO FOR":GOTO 1850
9190 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
9200 IF A$="" THEN A$="MAU VALOR INICIAL
     PARA CICLO":GOTO 1850
9210 BA$(LB)=BA$(LB)+A$+" TO "
9220 GOSUB 1940:REM >>>>> PROTEGER
     INDICADORES PASCAL
9230 ME$=" "
9240 FOR T=1 TO 6
9250 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9260 ME$=ME$+A$
9270 NEXT T
9280 MM=0
9290 IF LEFT$(ME$,2)="TO" THEN
MM=2:ME$="TO"
9300 IF ME$="DOWNTO" THEN MM=6
9310 IF MM=0 THEN A$="FALTA TO/DOWNTO EM
     INSTRUÇÃO FOR":GOTO 1850
9320 GOSUB 2000:REM >>>>> PROTEGER
     INDICADORES PASCAL
```

```
9330 FOR T=1 TO MM
9340 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9350 NEXT T
9360 GOSUB 4970:REM >>>>> OBTER
     EXPRESSÃO
9370 BA$(LB)=BA$(LB)+A$
9380 IF A$="" THEN A$="FALTA EXPRESSÃO EM
     INSTRUÇÃO FOR":GOTO 1850
9390 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9400 MM$=A$
9410 GOSUB 1770:REM >>>>> OBTER CARACTER
     SEGUINTE <> DE ESPAÇO
9420 MM$=MM$+A$
9430 IF MM$<>"DO" THEN A$=" FALTA DO EM
     INSTRUÇÃO FOR":GOTO 1850
9440 IF ME$="DOWNTO" THEN
     BA$(LB)=BA$(LB)+" STEP -1"
9450 LB=LB+1
9460 RETURN
```

APÊNDICE IV

COMPILADOR PARA O ZX SPECTRUM

```
  10 DATA "BEGIN"
  20 DATA "WRITELN ('OLÁ')"
  90 DATA END
1000 REM
1010 REM
1020 REM COMPILADOR
1030 REM
1040 REM PASCAL
1050 REM
1060 REM
1070 REM
1080 REM        Por Jeremy Ruston
1090 REM        Copyright (C) 1982
1100 REM
1110 REM
1120 REM
1130 REM
1140 GO SUB 1310
1150 REM
1160 DIM B$(LE*2+8,50): DIM V$(20,30): DIM T(20):
     DIM C$(20,50): DIM C(20): DIM S$(50,50)
1170 LET SP=50
1180 LET LP=1
```

```
1190 LET CP=1
1200 LET PV=1
1210 LET PC=1
1220 LET D$=""
1230 LET E$=""
1240 LET F$=""
1250 LET G$=""
1260 REM
1270 GO SUB 5170
1280 REM
1290 GO SUB 1500
1300 STOP
1310 REM
1320 REM OBTER PASCAL
1330 REM
1340 RESTORE
1350 PRINT "PASCAL:"
1360 LET LE=1
1370 READ A$
1380 PRINT A$
1390 IF A$="END." THEN GOTO 1420
1400 LET LE=LE+1
1410 GO TO 1370
1420 LET LE=LE+1
1430 DIM P$(LE,50)
1440 RESTORE
1450 FOR T=1 TO LE-1
1460 READ P$(T)
1470 NEXT T
1480 LET P$(LE)="....."
1490 RETURN
1500 REM
1510 REM SAÍDA BASIC
```

```
1520 REM
1530 PRINT "BASIC:"
1540 FOR T=1 TO LB-1
1545 LET Z$=B$(T): GO SUB 9500
1550 PRINT T*10;" ";Z$
1560 NEXT T
1570 RETURN
1580 REM
1590 REM INCREMENTAR INDICADORES PASCAL
1600 REM
1610 LET, CP=CP+1
1620 IF CP > LEN (P$(LP)) THEN LET CP=1: LET
     LP=LP+1
1630 RETURN
1640 REM
1650 REM DECREMENTAR INDICADORES PASCAL
1660 REM
1670 LET CP=CP-1
1680 IF CP=0 THEN LET LP=LP-1: LET CP=LEN
     (PS$(LP))
1690 RETURN
1700 REM
1710 REM OBTER CARACTER PASCAL SEGUINTE
1720 REM E INCREMENTAR INDICADORES
1730 REM
1740 LET A$=P$(LP,CP)
1750 GO SUB 1580
1760 RETURN
1770 REM
1780 REM OBTER CARACTER PASCAL SEGUINTE,
1790 REM IGNORANDO ESPAÇOS, E
1800 REM INCREMENTAR INDICADORES
1810 REM
```

```
1820 GO SUB 1700
1830 IF A$=" " THEN GO TO 1820
1840 RETURN
1850 REM
1860 REM ERRO
1870 REM
1880 PRINT
1890 IF LP<>1 THEN PRINT P$(LP-1)
1900 PRINT P$(LP)
1910 IF LP<>LE THEN PRINT P$(LP+1)
1920 PRINT ">>>>> ERRO -";A$
1930 STOP
1940 REM
1950 REM PROTEGER INDICADORES PASCAL
1960 REM
1970 LET LG=LP
1980 LET CG=CP
1990 RETURN
2000 REM
2010 REM RECUPERAR INDICADORES PASCAL
2020 REM
2030 LET LP=LG
2040 LET CP=CG
2050 RETURN
2060 REM
2070 REM OBTER ALGARISMO
2080 REM
2090 GO SUB 1940
2100 GO SUB 1770
2110 IF A$>="A" AND A$<="Z" THEN RETURN
2120 GO SUB 2000
2130 LET A$=" "
2140 RETURN
```

```
2150 REM
2160 REM OBTER LETRA
2170 REM
2180 GO SUB 1940
2190 GO SUB 1770
2200 IF A$>="A" AND A$<="Z" THEN RETURN
2210 GO SUB 2000
2220 LET A$=" "
2230 RETURN
2240 REM
2250 REM OBTER INTEIRO SEM SINAL
2260 REM
2270 LET H$=" "
2280 GO SUB 2060
2290 IF A$=" " THEN LET A$=H$: RETURN
2300 LET H$=H$+A$
2310 GO TO 2280
2320 REM
2330 REM OBTER IDENTIFICADOR
2340 REM
2350 LET I$=" "
2360 GO SUB 2150
2370 IF A$=" " THEN RETURN
2380 LET I$=I$+A$
2390 GO SUB 2060
2400 LET J$=A$
2410 LET H$=H$+A$
2420 GO SUB 2150
2430 IF A$=" " AND J$=" " THEN LET
     A$=H$: RETURN
2440 GO TO 2380
2450 REM
2460 REM OBTER NÚMERO SEM SINAL
```

```
2470 REM
2480 LET K$=" "
2490 GO SUB 2240
2500 IF A$=" " THEN RETURN
2510 LET K$=K$+A$
2520 GO SUB 1940
2530 GO SUB 1770
2540 GO SUB 2000
2550 IF A$<>"." THEN GO TO 2620
2560 GO SUB 1770
2570 LET K$=K$+A$
2580 GO SUB 2240
2590 IF A$=" " THEN RETURN
2600 LET K$=K$+A$
2610 GO SUB 1940
2620 GO SUB 1770
2630 IF A$<>"E" THEN LET A$=K$: GO SUB 2000:
     RETURN
2640 LET K$=K$+A$
2650 GO SUB 1940
2660 GO SUB 1770
2670 GO SUB 2000
2680 IF A$="+" OR A$="-" THEN LET
     K$=K$+A$: GO SUB 1770
2690 GO SUB 2240
2700 IF A$=" " THEN RETURN
2710 LET A$=K$+A$
2720 RETURN
2730 REM
2740 REM OBTER CADEIA
2750 REM
2760 GO SUB 1940
2770 GO SUB 1770
```

```
2780 GO SUB 2000
2790 IF A$<>"," THEN LET A$=" ": RETURN
2800 LET L$=CHR$ (34)
2810 GO SUB 1770
2820 GO SUB 1700
2830 LET L$=L$+A$
2840 IF A$<>"," THEN GO TO 2820
2850 LET A$=L$( TO LEN (L$) -1)+CHR$ (34)
2860 RETURN
2870 REM
2880 REM OBTER CONSTANTE SEM SINAL
2890 REM
2900 LET CD=CP: LET LD=LP
2910 GO SUB 2320
2920 IF A$=" " THEN LET CP=CD: LET LP=LD:
    GO TO 3000
2930 LET FL=0
2940 FOR E=1 TO PC-1
2950 IF C$(E)=A$ THEN LET FL=E
2960 NEXT E
2970 IF FL=0 THEN LET A$=" ": LET CP=CD: LET
    LP=LD: RETURN
2980 IF C(FL)=4 THEN LET A$=A$+"$"
2990 RETURN
3000 GO SUB 2450
3010 IF A$<>" " THEN RETURN
3020 GO SUB 2730
3030 IF A$<>" " THEN RETURN
3040 RETURN
3050 REM
3060 REM OBTER CONSTANTE
3070 REM
3080 GO SUB 2320
```

```
3090 IF A$=" " THEN GO TO 3170
3100 LET FL=0
3110 FOR F=1 TO PC=1
3120 IF C$(F)=A$ THEN LET FL=F
3130 NEXT F
3140 IF FL=0 THEN LET A$=" ": RETURN
3150 IF C(FL)=4 THEN LET A$=A$+"$"
3160 RETURN
3170 LET M$=" "
3180 GO SUB 1940
3190 GO SUB 1770
3200 GO SUB 2000
3210 IF A$="+" OR A$="-" THEN LET M$=A$:
    GO SUB 1770
3220 GO SUB 2450
3230 IF A$<>" " THEN LET A$=M$+A$: RETURN
3240 GO SUB 2730
3250 IF A$<>" " THEN LET A$=M$+A$: RETURN
3260 LET A$="MAU IDENTIFICADOR DE
    CONSTANTE"
3270 GO TO 1850
3280 REM
3290 REM OBTER VARIÁVEL
3300 REM
3310 LET LD=LP: LET CD=CP
3320 GO SUB 2320
3330 IF A$=" " THEN LET LP=LD: LET CP=CD:
    RETURN
3340 LET FL=0
3350 FOR G=1 TO PV-1
3360 IF V$(G)=A$ THEN LET FL=G
3370 NEXT G
3380 IF T(FL)=4 THEN LET A$=A$+"$"
```

```
3390 IF FL<>0 THEN RETURN
3400 LET A$=" "
3410 LET LP=LD: LET CP=CD
3420 RETURN
3430 REM
3440 REM PUSH VARIÁVEIS
3450 REM
3460 LET S$(SP)=D$
3470 LET S$(SP-1)=E$
3480 LET S$(SP-2)=F$
3490 LET S$(SP-3)=G$
3500 LET SP=SP-4
3510 RETURN
3520 REM
3530 REM PUSH A$ PARA STACK
3540 REM
3550 LET S$(SP)=A$
3560 LET SP=SP-1
3570 RETURN
3580 REM
3590 REM PULL ALGUMAS VARIÁVEIS
3600 REM
3610 LET G$=S$(SP+1)
3620 LET F$=S$(SP+2)
3630 LET E$=S$(SP+3)
3640 LET D$=S$(SP+4)
3650 LET SP=SP+4
3660 RETURN
3670 REM
3680 REM PULL A$ DO STACK
3690 REM
3700 LET SP=SP+1
3710 LET A$=S$(SP)
```

```
3720 RETURN
3730 REM
3740 REM OBTER FACTOR
3750 REM
3760 GO SUB 1940
3770 LET N$=" "
3780 FOR G=1 TO 3
3790 GO SUB 1770
3800 LET N$=N$+A$
3810 NEXT G
3820 GO SUB 2000
3830 IF N$<>"NOT" THEN LET N$=" "
3840 GO SUB 2870
3850 IF A$<>" " THEN LET A$=N$+A$: RETURN
3860 GO SUB 3280
3870 IF A$<>" " THEN LET A$=N$+A$: RETURN
3880 GO SUB 1940
3890 GO SUB 1770
3900 GO SUB 2000
3910 IF A$<>"(" THEN GO TO 4040
3920 GO SUB 1770
3930 LET A$=N$+"("
3940 GO SUB 3520
3950 GO SUB 3430
3960 GO SUB 4970
3970 GO SUB 3580
3980 LET N$=A$
3990 GO SUB 1770
4000 IF A$<>")" THEN LET A$=FALTA
    PARENTESIS": GO TO 1850
4010 GO SUB 3670
4020 LET A$=A$+N$+")"
4030 RETURN
```

```
4040 GO SUB 1940
4050 LET O$=N$
4060 LET N$=" "
4070 FOR G=1 TO 5
4080 GO SUB 1770
4090 LET N$=N$+A$
4100 NEXT G
4110 GO SUB 2000
4120 LET GN=0
4130 IF N$( TO 3)="ABS" THEN LET GN=3: LET
     A$="ABS"
4140 IF N$( TO 3)="ATN" THEN LET GN=3: LET
     A$="ATN"
4150 IF N$( TO 3)="CHR" THEN LET GN=3: LET
     A$="CHR"
4160 IF N$( TO 3)="COS" THEN LET GN=3: LET
     A$="COS"
4170 IF N$( TO 3)="EXP" THEN LET GN=3: LET
     A$="EXP"
4180 IF N$( TO 2)="LN" THEN LET GN=2: LET
     A$="LN"
4190 IF N$( TO 3)="ODD" THEN LET GN=3: LET
     A$="FN OD"
4200 IF N$( TO 3)="ORD" THEN LET GN=3: LET
     A$="CODE"
4210 IF N$( TO 4)="PRED" THEN LET GN=4: LET
     A$="FN PR"
4220 IF N$( TO 5)="ROUND" THEN LEN GN=5:
     LET A$="FN RO"
4230 IF N$( TO 3)="SIN" THEN LEN GN=3: LET
     A$="SIN"
4240 IF N$( TO 4)="SQRT" THEN LET GN=4: LET
     A$="SQR"
```

```
4250 IF N$( TO 3)="SQR" THEN LET GN=3: LET
     A$="FN SQ"
4260 IF N$( TO 4)="SUCC" THEN LET GN=4: LET
     A$="FN SU"
4270 IF N$( TO 5)="TRUNC" THEN LET GN=5: LET
     A$="INT"
4280 IF GN=0 THEN LET A$="MÁ CHAMADA DE
     FUNÇÃO": GO TO 1850
4290 LET O$=O$+A$
4300 FOR T=1 TO GN
4310 GO SUB 1770
4320 NEXT G
4330 GO SUB 1770
4340 IF A$ <>"(" THEN LET A$="FALTA
     PARENTESIS NA FUNÇÃO": GO TO 1850
4350 LET O$=O$+A$
4360 LET A$=O$
4370 GO SUB 3520
4380 GO SUB 3430
4390 GO SUB 4970
4400 GO SUB 3580
4410 LET O$=A$
4420 GOSUB 3670
4430 LET O$=A$+O$
4440 GO SUB 1770
4450 IF A$ <>")" THEN LET A$="FALTA
     PARENTESIS NA CHAMADA DE FUNÇÃO":
     GO TO 1850
4460 LET A$=O$+")"
4470 RETURN
4480 REM
4490 REM OBTER TERMO
4500 REM
```

```
4510 GO SUB 3730
4520 IF A$="" THEN LET A$="FALTA FACTOR":
     GO TO 1850
4530 LET D$=A$
4540 LET E$=" "
4550 GO SUB 1940
4560 FOR J=1 TO 3
4570 GO SUB 1770
4580 LET E$=E$+A$
4590 NEXT J
4600 GO SUB 2000
4610 IF E$<>"DIV" AND E$<>"MOD" AND
     E$<>"AND" AND E$(1)<>"*" AND
     E$(1)<>"/" THEN LET A$=D$: RETURN
4620 LET JA=1
4630 IF E$="AND" OR E$="DIV" OR E$="MOD"
     THEN LET JA=3
4640 FOR T=1 TO JA
4650 GO SUB 1770
4660 NEXT T
4670 GO SUB 3730
4680 IF A$="" THEN LET A$="FALTA FACTOR":
     GO TO 1850
4690 LET D$=D$+E$( TO JA)+A$
4700 GO TO 4540
4710 REM
4720 REM EXPRESSÃO SIMPLES
4730 REM
4740 GO SUB 1940
4750 GO SUB 1770
4760 GO SUB 2000
4770 LET F$=" "
4780 IF A$="+" OR A$="-" THEN LET F$=A$:
```

```
    GO SUB 1770
4790 GO SUB 4480
4800 IF A$="'" THEN LET A$="FALTA TERMO":
    GO TO 1850
4810 LET F$=F$+A$
4820 GO SUB 1940
4830 LET Q$=" "
4840 GO SUB 1770
4850 LET Q$=Q$+A$
4860 GO SUB 1770
4870 LET Q$=Q$+A$
4880 GO SUB 2000
4890 IF Q$(1) <> "+" AND Q$(1) <> "-" AND
    Q$ <> "OR" THEN LET A$=F$: RETURN
4900 LET F$=F$+Q$(1)
4910 GO SUB 1770
4920 IF Q$="OR" THEN LET F$=F$+"R":
    GO SUB 1770
4930 GO SUB 4480
4940 LET F$=F$+A$
4950 IF A$=" " THEN LET A$=F$: RETURN
4960 GO TO 4820
4970 REM
4980 REM OBTER EXPRESSÃO
4990 REM
5000 GO SUB 4710
5010 LET G$=A$
5020 GO SUB 1940
5030 LET R$=" "
5040 FOR M=1 TO 2
5050 GO SUB 1770
5060 LET R$=R$+A$
5070 NEXT M
```

```
5080 GO SUB 2000
5090 LET T$ = R$(1)
5100 IF T$ <> " < " AND T$ <> " = " AND
     T$ <> " > " THEN LET A$ = G$: RETURN
5110 IF R$ = " <> " OR R$ = "< = " OR R$ = " > = "
     THEN GO SUB 1770: GO SUB 1770: LET A$ = R$:
     GO TO 5130
5120 GO SUB 1770: LET A$ = T$
5130 LET G$ = G$ + A$
5140 GO SUB 4710
5150 LET A$ = G$ + A$
5160 RETURN
5170 REM
5180 REM
5190 REM
5200 REM
5210 REM
5220 REM
5230 REM
5240 REM PROGRAMA PRINCIPAL
5250 REM
5260 REM
5270 REM
5280 REM
5290 REM
5300 LET B$(1) = "DEF FN
     OD(X) = ((X - INT(X/2) * 2) = 1)"
5310 LET B$(2) = "DEF FN PR(X) = X - 1
5320 LET B$(3) = "DEF FN SU(X) = X + 1"
5330 LET B$(4) = DEF FN RO(X) = INT(X + 0.5)"
5340 LET B$(5) = "DEF FN SQ(X) = X * X"
5350 LET LB = 6
5360 GO SUB 5400
```

```
5370 GO SUB 5790
5380 GO SUB 6230
5390 RETURN
5400 REM
5410 REM SECÇÃO CONST
5420 REM
5430 LET C$(1)="TRUE": LET C$(2)="FALSE":
     LET PC=3
5440 LET B$(LB)="REM ——— CONST ————"
5450 LET B$(LB+1)="TRUE=-1"
5460 LET B$(LB+2)="FALSE=0"
5470 LET LB=LB+3
5480 LET U$=" "
5490 GO SUB 1940
5500 FOR M=1 TO 5
5510 GO SUB 1770
5520 LET U$=U$+A$
5530 NEXT M
5540 IF U$ <> "CONST" THEN GO SUB 2000:
     RETURN
5550 GO SUB 2320
5560 IF A$="" THEN LET A$="FALTA SINAL DE
     IGUAL": GO TO 1850
5570 LET C$(PC)=A$
5580 GO SUB 1770
5590 IF A$<>"=" THEN LET A$="FALTA SINAL
     DE IGUAL": GO TO 1850
5600 GO SUB 3050
5610 IF A$="" THEN LET A$=" FALTA
     CONSTANTE": GO TO 1850
5620 IF A$(1)=CHR$ (34) THEN LET T(PC)=4
5630 LET B$(LB)=C$(PC)
5640 IF T(PC)=4 THEN LET Z$=B$(LB):
```

```
    GO SUB 9500: LET B$(LB)=Z$+"$"
5650 LET Z$=B$(LB): GO SUB 9500: LET
    B$(LB)=Z$+"="+A$
5660 LET LB=LB+1
5670 LET PC=PC+1
5680 GO SUB 1770
5690 IF A$<>";" THEN LET A$="FALTA PONTO E
    VÍRGULA": GO TO 1850
5700 GO SUB 1940
5710 LET U$=" "
5720 FOR M=1 TO 5
5730 GO SUB 1770
5740 LET U$=U$+A$
5750 NEXT M
5760 GO SUB 2000
5770 IF U$="BEGIN" OR U$( TO 3)="VAR" THEN
    RETURN
5780 GO TO 5550
5790 REM
5800 REM SECÇÃO VAR
5810 REM
5820 GO SUB 1940
5830 LET U$=" "
5840 FOR T=1 TO 3
5850 GO SUB 1770
5860 LET U$=U$+A$
5870 NEXT T
5880 GO SUB 2000
5890 IF U$ <> "VAR" THEN RETURN
5900 GO SUB 1770
5910 LET A$="END"
5920 GO SUB 3520
5930 GO SUB 2320
```

```
5940 IF A$="'" THEN A$="FALTA
    IDENTIFICADOR": GO TO 1850
5950 GOSUB 3520
5960 GO SUB 1770
5970 IF A$=":" THEN GO TO 6000
5980 IF A$<>"," THEN LET A$="FALTA VÍRGULA
    NA SECÇÃO VAR": GO TO 1850
5990 GO TO 5930
6000 GO SUB 2320
6010 IF A$<>"REAL" AND A$<>"INTEGER" AND
    A$<>"CHAR" AND A$<>"BOOLEAN" THEN
    A$="TIPO ILEGAL": GO TO 1850
6020 IF A$="REAL" THEN LET MM=1
6030 IF A$="INTEGER" THEN LET MM=2
6040 IF A$="BOOLEAN" THEN LET MM=3
6050 IF A$="CHAR" THEN LET MM=4
6060 GO SUB 1770
6070 IF A$<>";" THEN LET A$="FALTA PONTO E
    VÍRGULA": GO TO 1850
6080 GO SUB 3670
6090 IF A$="END" THEN GO TO 6140
6100 LET V$(PV)=A$
6110 LET T(PV)=MM
6120 LET PV=PV+1
6130 GO TO 6080
6140 LET U$=" "
6150 GO SUB 1940
6160 FOR T=1 TO 5
6170 GO SUB 1770
6180 LET U$=U$+A$
6190 NEXT T
6200 GO SUB 2000
6210 IF U$="BEGIN" THEN RETURN
```

```
6220 GO TO 5910
6230 REM
6240 REM BLOCO
6250 REM
6260 LET B$(LB)="REM ---- BLOCK ----"
6270 LET LB=LB+1
6280 LET U$=" "
6290 LET EN=0
6300 FOR T=1 TO 5
6310 GO SUB 1770
6320 LET U$=U$+A$
6330 NEXT T
6340 IF U$<>"BEGIN" THEN LET A$="FALTA
    BEGIN": GO TO 1850
6350 GO SUB 7090
6360 IF EN=0 THEN GO TO 6350
6370 REM
6380 REM SEGUNDA PASSAGEM
6390 REM
6400 FOR T=1 TO LB-1
6410 LET A$=B$(T)
6420 IF A$( TO 13)="REM ----END----"
    THEN GO SUB 6460
6430 IF A$( TO 15)="REM ----UNTIL----"
    THEN GO SUB 6970
6440 NEXT T
6450 RETURN
6460 LET LI=T-1
6470 LET BE=0
6480 IF B$(LI, TO 15)="REM ---BEGIN---"
    AND BE=0 THEN GO TO 6540
6490 IF B$(LI, TO 15)="REM ---BEGIN---"
    THEN LET BE=BE+1
```

```
6500 IF B$(LI, TO 13)="REM ---END"""" THEN
     LET BE=BE-1
6510 LET LI=LI-1
6520 IF LI=0 THEN PRINT "BEGIN/END NÃO
     EMPARELHADOS": GOTO 1850
6530 GO TO 6480
6540 IF B$(LI-2, TO 12)="REM ---IF---"
     THEN GO TO 6750
6550 IF B$(LI-2, TO 15)="REM ---WHILE---"
     THEN GO TO 6700
6560 IF B$(LI-2, TO 13)="REM ---FOR---"
     THEN GO TO 6600
6570 LET B$(LI)="REM BEGIN"
6580 LET B$(T)='REM END"
6590 RETURN
6600 LET B$(LI)="REM BEGIN"
6610 LET B$(LI-2)="REM FOR"
6620 LET U$=" "
6630 LET H=5
6640 LET U$=U$+B$(LI-1,H)
6650 LET H=H+1
6660 IF B$(LI-1,H)="=" THEN GO TO 6680
6670 GO TO 6640
6680 LET B$(T)="NEXT "+H$+": REM END"
6690 RETURN
6700 LET B$(LI)="REM BEGIN"
6710 LET B$(LI-2)="REM WHILE"
6720 LET B$(T)="GO TO "+STR$
     ((LI-2)*10)+":REM END"
6730 LET Z$=B$(LI-1): GO SUB 9500: LET
B$(LI-1)=Z$+STR$ ((T+1)*10)
6740 RETURN
6750 LET EL=0
```

```
6760 IF B$(T+1, TO 14)="REM ---ELSE---"
     THEN LET EL=1
6770 LET B$(LI-1)="REM THEN"
6780 LET Z$=B$(LI-3): GO SUB 9500: LET
     B$(LI-3)=Z$+STR$ ((T+1)*10)
6790 LET B$(LI-2)="REM IF"
6800 LET Z$=B$(LI-4): LET B$(LI-4)=Z$+STR$
     ((LI-1)*10)
6810 LET B$(LI)="REM BEGIN"
6820 LET B$(T)="REM END"
6830 IF EL=0 THEN RETURN
6840 LET B$(T+1)="REM ELSE"
6850 LET B$(T+2)="REM BEGIN"
6860 LET EN=0
6870 LET LL=T+3
6880 IF B$(LL, TO 13)="REM ---END---" AND
     EN=0 THEN GO TO 6940
6890 IF B$(LL, TO 13)="REM ---END---"
     THEN LET EN=EN+1
6900 IF B$(LL, TO 15)="REM ---BEGIN---"
     THEN LET EN=EN-1
6910 LET LL=LL+1
6920 IF LL>=LB THEN PRINT "MÁ SECÇÃO
     ELSE": STOP
6930 GO TO 6880
6940 LET B$(T)="GOTO "+STR$ (LL*10)+"REM
     END"
6950 LET B$(LL)="REM END"
6960 RETURN
6970 LET LI=T-1
6980 LET RE=0
6990 IF B$(LI, TO 16)="REM ---REPEAT---"
     AND RE=0 THEN GO TO 7050
```

```
7000 IF B$(LI, TO 16)="REM ---REPEAT---"
    THEN LET RE=RE+1
7010 IF B$(LI, TO 15)="REM ---UNTIL---"
    THEN LET RE=RE-1
7020 LI=LI+1
7030 IF LI=0 THEN PRINT "REPEAT/UNTIL NÃO
    EMPARELHADOS": STOP
7040 GO TO 6990
7050 LET Z$=B$(T+1): GO SUB 9500: LET
    B$(T+1)=Z$+STR$ (LI*10)
7060 LET B$(LI)="REM REPEAT"
7070 LET B$(T)="REM UNTIL"
7080 RETURN
7090 REM
7100 REM OBTER DECLARAÇÃO
7110 REM
7120 LET U$=" "
7130 GO SUB 1940
7140 FOR T=1 TO 6
7150 GO SUB 1770
7160 LET U$=U$+A$
7170 NEXT T
7180 GO SUB 2000
7190 IF U$( TO 5)="WRITE" THEN GO TO 7480
7200 IF U$="READLN" THEN GO TO 7780
7210 IF U$( TO 5)="BEGIN" THEN GO TO 8000
7220 IF U$( TO 3)="END" THEN GO TO 8090
7230 IF U$( TO 2)="IF" THEN GO TO 8260
7240 IF U$( TO 4)="THEN" THEN GO TO 8460
7250 IF U$( TO 4)="ELSE" THEN GO TO 8550
7260 IF U="REPEAT" THEN GO TO 8640
7270 IF U$( TO 5)="UNTIL" THEN GO TO 8730
7280 IF U$( TO 5)="WHILE" THEN GO TO 8860
```

```
7290 I  U$  TO 3)=  FOR   T  EN GO TO 9040
7300 REM
7310 REM ATRIBUIÇÃO
7320 REM
7330 GO SUB 3280
7340 IF A$="" THEN LET A$="VARIÁVEL
     INCORRECTA": GO TO 1850
7350 LET B$(LB)="LET "+A$+"="
7360 GO SUB 1770
7370 IF A$<>":" THEN LET A$="FALTAM DOIS
     PONTOS": GO TO 1850
7380 GO SUB 1770
7390 IF A$<>"=" THEN LET A$="FALTA SINAL
     DE IGUAL": GO TO 1850
7400 GO SUB 4970
7410 IF A$="" THEN LET A$="FALTA
     EXPRESSÃO": GO TO 1850
7420 LET Z$=B$(LB): GO SUB 9500: LET
     B$(LB)=Z$+A$
7430 GO SUB 1940
7440 GO SUB 1770
7450 IF A$<>";" THEN GO SUB 2000
7460 LET LB=LB+1
7470 RETURN
7480 REM
7490 REM WRITE/WRITELN
7500 REM
7510 FOR T=  TO 5
7520 GOSUB 1770
7530 NEXT T
7540 GO SUB 1940
7550 GO SUB 1770
7560 LET U$=A$
```

```
7570 GO SUB 1770
7580 LET U$=U$+A$
7590 IF U$<>"LN" THEN LET U$="": GO SUB 2000
7600 LET B$(LB)="PRINT "
7610 GO SUB 1770
7620 IF A$<>"(" THEN LET A$="FALTA
     PARENTESIS": GO TO 1850
7630 GO SUB 1940
7640 GO SUB 1770
7650 GO SUB 2000
7660 IF A$=")" THEN GO SUB 1770: GO TO 7720
7670 GO SUB 4970
7680 LET Z$=B$(LB): GO SUB 9500: LET
     B$(LB)=Z$+A$
7690 GO SUB 1770
7700 IF A$="," THEN LET Z$=B$(LB): GO SUB
     9500: LET B$(LB)=Z$+A$: GO TO 7630
7710 IF A$<>")" THEN LET A$="SEPARADOR
     INCORRECTO": GO TO 1850
7720 IF U$<>"LN" THEN LET Z$=B$(LB): GO SUB
     9500: LET B$(LB)=Z$+";"
7730 LET LB=LB+1
7740 GO SUB 1940
7750 GO SUB 1770
7760 IF A$<>";" THEN GO SUB 2000
7770 RETURN
7780 REM
7790 REM READLN
7800 REM
7810 FOR T=1 TO 6
7820 GO SUB 1770
7830 NEXT T
7840 GO SUB 1770
```

```
7850 IF A$<>"(" THEN LET A$="FALTA
     PARENTESIS": GO TO 1850
7860 LET B$(LB)="INPUT"
7870 GO SUB 3280
7880 IF A$="" THEN LET A$="FALTA
     VARIÁVEL": GO TO 1850
7890 LET Z$=B$(LB): GO SUB 9500: LET
    B$(LB)=Z$+A$
7900 GO SUB 1770
7910 IF A$=")" THEN GO TO 7950
7920 IF A$<>"," THEN LET A$="FALTA
    VÍRGULA": GO TO 1850
7930 LET Z$=B$(LB): GO SUB 9500: LET
    B$(LB)=Z$+","
7940 GO TO 7870
7950 GO SUB 1940
7960 GO SUB 1770
7970 IF A$<>";" THEN GO SUB 2000
7980 LET LB=LB+1
7990 RETURN
8000 REM
8010 REM BEGIN
8020 REM
8030 LET B$(LB)=REM ---BEGIN---"
8040 LET LB=LB+1
8050 FOR T=1 TO 5
8060 GO SUB 1770
8070 NEXT T
8080 RETURN
8090 REM
8100 REM END
8110 REM
8120 FOR T=1 TO 3
```

```
8130 GO SUB 1770
8140 NEXT T
8150 GO SUB 1940
8160 GO SUB 1770
8170 GO SUB 2000
8180 IF A$ <>"." THEN GO TO 8230
8190 LET EN=1
8200 LET B$(LB)="STOP"
8210 LET LB=LB+1
8220 RETURN
8230 LET B$(LB)="REM ---END---"
8240 LET LB=LB+1
8250 RETURN
8260 REM
8270 REM IF
8280 REM
8290 GO SUB 1770: GO SUB 1770
8300 GO SUB 4970
8310 IF A$="" THEN LET A$="FALTA CONDIÇÃO:
     GO TO 1850
8320 LET B$(LB)="IF "+A$+" THEN "
8330 LET LB=LB+1
8340 LET B$(LB-1)="REM ---IF---"
8350 LET B$(LB)="IF NOT(" A$+) THEN "
8360 LET LB=LB+2
8370 LET U$=""
8380 GO SUB 1940
8390 FOR T=1 TO 4
8400 GO SUB 1770
8410 LET U$=U$+A$
8420 NEXT T
8430 IF U$<>"THEN" THEN LET A$="FALTA
     THEN": GO TO 1850
```

```
8440 GO SUB 2000
8450 RETURN
8460 REM
8470 REM THEN
8480 REM
8490 FOR T=1 TO 4
8500 GO SUB 1770
8510 NEXT T
8520 LET B$(LB)="REM ---THEN---"
8530 LET LB=LB+1
8540 RETURN
8550 REM
8560 REM ELSE
8570 REM
8580 FOR T=1 TO 4
8590 GO SUB 1770
8600 NEXT T
8610 LET B$(LB)="REM ---ELSE---"
8620 LET LB=LB+1
8630 RETURN
8640 REM
8560 REM REPEAT
8660 REM
8670 FOR T=1 TO 6
8680 GO SUB 1770
8690 NEXT T
8700 LET B$(LB)="REM ---REPEAT---"
8710 LET LB=LB+1
8720 RETURN
8730 REM
8740 REM UNTIL
8750 REM
8760 LET B$(LB)="REM ---UNTIL"
```

```
8770 LET LB=LB+1
8780 FOR T=1 TO 5
8790 GO SUB 1770
8800 NEXT T
8810 GO SUB 4970
8820 IF A$="" THEN LET A$="FALTA
     EXPRESSÃO": GO TO 1850
8830 LET B$(LB)="IF NOT("+A$+") THEN GOTO "
8840 LET LB=LB+1
8850 RETURN
8860 REM
8870 REM WHILE
8880 REM
8890 LET B$(LB)="REM ---WHILE---"
8900 LET LB=LB+1
8910 FOR T=1 TO 5
8920 GO SUB 1770
8930 NEXT T
8940 GO SUB 4970
8950 IF A$="" THEN LET A$="FALTA
     EXPRESSÃO": GO TO 1850
8960 LET B$(LB)="IF NOT("+A$+") THEN GOTO "
8970 GO SUB 1770
8980 LET U$=A$
8990 GO SUB 1770
9000 LET U$=U$+A$
9010 IF U$<>"DO" THEN LET A$="FALTA DO":
     GO TO 1850
9020 LET LB=LB+1
9030 RETURN
9040 REM
9050 REM FOR
9060 REM
```

```
9070 LET B$(LB)="REM ---FOR---"
9080 LET LB=LB+1
9090 FOR T=1 TO 3
9100 GO SUB 1770
9110 NEXT T
9120 GO SUB 3280
9130 IF A$="" OR T(FL)=4 THEN LET
    A$="VARIÁVEL INCORRECTA": GO TO 1850
9140 LET B$(LB)="FOR "+A$+"="
9150 GO SUB 1770
9160 IF A$<>":" THEN LET A$=" FALTAM DOIS
    PONTOS": GO TO 1850
9170 GO SUB 1770
9180 IF A$<>"=" THEN LET A$="FALTA SINAL
    DE IGUAL": GO TO 1850
9190 GO SUB 1970
9200 IF A$="" THEN LET A$="FALTA
    EXPRESSÃO": GO TO 1850
9210 LET Z$=B$(LB): GO SUB 9500: LET
    B$(LB)=Z$+A$+" TO "
9220 GO SUB 1940
9230 LET U$=" "
9240 FOR T=1 TO 6
9250 GO SUB 1770
9260 LET U$=U$+A$
9270 NEXT T
9280 LET MM=0
9290 IF U$( TO 2)="TO" THEN LET MM=2: LET
    U$="TO"
9300 IF U$="DOWNTO" THEN LET MM=6
9310 IF MM=0 THEN LET A$="FALTA TO
    DOWNTO": GO TO 1850
9320 GO SUB 2000
```

```
9330 FOR T=1 TO MM
9340 GO SUB 1770
9350 NEXT T
9360 GO SUB 4970
9370 LET Z$=B$(LB): GO SUB 9500: LET
     B$(LB)=Z$+A$
9380 IF A$="" THEN LET A$="FALTA
     EXPRESSÃO": GO TO 1850
9390 GO SUB 1770
9400 LET W$=A$
9410 GO SUB 1770
9420 LET W$=W$+A$
9430 IF W$<>"DO" THEN LET A$=" FALTA DO":
     GO TO 1850
9440 IF U$="DOWNTO" THEN LET Z$=B$(LB):
     GO SUB 9500: LET B$(LB)=Z$+" STEP -1"
9450 LET LB=LB+1
9460 RETURN
9470 REM
9480 REM
9490 REM
9500 REM RETIRAR ESPAÇOS DE Z$
9510 REM
9520 LET SPC=LEN Z$
9530 IF Z$(SPC)<>" " THEN GO TO 9600
9540 LET SPC=SPC-1
9550 IF SPC<>0 THEN GO TO 9530
9560 LET Z$=" ": RETURN
9600 LET Z$=Z$( TO SPC)
9610 RETURN
```

ÍNDICE

Este livro acabou de se
imprimir em Maio de 1984
para a
EDITORIAL PRESENÇA
na
Tipografia Nunes, Lda.
Porto

Este manual foi concebido com duas finalidades: ajudar os utilizadores de computadores a decidir se o Pascal é uma linguagem que serve as suas necessidades e, em caso afirmativo, ensiná-los a programar em Pascal. Uma primeira parte do livro destina-se tanto àqueles leitores que utilizam o pseudo-compilador fornecido no final da obra como àqueles que possuem um compilador Pascal profissional; a segunda parte trata apenas daquelas características do Pascal que só podem ser usadas com eficácia por um possuidor da versão comercial.

EDITORIAL PRESENÇA